entropy

MDPI

Article

Lossless Image Compression Using Context-Dependent Linear Prediction Based on Mean Absolute Error Minimization

Grzegorz Ulacha * and Mirosław Łazoryszczak *

Faculty of Computer Science and Information Technology, West Pomeranian University of Technology in Szczecin, ul. Żołnierska 49, 71-210 Szczecin, Poland

* Correspondence: grzegorz.ulacha@zut.edu.pl (G.U.); miroslaw.lazoryszczak@zut.edu.pl (M.Ł.)

Abstract: This paper presents a method for lossless compression of images with fast decoding time and the option to select encoder parameters for individual image characteristics to increase compression efficiency. The data modeling stage was based on linear and nonlinear prediction, which was complemented by a simple block for removing the context-dependent constant component. The prediction was based on the Iterative Reweighted Least Squares (*IRLS*) method which allowed the minimization of mean absolute error. Two-stage compression was used to encode prediction errors: an adaptive Golomb and a binary arithmetic coding. High compression efficiency was achieved by using an author's context-switching algorithm, which allows several prediction models tailored to the individual characteristics of each image area. In addition, an analysis of the impact of individual encoder parameters on efficiency and encoding time was conducted, and the efficiency of the proposed solution was shown against competing solutions, showing a 9.1% improvement in the bit average of files for the entire test base compared to *JPEG-LS*.

Keywords: entropy coding; iterative reweighted least squares; context-dependent coding; deep learning; lossless image coding

check for updates

Citation: Ulacha, G.; Łazoryszczak, M. Lossless Image Compression Using Context-Dependent Linear Prediction Based on Mean Absolute Error Minimization. *Entropy* **2024**, *26*, 1115. https://doi.org/10.3390/e26121115

T. Aaron Gulliver and Jun Chen

Received: 18 November 2024
Revised: 13 December 2024
Accepted: 18 December 2024
Published: 20 December 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In many image processing and archiving applications, the storage of lossless data is required. Significant applications of lossless compression of images and video sequences include archiving of 2D, 3D, and 4D medical images (3D video sequences) [1–5], astronomical images, and compression of satellite images [6,7]. In addition, lossless mode is often required in the graphic processing of photos and advertising materials and the post-production of television broadcasts and films [8].

Modern compression methods usually take into account two steps: data decorrelation and compression by one of the efficient entropy methods, among which the most effective are arithmetic coding, Huffman coding, and its specific variations (Golomb codes, Rice codes) [9]. The decorrelation is designed to significantly reduce data redundancy resulting from the high level of correlation between neighboring pixels. For this purpose, simple prediction methods (*JPEG-LS* [10], *CALIC* [11]), both linear and non-linear, described in works [12–16], are most often used. There are also other approaches, such as wavelet transforms (*JPEG 2000* [17] and *SPIHT* [18]), where their specific features are used to record image data efficiently.

The highest efficiency of lossless compression is those with high implementation complexity, which belongs to the class of time-symmetric (where the decoding time is as long as the encoding time). In such cases, the methods are based on linear prediction models with backward adaptation, using mechanisms known from the literature such as *RLS* [19], *OLS* [20–22], or *WLS* [23,24], where also the coding of each successive pixel is accompanied by a procedure for adapting or redetermining the coefficients of the linear predictor. A cascaded combination of predictors (with backward adaptation) of the *WLS*/*NLMS* type

achieves the highest compression efficiency, as shown in [23], but this is associated with decoding times that are too long. In addition, a branch of lossless image compression methods based on deep learning using neural networks has been developed recently. So, we have devoted a separate Section 4.2 to this issue, pointing out some disadvantages of this solution.

In this work, the authors proposed lossless compression of images with asymmetric encoding and decoding times with relatively short decoding times (since the encoding operation is usually performed once and the decoding multiple times) while maintaining high compression efficiency. This is made possible using linear prediction with forward adaptation (Section 2.1) supported by fast, nonlinear prediction based on simplified blending of constant predictors (Section 2.2). Section 2.3 briefly characterizes the entire data processing in the encoder and decoder. At the same time, the rest of the paper focuses on the main block using linear prediction (an improved *IRLS* algorithm to minimize mean absolute error described in Section 2.4) and how to increase prediction efficiency by introducing context switching (Section 3.1). Section 3.2 analyzes the impact of the proposed method's parameter selection on its efficiency and encoding and decoding times. Section 4 compares the compression efficiency with other codecs known from the literature.

2. Using Linear Prediction for Image Compression

2.1. Basics of Predictive Modeling

In the case of two-dimensional signals such as images, the data modeling stage involves removing as much of the causal information occurring between adjacent pixels as possible. Therefore, the predicted value $\hat{x}_n$ ([] denotes rounding operation in Formula (1)) of the encoded pixel x_n is determined, and then the differences between them are encoded:

$$e_n = x_n - [\hat{x}_n]. \tag{1}$$

Such differences, e_n, referred to as prediction errors, are often small values oscillating near zero with a probability distribution close to the Laplace distribution. A linear predictor of order r can be used as the predicted value:

$$\hat{x}_n = \sum_{j=1}^{r} b_j \cdot P(j), \tag{2}$$

where the elements b_j are the prediction coefficients that make up the vector $\mathbf{B} = [b_1, b_2, \ldots, b_r]$, while $P(j)$ denotes the jth pixel adjacent to the currently encoded x_n (see Figure 1 [23] showing the 46 numbered nearest neighbors of the encoded pixel $P(j)$, where a given jth number indicates a pixel with a value of $P(j)$ and a prediction error of $e(j)$). This is a relative numbering referring to the currently encoded pixel (with an index of 0; hence, you can use the relative assignment of $x_n = P(0)$ and $e_n = e(0)$ for the current prediction error). The image is encoded row by row, from left to right; hence, if we consider the nth time moment, the currently encoded pixel $x_n = P(0)$, and the pixel encoded at a time $(n-2)$, i.e., x_{n-2}, has a relative index $P(5)$, according to Figure 1.

The simplest predictors are just the values of individual pixels (with $r = 1$ and $b_1 = 1$), e.g., $\hat{x}_n = P(4)$ means that the predicted value of the encoded pixel x_n is the value of the upper right neighbor.

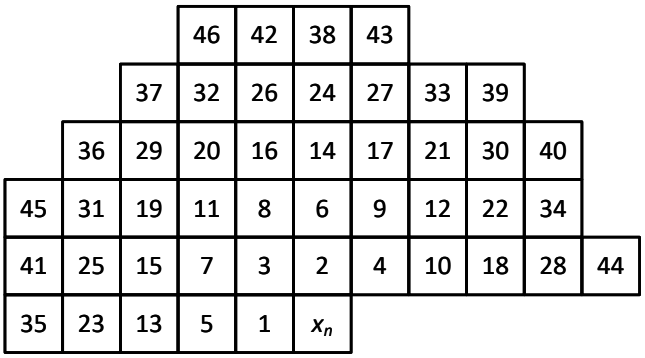


Figure 1. Numbering of neighborhood pixels relative to the currently encoded pixel x_n.

2.2. Non-Linear Predictive Model Support

The solution proposed here assumes an asymmetric approach between encoding and decoding times. Thus, increasing the implementation complexity of the encoder becomes acceptable by introducing forward adaptation, which requires access to the data of the whole image for multiple preliminary data modeling with parameter tuning before the final encoding stage. On the other hand, the decoding process is expected to have a reasonably short decoding time. This calls for a predictive method with low implementation complexity. The compromise solution is to use linear prediction enhanced by two additional techniques to obtain an auxiliary predicted value based on nonlinear prediction. This applies to GAP_+ (Gradient-Adjusted Predictor) [25] and $GBSW_+$ (Gradient-Based Selection and Weighting) [26], which, despite using backward adaptation, have low implementation complexity. The "+" symbol in the name denotes enhancements to the original algorithms, which we discussed in [27]. The final form of the predicted value, which is an expansion of the Formula (2) for the encoder proposed in this work, is determined as follows:

$$\hat{x}_n = b_1 \cdot \hat{x}_{GBSW_+} + b_2 \cdot \hat{x}_{GAP_+} + \sum_{j=3}^{r} b_j \cdot P(j-2). \tag{3}$$

This means that, for example, with order $r = 14$, in addition to the auxiliary predicted values $\hat{x}_{GBSW_+}$ and $\hat{x}_{GAP_+}$, we also use the 12 nearest neighbors whose Euclidean distance from the encoded pixel x_n is less than 3 (see Figure 1).

After an in-depth analysis of the *GBSW* method [26], its strong affinity with the simple blending version of the [28,29] predictors becomes apparent, as will be demonstrated below. This leads to the possibility of further improving $GBSW_+$ relative to the version presented in the work of [27], where a weighted average of two of the four nearest neighborhood pixels constituting the set of potential sub-predictors is determined as the predicted value of $\hat{x}_{GBSW_+}$. The predicted value depends on the initial determination of the two currently most minor of the four gradients d_W, d_N, d_{NW} and d_{NE} (associated, respectively, with simple predictors of order one: $P(1)$, $P(2)$, $P(3)$ and $P(4)$ referred to hereafter as sub-predictors), where the subscripts indicate the direction of the gradient: W—west, N—north, NW—northwest and NE—northeast.

For example, if the two smallest values are d_W and d_N, the prediction value is determined as follows:

$$\hat{x}_{GBSW_+} = \frac{d_W \cdot P(2) + d_N \cdot P(1)}{d_W + d_N}. \tag{4}$$

These gradients are a weighted average of the absolute differences between adjacent nearest-neighbor pixels in a specific direction (to the nearest 45°). These differences can also be viewed as prediction errors of the individual four sub-predictors from the set of $P(1), P(2), P(3), P(4)$ applied to the previously coded pixels. For example, if at the nth time instant the component of the d_{NE} gradient is $|P(5) - P(3)|$, this corresponds to the prediction error $|e(5)_{NE}|$ (where $e(j)_{NE}$ specifies the jth prediction error using the

northeast-type sub-predictor at previous time instants). It means that according to Figure 1, the northeast type predictor for pixel $P(5)$ at time $(n-2)$ was its upper right neighbor $\hat{x}_{n-2} = P(3)$, then the prediction error is $e(5)_{NE} = P(5) - P(3)$.

In such notation, the gradients (multiplied by 30 to dispense with floating-point operations) take the following form:

$$\begin{aligned} d_W =& 3 \cdot (2 \cdot (|e(1)_W| + |e(2)_W| + |e(3)_W| + \\ & |e(4)_W|) + |e(6)_W| + |e(9)_W|) \\ d_N =& 3 \cdot (2 \cdot (|e(1)_N| + |e(2)_N| + |e(3)_N| + \\ & |e(4)_N|) + |e(5)_N| + |e(7)_N|) \\ d_{NW} =& 5 \cdot (2 \cdot (|e(1)_{NW}| + |e(2)_{NW}|) + \\ & |e(3)_{NW}| + |e(4)_{NW}|) \\ d_{NE} =& 5 \cdot (2 \cdot (|e(2)_{NE}| + |e(5)_{NE}|) + \\ & |e(1)_{NE}| + |e(3)_{NE}|) \end{aligned} . \tag{5}$$

The same principle should be adopted to compare the predictive blending method with $GBSW_+$: only two of the four sub-predictors are of interest (in the basic approach, the blending method uses all available sub-predictors from the defined set). Then (assuming that the nth time instant most minor gradients are, for example, d_W and d_N) for sub-predictors $P(1)$ and $P(2)$, we obtain the following formula:

$$\hat{x}_n = \frac{\frac{1}{d_W^\alpha} \cdot P(1) + \frac{1}{d_N^\alpha} \cdot P(2)}{\frac{1}{d_W^\alpha} + \frac{1}{d_N^\alpha}}. \tag{6}$$

Here, we use the principle that the predicted value in the method of predictor blending is their weighted average with weights inversely proportional to the average level of prediction errors. The corresponding values are d_W and d_N. In addition, the sum of the weights must be equal to 1.

Simplifying, at $\alpha = 1$, the Formula (6) reduces to the form of the Formula (4) known from the $GBSW_+$ method.

With the introduction of the additional parameter α, after performing experiments using the final Formula (3), more favorable results were obtained in this work with $\alpha = 2$, which led to the modification of Formula (4) to the form:

$$\hat{x}_{GBSW_+} = \frac{d_W^2 \cdot P(2) + d_N^2 \cdot P(1)}{d_W^2 + d_N^2}. \tag{7}$$

When the denominator in Formula (7) is equal to 0, then the predicted value is obtained from the GAP_+ model.

2.3. Component Blocks of the Proposed Codec

The solution proposed in this work was based on a cascade approach (see Figure 2), in which, in addition to the predicted value calculated using linear prediction, an additional *CDCCR* (Context-Dependent Constant Component Removing) block was used to remove the constant component C_{mix} associated with a specific context (detailed description of *CDCCR* in [30]). This idea has been used before in codecs such as *JPEG-LS* and *CALIC*. Removing the constant component C_{mix} associated with one of the 1024 contexts results in a slight modification of Formula (1) to the form:

$$e_n = x_n - [\hat{x}_n + C_{mix}], \tag{8}$$

where $\hat{x}_n$ is calculated from the Formula (3).

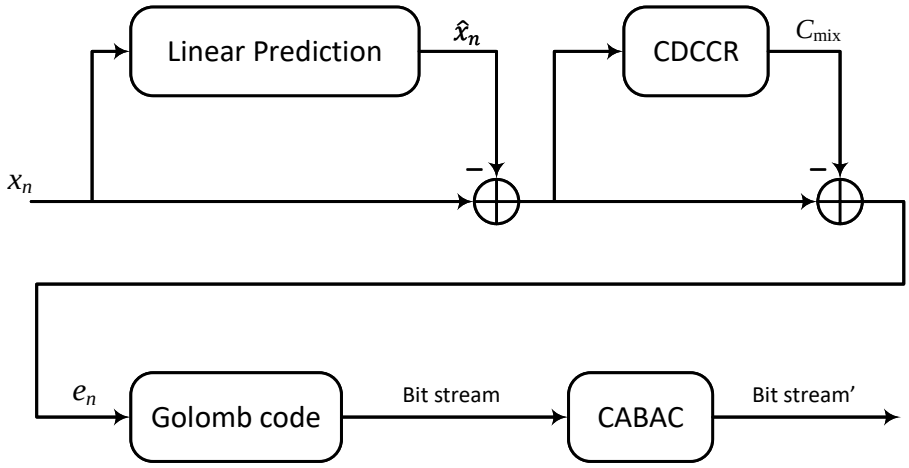


Figure 2. Block diagram of the cascade coding proposed in this paper.

The final blocks of cascade processing are used to efficiently encode prediction errors using an adaptive Golomb encoder and a binary arithmetic encoder (details are discussed in the paper [23]). Algorithms 1 and 2 below show the encoder and decoder processing, respectively.

Algorithm 1 Encoder data processing

1: For each successively encoded pixel x_n:
2: Calculate the predicted value (Formula (3)—stage 1) and prediction error e_n after considering context-dependent constant component C_{mix} (Formula (8)—stage 2).
3: Convert the prediction error value e_n to the bit sequence using adaptive Golomb coder (stage 3).
4: Encode the bit sequence from stage 3 using adaptive binary arithmetic coder.
5: Return to step 2 if there are still pixels to encode.

Algorithm 2 Decoder data processing

1: For each successively decoded pixel x_n:
2: Decode the input bit sequence using adaptive binary arithmetic coder and obtain a word of the Golomb code.
3: Convert the Golomb word to the prediction error value e_n.
4: Calculate the predicted value according to (3) and the C_{mix} value, and add these values to e_n obtaining decoded pixel value x_n.
5: Return to step 2 if there are still pixels to decode.

In this work, we will mainly focus on the detailed aspects of the first block of the encoder related to linear prediction, discussing the method of fast mean absolute error minimization in Section 2.4.1 and the use of context-dependent linear prediction in Section 3. The proposed solution uses forward-adaptive prediction, where at the stage of calculating prediction coefficients, we have access to the entire encoded image. This allows us to achieve a relatively fast decoding process.

2.4. Fast Minimization of Mean Absolute Error

Although it has been widely accepted that for images, determining prediction coefficients using the Minimum Mean Square Error (MMSE) method yields satisfactory results [9,23], in the paper [27], we have shown that an even smaller bit average can be obtained using other methods, including after using mean absolute error minimization. We analyzed a brute-force method with exponential complexity against the number of repetitions of precoding, a method based on selective iterative search with polynomial complexity, and a method with a fixed number of five repetitions of precoding based on a

modified form of $ALCM_+$ (Activity Level Classification Model). The latter method makes it possible to reduce the bit average's value relative to the classic MMSE approach, but it is pretty cumbersome. This is because for the time of adaptive calculation of prediction coefficients, an internal transformation of data from the model of order r to a higher order r_{ALCM} is required, and in the final stage, it is necessary to use an inverse transformation (individually, designed for the pair r, r_{ALCM}). Two examples for $r = 14, r_{ALCM} = 27$ and $r = 24, r_{ALCM} = 43$ are presented in the paper [27]. The second disadvantage of this solution is that it requires more precision for saving the prediction coefficients (which must be passed to the decoder in the file header) than using classic MMSE. Assuming that the prediction coefficient is in the range $[-1.999, 1.999]$, we can store each of the coefficients on some small number of $(N + 2)$ bits (allocating one bit each for saving the sign of the number and the integer value, respectively, and N bits of precision for saving the fractional part of the coefficient).

The modified $IRLS_+$ method is devoid of these disadvantages, allowing the calculation of a predictive model of any order in less time, and saving the prediction coefficients requires a precision of only $N = 9$ bits of the fractional part, which is 3 bits less per coefficient than in the $ALCM_+$-based solution.

2.4.1. Iterative Reweighted Least Squares Algorithm

Reference [31] proposed a minimum mean absolute error (*MMAE*) to obtain better results (when the image was divided into 8×8 squares) compared to using *MMSE*. To determine the predictive model, we do not need to know the optimal solution that guarantees the minimization of mean absolute error, and one of the approximate solutions will suffice. For this purpose, instead of using improved simplex algorithms, a method can be used that uses the classical *MMSE* by reducing it to a Weighted Least Squares (*WLS*) minimization problem in an iterative manner, allowing a suboptimal solution minimizing the mean absolute error to be obtained relatively quickly. To be more specific, it is *IRLS* with the parameter $p = 1$.

IRLS allows for an approximate solution that minimizes the average error in the Q data area according to the l_p norm [32]:

$$\|e\|_p = \left(\sum_{n \in Q} |e_n|^p \right)^{1/p}, \tag{9}$$

by reducing the issue to solving a WLS problem that offers relatively low computational complexity:

$$\|e\|_p = \left(\sum_{n \in Q} |e_n|^{(p-2)} \cdot |e_n|^2 \right)^{1/p}. \tag{10}$$

We can perform an approximate minimization for any l_p norm using an iterative expression minimization algorithm:

$$\|e\|_p = \left(\sum_{n \in Q} w_n^2 \cdot |e_n|^2 \right)^{1/p}. \tag{11}$$

For this purpose, we set the weights $w_n = 1$ in the first iteration and minimize the weighted mean squared error following the Formula (11). We obtain this by solving the matrix equation (using Cholesky decomposition):

$$\mathbf{B} = \mathbf{R}^{-1} \cdot \mathbf{P}, \tag{12}$$

where $\mathbf{R}$ is a square matrix of $r \times r$ elements $\mathbf{R}(j,i)$ such that:

$$\mathbf{R}(j,i) = \sum_{n \in Q} w_n^2 \cdot y_{n,i} \cdot y_{n,j}, \tag{13}$$

at $j = \{1,2,\ldots,r\}$, $i = \{1,2,\ldots,r\}$ while $\mathbf{P}$ is a vector of $r \times 1$ elements $\mathbf{P}(j)$ such that:

$$\mathbf{P}(j) = \sum_{n \in Q} w_n^2 \cdot x_n \cdot y_{n,j}, \tag{14}$$

where x_n specifies the value of the next nth encoded pixel.

In the proposed solution, using an extended prediction model according to Formula (3), we use two nonlinear predicted values $\hat{x}_{GBSW_+(n)}$, $\hat{x}_{GAP_+(n)}$ and $(r-2)$ nearest neighbor pixels (see Figure 1); therefore, the vector $\mathbf{Y}_n = [y_{n,1}, y_{n,2}, \ldots, y_{n,r}]$ can be represented as $\left[\hat{x}_{GBSW_+(n)}, \hat{x}_{GAP_+(n)}, P_n(1), P_n(2), \ldots, P_n(r-2)\right]$.

In each subsequent iteration of the *IRLS*, the weights are determined using the e_n^{old} errors obtained from using the linear prediction model obtained from the previous iteration, using the formula:

$$w_n = \left|e_n^{old}\right|^{(p-2)/2}. \tag{15}$$

Each successive iteration in the *IRLS* algorithm brings us closer to the expected minimization accuracy, with minimization of the final bit average L_{avg} obtained after taking into account the header data (the cost of saving the prediction coefficients) and entropy coding of prediction errors playing a key role. For eight-bit image data at $p = 1$, good results are obtained after only 8–10 iterations. In the case of $p = 2$, we obtain the classic minimization of the mean-square error, in which case only the first iteration is sufficient.

To minimize the mean absolute error in *IRLS*, we set the parameter $p = 1$. Then, substituting the Formula (15) into (11), we obtain the classical *WLS* problem, with which the approximate minimization of the mean absolute error can be obtained:

$$\|e\|_1 = \sum_{n \in Q} \frac{1}{\left|e_n^{old}\right|} \cdot |e_n|^2 \approx \sum_{n \in Q} |e_n|. \tag{16}$$

Particular attention should be paid to prediction errors oscillating near zero, and protection against dividing by 0 should be made. With eight-bit data, where the absolute error is in the range $\left|e_n^{old}\right| \in [0, 255]$, it works well to introduce a limiter $\left|e_n^{old}\right| \leftarrow \max\left\{0.6, \left|e_n^{old}\right|\right\}$ protecting against excessive weights for errors close to zero ($\left|e_n^{old}\right| < 0.6$). The value of 0.6 was chosen experimentally (with an accuracy of 0.1) to minimize the bit average of the set of learning images. Finally, the approximate solution to minimize the average error in the Q data area according to the l_1 norm takes the form:

$$\|e\|_1 = \sum_{n \in Q} \frac{1}{\max\left\{0.6, \left|e_n^{old}\right|\right\}} \cdot |e_n|^2. \tag{17}$$

It should be noted that at the stage of determining successive iterations, the prediction values (Formula (3)) are not rounded to an integer when using the Formula (1), which increases the computational convergence. Only after the final form of the predictor $\mathbf{B}$ has been determined does the differential image coding procedure use Formula (8) to encode the total prediction errors.

2.4.2. Improvements to the *IRLS* Algorithm

Our contribution to accelerating the convergence of *IRLS* (in terms of the approximate solution and eight-bit data) in the case of mean absolute error minimization ($p = 1$) is to

replace the weight w_n^2 (Formula (15)) inversely proportional to the absolute value of the prediction error by an exponential function $0.8^{e_n^{old}}$. Formula (17) is changed to the form:

$$\|e\|_1 = \sum_{n \in Q} 0.8^{e_n^{old}} \cdot |e_n|^2. \tag{18}$$

This halved the number of iterations in the *IRLS* to just five and eliminated problems of dividing by 0. The value of 0.8 was selected by experiment (with an accuracy of 0.01) to minimize the bit average of the set of learning images.

If we assume that the weights in Formulas (17) and (18) are treated as some kind of probability density functions ($f_1(x)$ and $f_2(x)$, respectively) over the range of values of the variable x in the interval $[0, 255]$, then we get a distribution:

$$f_1(x) = \frac{1}{a \cdot \max\{0.6, x\}}, \tag{19}$$

where

$$a = \int_0^{0.6} \frac{5}{3}\, dx + \int_{0.6}^{255} \frac{1}{x}\, dx = 1 + \ln 425, \tag{20}$$

and the one-sided Laplace distribution:

$$f_2(x) = \ln(1.25) \cdot 0.8^x. \tag{21}$$

The introduction of a new weight in the Formula (18) is, therefore, justified by the similarity of the shape of the graphs reflecting the probability distributions of $f_1(x)$ and $f_2(x)$ (as shown in Figure 3).

A second idea to improve the minimization of the bit average L_{avg} using *IRLS* is to slightly reduce the weights w_n for encoded pixels whose nearest neighborhood (or more precisely, elements of the $\mathbf{Y}_n$ vector) are characterized by high variability (e.g., due to noisy data). We can calculate the level of this variability using the mean absolute deviation:

$$D_n = \frac{1}{r} \sum_{i=1}^{r} |y_{n,j} - \overline{y}_n|, \tag{22}$$

where $\overline{y}_n$ is the arithmetic mean of the elements of vector $\mathbf{Y}_n$. Then, Formula (18) is extended to the form:

$$\|e\|_1 = \sum_{n \in Q} \frac{0.8^{e_n^{old}}}{\frac{200}{MAE_{old}} + D_n} \cdot |e_n|^2, \tag{23}$$

where MAE_{old} denotes the mean absolute error obtained in the previous *IRLS* iteration (in the first iteration, it is initialized with $MAE_{old} = 10$). In summary, the corrections made here in the form of a new weight:

$$w_n = \left(\frac{0.8^{e_n^{old}}}{\frac{200}{MAE_{old}} + D_n} \right)^{\frac{1}{2}}, \tag{24}$$

we can refer to as $IRLS_+$.

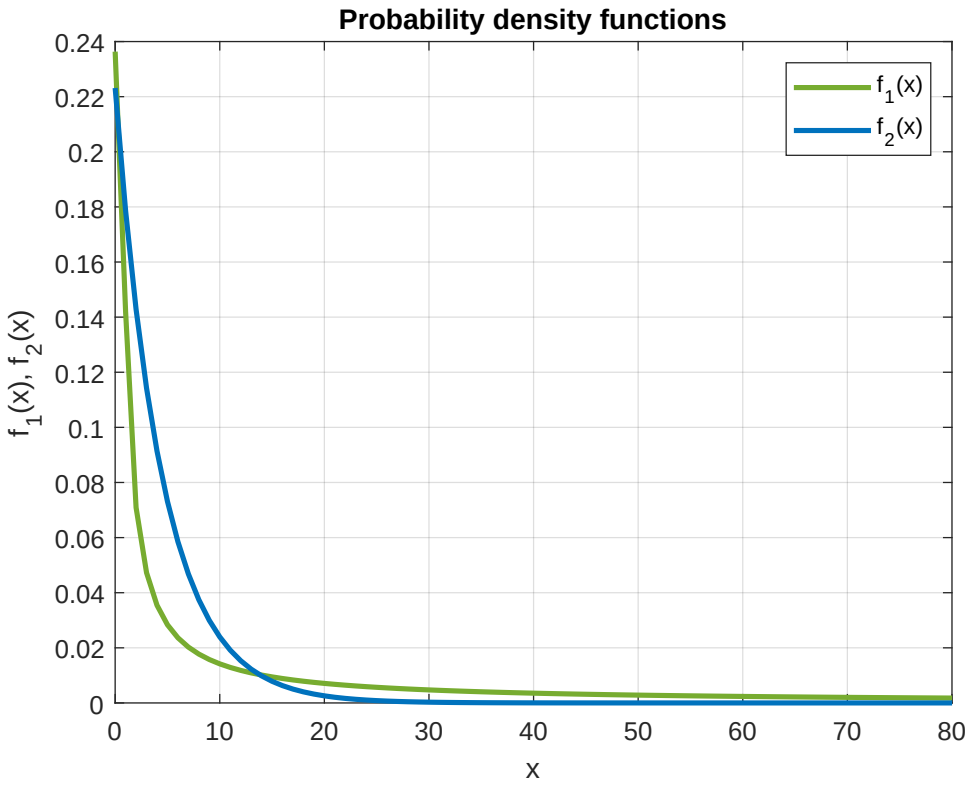


Figure 3. Comparison of $f_1(x)$ and $f_2(x)$ probability distributions.

2.4.3. Effectiveness of Improvements

By using $IRLS_+$ and improving the $GBSW_+$ method with the help of Formula (7), we were able to improve the bit average and encoding/decoding time relative to the $ALCM_+$ codec presented in our previous work [27], which used the same Formula (3) for the predicted value and the same number of five iterations of scanning (pre-coding) the entire image to determine the prediction coefficients. The decoding time (Dell Precision 5690 mobile workstation (Dell, Round Rock, TX, USA) with Intel® Core™ Ultra 7 165H processor, 3.8 GHz) of a `Lennagrey` image with a resolution of 512×512 pixels decreased by about 12% to 0.133 s for the order of $r = 14$ and to 0.152 s at $r = 24$. It should be noted here that the decoding time is linearly dependent on both the number of image pixels and the prediction order. On the other hand, the coding time decreased by 29.7% (0.671 s) at $r = 14$ and by 16.9% (1.01 s) at $r = 24$. Due to the non-linear dependence of coding time on the prediction order, a more detailed analysis of this aspect will be presented in Section 3.2.

Table 1 compares the bit averages obtained by the proposed $IRLS_+$ method with other methods of determining prediction coefficients (using the same prediction error encoder). Two sets of most popular standard test images were used for testing. The first consists of 9 images and the second consists of 45 images; they are a subset of the entire image database available at [33].

Table 1. Comparison of bit averages in bits per pixel (bpp) for two sets of standard test images and two prediction orders, obtained by proposed $IRLS_+$ method with other methods of prediction coefficient calculation.

Image	$MMSE$ $r = 14$	$ALCM_+$ [27] $r = 14$	$IRLS_+$ $r = 14$	$MMSE$ $r = 24$	$ALCM_+$ [27] $r = 24$	$IRLS_+$ $r = 24$
Balloon	2.742	2.748	2.731	2.733	2.733	2.720
Barb	4.238	4.238	4.210	4.221	4.211	4.196
Barb2	4.416	4.421	4.403	4.406	4.401	4.393
Board	3.446	3.449	3.426	3.436	3.425	3.417
Boats	3.717	3.711	3.694	3.707	3.696	3.688
Girl	3.612	3.624	3.594	3.609	3.612	3.595
Gold	4.310	4.304	4.298	4.299	4.293	4.290
Hotel	4.180	4.176	4.151	4.179	4.169	4.153
Zelda	3.634	3.620	3.621	3.621	3.598	3.603
Avg. (9 images)	3.810	3.810	3.792	3.801	3.793	3.776
Avg. (45 images)	4.045	4.038	4.025	4.032	4.021	4.007

3. Context-Dependent Linear Prediction

Section 2.4 presents how to determine the prediction coefficients that form the vector $\mathbf{B} = [b_1, b_2, ..., b_r]$ to be passed as header information to the decoder. Assuming that the prediction coefficient is between -1.999 and 1.999, we can write each coefficient on a certain small number of $(N + 2)$ bits (one bit each for the sign of the number and the integer value, respectively, and N bits of precision for the fractional part). In addition, since the sum of the coefficients is 1, it is sufficient to write only $r - 1$ coefficients. For example, for an image with a resolution of 512×512 pixels at $r = 24$, $N = 9$, the header cost is only 0.000965 bits per pixel.

This is a small overhead, hence the idea to increase compression efficiency by expanding the prediction order and introducing a context-dependent predictor switching mechanism. Due to the variability of the neighborhood characteristics of the nearest pixel, it is possible to select several predictors individually tailored to the different types of neighborhoods of the currently encoded pixel. In the paper [34], we demonstrated the validity of this approach using simple context partitioning algorithms based on *MED* (Median Edge Detector—known from *JPEG-LS* [10]) and *GAP* (Gradient Adjust Predictor used in the *CALIC* codec [11]). In the first case, three and seven different contexts were obtained, and their corresponding predictors were obtained in the second case. Unlike the original versions of *JPEG-LS* and *CALIC*, where the predictors were fixed, in the work of [34], we used predictors selected individually for each image using the *MMSE* method. Although with seven contexts at $N = 9$ and a higher prediction order $r = 80$, the header cost for an image with a resolution of 512×512 pixels of 0.0232 bits per pixel is already noticeably higher, it is still acceptable. These are the upper limits on the number of contexts and prediction order we decided to introduce in the codec proposed in this work.

3.1. Algorithm to Determine the Context

In the case of using *GAP*, the main problem was the inefficient method of dividing seven contexts using a scalar quantizer with six fixed thresholds, which needed to sufficiently provide an individual for each image to match the dynamics of the digital signal.

In this work, we want to present a much more efficient contextual partitioning and use the $IRLS_+$ algorithm discussed in Section 2.4 to determine the prediction coefficients for each context. Depending on user demand, the two-stage decision-making algorithm divides it into three and seven contexts.

In the case of splitting into three contexts to take into account the average level of variation of the digital signal, the average value from all the weighted variances m of the pixels closest to each encoded image pixel should be determined for a given image (Figure 1). We calculate the weighted variance from the formula:

$$\widetilde{\sigma}^2 = \frac{1}{\delta}\sum_{j=1}^{m} \overline{d}_j \cdot (P(j) - \overline{p})^2, \tag{25}$$

where the weights $\overline{d}_j$ are determined as the inverse of the Euclidean distance between pixels $P(j)$ and $P(0)$:

$$\overline{d}_j = \frac{1}{\sqrt{(\Delta x_j)^2 + (\Delta y_j)^2}}, \tag{26}$$

where Δx_j and Δy_j are the horizontal and vertical distances of pixel $P(j)$ relative to $P(0)$, respectively.

Furthermore:

$$\delta = \sum_{j=1}^{m} \overline{d}_j, \tag{27}$$

is the normalizing factor, while $\overline{p}$ is the arithmetic mean of the nearest neighbor:

$$\overline{p} = \frac{1}{m}\sum_{j=1}^{m} P(j). \tag{28}$$

The first step in context partitioning is to quantize the surrounding weighted variance values $\widetilde{\sigma}^2$ into three intervals (low, medium, and large variances) with thresholds of $\beta_1 \cdot \overline{\sigma}^2$ and $\beta_2 \cdot \overline{\sigma}^2$ (where $\overline{\sigma}^2$ is the arithmetic mean of all $\widetilde{\sigma}^2$ weighted variances determined for the encoded image). Good results were obtained with $m = 10$, $\beta_1 = 0.15$ and $\beta_2 = 1.2$. Determining the context number (denoted as ctx) from 0 to 2 is presented in detail by Algorithm 3.

Algorithm 3 The first method of the context number calculation for pixel x_n being encoded.

```
Calculate the weighted variance (25):
if (σ̃² < β1 · σ̄²) then
    ctx ← 0;
else if (σ̃² < β2 · σ̄²) then
    ctx ← 1;
else
    ctx ← 2;
end if
```

The second step is used when to obtain a split into seven instead of three contexts. For this purpose (in the situation of obtaining a mean or large variance), we check for a large discrepancy in the nearest neighborhood between the horizontal gradient d_h and the vertical gradient d_v, where:

$$\begin{aligned} d_h =& 2 \cdot (|P(1) - P(5)| + |P(2) - P(3)| + |P(3) - P(7)| + \\ & |P(2) - P(4)|) + |P(6) - P(8)| + |P(6) - P(9)| \\ d_v =& 2 \cdot (|P(2) - P(6)| + |P(1) - P(3)| + |P(4) - P(9)| + \\ & |P(3) - P(8)|) + |P(5) - P(7)| + |P(7) - P(11)| \end{aligned}. \tag{29}$$

The range of proportions $d_h : d_v$ allows us to divide the context (with medium or large variance) into three sub-contexts. In detail, the determination of the context number

(denoted as ctx) from 0 to 6 is presented by Algorithm 4, where the additional parameter is set to $\beta_3 = 1.4$.

Algorithm 4 The second method of the context number calculation for pixel x_n being encoded.

Calculate the weighted variance (25):
if $(\widetilde{\sigma}^2 < \beta_1 \cdot \overline{\sigma}^2)$ **then**
    $ctx \leftarrow 0$;
else
    Calculate gradients using (29);
    if $(d_h > \beta_3 \cdot d_v)$ **then**
        $ctx \leftarrow 1$;
    else if $(d_v > \beta_3 \cdot d_h)$ **then**
        $ctx \leftarrow 2$;
    else
        $ctx \leftarrow 0$
    end if
    if $(\widetilde{\sigma}^2 < \beta_2 \cdot \overline{\sigma}^2)$ **then**
        $ctx \leftarrow ctx + 1$;
    else
        $ctx \leftarrow ctx + 4$;
    end if
end if

Depending on the number of contexts in the solution proposed here, perform Algorithm 3 or Algorithm 4 as an additional step in Algorithm 1 (between steps 1 and 2) and Algorithm 2 (between steps 3 and 4).

3.2. Selection of the Parameters of the Compression

3.2.1. Impact of Prediction Order on Coding Time

The introduction of context splitting does not significantly affect the decoding time, which (for a `Lennagrey` image with a resolution of 512×512 pixels) in the seven-context version at $r = 10$ is 0.145 s and 0.165 s at $r = 80$.

The same is true for coding time, but there is a non-linear relationship concerning the prediction order. For the orders $r = 10$ and $r = 80$, the encoding times are 0.49 s and 5.06 s, respectively. Figure 4 presents an encoding time as a function of the prediction order.

Although as many as seven matrix equations must be solved in the seven-context case (see the Formula (12)), the time to solve them is negligibly short (24 µs at $r = 10$ and 2559 µs at $r = 80$) compared to the time to pre-determine the set of seven matrices $\mathbf{R}$ and vectors $\mathbf{P}$. In the case of $r = 10$, it is only 0.0285 s, but already at $r = 80$ the time increases to 0.927 s, which, in this case, is more than 90 percent of the total image encoding time (it should be noted here that the procedure for calculating prediction coefficients in the $IRLS_+$ algorithm requires as many as five iterations). Since the $\mathbf{R}$ matrix is triangular, it is sufficient to fill its $r(r+1)/2$ elements. After taking into account the filling of r elements of the $\mathbf{P}$ vector, this requires a total of $S(r(r+1)/2 + 2)$ multiplications and additions, where S is the number of pixels of the image ($S >> r$). For example, for an image with a resolution of 512×512 pixels at $r = 80$ filling the $\mathbf{R}$ matrix and $\mathbf{P}$ vector requires $2^9 \cdot 2^9 \cdot (80 \cdot 81/2 + 2) = 849,870,848$ multiplications and additions. This work set the maximum order as $r = 80$ as a compromise value, offering high efficiency with moderate coding time. It is important to emphasize that the encoding of an image is most often performed once by the user, and the decoding process is performed multiple times; hence, with $r = 80$ from the user's point of view, we obtain a good ratio of encoding versus decoding time of 30.77 : 1. Decoding time is linearly dependent on the prediction order and the number of pixels in the image.

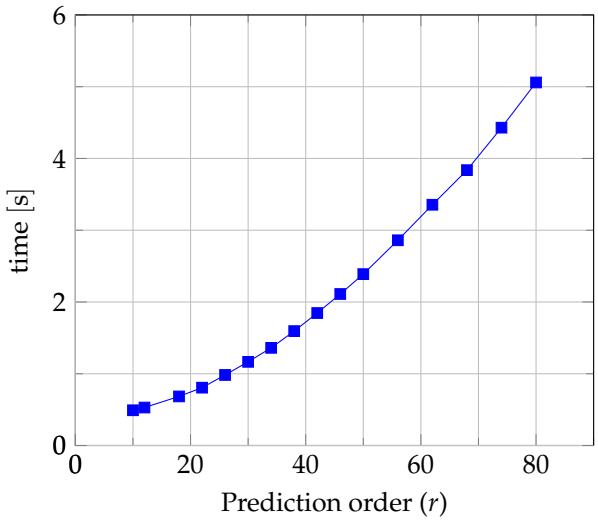


Figure 4. The encoding time as a function of the prediction order for a `Lennagrey` image with a resolution of 512 × 512 pixels.

3.2.2. Cost of Prediction Coefficient Storage

Minimizing the bit average is a manageable problem due to the large number of variables affecting the final length of the encoded file. As the prediction order and the number of contexts increases, the mean absolute error can be expected to decrease. On the other hand, due to the requirement to save prediction coefficients, the size of the file header is negatively affected.

The solution proposed here defines three image classes in terms of size. Small, medium, and large images are characterized by their sizes: up to 2^{16} pixels, up to 2^{18} pixels, and counting above 2^{18} pixels, respectively. By using the Algorithms 1 and 2, we can adjust the size of the header for each of the three image classes by changing the number of contexts and the accuracy of the prediction coefficients, where each is saved using $(N + 2)$ bits. Table 2 presents the parameters for the images at three resolutions (each representing a separate class), with $r = 80$.

Table 2. Header cost depending on image size at order $r = 80$.

Resolution	**256 × 256**	**512 × 512**	**720 × 576**
Number of contexts	3	7	7
$(N + 2)$	9	10	11
Header cost [b/pixel]	0.03255	0.02110	0.01467

For the smallest images of resolution 256 × 256, despite the significant reduction in the precision of the prediction coefficients to $N = 7$ bits of the fractional part and the restriction to splitting into just three contexts, the header cost is not marginal, signaling the need for possible header data compression.

Although a reasonably wide range has been defined (from -1.999 to 1.999) for each of the coefficients, they are usually small values for which the arithmetic mean (after analyzing the probability distribution of a given coefficient) oscillates near zero. The higher the jth coefficient number b_j, the smaller its absolute value usually becomes. This can be used to compress coefficients using a context-dependent binary arithmetic encoder.

From a base of training images, approximate probability distributions can be constructed for each of the r coefficients. We then convert these distributions into probability distributions of the individual bits of a given coefficient (with required bit space of $(N + 2)$

bits). The sign bit of the coefficient is encoded first, followed by the bit specifying the integer part of the number (with a value of 0 or 1). Next, we encode N bits of the fractional part of a given coefficient sequentially from highest to lowest, respectively.

For the fractional part of the coefficient, we can introduce a contextual division that considers several distributions for a given bit by obtaining preliminary information about the previously encoded bits (used further as contextual bits). We first use the information in the two encoded bits to do this. The first is the sign bit (s_1) of the encoded coefficient. The second context bit is determined on the fly based on information about whether at least one bit with a value of 1 has ascended among the previously encoded most significant bits of a given coefficient, in which case $s_2 = 1$, otherwise $s_2 = 0$. This gives us a two-bit number $s_1 s_2$ with values between 0 and 3, which, when multiplied by four times the bit number of the fractional part in the range from 1 to N, gives us the number of the context in which the binary distribution is stored (in practice, a single number storing the probability of occurrence of a bit with a value of 0 is sufficient) corresponding to the currently encoded bit. The encoder and decoder have a built-in database with a set of such $4\dot{N}$ distributions for each of the $(r-1)$ prediction coefficients (with the first coefficient not encoded, since on the decoder side, it can be calculated by assuming that the sum of the coefficients of the **B** vector is 1).

Two facts are noticeable: First, the less significant the coefficient, the more random it is, i.e., it is less susceptible to compression. Second, the higher we set the prediction order, the lower the average cost per coded coefficient. Table 3 presents (obtained for the test image database) the average cost of saving one prediction coefficient depending on the precision of writing these coefficients using two cases for $r = 42$ and $r = 80$, respectively. It shows that with $r = 80$, $N = 8$, it was possible to reduce the header's size by an average of 46.64%.

Table 3. Average bit numbers of the prediction coefficient after arithmetic coding as a function of input data word length.

Coefficient Precision $(N+2)$ Bits	**8**	**9**	**10**	**11**
Average cost $(r = 42)$	3.96113	4.90561	5.88912	6.88633
Average cost $(r = 80)$	3.46996	4.37203	5.33586	6.32542

3.2.3. Influence of the Choice of the Parameters on the Efficiency of Compression

In the proposed codec, using Algorithms 1 and 2, further supported by contextual partitioning (see Algorithms 3 and 4), referred to hereafter as the base version, several parameters were fixed. This is the prediction order $r = 80$, the neighborhood size $m = 10$ (see Formula (25)), three quantization thresholds: $\beta_1 = 0.15$, $\beta_2 = 1.2$ and $\beta_3 = 1.4$, and the five iterations in the $IRLS_+$ algorithm. In addition, depending on the size of the images (small, medium, and large images of up to 2^{16} pixels, up to 2^{18} pixels and those counting more than 2^{18} pixels, respectively), two fixed parameters were set: the number of contexts (which is three for small images and seven in other cases) and the precision of the prediction coefficients save of 9 (for small), 10 (for medium) and 11 bits (for large images), respectively.

In addition to the proposal for encoding of the prediction coefficients (which the base version of the encoder does not use), discussed in Section 3.2.2, it is possible to reduce the bit average by individually adjusting the eight encoder parameters listed here to the characteristics of a particular image. However, this requires multiple image encodings. Additional ideas for improvement can be introduced, requiring additional iterations of image encoding (such as choosing the correct parameters in the arithmetic encoder). To ensure that the entire tuning stage takes little time and that the modified parameter information itself does not unduly increase the size of the header, it is necessary to narrow the search area and determine their respective order of parameter selection. For this purpose, we designed an Algorithm 5 to select the best compression parameters.

Disregarding the last (12th) step of the Algorithm 5, more than 2 million combinations of parameter settings listed in steps 2 through 11 are possible. However, executing the individual steps of the Algorithm 5 appropriately limits the number of tests (involving full image encoding) to just 43.

Algorithm 5 Tuning compression parameters to reduce the bit average.

1: Set start parameters according to base version (except for one less number of iterations within the $IRLS_+$ algorithm).
2: Calculate the one of best of 8 lossless phases of image rotation (at $r = 22$).
3: Calculate the best class of prediction order (three tests, one representative each from the following classes, respectively: $r = 18$ for the low-order class, $r = 42$ for the medium-order class, and $r = 68$ for the high-order class) for the version with seven contexts.
4: Repeat step #3 for the three-context version, choosing a winner within steps #3 and #4.
5: Perform more accurate prediction order selection (five tests) for the winning class.
6: Choose the best parameter m (see Formula (25)) from the set $\{8, 10, 12, 28\}$.
7: Choose the best pair of quantization thresholds $\{\beta_1, \beta_2\}$ from a set of seven settings.
8: Choose the best accuracy for saving prediction coefficients on N bits after the decimal point from the set $\{6, 7, 8, 9\}$.
9: If the seven-context version is chosen, select the best quantization threshold β_3 from the set $\{1.3, 1.4, 1.5, 2.4\}$
10: Check if the version with the $CDCCR$ block disabled (with $C_{mix} = 0$) improves the bit average.
11: Check how many iterations at the $IRLS_+$ stage gives the best result (nine tests).
12: Perform a set of arithmetic coding tests for different encoder settings.

Due to the slight difference in efficiency between four and five iterations in $IRLS_+$, it was decided to reduce (relative to the base version of the encoder) to four iterations as a first step, which reduces the time of most tests by almost 20%.

Each rectangular image can be rotated losslessly in 90° increments, resulting in four different settings. Suppose we combine this with the mirroring capability. In that case, we obtain a doubling of these lossless image transformations (which the decoder can perform in reverse, restoring the original image). This transformation can affect the final compression efficiency due to the different neighborhood selections and encoded pixel order. Therefore, eight turnover phase selection tests are performed in step 2 (with a relatively small but representative order $r = 22$).

Steps 3 and 4 are where the decision is made to choose the number of contexts: three or seven. To reduce the total number of tests for selecting a prediction order (from the considered 16-element set of orders between 10 and 80), an initial selection of this order is made from among the three possibilities of $\{18, 42, 68\}$ and twice, as both the three-context and seven-context versions. If the bit average is lowest for $r = 18$, in step 5, we make a complementary fine-tuning of the order from the set $\{10, 12, 22, 26, 30\}$. On the other hand, when the order $r = 42$ or $r = 68$ won in steps 3 and 4, the sets of complementary orders in step 5 are, respectively: $\{30, 34, 38, 46, 50\}$ and $\{50, 56, 62, 74, 80\}$. As a result, instead of 32 tests (16 different orders each in the 3-context and 7-context versions), only 11 tests are performed in steps 3–5.

In step 6, the best parameter m is selected from a set of $\{8, 10, 12, 28\}$, and in step 7, the best pair of quantization thresholds $\{\beta_1, \beta_2\}$ is selected from a set of 7 experimentally selected settings: $\{\{0.02, 1.00\}, \{0.08, 0.80\}, \{0.08, 1.00\}, \{0.10, 1.00\}, \{0.10, 1.20\}, \{0.15, 1.20\}, \{0.30, 1.40\}\}$. Step 8 allows us to choose the best accuracy for saving the prediction coefficients at N decimal bits from the set of $\{6, 7, 8, 9\}$, taking into account the compression of the prediction coefficients discussed in Section 3.2.2. Step 9 is performed if the decision to use the division into seven contexts was previously made. The quantization threshold β_3 is tuned from the allowed $\{1.3, 1.4, 1.5, 2.4\}$. Step 10 is just one test to see if the version

with the CDCCR block disabled (i.e., with $C_{mix} = 0$) will improve the bit average, which happens less than 10% of the time.

The final (11) step affecting the determination of the final set of prediction coefficients is the selection of the number of iterations in $IRLS_+$, where we check it in the range of 3 to 10. After the first iteration, the result is also considered, corresponding to the classic *MMSE* method.

Step 12 involves only minor modifications to the parameters of the arithmetic encoder. Although there are quite a lot of tests, 140, they are performed relatively quickly, as they use the already coded prediction error map created after step 11 (unlike the earlier steps of the Algorithm 5, the time-consuming step of determining a new set of prediction coefficients based on $IRLS_+$ is absent here). Information about the winning parameters is just a few extra header bytes.

Table 4 shows the impact of disabling individual parameter tuning steps within the Algorithm 5. The results show the bit average's growth level based on averaged results for 45 test images. It should be emphasized that the actions in the individual steps to be turned off affect each other, so these losses should not be directly summed up to assess the overall suitability of the Algorithm 5. Using Algorithm 5 allowed us to improve the bit average by 0.02679 bit/pixel on average.

A check was also made on the usefulness of introducing the denominator weight w_n in the Formula (24). The gain averaged 0.001 bit/pixel.

To estimate the implementation complexity of the process of encoding a `Lennagrey` image with a resolution of 512×512 preceded by the execution of the selection of the best parameters (all steps of the Algorithm 5), we introduced forced actions to be able to analyze two extreme situations. In the first (fastest) case after step 2 of the Algorithm 5, we forced the transfer of the winning order $r = 18$, and after step 4, the triple-context version with order $r = 10$ was indicated as the winner. The coding time was 58.1 s. In the second (slowest) case after step 2 of the Algorithm 5, we forced the transfer of the winning order $r = 68$, and after step 4, the 7-context version with order $r = 80$ was indicated as the winner. In this case, the encoding time has increased to 184.7 s.

Table 4. Influence of disabling individual steps of Algorithm 5 on the average degradation of the bit average (for 45 test images).

Detachable Fine-Tuning Step in Algorithm 5	Step Number	Average Degradation in b/pixel
Rotating phase	2	0.00623
Order and number of contexts	3, 4, 5	0.00210
m parameter	6	0.00172
Quantization levels $\{\beta_1, \beta_2, \beta_3\}$	7, 9	0.00236
Precision of prediction coefficients	8	0.00032
Additional coding of prediction coefficients	8	0.00449
Testing of disabling *CDCCR* block	10	0.00315
Selection the iteration number of $IRLS_+$ step	11	0.00268
Parameters selection of the arithmetic coder	12	0.00688

4. Experiments and Analysis of the Effectiveness of Existing Solutions

This section compares the encoding and decoding time and efficiency of the proposed method and other well-known classical codecs (without the machine learning category) as representing the deep learning category.

4.1. Analysis of Encoding and Decoding Time

Today's computer systems, with the development of technology, are becoming less and less stable every year regarding reliable measurement of coding time. This is mainly due to multicore (which only a few codecs take advantage of, as most perform their tasks sequentially concerning single image compression). But even more problematic are the mechanisms for auto-adjusting the CPU clock frequency depending on the number of currently running computing threads and the current CPU temperature (which depends on the performance of the automatically regulated cooling system).

For example, an attempt to use a computer Dream Machines (Warsaw, Poland) NP50DE (Windows 10, 16 GB RAM) with a six-core Intel i7 10750H processor with a nominal frequency of 2.6 GHz (Windows 10, 16 GB RAM) for encoding time measurements showed that running a single encoder causes the clock frequency to oscillate around 4.7–4.8 GHz (thanks to Turbo mode, the nominal value was significantly exceeded), decreasing during moments of increased data transfer from the SSD to 3.7 GHz (during the final stage under Algorithm 5 involving multiple encoders tuning parameters in an adaptive arithmetic encoder).

To increase the measurement stability of the encoding/decoding time, we used an up-to-date Dell Precision 5690 mobile workstation (Intel® Core™ Ultra 7 165H processor, 3.8 GHz, 32 GB DDR5, 2 TB SSD, Windows 11), powered by an external power supply, with the ultra performance option set in Dell Optimizer management application. In addition, only one user application was run (minimizing the level of disturbance to the stability of the total load on the quad-core processor by any applications executing in the background). The time measurement was calculated as the arithmetic average of a sequentially repeated 100 or 1000 times encoding/decoding process.

In contrast, the difficulty of practically assessing encoding times based on theoretical, computational complexity nowadays stems from the use of highly efficient processor caches (the impact of which on the variation of measured time can be counted in the hundreds of percent depending on the arrangement of memory references in a given algorithm), as well as the possible opportunities of parallelizing encoding in the era of commonly used multicore processors. Despite these caveats, we have presented an analysis of both encoding and decoding times and the number of multiplications and additions for the most complex operation of processing encoded data in Section 3.2.

In recent publications, the issue of measurement reliability can be further complicated by the possibility of using highly specialized, multi-core graphics cards, which already happens with some of the latest solutions using neural networks. This aspect will be discussed more extensively in the next section.

4.2. Overview of Deep Learning-Based Solutions

Since a whole branch of lossless image compression methods based on deep learning using neural networks has been developed in recent years, we have decided to devote a separate section to this issue, where we want to highlight some of the drawbacks of this type of solution.

The first works using neural networks for lossless image encoding used an adaptive method in which the network weights are modified on the fly after each successive pixel is encoded (backward-adaptive methods)—this is learning the network on the fly without pre-learning using training image bases. In the work of [12–14], an Adaptive Neural Network (AdNN) was used, whose design was based on the Multilayer Perceptron Network (MLP), while in the work of [15], Cellular Neural Networks (CNNs) were used. Our attempts to achieve the highest possible compression efficiency (without pre-learning the network) using AdNN enhanced with context-sensitive switching showed that using such networks in a backward-adaptive version requires high implementation complexity of the encoder and decoder, comparable to that of *WLS*, with the bit-average results obtained with neural networks proving to be noticeably inferior to *WLS* [35].

Other, newer approaches based on neural network deep learning are widely described in papers [36,37]. Different categories exist within the entire pool of state-of-the-art codecs using deep learning with neural networks. For example, the work of [38] proposes a combination of lossy *BPG* coding (which generates a map of $\hat{x}_n$ prediction values) and lossless prediction error compression (an idea known even from the work of [8]). In contrast, the work of [39] uses sub-image (lower resolution) partitioning, where one of the sub-images is encoded using classic *JPEG-XL* (this approach is known from the original version of *CALIC*, and also used in multi-resolution codecs using wavelet transforms), and only the remaining data are encoded using deep neural networks (*LC-FDNet*).

Unfortunately, in the case of solutions based on neural networks, the authors usually skip the analysis of the entire literature (without noticing the duplication of ideas, for example) on lossless image compression – calling Non-Learned Methods as classical approaches or codecs (for short) without learning. Just as often, they seem to overlook the distinction between encoding and decoding times (e.g., the paper [37] presented a graphical comparison of only the summed encoding and decoding times for 11 codecs). In doing so, using an interpretable programming language such as Python 3, in many cases, does not facilitate the reliable analysis of encoding/decoding times against fully compiled and optimized classical codecs (e.g., *CALIC*, *WebP*). Even in the paper [40] summarizing the achievements of deep neural networks, there needs to be a comparative mention of the encoding and decoding times of the solutions discussed there.

In the work of [41], we can come across a comparison of several neural network-based solutions, including the conclusion that *PixelCNN*-type methods (e.g., [42]) are generally computationally expensive. In addition, a disadvantage of these solutions is the strong dependence on the learning base of such networks (it is easy to have a network overfitting effect, especially for images with very low resolution 32×32 or 64×64, resulting in too low efficiency in more comprehensive application compared to classical solutions), which was criticized in the paper [43].

Based on bibliography data, published proportions of coding times can be used. For the work of *IDF*, [44] shows that the *IDF* [41] method is (depending on the image base) 63 to 143 times slower concerning *WebP* encoding (for decoding, the ratio is even more unfavorable, ranging from 4857:1 to as much as 89,043:1). Against this background, the GPU-processing-based *L3C* [38] codec (1.5 to 3.33 times slower than *WebP* at the encoding stage and between 110 and 1614 at the decoding stage) and *SReC* (2.1 to 2.5 times slower than *WebP* at the encoding stage and between 338 and 1643 at the decoding stage) fare significantly better. In contrast, measurements presented by the authors of *L3C* [38] (most likely due to the use of a suitably robust GPU system) show a 1.5:1 encoding time ratio and a 5.25:1 decoding time ratio relative to *WebP* (for a 512×512 pixel image). At the same time, the work of [45] showed that for a set of classic Kodak test images (24 images with a resolution of 768×512 pixels), the *L3C* codec proved not only slower but achieved lower lossless compression efficiency compared to *WebP 1.0.2*. Much, therefore, depends on the selected set of test images, and in the modern world, it is rare to find pictures with such small resolutions as 32×32 or 64×64 preferred by the "deep learning world". Confirmation of this uncomfortable conclusion can be found in the work of [46], where explicitly, the authors state that for high-resolution images, conventional codecs still have an advantage (offering a lower bit average) over learning-based methods. Another research team in the paper [36], after an in-depth analysis of the deep neural networks (DNNs) literature, also came to the opinion that existing lossless image methods based on learning tend to suffer from too low coding speed and are difficult to apply to practical image compression tasks at full resolution.

In a paper by [47], the authors boasted the higher efficiency of their *HiLLoC* solution relative to *IDF* while showing the possibility of parallelizing the coding process. The only published result of 500×374 image encoding time is about 29 s (on a six-core computer with GTX 1060 GPU support).

The paper [36] also did not specify the CPU clock, instead comparing several codecs for moderately high-resolution images (against other papers on the subject), including 768×512 pixels, where the authors presented a method *DLPR* characterized by high efficiency (for moderately high-resolution images) and 6.5 times shorter encoding time of 1.26 s (and 4.4 times shorter decoding time of 1.8 s) relative to *L3C*. Interestingly, the results of testing the same *L3C* codec can be found in the work of [48], where the encoding time (for images with a resolution of 512×512 pixels) was found to be 12.8 times shorter than for *WebP 1.0*. Again, the authors should have specified which hardware the experiments were conducted on. However, this shows a discrepancy in the conclusions about encoding/decoding times that can occur depending on the GPU technology used, which the *L3C* codec uses.

In the work [49], we obtained reliable information about the architecture of the hardware used for testing: NVIDIA GeForce RTX 2080 GPU + Intel i7-9700 CPU (8 cores, 3.0/4.7 GHz in Turbo mode). From this research, the average encoding/decoding times of an image with a resolution of 512×512 pixels using *L3C* are, respectively, 0.49/0.54 (unfortunately, no comparison with the encoding times of classical solutions is included). In the test mentioned above, the authors also present their own *LPPLIC* solution, where they honestly demonstrate how much, in the case of methods using neural networks, the encoding and decoding time depends on the ability to parallelize the algorithm and the use of GPU technology to support this parallelization. In the sequential implementation of the decoding algorithm, the ratio of decoding time to encoding time is 433.5:1 (with the decoding time for an image with a resolution of 512×512 pixels being 711.03 s, with the *LPPLIC* decoding time decreasing to 26.26 s when using the parallelized version).

None of those mentioned above indicate which compression option was used for *WebP* (representing the classic codecs in the comparisons), directly affecting its compression efficiency (bit average) and coding time. Only the authors of the paper [50] indicated that they used the switch offering the shortest encoding time for compression (which translates into less than the minimum bit average that *WebP 1.2.1* can offer). They also changed the original encoder to their own to speed up encoding in *L3C*.

In summary, the above analysis of recent work indicates that the ability to parallelize the encoding/decoding process and GPU performance plays a crucial role in methods with learning (using neural networks). Versions of eminently sequential algorithms require (compared to classical solutions) very long encoding and especially decoding times. Despite the shortcomings mentioned here, the next section will analyze several representatives of DNN-type solutions in terms of compression efficiency against classical solutions (including our proposed solution).

4.3. Comparison of the Effectiveness of the Proposed Solution

In Section 3, we proposed a new way to compress images based on, among other things, context splitting and $IRLS_+$ minimization. Because it is time asymmetric (with a reasonably short decoding time), we proposed two versions of the encoding: a baseline (with a fixed set of parameters, such as the prediction order $r = 80$) and an extended version that includes a pre-selection of parameters (12 steps of the Algorithm 5). An analysis of the encoding time shows that for an image with a resolution of 512×512 pixels, the encoding time of the base version is 5.06 s, and in the enhanced version it varies within 36.6–116.36 s. Concerning encoding/decoding time measurements, it should be noted that the current C-language codec implementation has not been parallelized, the code optimization process has not been performed, and the latest technologies, such as those using the GPU, have not been used. At the same time, it should be noted that due to its simplicity, the decoding algorithm lends itself to easy hardware implementation.

The advantage of the proposed solution is the short decoding time, which is (depending on the prediction order) between 0.145 s and 0.165 s for the `Lennagrey` image, which can be considered an acceptable value. Although faster solutions offer faster encoding and decoding times (e.g., *JPEG-LS* or *WebP*), they cannot compare in terms of compression

efficiency. Table 5 compares the bit averages for nine most popular test images of the proposed solution with those obtained by several methods known from the literature with fast encoding and decoding times. Based on this comparison, it can be seen that our proposed solution offers, on average, 9.1% lower bit average against *JPEG-LS* and 8.7% against *WebP 2* (in the highest efficiency mode using the following settings: "`-q 100 -alpha_q 100 -effort 9`".

On the other hand, among time-symmetric solutions with a comparable, albeit slightly higher, bit average (relative to our solution), one can point out, for example, such as *Glicbawls* [51], where the decoding time is 4.47 s). Among the methods using neural networks, there is an extensive range of encoding times; one example characterized by a reasonable time can be $LCIC_{\text{duplex}}$ (1.14 s on an i5-7600 processor 3.5 GHz) [37]. Again, the proposed solution offers, on average, a 4.55% shorter bit average, as presented in Table 6, where a comparison of both classical codecs (no-learning category) and those representing the deep learning category (*L3C*, *CWPLIC*, $LCIC_{\text{duplex}}$) could be found.

Table 5. Comparison of bit averages (for nine standard test images) obtained by known methods.

Image	*WebP 2*	*JPEG-LS*	*CALIC*	*JPEG XL*	*Blend-7* [29]	*HBB* [28]	$ALCM_+$ [27] $r = 24$	*Glicbawls* [51]	Prop. Soln (Base)	Prop. Soln (+Alg. 5)
Balloon	2.92	2.889	2.78	2.757	2.84	2.80	2.733	2.640	2.622	2.617
Barb	4.51	4.690	4.31	4.271	4.43	4.28	4.211	3.916	4.049	4.035
Barb2	4.67	4.684	4.46	4.483	4.57	4.48	4.401	4.318	4.303	4.292
Board	3.64	3.674	3.51	3.497	3.57	3.54	3.425	3.392	3.336	3.317
Boats	3.94	3.930	3.78	3.786	3.84	3.80	3.696	3.628	3.605	3.591
Girl	3.89	3.922	3.72	3.696	3.76	3.74	3.612	3.565	3.528	3.520
Gold	4.45	4.475	4.35	4.352	4.42	4.37	4.293	4.276	4.243	4.235
Hotel	4.37	4.378	4.18	4.193	4.29	4.27	4.169	4.177	4.084	4.067
Zelda	3.81	3.884	3.69	3.695	3.79	3.72	3.598	3.537	3.544	3.531
Average	4.022	4.058	3.864	3.859	3.946	3.889	3.793	3.717	3.702	3.689

Table 6. Measurement of bit average values—second set of test images [37].

Image	*BPG*	*PNG*	*LCIC*	*JPEG 2000*	*JPEG LS*	*JPEG XL*	*FLIF*	*WebP 2*	*L3C*	*CWPLIC*	*LCIC* Dupl. [37]	Prop. Soln (+Alg. 5)
Airplane	4.32	4.26	3.99	4.00	3.80	3.71	3.82	3.84	4.56	3.69	3.69	3.563
Barbara	5.06	5.22	4.61	4.61	4.70	4.40	4.56	4.51	5.44	4.35	4.36	4.046
Coastguard	5.70	5.06	4.82	4.83	4.86	4.73	4.93	4.82	5.82	4.80	4.83	4.338
Comic	6.15	5.84	5.63	5.65	5.30	5.07	5.50	5.39	6.60	4.83	4.83	4.805
Flowers	5.18	5.08	4.91	4.92	4.62	4.51	4.74	4.70	5.53	4.41	4.35	4.338
Goldhill	4.95	4.70	4.58	4.59	4.43	4.37	4.50	4.41	5.27	4.33	4.33	4.175
Lennagrey	4.54	4.61	4.31	4.31	4.24	4.16	4.28	4.13	4.95	4.13	4.08	3.955
Mandrill	6.61	6.23	6.11	6.11	6.04	5.98	6.14	5.90	6.97	5.95	5.89	5.724
Monarch	4.10	4.26	3.82	3.82	3.70	3.54	3.68	3.72	4.37	3.40	3.45	3.359
Pepper	4.77	4.90	4.63	4.63	4.51	4.48	4.58	4.47	5.38	4.67	4.38	4.189
Ppt3	2.20	2.35	2.41	2.41	2.04	1.84	1.87	2.01	3.71	2.14	2.07	1.769
Zebra	5.83	5.19	4.89	4.89	4.81	4.66	4.84	4.84	6.08	4.65	4.68	4.363
Average	4.951	4.808	4.559	4.564	4.421	4.288	4.453	4.395	5.390	4.279	4.245	4.052

5. Conclusions

Based on the assumption that we usually encode an image once and decode it multiple times, users of computer systems should care about a reasonably fast decoding process. Therefore, in this work, we proposed a time-asymmetric method that uses linear and nonlinear prediction with forward adaptation at the data modeling stage. This treatment yields prediction errors with a high susceptibility to compression using adaptive variations of the Golomb code and a binary arithmetic encoder.

The solution proposed here uses an improved $IRLS_+$ method to determine linear prediction coefficients, which allows the mean absolute error to be minimized. In addition, high compression efficiency was achieved by using the proposed context-switching

algorithm, which allows the use of several prediction models tailored to the individual characteristics of each image area. In addition to the base version (with a relatively low encoding time despite the lack of need for highly efficient GPU systems, as is often the case with the latest deep learning-based solutions), we also presented an optional version extended with a method of pre-selecting compression parameters to increase its efficiency (in both cases, a satisfactorily short decoding time was achieved). Our proposed solution offers, on average, 9.1% lower bit average against *JPEG-LS* and 8.7% against *WebP 2*.

Author Contributions: Conceptualization, G.U.; methodology, G.U.; software, G.U.; validation, G.U., and M.Ł.; formal analysis, G.U., and M.Ł.; investigation, G.U. and M.Ł.; resources, G.U.; data curation, G.U.; writing—original draft preparation, G.U. and M.Ł.; writing—review and editing, G.U., and M.Ł.; visualization, G.U. and M.Ł.; supervision, G.U.; project administration, G.U.; funding acquisition, G.U. and M.Ł. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data sets for this study are available upon request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kassim, A.; Yan, P.; Lee, W.S.; Sengupta, K. Motion compensated lossy-to-lossless compression of 4-D medical images using integer wavelet transforms. *IEEE Trans. Inf. Technol. Biomed.* **2005**, *9*, 132–138. [CrossRef]
2. Sanchez, V.; Nasiopoulos, P.; Abugharbieh, R. Efficient 4D motion compensated lossless compression of dynamic volumetric medical image data. In Proceedings of the 2008 IEEE International Conference on Acoustics, Speech and Signal Processing, Las Vegas, NV, USA, 31 March–4 April 2008; pp. 549–552. [CrossRef]
3. Scharcanski, J. Lossless and Near-Lossless Compression for Mammographic Digital Images. In Proceedings of the 2006 International Conference on Image Processing, Atlanta, GA, USA, 8–11 October 2006; pp. 2253–2256. [CrossRef]
4. Ström, J.; Cosman, P.C. Medical image compression with lossless regions of interest. *Signal Process.* **1997**, *59*, 155–171. [CrossRef]
5. Xie, X.; Li, G.L.; Wang, Z.H. A Near-Lossless Image Compression Algorithm Suitable for Hardware Design in Wireless Endoscopy System. *EURASIP J. Adv. Signal Process.* **2006**, *2007*, 48–61. [CrossRef]
6. Chen, X.; Canagarajah, C.N.; Vitulli, R.; Nunez-Yanez, J.L. Lossless Compression for Space Imagery in a Dynamically Reconfigurable Architecture. In Proceedings of the Reconfigurable Computing: Architectures, Tools and Applications, London, UK, 26–28 March 2008; Woods, R., Compton, K., Bouganis, C., Diniz, P.C., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 336–341.
7. Image Data Compression. Green Book. In *Informational Report CCSDS 120.1-G-3*; The Consultative Committee for Space Data Systems: Washington, DC, USA, 2021.
8. Andriani, S.; Calvagno, G.; Erseghe, T.; Mian, G.; Durigon, M.; Rinaldo, R.; Knee, M.; Walland, P.; Koppetz, M. Comparison of lossy to lossless compression techniques for digital cinema. In Proceedings of the 2004 International Conference on Image Processing, Singapore, 24–27 October 2004; ICIP '04., Volume 1, pp. 513–516. [CrossRef]
9. Sayood, K. (Ed.) *Introduction to Data Compression*, 5th ed.; Morgan Kaufmann: San Mateo, CA, USA, 2018.
10. Weinberger, M.; Seroussi, G.; Sapiro, G. The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS. *IEEE Trans. Image Process.* **2000**, *9*, 1309–1324. [CrossRef]
11. Wu, X.; Memon, N. CALIC—A Context Based Adaptive Lossless Image Codec. In Proceedings of the 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, Atlanta, GA, USA, 9 May 1996; Volume 4, pp. 1890–1893. [CrossRef]
12. Kau, L.J.; Lin, Y.P.; Lin, C.T. Lossless image coding using adaptive, switching algorithm with automatic fuzzy context modelling. *Vis. Image Signal Process. IEE Proc.* **2006**, *153*, 684–694. [CrossRef]
13. Marusic, S.; Deng, G. A neural network based adaptive non-linear lossless predictive coding technique. In Proceedings of the ISSPA '99. Proceedings of the Fifth International Symposium on Signal Processing and Its Applications (IEEE Cat. No.99EX359), Brisbane, Australia, 22–25 August 1999; Volume 2, pp. 653–656. [CrossRef]
14. Marusic, S.; Deng, G. Adaptive prediction for lossless image compression. *Signal Process. Image Commun.* **2002**, *17*, 363–372. [CrossRef]
15. Takizawa, K.; Takenouchi, S.; Aomori, H.; Otake, T.; Tanaka, M.; Matsuda, I.; Itoh, S. Lossless image coding by cellular neural networks with minimum coding rate learning. In Proceedings of the 2011 20th European Conference on Circuit Theory and Design (ECCTD), Linkoping, Sweden, 29–31 August 2011; pp. 33–36. [CrossRef]
16. Topal, C.; Gerek, O.N. Pdf sharpening for multichannel predictive coders. In Proceedings of the 2006 14th European Signal Processing Conference, Florence, Italy, 4–8 September 2006; pp. 1–4.

17. Marcellin, M.; Gormish, M.; Bilgin, A.; Boliek, M. An overview of JPEG-2000. In Proceedings of the Proceedings DCC 2000. Data Compression Conference, Snowbird, UT, USA, 28–30 March 2000; pp. 523–541. [CrossRef]
18. Said, A.; Pearlman, W. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Trans. Circuits Syst. Video Technol.* **1996**, *6*, 243–250. [CrossRef]
19. Ulacha, G.; Stasinski, R. Context based lossless coder based on RLS predictor adaption scheme. In Proceedings of the 2009 16th IEEE International Conference on Image Processing (ICIP), Cairo, Egypt, 7–10 November 2009; pp. 1917–1920. [CrossRef]
20. Wu, X.; Barthel, E.; Zhang, W. Piecewise 2D autoregression for predictive image coding. In Proceedings of the Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269), Chicago, IL, USA, 4–7 October 1998; Volume 3, pp. 901–904. [CrossRef]
21. Ye, H.; Deng, G.; Devlin, J. Adaptive linear prediction for lossless coding of greyscale images. In Proceedings of the Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101), Vancouver, BC, Canada, 10–13 September 2000; Volume 1, pp. 128–131. [CrossRef]
22. Ye, H.; Deng, G.; Devlin, J. Least squares approach for lossless image coding. In Proceedings of the ISSPA '99. Proceedings of the Fifth International Symposium on Signal Processing and its Applications (IEEE Cat. No.99EX359), Brisbane, Australia, 22–25 August 1999; Volume 1, pp. 63–66. [CrossRef]
23. Ulacha, G.; Stasinski, R.; Wernik, C. Extended Multi WLS Method for Lossless Image Coding. *Entropy* **2020**, *22*, 919. [CrossRef] [PubMed]
24. Ye, H. A Study of Lossless Compression of Greyscale Images. Ph.D. Thesis, La Trobe University, Melbourne, Australia, 2002.
25. Wang, H.; Zhang, D. A linear edge model and its application in lossless image coding. *Signal Process. Image Commun.* **2004**, *19*, 955–958. [CrossRef]
26. Knezovic, J.; Kovac, M. Gradient based selective weighting of neighboring pixels for predictive lossless image coding. In Proceedings of the 25th International Conference on Information Technology Interfaces, 2003. ITI 2003., Cavtat, Croatia, 19 June 2003; pp. 483–488. [CrossRef]
27. Ulacha, G.; Łazoryszczak, M. Lossless Image Coding Using Non-MMSE Algorithms to Calculate Linear Prediction Coefficients. *Entropy* **2023**, *25*, 156. [CrossRef] [PubMed]
28. Seemann, T.; Tischer, P.; Meyer, B. History-Based Blending of Image Sub-Predictors. In Proceedings of the Proc. Picture Coding Symposium, Berlin, Germany, 10–12 September 1997; pp. 147–151.
29. Seemann, T.; Tischer, P. Generalized locally adaptive DPCM. *Dep. Comput. Sci. Tech. Rep. Cs97/301* **1997**, 1–15. Available online: https://dl.acm.org/doi/10.5555/789085.789575 (accessed on 13 December 2024).
30. Ulacha, G.; Stasinski, R. A New Fast Multi-Context Method for Lossless Image Coding. In Proceedings of the 2018 International Conference on Sensors, Signal and Image Processing, New York, NY, USA, 12–14 October 2018; pp. 69–72. [CrossRef]
31. Hashidume, Y.; Morikawa, Y. Lossless image coding based on minimum mean absolute error predictors. In Proceedings of the SICE Annual Conference 2007, Takamatsu, Japan, 17–20 September 2007; pp. 2832–2836. [CrossRef]
32. Burrus, C.S. Iterative Reweighted Least Squares. *OpenStax CNX* **2012**, *12*, 6.
33. Ulacha, G. Dataset of 45 Images. Available online: https://kakit.zut.edu.pl/fileadmin/Test_Images.zip (accessed on 18 November 2022).
34. Ulacha, G.; Stasinski, R. On context-based predictive techniques for lossless image compression. In Proceedings of the IWSSIP 2005-Proceedings of 12th International Workshop on Systems, Signals and Image Processing, Chalkida, Greece, 22–24 September 2005; pp. 345–348.
35. Ulacha, G.; Stasinski, R. Paths to future image lossless coding. In Proceedings of the Proceedings ELMAR-2012, Zadar, Croatia, 12–14 September 2012; pp. 63–66.
36. Bai, Y.; Liu, X.; Wang, K.; Ji, X.; Wu, X.; Gao, W. Deep Lossy Plus Residual Coding for Lossless and Near-lossless Image Compression. *arXiv* **2022**, arXiv:2209.04847. [CrossRef] [PubMed]
37. Rhee, H.; Jang, Y.I.; Kim, S.; Cho, N.I. Lossless Image Compression by Joint Prediction of Pixel and Context Using Duplex Neural Networks. *IEEE Access* **2021**, *9*, 86632–86645. [CrossRef]
38. Mentzer, F.; Gool, L.V.; Tschannen, M. Learning Better Lossless Compression Using Lossy Compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2020.
39. Rhee, H.; Jang, Y.I.; Kim, S.; Cho, N.I. LC-FDNet: Learned Lossless Image Compression with Frequency Decomposition Network. In Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 18–24 June 2022; pp. 6023–6032. [CrossRef]
40. Rahman, A.; Hamada, M.; Rahman, A. A comparative analysis of the state-of-the-art lossless image compression techniques. *SHS Web Conf.* **2022**, *139*, 03001. [CrossRef]
41. Hoogeboom, E.; Peters, J.W.T.; van den Berg, R.; Welling, M. Integer Discrete Flows and Lossless Compression. *arXiv* **2019**, arXiv:1905.07376.
42. Salimans, T.; Karpathy, A.; Chen, X.; Kingma, D.P. PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications. In Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, 24–26 April 2017.

43. Kojima, H.; Kita, Y.; Matsuda, I.; Itoh, S.; Kameda, Y.; Unno, K.; Kawamura, K. Improved Probability Modeling for Lossless Image Coding Using Example Search and Adaptive Prediction. In Proceedings of the International Workshop on Advanced Imaging Technology, IWAIT 2022, Hong Kong, China, 4–6 January 2022; Nakajima, M., Muramatsu, S., Kim, J.G., Guo, J.M., Kemao, Q., Eds.; SPIE: Bellingham, WA, USA, 2022; [CrossRef]
44. Cao, S.; Wu, C.Y.; Krähenbühl, P. Lossless Image Compression Through Super-Resolution. *arXiv* **2020**, arXiv:2004.02872.
45. Cheng, Z.; Sun, H.; Takeuchi, M.; Katto, J. Learned Lossless Image Compression with A Hyperprior and Discretized Gaussian Mixture Likelihoods. In Proceedings of the ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 2158–2162. [CrossRef]
46. Zhang, S.; Zhang, C.; Kang, N.; Li, Z. iVPF: Numerical Invertible Volume Preserving Flow for Efficient Lossless Compression. In Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2021; pp. 1–10. [CrossRef]
47. Townsend, J.; Bird, T.; Kunze, J.; Barber, D. HiLLoC: Lossless image compression with hierarchical latent variable models. In Proceedings of the International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, 26–30 April 2020. [CrossRef]
48. Shim, J.H.; Rhee, H.; Jang, Y.I.; Lee, G.; Kim, S.; Cho, N.I. Lossless Image Compression Based on Image Decomposition and Progressive Prediction Using Convolutional Neural Networks. In Proceedings of the 2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Tokyo, Japan, 14–17 December 2021; pp. 158–163.
49. Gumus, S.; Kamisli, F. A Learned Pixel-by-Pixel Lossless Image Compression Method with 59K Parameters and Parallel Decoding. *arXiv* **2022**, arXiv:2212.01185.
50. Kang, N.; Qiu, S.; Zhang, S.; Li, Z.; Xia, S. PILC: Practical Image Lossless Compression with an End-to-end GPU Oriented Neural Framework. *arXiv* **2022**, arXiv:2206.05279.
51. Meyer, B.; Tischer, P. Glicbawls—Grey Level Image Compression by Adaptive Weighted Least Squares. In Proceedings of the Data Compression Conference, Los Alamitos, CA, USA, 27–29 March 2001; p. 0503. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.