Advances in Science and Technology Research Journal

Advances in Science and Technology Research Journal, 18(6), 62–85
https://doi.org/10.12913/22998624/191111
ISSN 2299-8624, License CC-BY 4.0

Received: 2024.06.18
Accepted: 2024.08.15
Published: 2024.09.08

Two-Stage Golomb – Context-Adaptive Binary Arithmetic Coders Coding in Lossless Image Compression

Małgorzata Frydrychowicz[1*], Grzegorz Ulacha[1]

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

* Corresponding author's e-mail: malgorzata.frydrychowicz@zut.edu.pl

ABSTRACT

In this paper, a broad view of the individual stages of data modeling and encoding in the field of lossless image compression was presented. The main emphasis was placed on encoding prediction errors with the assumption of its geometric distribution. Basic and less common techniques of predictive data modeling for improving prediction efficiency by better fitting to a one-sided probability distribution were discussed. Among them, authors' own mechanism Conditional Move To Front (CMTF), which can be useful for encoding images with high variation of input data, was described. Additionally, an original two-stage mechanism for efficient prediction error encoding (used in three codecs of different computational complexities: Multi-ctx 2, 7-ctx MMAE, and Blend-28), which uses adaptive Golomb code at the initial stage and passes its binary output to context-adaptive binary arithmetic coders (CABAC), was described in detail. Separate coders for prediction error bit signs and prediction coefficients (often forming large block in header data) were also introduced. In particular, the authors focused on the important role of correct contextual division, when sections with different characteristics of their nearest neighborhood are grouped into separate classes to compress the data within each of them as efficiently as possible. From experimental studies, it was concluded that the minimum mean absolute error (MMAE) method is superior to the minimum mean square error (MMSE) method in determining linear prediction models, especially for images with low noise level. Connecting the mechanisms known from the literature with authors' ideas in the Blend-28 codec enabled to increase compression efficiency in comparison to the modern and popular WebP codec by achieving an approximately 11% shorter bit average.

Keywords: lossless image compression, adaptive arithmetic coding, linear prediction, context-dependent constant component removing, prediction coding.

INTRODUCTION

Regardless of technological progress, where disc capacity (in local and cloud environments) continues to grow, offering steady cost reduction of storing data, compression still finds its use driven mainly by economic reasons. Images and video compression can be divided into lossy and lossless, and the selection of one of them depends on the application. This study focuses on lossless compression; however, some of the aspects described here can be used to improve lossy methods as well. For simplicity, the presented results refer to the coding of the grayscale images.

One of the main applications of lossless image and video compression is archiving medical images [1, 2]. This includes not only 2D and 3D but also 4D (three-dimensional video sequences) [3, 4]. Another significant use is a compression of astronomical and satellite images [5, 6]. Moreover, the lossless mode is often required during digital photo processing, creating advertising materials, and film post-production [7]. In such cases, lossy methods, such as JPEG or JPEG2000 cannot be used. Although these standards have lossless modes, they are not particularly efficient. For this reason, new solutions to improve compression efficiency are still being developed.

Modern compression methods typically consist of two phases: decorrelation (data decomposition) and entropy coding, among which arithmetic and Huffman codes are the most efficient [8]. The purpose of data decorrelation is to remove most of the redundancy from it to achieve the best efficiency during the final coding. Linear and nonlinear prediction methods are often used at this stage, and this stage requires the highest computational complexity. For this reason, most papers discussing new approaches to lossless image compression treat descriptions of the second phase (coding of data after redundancy removal using prediction methods) in a limited manner. In this paper, the authors wanted to show the importance of connecting these two stages together in order to achieve the highest possible compression efficiency.

Depending on different requirements, compression methods can be divided into three main categories in terms of encoding and decoding time. The first one requires fast time of encoding and decoding (time symmetric). Typical examples of this category are JPEG-LS [9] and CALIC [10]. WebP [11] is one of the newest in this class. The second one is characterized by long encoding and short decoding time. This is due to the fact that compression requires adaptation of parameters and multiple attempts to data coding. It uses forward-adaptation (prior knowledge about the coded image) and requires saving large-sized information (already tuned final parameters for the decompression algorithm) in the header section of output file. In this category, one can find e.g. TMW (1997) [12] or its later enhancement TMW^{LEGO} (2001) [13]. The most efficient in this category is the new version of MRP 0.5 codec presented under the name 'VBS & new-cost' (2005) [14]. The third category is the most complex in terms of computation and is characterized by long encoding and decoding time. In such solutions, backward-adaptation is generally used (parameters are tuned on the fly after receiving new data), and there is no need to store a large header block for the decoder. Several known mechanisms can be found in the algorithms of this category, such as RLS [15] and OLS/WLS [16], where coding and decoding of each consecutive pixel is accompanied by a linear predictor's coefficients adaptation. This adversely affects the encoding and decoding time, but also enables the use of high-order predictors without the need to increase the size of header. In this paper, the results of the three proposed solutions were presented, one for each of the three described categories. There are other approaches to data modeling. Some of these methods use wavelet transforms e.g. JPEG2000 [17] or ICER [18]. In recent years, methods based on deep learning that fully utilize nonlinear neural networks [19, 20] have also been developed. They are also characterized by high computational complexity and short (not in all cases) encoding/decoding time is achieved owing to the possibility of high-level parallelization using GPU/NPU technologies only. Insight into different approaches to lossless compression can be found in the literature [21, 22].

The basics of linear prediction, context-dependent constant removal, and other mechanisms for decreasing the average prediction error are described in the Basics of data modeling section. Unlike most works which focus on describing prediction methods, this one is dedicated to show various aspects of prediction error encoding widely. The Efficient prediction errors encoding section presents a broad view of prediction error encoding methods, such as context modeling or nonlinear quantization of prediction errors. The two-stage method proposed as a final encoding solution for data with geometrical distribution utilizes an adaptive Golomb coder of prediction errors as a first step, and then, as a second, its binary output is coded with a context-adaptive binary arithmetic coder. A separate mechanism for encoding the sign information of the prediction error was also presented. A similar context arithmetic coder concept is used to compress the prediction coefficients. All three codecs proposed in this paper were compared to the solutions known from literature against the compression efficiency and implementation complexity (with distinction to encoding and decoding time). In addition, the usefulness of individual improvements, which may be used to design new efficient methods for lossless image compression, was analyzed.

BASICS OF DATA MODELING

Application of linear prediction

In two-dimensional signals, such as images, the data modeling phase boils down to remove as much mutual information between neighboring pixels (spatial correlation) as possible. To do this, the predicted value $\hat{x}_n$ (rounded to an integer value) of the encoded pixel x_n is calculated, and

then the difference between these two, known as prediction error, is encoded:

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

In general, prediction errors e_n tend to be small values oscillating around zero and their probability distribution is close to the Laplace distribution. The predicted value can be calculated using the r-order linear predictor:

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

where: b_j are the prediction coefficients that form a vector $\mathbf{B} = [b_1, b_2, ..., b_r]$ and $P(j)$ is the j-th nearest neighbor of the currently encoded pixel x_n.

Figure 1 illustrates the 48 numbered nearest neighbors of a coded pixel x_n, where j-th number is used to indicate pixel $P(j)$ and prediction error $e(j)$. This is a relative numbering to the currently encoded pixel (index 0), meaning that it can be translated into $x_n = P(0)$ and $e_n = e(0)$. The image is coded row by row from left to right. The simplest possible predictors are the values of the individual pixels ($r = 1$ and $b_1 = 1$), for example, $\hat{x}_n = P(4)$ means that the predicted value of pixel x_n is the value of its 4th neighbour.

The methods utilizing higher-order predictors are usually based on MMSE because of the simplicity of calculating the vector **B** of prediction coefficients [8]. However, this implies that the distribution of prediction errors is close to the Gaussian distribution. Experiments have proven that a better distribution for achieving higher compression efficiency is the Laplace distribution, which can be obtained by MMAE [23]. A wider explanation of the influence of MMSE on entropy was discussed in [24].

Constant component removal

Prediction methods tend to introduce a constant component (bias) in the calculated prediction errors, which depend on the characteristics of the individual context. Hence, many solutions propose an adaptive method for constant component removal (bias cancelation), also known as context-based error correction techniques. By adding a new context-dependent constant component removing (CDCCR) block, which removes the constant component C_{mix} correlated with a certain context, it is possible to continue increasing precision of prediction. This idea has been used in earlier codecs, such as JPEG-LS and CALIC. The constant component C_{mix} removal correlated with one of the 1024 contexts results in a slight modification of Equation 1:

$$e_n = x_n - [\hat{x}_n + C_{mix}] \tag{3}$$

where: $\hat{x}_n$ is calculated using Equation 2.

The context number κ for each of subsequential encoded pixel x_n can be determined using a set of binary rules. If $z_i > \hat{x}_n$, then bit $\kappa_i = 1$, otherwise $\kappa_i = 0$. The nearest neighboring pixels or their linear combination can be used as a z_i. The example set of eight z_i values to create an 8-bit number $\kappa_7\kappa_6\kappa_5\kappa_4\kappa_3\kappa_2\kappa_1\kappa_0$ which offers highly efficient contextual division is following: {$P(1)$, $P(2)$, $P(3)$, $P(4)$, $P(5)$, $P(6)$, $2P(1) - P(5)$, $2P(2) - P(6)$}. If an additional 2-bit (four-state) quantizer with three thresholds {300, 2000, 8000} is used to quantify the number:

$$\hat{\sigma}^2 = \sum_{i=1}^{8} (\hat{x}_n - z_i)^2 \tag{4}$$

then 10-bit context number is obtained. Each of the 1024 contexts has its own sum of prediction errors (updated on the fly). This sum, divided

			46	42	38	43	47		
		37	32	26	24	27	33	39	
	36	29	20	16	14	17	21	30	40	
45	31	19	11	8	6	9	12	22	34	48
41	25	15	7	3	2	4	10	18	28	44
35	23	13	5	1	x_n					

Figure 1. Neighborhood pixel numbering

by the current number of occurrences in this context, gives us a context-dependent constant C_{mix}. The value of C_{mix} changes over time in each context. These fluctuations, recorded in an example context, are shown in Figure 2. More details about the building of context number and CDCCR block usage are discussed in [25]. Figure 3 shows a fragment of the probability distribution for the Lennagrey image after coding using linear prediction. The solid line represents the probability obtained without context component removal, whereas the dashed line shows the

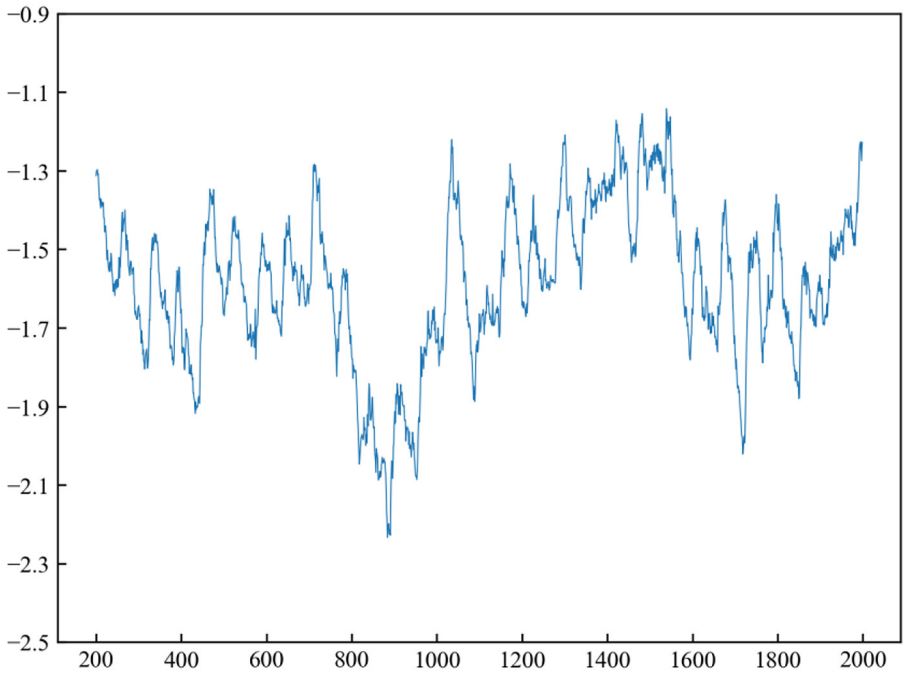

Figure 2. Value C_{mix} of example context, which adapts during encoding of subsequent pixels of the image

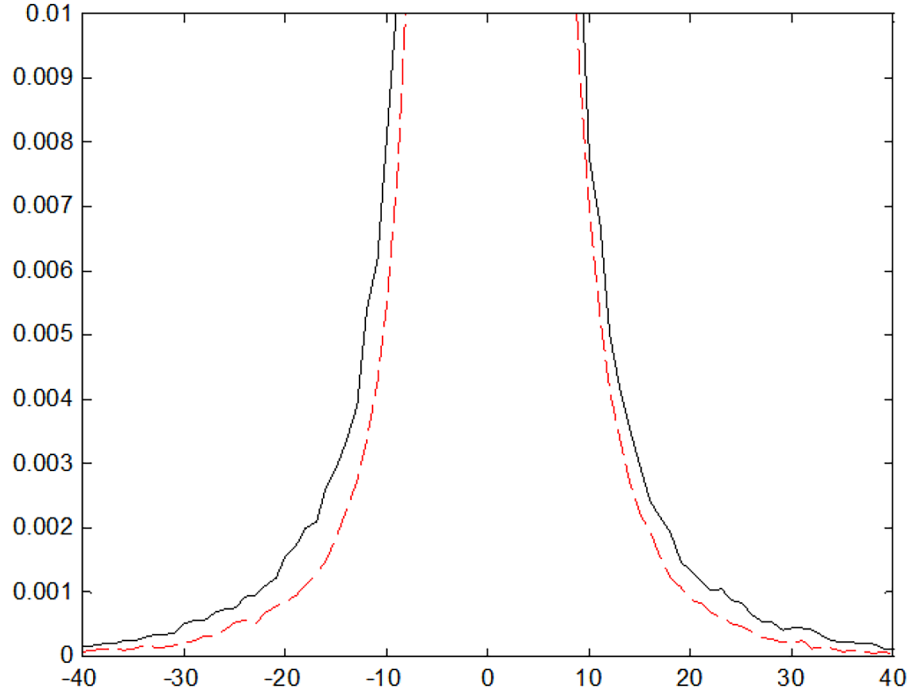

Figure 3. Probability distribution for Lennagrey after linear prediction encoding without CDCCR (solid line) and with CDCCR (dashed line)

results with context-dependent constant component removal. In the latter case, it is possible to notice a greater concentration of prediction errors among values that are close to zero.

Basic improvements in codecs

The vast majority of publications skips less important implementation details, leaving readers clueless about the integrated improvements. For this reason, the paper closely looked at the basic mechanisms that can be used to improve compression efficiency.

Pixel encoding and decoding process usually has a sequential flow – row by row, starting from left to right. Linear prediction uses the left and upper neighboring pixels of the currently coded one. The fundamental problem of using linear prediction of a higher order is the need to access data outside the image boundaries. Taking the first row as an example, it was quickly found out that there are no upper pixels that can be accessed following the numbering scheme (Figure 1). To overcome this, a special margin is being built around the original image (excluding the bottom). Different approaches to fill this margin exist, and the easiest approach is to fill every cell with a 0. A much better solution is to encode the first row with differential code (prediction error is calculated as the difference between current pixel x_n and its left neighbour $P(1)$). A similar procedure is applied to the first left column, but everything is done vertically (difference between current pixel x_n and its top neighbor $P(2)$). The first row and first left column can then be copied n times into the margin area, as shown in Figure 4.

The second improvement is a mechanism for detecting flat regions, useful especially for artificially generated images. The detection is limited to checking whether the four nearest pixels $\{P(1), P(2), P(3), P(4)\}$ are equal. When this condition is satisfied, the default predictor is replaced with the value of the pixel $P(1)$.

The next improvement can be optional, depending on the number of different shades used in a certain image. If the number of shades is significantly lower than the maximum (in the case of 8-bit images, there can be up to 256 different shades), then the range of shades can be reduced. The information which shades have been used in original image must be saved in header data for decoder. Each consecutive shade present in an image is assigned with a number from 0 to $h - 1$, where h is the total number of shades in use [26]. Figure 5 shows an example of reducing 9 shades present in an image with a 16 – shade palette into new shades in a form of consecutive numbers (symbol X indicates a non-existent shade in image). In the last row, the index values next to the new shade values indicate the original shade number. In the header section of the file, a vector consisting of 256 bits is placed with h ones (representing the shades present in the image) and $256 - h$ zeros (representing the shades that did not appear in the image). Thus, the decoder can reconstruct the original shades after main decoding process. In addition, this vector can be encoded using a binary adaptive arithmetic coder.

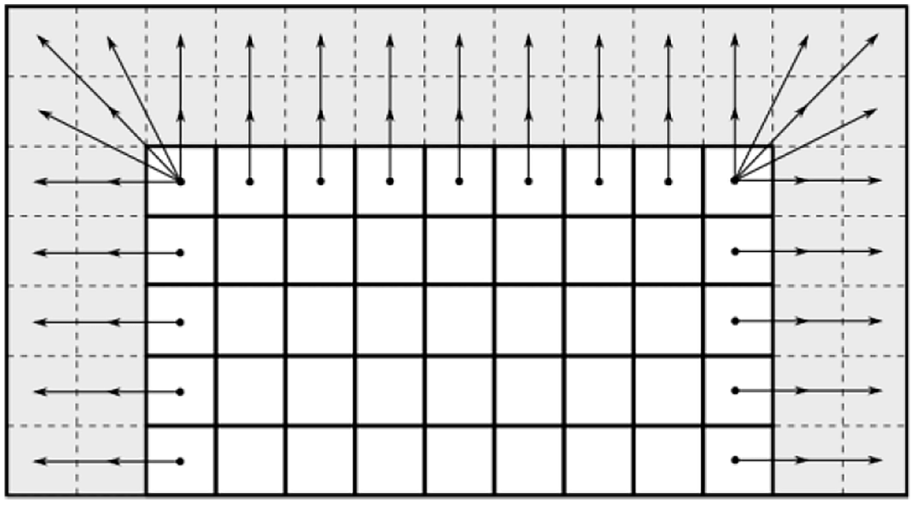

Figure 4. Scheme of filling margins surrounding coded image

Input shade	0	1	X	3	4	X	X	7	8	9	10	X	X	X	14	X
Header metadata	1	1	0	1	1	0	0	1	1	1	1	0	0	0	1	0
Output shade	0_0	1_1	2_3	3_4	4_7	5_8	6_9	7_{10}	8_{14}	X	X	X	X	X	X	X

Figure 5. Shade range reduction scheme

There is also a possibility of utilizing reversible image transformation involving n rotations by 90° (with $n = \{0, 1, 2, 3\}$, there are four phases of rotation). If mirroring is included, eight lossless image transformations are obtained. This improvement requires eight complete or simplified encodings, which has a negative impact on encoding time, but it has no significant impact on decoding time.

There are few examples when even the most complex high-order prediction methods are less efficient than a first-order predictor that uses one of the neighboring values e.g. 4 – th or 6 – th neighbor (known as the special mode). After coding with the main method, a mechanism that checks the outcome influence of fast encoding with a first-order predictor multiple times can be integrated. An example of image with such characteristics is discussed in the Performance analysis of the individual improvements section.

Further data modeling

Linear and nonlinear prediction may result in prediction errors of normal distribution when using MMSE or Laplace distribution when using MMAE. MMAE usually has a better efficiency, but the downside is a computational complexity, as for example in simplex method it has exponential worst-case complexity. In the paper [23] lowering of the complexity is discussed; however, the taken steps seem to be insufficient. If it is assumed that the optimal solution is not as much important because at the end, the bit average of file is to be minimized, and not mean absolute error, then minimizing methods that give approximate local minimum can be used. The second argument for introducing simplifications in minimizing mean absolute error is the fact that according to Equation 1 predicted values are rounded to the nearest integer. The iterative reweighted least squares (IRLS) method offers that with relatively short execution time. Its improved variant was proposed in [27]. It requires approximately two times fewer iterations than the classic solutions. This approach has been integrated in the Multi-ctx 2 and 7-ctx MMAE codecs, which will be described in the upcoming sections.

For practical implementation of Laplace distribution for integer numbers n from range $[-N,N]$ (e.g. of prediction error, n is replaced by e_n) with the assumption that the average is equal to 0 and variation equals σ^2, the following equation can be used [28, 29]:

$$L(n) = \begin{cases} 1 - e^{-\frac{1}{\sqrt{2\sigma}}} & n = 0 \\ \frac{1}{2}\left(e^{-\frac{|n|-0.5}{\sigma/\sqrt{2}}} - e^{-\frac{|n|+0.5}{\sigma/\sqrt{2}}}\right) & 0 < |n| < N \\ \frac{1}{2}e^{-\frac{|n|-0.5}{\sigma/\sqrt{2}}} & |n| = N \end{cases} \quad (5)$$

Considering usage of discrete probability distribution suited for integer numbers, in the paper [30] it was pointed out, that if $e_n = 0$, then noticeable deviation of the actual probability value from the theoretical value can occur. Therefore, a special modified distribution called Laplace+δ can be used.

In the research [31], Deng and Ye have emphasized the importance of differentiating between probability distributions, suggesting that Laplace distribution works well when it comes to the distribution of prediction errors for the entire image. However, the Gaussian distribution (associated with minimizing mean square error) is more suitable for modeling local characteristics. This is especially true for the methods incorporating forward adaptation, where predicted values are determined based on causal neighboring pixels, which was confirmed by the experiments presented in [16]. The hardest part is to specify which distribution is most favorable when forward adaptation is in use with a simultaneous image division into small blocks [23].

Shrinking 8-bit prediction errors range by half from [-255, 255] to [0, 255], can be beneficial for adaptive coding of prediction errors. Shrinking algorithm has two stages [32, 33], the first one is:

$$\begin{aligned} e_n &\leftarrow e_n + 256, \text{if } e_n < -128 \\ e_n &\leftarrow e_n - 256, \text{if } e_n > 128 \end{aligned} \quad (6)$$

The second stage, which is often used in audio compression, enables building of a one-sided distribution:

$$\begin{aligned} e_n &\leftarrow 2 \cdot |e_n|, \text{if } e_n \geq 0 \\ e_n &\leftarrow 2 \cdot |e_n| - 1, \text{if } e_n < 0 \end{aligned} \tag{7}$$

However, another more efficient error mapping approach that works smoothly along with the context arithmetic coder (described later in this paper) exists. It is noteworthy that the prediction error can take 256 different values. For example, when x_n = 5, the prediction error range is [-250, 5]. The distribution is symmetric between [-5, 5] and remaining [-250, -6] forms a sort of long tail. The tail can be balanced by alternatingly casting each next value from that range into negative and positive: -6 and 6, -7 and 7 etc. By doing so, the distribution becomes symmetric and mean absolute error is decreased. To perform prediction error mapping, the algorithm 1 is used. x_{max} is the maximum shade value, and n & 1 operation returns the least significant bit of n (the same operation can be achieved with n mod 2). Algorithm 1 limits e_n range to [-128, 128]. For example, when x_{max} = 255, x_n = 105, and $\hat{x}_n$ = 25 prediction error equals e_n = 105 – 25 = 80. The application of algorithm 1 reduces the value of coded error to e_n = -53, which cannot be achieved using Equation 6. Algorithm 2 is used in decoder to reverse mapping process and recover the original prediction error value.

Algorithm 1. Prediction error mapping

$\theta = \hat{x}_n$;
if ($\theta > (x_{max} - \theta)$) $\theta \leftarrow x_{max} - \theta$;
if ($|e_n| > \theta$) $e_n \leftarrow \left\lfloor \frac{|e_n|+\theta+1}{2} \right\rfloor \cdot \left(1 - 2 \cdot ((|e_n| + \theta)\&1)\right)$

Algorithm 2. Reversing prediction error mapping

```
if ( x̂_n ≤ (x_max − x̂_n )) {
        θ = x̂_n ;
        if (|e_n| > θ) {
                if (e_n < 0) e_n ← 2·|e_n| − θ − 1
                else e_n ← 2·|e_n| − θ ;
        }
} else {
        θ = x_max − x̂_n ;
        if (|e_n| > θ) {
                if (e_n < 0) e_n ← θ − 2·|e_n| + 1
                else e_n ← θ − 2·|e_n|;
        }
}
```

Because of the practical aspects of designing an adaptive arithmetic coder, one should strive towards a one-sided Laplace distribution. For this purpose, instead of using Equation 7, the symmetry of distribution received after applying algorithm 1 shall be used, then both absolute value of prediction error $|e_n|$ and its sign shall be coded separately. Hence, instead of using the Laplace distribution, it is easier to use the geometric distribution, which in classic approach is calculated as:

$$G(n) = (1 - p) \cdot p^{n-1} \tag{8}$$

where: n denotes a positive natural number.

Because in place of n the integer value of absolute prediction error $|e_n|$ is to be used, to include value 0 as well, it would be more convenient to modify equation into:

$$\bar{G}(n) = (1 - p) \cdot p^{n} \tag{9}$$

where: n is a non-negative integer.

EFFICIENT PREDICTION ERRORS ENCODING

Practical basics of prediction error coding

The data modeling methods discussed above are just preparing the data for the main coding procedure, during which prediction errors are transformed into a sequence of bits to save in the output file. In addition, there is an approach that entirely skips the prediction stage and instead of prediction errors it directly adaptively encodes pixel values [34], however it is the least popular. Usually, this results in very high computational complexity compared to the methods described in this chapter. To achieve high compression efficiency, the Huffman code (or its variations appropriate for geometric distribution, such as Golomb or Rice codes) and arithmetic code are usually used.

Both Huffman and multi-value arithmetic code have an advantage over Rice and Golomb codes that the probability distribution is unrestricted to any particular type and can be non-monotonous. It must be stressed, that the classic Huffman code is optimal among codes, which can be described in the form of a tree. Despite this, it also has some disadvantages, for example the shortest possible codeword can be 1 bit long, and the optimal probability of such coded symbol is 1/2 (in the case of higher probabilities, the efficiency of Huffman code decreases). Arithmetic

code does not suffer from flaws like this, owing to algorithm which allows information to be saved on a "bit fraction" (sending one bit to the output can be a consequence of encoding several information modifying floating point base value). For this reason, many modern lossless image compression methods use various variants of arithmetic code, including adaptive binary arithmetic code. The integer implementation is described in detail in [8]. The use of any of the mentioned codes requires a high level of knowledge about the data probability distribution. It can be either complete knowledge about distribution (static data saved in header section of output file, which size can have noticeable negative influence on compression efficiency) or partial – in form of concise parameters description of one of the classical probability distributions.

Both approaches can be used in adaptive algorithms. In this case, an initial probability distribution is determined by knowing the values of the parameters. For example, with a geometric distribution, the initialization of the probability vector is simple and can be defined using a single-parameter equation [35]:

$$\mathrm{N}_e(i) = \lfloor f \cdot 0.8^i \rfloor + 1 \tag{10}$$

where: i is a value from [0.255] range, and f is an amplitude.

Good results were obtained when $f = 10$. Here, symmetry of the distribution for negative numbers is assumed.

Adaptation means that after reading and encoding each next absolute prediction error $|e_n| = |e(0)|$, the values stored in vector N_e under index $|e_n|$ are updated by 1: $\mathrm{N}_e(|e_n|) \neg \mathrm{N}_e(|e_n|) + 1$. The current probability of value $|e_n|$ is calculated as a ratio of the $\mathrm{N}_e(|e_n|)$ to the total count of all encoded prediction errors so far. It is noteworthy that the initial setting is usually non-optimal. Secondly, randomly incoming data at the beginning can noticeably alert temporary probability distribution (comparing to the real distribution of the entire file content). For this reason, it is worth to experimentally estimate the initial probabilities.

As an example, the binary distribution was analyzed, where two fractions 1/2 and 6/18 give the same result 0.5, which is the initial probability of occurrence of bit 0. During encoding phase, after receiving another 0, the probabilities were updated to 2/3 in the first case, and to 9/17 in the second case (here, the percentage increase is much lower). Without knowledge about the real distribution, even in such trivial case it cannot be judged which version is more efficient. A lot of depends on what was the expected probability value (average of whole file, which could be equal to e.g. 0.3) and if the first adaptation causes change forwards correct or the opposite (distancing from expected decisively) direction. Therefore, when designing a codec that uses the adaptive form of probability distribution, the value of the parameter f in Equation 10 should be determined on the basis of preliminary statistical analysis using a training set.

Moreover, the possibility of variability in distribution features in different areas of the encoded image (non-stationary data) should be considered. This is why it is worth introducing the forgetting mechanism to reduce occurrence counters that have not been encoded lately or have occurred rarely in comparison to earlier encoding stages [36]. For this purpose, it is necessary to monitor the total count of values encoded so far (more specifically, the counter that increases by one each time $|e_n|$ is encoded). When the counter reaches $N_{\max} = 2^s$, then all elements from vector of occurrences are halved using the assignment:

$$\mathrm{N_e}(i) \leftarrow \left\lfloor \frac{\mathrm{N_e}(i)}{2} \right\rfloor + 1, \text{for } i \in [0, e_{max}] \tag{11}$$

After scaling, the total occurrence counter is recalculated as the sum of all i elements from the vector of occurrences N_e. Good results are offered by $s \geq 10$. This mechanism can be described as a long-term adaptation.

Use cases of CMTF method

Similarly to linear prediction a classic Move To Front (MTF) method is used for data modeling. Data obtained after transformation with MTF is characterized by a more favorable probability distribution.

For some images (especially those with a high level of noise), even a highly efficient prediction method does not yield a decreasing probability distribution function of the absolute prediction errors $|e_n|$, which is the case with a geometric distribution. In this situation, the modified conditional MTF (CMTF) method can be used. The version proposed here is beneficial in arrangement of values $|e_n|$ by the decreasing occurrence number, but it is happening gradually after successively encoded prediction errors. Assuming that the probability distribution of $|e_n|$ is unknown, a 2-row

table is built before coding begins. The first row stores sequentially arranged values {0, 1, 2,..., $|e_{max}|$}, while the second row stores the occurrence counters of each $|e_n|$. Initially, all elements in the second row are set to 0. During encoding phase of $|e_n|$, the value of index I_n to the table element associated with $|e_n|$ is determined. For example, if $|e_n| = 5$, then the initial index is $I_n = 5$. The index is then encoded with arithmetic code, and the occurrence counter associated with the prediction error $|e_n| = 5$ is incremented by 1. In addition, a comparison between the two adjacent indices is performed: $I_n = 5$ and its preceding index $I_{\text{left}} = 4$. If occurrence counters are not sorted descendingly, they are swapped (like in bubble sort). This means that if the next encoded value would be $|e_n| = 5$ once again, then its index in table would be $I_n = 4$, therefore 4 would be encoded. In this way, the most frequent $|e_n|$ values in table are gradually moving towards the beginning of index table (in classic MTF each $|e_n|$ occurrence moves itself to the front without checking how many times this value occurred earlier).

The proposed CMTF method can be optionally activated at the stage of tuning the parameters of the arithmetic encoder.

Context based division in arithmetic code

Beside long-term adaptation described earlier in the Practical basics of prediction error coding chapter, the fact that there are short-term dependencies between successively encoded data can also be used by analyzing the nearest neighborhood consisting of pixels $P(j)$ and prediction errors $e(j)$ (consistent with the spatial arrangement presented in Figure 1).

On the basis of the characteristics of neighboring prediction errors, it is possible to determine approximate type of probability distribution of currently encoded $|e_n|$ value quite accurately. According to this assumption, a contextual arithmetic coder that does not have one, but t probability distributions associated with appropriate context numbers (classes of different characteristics) from 0 to $t - 1$ can be designed. Theoretically, with an increase in the number of contexts, improvement of compression efficiency is expected. Assuming that the distributions are not known beforehand, because their form is being built with the backward adaptation method, the problem of context dilution is encountered, which results in too slow adaptation of distributions. Adaptive distribution adaptation is expected to quickly estimate the target form for each of t distribution. Therefore, a compromise must be made between the number of contexts and the speed of the distribution adaptation. Sometimes, only few contexts are used (4 in [37] or 8 in [38]). However, the most common method is to use a dozen, for example, 12 used in [39] or 16 used in [40]. In one case, as many as 20 contexts were used [41]. An analysis of the influence of the number of contexts on the bit average was presented in [42]. In this paper, using 16 contexts was proposed, and the context division details were described in the Rules of context number calculation – proposal 1 section.

The first idea for a twice as fast adaptation of distributions in the case of many contexts is the fact of prediction error distribution symmetry and separate encoding of the absolute error $|e_n|$ value and its sign bit (for 0, there is no need to store sign data). The second idea is to use range encoder.

Range encoder

In contextual encoding, additional mechanisms are often used to increase the speed of adaptation of distributions associated with an appropriate context. One of them is to divide the $|e_n|$ values into two ranges [43]. The first one is used in 95% of the cases and it covers typical errors from range [0; Δ]. The second one covers values larger than Δ, which rarely occur. Both ranges are encoded using different probability distributions. Saving of value $|e_n| > \Delta$ requires encoding of escape signal in the form: $(\Delta + 1)$, and then complementary value in the form $|e_n| - \Delta$, but based on the probability of rare values. This idea can be extended to more ranges, which requires iterative usage of escape signal [37].

The second approach, which also divides $|e_n|$ values into ranges is based on $|e_n|$ quantization [22]. The $|e_n|$ values from range [0, 255] are casted into a smaller range k, for example from 0 to 17. This technique is used in many coding methods, including the JPEG standard [8] being one of them. The quantization technique, together with lossless prediction error encoding, is designed to divide a single $|e_n|$ into two values. Then, they are encoded with different distributions (it is sometimes assumed that the residue value left after quantization is encoded as a bit stream without compression). The quantization and encoding phases are described below.

At the beginning, having an example set of consecutive quantization thresholds $\mathbf{T}$ = {0, 1,

2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20, 24, 32, 64, 128} the value of k is calculated using relation $\mathbf{T}(k) \leq |e_n| < \mathbf{T}(k+1)$. The quantized value k is then sent to multi-value arithmetic coder, and encoded after associating it with an appropriate probability distribution assigned to given context. Residue $e_q = |e_n| - \mathbf{T}(k)$ produced after quantization is treated as $\mathbf{q}(k)$ bit value where $\mathbf{q}(k)$ is read as k-th value from vector $\mathbf{q}$ = {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 3, 5, 6, 7}. If $\mathbf{q}(k) > 0$, then e_q can be sent to the output as a sequence of $\mathbf{q}(k)$ bits or encoded via one of the seven adaptive arithmetic coders (of index $\mathbf{q}(k)$).

To decode, the k value is read. k is an index pointing to the threshold vector $\mathbf{T}$ and to the vector $\mathbf{q}$ that stores the number of quantization bits. When $\mathbf{q}(k) > 0$, then decoder (of index $\mathbf{q}(k)$) reads residue e_q value, otherwise $e_q = 0$. In the next step, $|e_n|$ is determined using relation: $|e_n| = \mathbf{T}(k) + e_q$.

Let $|e_n| = 29$, then $\mathbf{T}(14) = 24 \leq 29 < \mathbf{T}(15) = 32$, which means $k = 14$, $\mathbf{T}(14) = 24$, $e_q = 29 - 24 = 5$, $\mathbf{q}(14) = 3$, therefore value $e_q = 5$ is encoded using third arithmetic coder for quantization residues (which encodes 3-bit integer values from 0 to 7). On the decoder side, value is reconstructed as $|e_n| = \mathbf{T}(14) + e_q = 24 + 5 = 29$.

Instead of quantizing prediction errors to encode $|e_n|$, a binary arithmetic coder can be used. This will be discussed in the Binary adaptive arithmetic coder section.

In addition to coding with backward adaptation for updating probability distributions, another possibility emerges, in which initial and fairly accurate average probability distributions are assumed for each context. The version with forward adaptation requires saving of distribution parameters in header section. In this case, an arithmetic coder can be designed that determines the individual probability distribution for each consecutively encoded pixel. This is a very complex problem, because it is needed to find the best possible mathematical description of distribution that matches a certain image or class of images. An analysis of this approach, based on Laplace, Gaussian, Student's t-distribution and generalized Gaussian distributions was performed in [44]. Its conclusions were used in [40], where generalized Gaussian distribution was utilized to set individual parameter for every context. A similar solution, but based on Student's t distribution, was applied in the TMW method [12], where probability distribution blending was also introduced.

Rules of context number calculation – proposal 1

The proposed solution consists of 16 main contexts with pre-initialized distributions and an adaptive rule for their update along with each next encoded pixel. This enables to use both multi-value and binary variant of arithmetic code.

An important factor affecting compression efficiency is the correct choice of decision rule that determines the context number (it is a form of short-term adaptation). The first rule presented here extends the concept proposed by Deng et al. [41, 45]. Value ω_1 is calculated based on prediction errors $e(j)$ from nearest neighborhood, where j is a neighbor index (see Figure 1) (Equation 12).

Slightly different approaches were used in [14, 42]. In the solution proposed here, similarly to [14], the next parameter ω_2 is calculated as weighted average of $m = 28$ absolute errors $|e(j)|$ from nearest neighborhood:

$$\omega_2 = \frac{1}{\delta}\sum_{j=1}^{m} \bar{d}_j \cdot |e(j)| \tag{13}$$

where the normalizing factor δ is calculated as:

$$\delta = \sum_{j=1}^{m} \bar{d}_j \tag{14}$$

and weights as:

$$\bar{d}_j = \frac{1}{\sqrt{(\Delta x_j)^2 + (\Delta y_j)^2}} \tag{15}$$

where: Δx_j and Δy_j are horizontal and vertical distance between pixels $P(j)$ and $P(0)$.

The next step is to use the correlation between 4 nearest neighboring pixels to calculate parameter ω_3(Equation 16).

$$\omega_1 = \max\{2.3|e(1)|, 2|e(2)|1.6|e(4)|, 0.95(|e(3)| + |e(4)|), 1.25(|e(5)| + |e(10)|), 1.3|(e(3)|,1.375(|e(1)| + |e(2)|),0.4(|e(6)| + |e(7)|),0.4(|e(8)| + |e(9)|)\} \tag{12}$$

$$\omega_3 = max\{|P(1) - P(3)|, |P(2) - P(4)|, 1.1|P(1) - P(2)|, 0.7|P(2) - P(3)|, 0.9|P(1) - P(4)|, 0.9|P(3) - P(4)|\} \tag{16}$$

On the basis of parameters ω_i the final value of ω can be calculated:

$$\omega = h_1 \cdot max\{2.1 \cdot h_3 \cdot \omega_1, 11.5 \cdot \omega_2\} + \\ + \frac{0.5 \cdot h_2}{3d+1} \cdot \omega_3 \quad (17)$$

where: parameter d is the maximal allowed error value in near-lossless mode [46, 47] (lossless mode is obtained with $d = 0$).

In basic version the scaling parameters h_1, h_2 and h_3 are equal to 1. They can be used for additional adaptation to currently encoded image, however, it requires many repetitions of prediction errors encoding.

The value ω is a subject to non-uniform quantization (of an approximately logarithmic nature) with t –1 thresholds $\mathbf{T}_h(i)$ to obtain arithmetic context number connected to current probability distribution. For example, with $t = 16$ the thresholds are $\mathbf{T}_h = \{3, 7, 12, 18, 24, 31, 39, 49, 59, 72, 90, 115, 140, 170, 210\}$.

Figures 6a and 6b show two examples of probability distributions of k (quantized according to the description in Range encoder section) in context of number 0 and 4 for test image Hotel.

Rules of context number calculation – proposal 2

Although in most cases the contextual division based on the set of rules described in previous section works well, for some images, a completely different approach may be more cost-efficient. The next proposition also uses $t = 16$ main contexts. It can be used for images with clusters of visibly uneven distribution of grayscale values, to which the adaptive scalar quantizer easily adjusts.

After using shade reduction mechanism, input data values from 0 to h – 1 are obtained. Therefore, the use of any prediction method will lead to calculating predicted value $\hat{x}_n$ within the same range. It can be quantized to one of 16 values, which then will become a main context number. Because of adaptive nature of the algorithm, a centroid is initialized as:

$$V(i) = \frac{(2 \cdot i + 1) \cdot h}{32}, \text{for } i = 1, \ldots, 16 \quad (18)$$

The initial occurrence counter for each centroid was set to 8. Then, during encoding of each subsequent pixel, the number of the winning i-th centroid is adapted (occurrence counter is incremented by 1) and based on $\hat{x}_n$ centroid value is also updated (arithmetic mean of previous values assigned to i-th centroid). After initial learning phase centroid values reflects central points of clusters quite well.

Binary adaptive arithmetic coder

Although in the case of a multi-valued arithmetic encoder, the proposition to quantize prediction errors and to make use of prediction errors distribution symmetry increases speed of distribution adaptation in many contexts, there is competitive solution that can offer even higher efficiency, namely CABAC. CABAC uses a higher number of contexts and an individual approach towards every bit. Moreover, assuming geometric distribution, adaptive Golomb code can be used at the initial stage and its binary output can be passed to one of the three adaptive binary arithmetic coders. Each of these coders has its own method for determining the context number, which is associated with the individual occurrence probability distribution of bits 0 and 1.

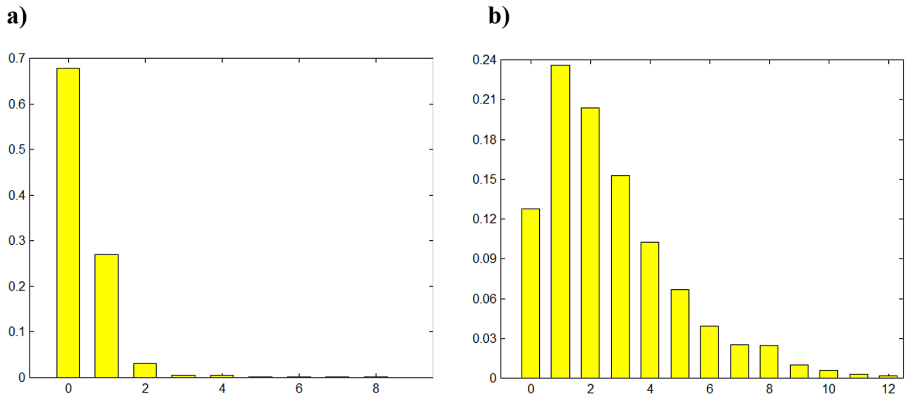

Figure 6. Probability distribution of k in image Hotel in context: (a) 0, (b) 4

Adaptive Golomb code

Division into 16 main contexts described in the Rules of context number calculation – proposal 1 section refers to the short-term energy variation of the prediction error signal, but it is not sufficient to satisfactorily consider all the dependencies that affect the best possible estimation of the probability distribution for the currently encoded value $|e_n|$. In the case of the prediction error stream, the assumption is made about the medium-term stationarity of its distribution, which is usually close to the geometrical distribution (see Equation 9). Data with such characteristics can be encoded in a highly efficient manner using the Golomb code family, which was used, i.a. LOCO-I [9].

The integer group value m (also called the Golomb code order) is selected according to the fit of the data to the distribution $G(n)$, which is dependent on the parameter p. This translates into approximation $p^m \approx 1/2$.

To consider the variability of the medium-term characteristics of the probability distribution, an individual group number m is calculated for each sequentially coded $|e_n|$. For this purpose, the method of adaptive average cost of coding with the Golomb code using one of six appropriately selected probability distributions can be used. This means, that six values of m are allowed to be in the vector **m**. To calculate the current predicted value of m, it is needed to firstly calculate p using the following formula:

$$p = \frac{S}{S+1} \tag{19}$$

where: the expected value $S = \omega_2$ (see Equation 13) with $m = 48$.

Having p calculated, m can be calculated using formula presented in [48, 49]:

$$m = \left\lceil -\frac{\log_{10}(1+p)}{\log_{10} p} \right\rceil \tag{20}$$

It is also possible to use the fact that there is a linear relationship between m and S values that equals $m \approx \ln(2) \cdot S$ [50]. Experimental studies have shown that in the case of image encoding, the best fit to vector m is achieved by quantizing the value $\ln(2) \cdot S$ into one of six intervals with the following quantization thresholds {0.01, 1.5, 3.6, 11.0, 16.0}. This gives us an index (integer value from 0 to 5, hereinafter referred to as b_{Golomb}) to corresponding value in the vector m = {1, 1, 2, 3, 4, 12}. It automatically becomes the number of probability distribution (one of six). The value b_{Golomb} is a fragment of the context number and is invariant during encoding all bits of next Golomb codeword that represents $|e_n|$.

Such mechanism of selecting parameter b_{Golomb} in a highly flexible manner adapts to the local features of probability distribution of currently encoded prediction errors, as well as to the rate of change of the features of this distribution. Selection of the parameter m from only six values is a compromise approach with regard to the constant adaptation of each probability distribution, which are used by binary arithmetic coder that encodes the binary output of Golomb coder.

The Golomb codeword encoding $|e_n|$ value consist of two elements. The first one is the u_G which determines group number:

$$u_G = \left\lfloor \frac{|e_n|}{m} \right\rfloor \tag{21}$$

and is saved in unary form (sequence of u_G zeros terminated with one). The second part is v_G, which is known as number of elements in group (remainer of division by m):

$$v_G = |e_n| - u_G \cdot m \tag{22}$$

where: v_G is coded by phased-in binary code (variant of Huffman code for sources with m equally probable symbols [49]).

Specifying the parameter k as $[\log_2 m]$, means that, in each group, the first $l = 2^k - m$ elements of v_G are encoded using $k - 1$ bits, and remaining $m - l$ are encoded using k bits as a number $v_G + l$ [8]. An example set of Golomb codewords for $|e_n| < 32$ with the parameters $m = 3$ and $m = 12$ is presented in Table 1.

Rules for determining context number

Binary streams representing u_G and v_G are encoded by separate arithmetic coders, and hence, also by using a separate method for determining the context number. The context number ctx_u used for encoding series of u_G zeros terminated with one is calculated for each consecutively encoded bit as follows:

$$ctx_u = 6 \cdot (2^4 \cdot b_{Golomb} + b_{main\ ctx}) + b_{unary} \tag{23}$$

The b_{unary} value is an integer number from 0 to 5, indicating current bit number of currently encoded unary representation of u_G. When the bit number is greater than 5, then $b_{\text{unary}} = 5$.

Table 1. Golomb codewords for parameters $m = 3$ and $m = 12$, where u_G i v_G are separated with colon

$m = 3$				$m = 12$			
$\|e_n\|$	Codeword	$\|e_n\|$	Codeword	$\|e_n\|$	Codeword	$\|e_n\|$	Codeword
0	1:0	16	000001:10	0	1:000	16	01:1000
1	1:10	17	000001:11	1	1:001	17	01:1001
2	1:11	18	0000001:0	2	1:010	18	01:1010
3	01:0	19	0000001:10	3	1:011	19	01:1011
4	01:10	20	0000001:11	4	1:1000	20	01:1100
5	01:11	21	00000001:0	5	1:1001	21	01:1101
6	001:0	22	00000001:10	6	1:1010	22	01:1110
7	001:10	23	00000001:11	7	1:1011	23	01:1111
8	001:11	24	000000001:0	8	1:1100	24	001:000
9	0001:0	25	000000001:10	9	1:1101	25	001:001
10	0001:10	26	000000001:11	10	1:1110	26	001:010
11	0001:11	27	0000000001:0	11	1:1111	27	001:011
12	00001:0	28	0000000001:10	12	01:000	28	001:1000
13	00001:10	29	0000000001:11	13	01:001	29	001:1001
14	00001:11	30	00000000001:0	14	01:010	30	001:1010
15	000001:0	31	00000000001:10	15	01:011	31	001:1011

$b_{\text{main ctx}}$ is a 4-bit main context number determined according to the description presented in the Rules of context number calculation – proposal 1 section. Therefore, $2^4 \cdot 6 \cdot 6 = 576$ contexts ctx_u are obtained.

In the case of the second coder used to encode the v_G number, there are slightly fewer, since $2^5 \cdot 6 = 192$ contexts. The ctx_v number is calculated as:

$$ctx_v = 2^4 \cdot \left(2 \cdot b_{Golomb} + b_{phased\text{-}in}\right) + \\ + 2^3 \cdot b_\omega + 2^2 \cdot b_{binary} + b_{unary2} \quad (24)$$

Value $b_{\text{unary2}} = \min\{b_{\text{unary}}, 3\}$. b_{binary} is the MSB of v_G (the first encoded bit). b_ω is a 1-bit value obtained after quantizing ω with 49 threshold (see Equation 17), and $b_{\text{phased-in}}$ is a value indicating current bit number (counting from most to least significant) of currently encoded v_G. When bit number $b_{\text{phased-in}}$ is greater than 0, then $b_{\text{phased-in}} = 1$. In must be stressed, that when $b_{\text{phased-in}} = 0$, then b_{binary} is being set to 0 by default.

Long-term adaptation of probability distribution

In the case of context-adaptive binary arithmetic coder, every context number is associated with individual probability distribution. This distribution consists of only two values: $p(0)$ and $p(1)$ – corresponding to frequency of appearance of zeros and ones, respectively. In practice, occurrence counters denoted as $\mathbf{N}_e(0)$ and $\mathbf{N}_e(1)$ are used, then $p(0) = \mathbf{N}_e(0)/(\mathbf{N}_e(0) + \mathbf{N}_e(1))$, and $p(1) = 1 - p(0)$. The ongoing adaptation of distribution consists of incrementing (in a given context) occurrence counter of currently encoded bit by 1 – which is why even with multi-context approach, there is no need to provide the decoder with information about hundreds of probability distributions.

In the proposed solution, a form of long-term adaptation is also used. When the total amount of occurrences of zeros and ones in a given context exceeds a predefined $N_{\max}$, then both counters are divided by 2 – which is an equivalent to the long-term forgetting mechanism (see Equation 11). In all contexts of u_G coder $N_{\max} = 2^{10}$ was set, and the counters $\mathbf{N}_e(0)$ and $\mathbf{N}_e(1)$ were initialized to the value 1. In the case of v_G coder $N_{\max} = 2^{11}$ and the counters were initialized to: $\mathbf{N}_e(0) = \mathbf{N}_e(1) = 16$.

Sign bit encoding

Due to symmetry of probability distribution of prediction errors, it is more convenient to encode their absolute values $|e_n|$. This enables a faster distribution adaptation in the individual contexts of arithmetic coder. The bit sign is encoded separately, whereby information about it must be preserved only for non-zero values of e_n, which is particular feature of the coder presented in this paper, and it requires the sign bit to be encoded after encoding of $|e_n|$ (regardless of whether the

prediction error is encoded with multi-value or binary arithmetic code).

Unlike the solution proposed here, in most known methods (especially in lossless audio compression) negative numbers are cast by calculating their absolute value and subtracting 1 from it. This causes two, not entirely symmetric, probability distributions to be combined (the highest disproportion can be observed, when number –1 is casted to 0 with information about negative bit sign). Some algorithms do not introduce a special method of bit sign encoding and save unchanged value of this bit to the output file. Work [51] is one of the few exceptions in which contextual division has been connected with binary arithmetic code.

The proposed method is also based on an adaptive arithmetic coder with division into 128 contexts. The distribution of this two-symbol source is initially uniform in each context (initial values of occurrence counters for bit 0 and 1 are set to $\mathbf{N}_e(0) = \mathbf{N}_e(1) = 1$).

The context number consists of 7 bits: $\kappa_6\kappa_5\kappa_4\kappa_3\kappa_2\kappa_1\kappa_0$. The first two bits $\kappa_1\kappa_0$ are determined as a integer number from 0 to 3, based on a four-level (2-bit) quantizer of the value $|e_n|$ with thresholds {3, 6, 16}. The next two bits depend on the predicted value. If $\hat{x}_n > 0.5 \cdot x_{\max}$, then $\kappa_2 = 1$, otherwise $\kappa_2 = 0$, where $x_{\max}$ represents the maximum shade value of encoded image. When shade reduction is applied, for some images, $x_{\max}$ can be significantly less than 255. The second condition is based on predicted value which is compared with expression $x_{causal} = 0.1 \cdot (2 \cdot (P(1) + P(2)) + P(3) + P(4) + P(5) + P(10) + P(18) + P(28))$. If $\hat{x}_n > x_{causal}$, then $\kappa_3 = 1$, otherwise $\kappa_3 = 0$.

The experiments showed that the fractional part of the predicted value (including the value of the C_{mix} constant component – see Equation 3), which occurs at the calculation stage in most of more complex prediction methods, also has a significant impact on the efficiency of encoding of prediction error sign bit. Therefore, a condition can be introduced to check the sign of $(\hat{x}_n - [\hat{x}_n])$, where $[\hat{x}_n]$ means rounding to the nearest integer. When $\hat{x}_n > [\hat{x}_n]$, then $\kappa_4 = 1$, otherwise $\kappa_4 = 0$.

The two most significant bits $\kappa_6\kappa_5$ of context number are the sign of prediction error of left and top neighbor of currently encoded $|e_n|$: sgn(e(1)) and sgn(e(2)).

Prediction coefficients encoding

The usage of variability of features in different regions of image is not limited to the arithmetic coding stage, where the main context number is calculated. Local features can also be used by dividing the image into small blocks (e.g. 8 × 8 pixel) and assigning to them an individual predictive model (from a limited set). The introduction of such solution dictates a need to save additional header information containing a set of e.g. 16 linear predictors (including information about assignment of predictor number to block). Such method is implemented in MRP 0.5 coder [40]. Instead of dividing into blocks, a condition for dividing into many contexts depending on closest neighborhood of encoded pixel (backward adaptation) can be introduced. Then, for each pixel, a context number is determined individually, which has been assigned with a separate predictor.

Minimizing the bit average is not a trivial problem because of the large number of factors affecting the final length of the encoded file. Along with increasing prediction order and with increasing number of contexts, it is expected to decrease average absolute error. On the other hand, the requirement to save coefficients of many predictors negatively influences the header section size. For this reason, in the developed 7-ctx MMAE codec, a compromise number of 7 predictors with maximal order $r = 80$ was set. It can be assumed that every prediction coefficient is saved using (N + 2) bits, where 1 bit is dedicated for sign, 1 bit for integer part and N bits for fractional part. Despite the relatively small number of only 7 contexts for 512 × 512 images the cost of the header when N = 8, r = 80 equals 0.0211 b/pixel.

Even though the defined acceptable range for coefficients is quite wide (from –1.999 to 1.999), for the majority of cases they are small values, and their arithmetic mean (after analysis of probability distribution of selected coefficient) oscillates around zero. In general, the higher the index j of coefficient b_j, the smaller its absolute value. This fact can be used to compress the prediction coefficients with a context binary arithmetic coder.

Unlike the previously discussed binary coders, in this case the distribution is static, therefore coder is not adaptive. With training images dataset one can build approximate probability distributions for each of r coefficients. These

distributions are then transformed into probability distributions of individual bits of a given coefficient (saved using $N + 2$ bits). The bit sign of coefficient is encoded in the first place, followed by the bit of integer part (0 or 1), and then the N bits of fractional part in order from MSB to LSB.

At the same time, a contextual division for the bits of the fractional part of the coefficient can be introduced, which takes into account several distributions for a given bit owing to prior information about previously encoded bits (used to build the context number). For this purpose, the information contained in two bits which were encoded in the first place is used. The first one is a sign bit s_1 of prediction coefficient. The second bit of a context number is calculated on an ongoing basis based on information whether at least one bit of value 1 occurred among previously encoded most significant bits of given coefficient, if yes, then $s_2 = 1$, otherwise $s_2 = 0$. In this way, 2 – bit value $s_2 s_1$ with values bounded between 0 and 3 is obtained. It is added to the bit number of the fractional part in the range from 1 to N multiplied by 4, producing the current context number. This number refers to the individual binary distribution corresponding to the currently encoded bit (in practice, it is sufficient to store only one value – probability of occurrence of bit 0). The coder and decoder have an embedded database with a set of $4 \cdot N$ probabilities for each of $(r - 1)$ prediction coefficients (first coefficient is not encoded, since on decoder side it can be reconstructed, with an assumption that sum of all coefficients from vector **B** is equal to 1).

Two observations were made. Firstly, the less significant the bit of precision coefficient, the more random it is, i.e. less susceptible to compression. Secondly, the higher the prediction order, the lower the average cost per encoded coefficient. The average cost of saving single prediction coefficient in relation to precision is presented in Table 2 for two cases $r = 42$ and $r = 80$ (results obtained for test images dataset). For example, with $r = 80$, $N = 8$ it was possible to achieve a reduction of header size by 46.64% on average.

Experiments

The solutions proposed in this chapter were used in the Multi-ctx 2 codec. Table 3 shows bit average of encoding a sequence of sign bits and sequences of Golomb codeword elements u_G and v_G. This table contains the total bit average, which, in addition to the sum of the values from the last three columns, takes into account the size of file header. The results are the average obtained by encoding 45 images from the test image dataset [52]. Similarly to the solution proposed in Long-term adaptation of probability distribution section, two-stage codec can be designed for prediction errors produced after prediction phase in case of audio data, which is characterized by higher than 8 number of bits per sample. This solution is presented in [53].

EFFICINECY ANALYSIS OF VARIOUS CODECS

Three proposed codecs

In this work, the experiments were carried out using three proposed codecs of different implementation complexities, and each of them belonging to one of three different classes defined at the beginning of this paper. A representative of the first class, characterized by fast coding and decoding time is Multi-ctx 2, which is an improved variant of codec described in [25]. Both coder and decoder of Multi-ctx 2 have relatively low complexity, close to other fast methods like CALIC. The coding and decoding times for Lennagrey image (512 × 512 pixels) (i5-4670 3,4 GHz processor, Windows 7, 8 GB RAM, SSD and non-optimized code of Multi-ctx 2) equals

Table 3. Bit average of individual streams encoded using the original CABAC method implemented in the Multi-ctx 2 codec obtained for a dataset of 45 test images

Total bit-average	u_G	v_G	Sign bit
4.02799	2.37562	0.84097	0.81140

Table 2. Average bit number required to save prediction coefficient after arithmetic encoding depending on precision of input data

Coefficients precision ($N + 2$) bits	8	9	10	11
Average cost at $r = 42$	3.961	4.906	5.889	6.886
Average cost at $r = 80$	3.470	4.372	5.336	6.325

0.307 and 0.303 s, respectively, and do not depend on the coded image content, but linearly on its number of pixels.

The second proposed codec is 7-ctx MMAE [27], which represents a class with a long encoding and short decoding time. It incorporates an initial training mechanism, in which a set of 7 predictors of maximum order $r = 80$ is optimized. The encoding time is variable (parameter adaptation phase depends on the encoded image content) and for images with a resolution 512 × 512 pixels ranges between 58.1 s (with $r = 10$) and 184.7 s (with $r = 80$). The 7-ctx MMAE decoding time is even shorter than in Multi-ctx 2, ranging between 0.2295 s and 0.261 s for $r = 10$ and $r = 80$, respectively.

The third proposed codec, Blend-28, is an improved version of the method described in [54]. It represents a class with a long encoding and decoding time. For the Lennagrey image, the measured times are 222.1 s and 189.55 s, respectively. At the encoding phase of Blend-28 there is a pre-checking stage of the improvements described in the Basic improvements in codecs section (lossless rotation by a multiple of 90 degrees and the possibility of using a first order predictor) – this makes 4.32% of total encoding time (9.6 s). In addition, in encoding stage of Blend-28 (similarly to 7-ctx MMAE) there is an additional parameters adaptation step. Not only for prediction parameters but for arithmetic encoding as well (selecting values of parameters: h_1, h_2, h_3. See Equation 17), and also checking the cost-efficiency of the alternative way of determining main contexts number (see Rules of context number calculation – proposal 2 section). The adaptations in Blend-28 case take 9.77% of the total encoding time (21.7 s).

The general encoding scheme in all three codecs is quite similar. Basic blocs are presented in Figure 7. The differences are mainly in the complexity of the prediction block, which can be based on linear or nonlinear prediction.

In all of the three codecs a two-stage prediction error compression method was used. The adaptive Golomb coder of prediction errors as a first step, and then, as a second, its binary output is coded with the context-adaptive binary arithmetic coder CABAC as described in the Binary adaptive arithmetic coder section. In the last step, the sign of prediction error is encoded. For this purpose, an adaptive arithmetic coder with 128 contexts is used (see Sign bit encoding section).

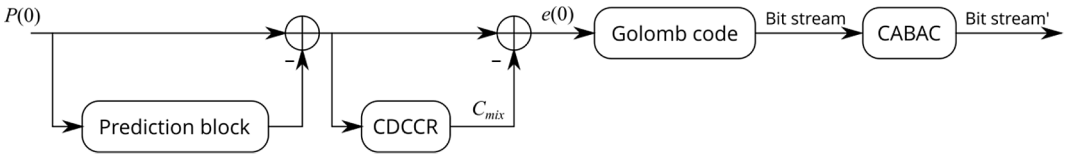

Figure 7. Block diagram of the proposed cascade coding

Table 4. Set of results for the Lennagrey image

Coding method	MSE	MAE	p_0	Entropy	Bit average
Predictor $P(1)$	128.558	6.459	8.516	5.04881	4.35257
Predictor $P(3)$	186.100	7.545	7.580	5.25041	4.45274
Predictor MMSE $r = 10$	28.225	3.548	11.679	4.26207	4.02489
Predictor MMAE $r = 10$	28.843	3.544	11.791	4.25798	4.01717
+ Active colour casting	28.787	3.542	11.795	4.25736	4.01736
+ Active flat region	28.787	3.542	11.794	4.25734	4.01733
+ Active Algorithm 1	28.701	3.540	11.794	4.25682	4.01700
+ Active C_{mix}	27.438	3.460	12.280	4.22477	3.99124
+ Additional adaptation of arithmetic coder	27.438	3.460	12.280	4.22477	3.98729
Multi-ctx 2 method	25.587	3.366	12.553	4.18780	3.96701
7-ctx MMAE method	24.786	3.284	12.846	4.14969	3.94393
Blend-28 method	20.625	3.013	14.110	4.02887	3.83181

Table 5. Set of results for the Frog image

Coding method	MSE	MAE	p_0	Entropy	Bit average
Predictor *P*(1)	483.973	16.231	20.271	5.28230	5.42532
Predictor *P*(3)	538.836	16.063	27.617	4.95369	5.09851
Predictor MMSE *r* = 10	274.644	12.687	3.130	6.06386	5.96582
Predictor MMAE *r* = 10	278.146	12.692	3.158	6.06722	5.96714
+ Active colour casting	77.287	6.691	5.811	5.15122	5.02971
+ Active flat region	77.338	6.683	6.559	5.14895	5.00404
+ Active Algorithm 1	76.199	6.660	6.559	5.14245	4.99978
+ Active C_{mix}	73.539	6.553	6.904	5.11793	4.99136
+ Additional adaptation of arithmetic coder	73.539	6.553	6.904	5.11793	4.95583
Multi-ctx 2 method	74.942	6.630	6.740	5.13740	5.00284
7-ctx MMAE method	68.427	6.290	7.429	5.06304	4.91232
Blend-28 method	64.897	6.023	10.791	4.98775	4.84719
7-ctx MMAE method + special mode	179.064	9.002	28.829	4.00259	3.97763
Blend-28 method + special mode	179.064	9.002	28.829	4.00259	3.98395

Table 6. Set of results for the Noisesquare image

Coding method	MSE	MAE	p_0	Entropy	Bit average
Predictor *P*(1)	224.766	11.031	2.971	5.75220	5.50798
Predictor *P*(3)	279.938	11.409	3.033	5.78073	5.53577
Predictor MMSE *r* = 10	121.949	8.826	3.069	5.40800	5.34126
Predictor MMAE *r* = 10	129.337	8.855	3.127	5.39361	5.33018
+ Active colour casting	99.085	8.406	3.076	5.30218	5.29626
+ Active flat region	99.089	8.406	3.076	5.30223	5.29631
+ Active Algorithm 1	93.534	8.255	3.076	5.21611	5.22089
+ Active C_{mix}	92.826	8.233	3.146	5.21042	5.21753
+ Additional adaptation of arithmetic coder	92.826	8.233	3.146	5.21042	5.16766
Multi-ctx 2 method	98.826	8.456	3.188	5.28050	5.26953
7-ctx MMAE method	87.291	8.036	3.198	5.12086	5.12540
Blend-28 method	87.498	8.058	2.977	5.11741	5.11435

Table 7. Set of results for the Shapes image

Coding method	MSE	MAE	p_0	Entropy	Bit average
Predictor *P*(1)	106.789	2.406	70.786	2.20811	1.23396
Predictor *P*(3)	147.060	3.587	59.671	2.95044	1.49407
Predictor MMSE *r* = 10	32.552	1.347	72.697	1.93634	1.23753
Predictor MMAE *r* = 10	38.808	1.069	79.171	1.46343	0.98922
+ Active colour casting	38.806	1.069	79.172	1.46335	0.98920
+ Active flat region	38.805	1.067	79.358	1.45653	0.98053
+ Active Algorithm 1	31.142	1.000	79.358	1.45208	0.97975
+ Active C_{mix}	30.248	0.935	80.411	1.35503	0.96549
+ Additional adaptation of arithmetic coder	30.248	0.935	80.411	1.35503	0.93919
Multi-ctx 2 method	28.616	0.975	80.941	1.37009	0.97629
7-ctx MMAE method	31.021	0.899	83.359	1.18996	0.86839
Blend-28 method	21.982	0.646	89.255	0.87503	0.65968

Performance analysis of the individual improvements

In Tables 4 to 7 the results obtained for 4 test images: Lennagrey, Frog, Noisesquare and Shapes are shown. The first image represents the most typical category of photos. The other three were selected for their unique features, enabling a broader view of the individual stages of coding in modern codecs, which should be able to adjust to many different categories of encoded images. Among these atypical images there is Noisesquare – an image with high noise level, and Shapes – artificially generated image with soft tonal gradations and low noise level. To draw conclusions, five measures were used in relation to the prediction errors stream. These are the mean square error (MSE), mean absolute error (MAE), percentage probability of prediction errors equal to 0 (marked as p_0), first-order entropy, and bit average – the real average cost of encoding a single pixel expressed in bits. Although there is a large correlation between these parameters, it should be highlighted that the main goal is to minimize the bit average.

It must be stressed that the bit average results refer to the complex two-stage codec based on the Golomb and context-adaptive binary arithmetic code. Owing to its adaptive nature and large number of contexts, the proposed encoder to a large extent takes into account the local variability of image characteristics, hence in most of the cases the bit average is lower than the first-order entropy. The better the prediction, the entropy is usually lower, but at the same time, the difference between entropy and bit average become smaller. In Tables 4 to 7, the first two rows with results refer to simple prediction method, where predicted value is a neighboring pixel $P(1)$ and $P(3)$, respectively (first-order predictor). In the third row a linear predictor of order $r = 10$ (using 10 nearest pixel neighbors) determined based on MMSE is used. In most cases, the improvement was significant for each of the five tested parameters. The Frog image is an exception, where regardless of a significant decrease in MSE and MAE, the result of entropy and bit average turned out to be worse. Therefore, it is worth examining the parameter p_0, which has decreased several times (from 20.27% with predictor $P(1)$ to only 3.13%). This shows, that classical approach to minimizing the mean squared error does not always translate into compression improvement. One of the reasons might be the fact that the coder works on prediction errors rounded to the nearest integer, while mathematical approach of MMSE uses also fractional parts. In addition, the problem of individual treatment of prediction errors equal to 0 is discussed in [30].

In the fourth row MMSE method has been replaced with our novel version of minimizing mean absolute error (MMAE) based on IRLS algorithm. With the same order of $r = 10$, the results improved slightly, but it depends on data. The largest improvement in measurements (excluding MSE, which rose in all cases) was achieved for the Shapes image (with low noise level).

The next five rows refer to the linear prediction method of order $r = 10$ (MMAE), and an additional coder mechanism was activated with each next row. The first mechanism is shade reduction, which gives a noticeable improvement for images with number of shades in use being significantly lower than the maximal 256 (for example, Frog image has only 102 shades in use). The second mechanism is flat region detection, and it was the most beneficial for the Shapes image. The third mechanism in an introduction of Algorithm 1, which was the most efficient for the Noisesquare image (with the highest variance of prediction errors). The fourth mechanism removes context-dependent constant component C_{mix}. This mechanism is beneficial for all cases. The fifth, and the last mechanism, is to adapt the parameters of arithmetic coder (which requires performing a multiple encodings). This mechanism was used in 7-ctx MMAE and Blend-28 codecs. A noticeable improvement was also observed for the Noisesquare image, among others, owing to activation of second proposition of determining main context number (see the Rules of context number calculation – proposal 2 section) and usage of CMTF method. The last three rows present the results of three codecs proposed in this paper. The exceptions are two additional rows of Table 5, where two versions of the 7-ctx MMAE and Blend-28 codecs with and without enabling special mode can be compared. Having in mind that the Frog image is an exception, there is rationale for the implementation of a preliminary analysis of the special mode usage, where instead of complex prediction methods, it may be more beneficial to use one of the 4 to 6 nearest pixels from the neighborhood as the predicted value.

Coding time in competitive solutions

The three codecs proposed by us are characterized by reasonable encoding and decoding time compared to the other solutions (among the same class of codecs). For example, encoding time of the Lennagrey image with MRP 0.5 (second category of codecs with long encoding and short decoding time) is 420 s. Methods such as xMRP [55], or GPR-BP [56] are extensions of MRP 0.5, therefore they have even higher estimated implementation complexity of the encoder (although the proportions of the increase in coding time, and especially in decoding time, compared to the relatively fast MRP 0.5, are not known). In the case of MRP-SSP [57], the authors provided average encoding performance of 600 pixels/s (i7 3.2 GHz processor), which translates into approximately 7 min for the Lennagrey image. In the third category (long both encoding and decoding time), a wide range of encoding time is encountered e.g. for LA-OLS it is only 5.86 s [16]. Other representative Vanilc WLS-D (52.5 s) [58] and Extended Multi-WLS (107.4 s) [16]. The newest high-efficient approach PMO 2019 [59] deserves particular attention. It does not have separate stages for data decomposition and encoding, although its implementation complexity is much higher than of the Extended Multi-WLS and Blend-28. The average time of encoding and decoding 512 × 512 image (Intel Xeon@2.6 GHz, 64 GB RAM) with PMO 2019 is said to be 102 min and 18 min, respectively. Contrary to Blend-28, where encoding/decoding time is almost non-dependent from image structure (only approximately linearly from the number of pixels of the image), in PMO 2019 the discrepancies are significant e.g. encoding of the 512 × 512 images Airplane and Baboon equals 4771 and 6498 s, respectively. Later introduction of neural networks in its next version PMO 2022 [60] caused a drastic reduction (by over 96%) in the coding time. It is characterized by improved efficiency in comparison to PMO 2019 and it is approximately 1/3 faster in encoding than Blend-28. There is no information about LTAP Pixel CNN++ (Locally Trained Adaptive Prediction PixelCNN++ [61]) complexity; however, preliminary analysis of the algorithm indicates that the coding time can be expected to be no less than in the case of PMO 2019.

Experimental results

One of the most popular sets of test images consist of 13 images in 3 different resolutions: (three 256 × 256 images: Camera, Couple, Noisesquare, six 512 × 512 images: Airplane, Baboon, Peppers, Lena_{TMW}, Lennagrey, Shapes, and four 720 × 576 images from ISO/IEC 10918-1 image test set: Balloon, Barb, Barb2, Gold), which was used for the first time in the work [12] (archive available at [52]). Because of the presence of small images, as well as two atypical (artificially generated Shapes with soft tonal gradations and Noisesquare with high noise level) using this set, one can indicate how universal a given codec is.

Unfortunately, in our opinion, the authors of some publications deliberately omitted some of these 13 images from their results (not making the implementation publicly available). Therefore, to provide fairly broad comparison with other papers, it was decided to extract a few comparisons

Table 8. Bit average for various codecs – image set I

Images	JPEG2000 [34]	WebP lossless 1.3 [11]	FLIF 0.3 [34]	SWAP [64]	RALP [65]	Multi-ctx 2	TMW [12]	PMO 2018 [34]	GLICBA WLS [35]	Vanilc WLS-D [58]	7-ctx MMAE
Camera	4.535	4.329	4.285	4.39	4.24	4.042	4.098	3.960	4.208	3.995	4.000
Couple256	3.915	3.727	3.677	3.75	3.63	3.439	3.446	3.415	3.543	3.459	3.400
Airplane	4.013	3.936	3.794	3.58	3.71	3.603	3.601	3.632	3.668	3.575	3.572
Baboon	6.107	5.899	6.078	5.86	5.81	5.762	5.738	5.727	5.666	5.678	5.718
Lennagrey	4.303	4.158	4.252	3.95	3.95	3.966	3.908	3.944	3.901	3.856	3.944
Peppers	4.629	4.512	4.595	4.25	4.27	4.306	4.251	4.267	4.246	4.187	4.191
Balloon	3.031	2.961	2.856	2.49	2.55	2.686	2.649	2.673	2.640	2.626	2.600
Barb	4.600	4.589	4.500	4.12	4.12	4.184	4.084	3.997	3.916	3.815	4.029
Barb2	4.789	4.693	4.656	4.55	4.51	4.383	4.378	4.287	4.318	4.231	4.288
Gold	4.603	4.495	4.518	4.30	4.32	4.292	4.266	4.476	4.276	4.229	4.229
Average	4.453	4.330	4.321	4.124	4.111	4.066	4.042	4.038	4.038	3.965	3.997

Table 9. Bit average for various codecs – image set I

Images	xMRP [55]	MRP 0.5 [14]	LA-OLS [16]	BMF [59]	GPR-BP [56]	MRP-SSP [57]	PMO 2019 [59]	Extended Multi-WLS [16]	Selective PMO 2022 [60]	LTAP Pixel CNN++ [61]	Blend-28
Camera	3.971	3.949	4.001	3.952	3.964	3.901	3.833	3.920	3.804	3.753	3.862
Couple256	3.389	3.388	3.414	3.275	3.339	3.323	3.281	3.345	3.269	3.180	3.297
Airplane	3.590	3.591	3.568	3.535	3.451	3.536	3.546	3.547	3.529	3.474	3.515
Baboon	5.662	5.663	5.643	5.677	5.641	5.635	5.698	5.622	5.611	5.598	5.602
Lennagrey	3.885	3.889	3.881	3.863	3.880	3.877	3.845	3.847	3.825	3.824	3.832
Peppers	4.208	4.199	4.153	4.177	4.170	4.163	4.176	4.101	4.161	4.173	4.087
Balloon	2.613	2.579	2.576	2.649	2.544	2.548	2.584	2.546	2.573	2.563	2.509
Barb	3.817	3.815	3.832	3.804	3.821	3.764	3.733	3.705	3.708	3.805	3.663
Barb2	4.226	4.216	4.214	4.163	4.184	4.175	4.146	4.126	4.122	4.197	4.079
Gold	4.216	4.207	4.198	4.179	4.178	4.173	4.191	4.170	4.171	4.157	4.136
Average	3.958	3.950	3.948	3.927	3.917	3.910	3.903	3.893	3.877	3.872	3.856

Table 10. Bit average for various codecs – image set II

Images	Multi-ctx 2	Vanilc WLS-D [58]	7-ctx MMAE	LA-OLS [16]	xMRP [55]	MRP 0.5 [14]	Pixel CNN++ [62]	PMA CNN [63]	Extended Multi-WLS+ [16]	PMO 2019 [59]	Selective PMO 2022 [60]	Blend-28
Camera	4.042	3.995	4.000	4.001	3.971	3.949	3.749	3.748	3.917	3.833	3.804	3.862
Couple256	3.439	3.459	3.400	3.414	3.389	3.388	3.176	3.176	3.344	3.281	3.269	3.297
Noisesquare	5.283	5.159	5.128	5.194	5.301	5.270	5.387	5.375	5.167	5.296	5.274	5.114
Airplane	3.603	3.575	3.572	3.568	3.590	3.591	3.486	3.481	3.547	3.546	3.529	3.515
Baboon	5.762	5.678	5.718	5.643	5.662	5.663	5.610	5.608	5.622	5.698	5.611	5.602
Peppers	4.306	4.187	4.191	4.153	4.208	4.199	4.194	4.192	4.100	4.176	4.161	4.087
Shapes	1.087	1.302	0.868	1.109	0.769	0.685	0.747	0.720	0.903	0.497	0.490	0.660
Balloon	2.686	2.626	2.600	2.576	2.613	2.579	2.579	2.573	2.547	2.584	2.573	2.509
Barb	4.184	3.815	4.029	3.832	3.817	3.815	3.914	3.905	3.704	3.733	3.708	3.663
Barb2	4.383	4.231	4.288	4.214	4.226	4.216	4.270	4.266	4.121	4.146	4.122	4.079
Gold	4.292	4.229	4.229	4.198	4.216	4.207	4.170	4.166	4.168	4.191	4.171	4.136
Average	3.915	3.841	3.820	3.809	3.797	3.778	3.753	3.746	3.740	3.726	3.701	3.684

Table 11. Bit average for various codecs – image set III [19]

Images	BPG	PNG	LCIC	JPEG 2000	JPLEG-LS	JPEG-XL	FLIF	WebP	L3C	CWPLIC	LCIC duplex [19]	Multi-ctx 2	7-ctx MMAE	Blend-28
Airplane	4.32	4.26	3.99	4.00	3.80	3.71	3.82	3.87	4.56	3.69	3.69	3.579	3.546	3.487
Barbara	5.06	5.22	4.61	4.61	4.70	4.40	4.56	4.55	5.44	4.35	4.36	4.186	4.034	3.670
Coastguard	5.70	5.06	4.82	4.83	4.86	4.73	4.93	4.81	5.82	4.80	4.83	4.579	4.332	4.258
Comic	6.15	5.84	5.63	5.65	5.30	5.07	5.50	5.45	6.60	4.83	4.83	4.816	4.793	4.650
Flowers	5.18	5.08	4.91	4.92	4.62	4.51	4.74	4.76	5.53	4.41	4.35	4.329	4.323	4.209
Goldhill	4.95	4.70	4.58	4.59	4.43	4.37	4.50	4.47	5.27	4.33	4.33	4.208	4.170	4.079
Lennagrey	4.54	4.61	4.31	4.31	4.24	4.16	4.28	4.14	4.95	4.13	4.08	3.966	3.944	3.832
Mandrill	6.61	6.23	6.11	6.11	6.04	5.98	6.14	5.89	6.97	5.95	5.89	5.764	5.721	5.604
Monarch	4.10	4.26	3.82	3.82	3.70	3.54	3.68	3.73	4.37	3.40	3.45	3.394	3.347	3.247
Pepper	4.77	4.90	4.63	4.63	4.51	4.48	4.58	4.50	5.38	4.67	4.38	4.297	4.185	4.080
Ppt3	2.20	2.35	2.41	2.41	2.04	1.84	1.87	2.06	3.71	2.14	2.07	1.761	1.745	1.615
Zebra	5.83	5.19	4.89	4.89	4.81	4.66	4.84	4.86	6.08	4.65	4.68	4.507	4.361	4.243
Average	4.951	4.808	4.559	4.564	4.421	4.288	4.453	4.424	5.390	4.279	4.245	4.116	4.042	3.915

(in presented tables the best result of bit average is underlined). Because of GPR-BP and MRP-SSP codecs, in Tables 8 and 9 the comparison of bit average for 10 test images is presented. Due to the newest codecs: Pixel CNN++ [62] and PMA CNN [63], which are based on neural networks, the results in Table 10 are limited to 11 images. In each comparison, the best average among the entire set was achieved using the proposed Blend-28 method. However, it can be seen that for some images with lower resolutions, competing solutions have proven to be better. The Selective PMO 2022 showed its advantage over all other solutions mainly for the artificially generated Shapes image, while Pixel CNN++ and PMA CNN won for two images (digital photos) with the lowest resolution (although they did not manage to achieve good results for the atypically noisy Noisesquare image with the same resolution of 256 × 256 pixel).

From [19] another set of test images was included (8-bit grayscale – see Table 11), because it is one of the few papers referring to neural networks, in which the bit average for individual encoded images is shown (unfortunately, in the case of many deep learning works, there is usually one average result for the entire image database, which makes it difficult to analyzes in depth the capabilities offered by individual codecs for different types of images).

The obtained results allow stating that, in comparison to the classic modern WebP codec (in lossless version 1.3 offering compression efficiency similar to JPEG-LS), the Blend-28 method achieved the bit average smaller by about 10.95% and 11.51% for test image set I and III, respectively. In addition, this method turned out to be, on average, the most efficient among state-of-the-art solutions known from the literature (including those based on deep learning).

CONCLUSIONS

As a part of this review paper, the mechanism for efficient prediction errors encoding (used in three codecs of varying degrees of implementation complexity: Multi-ctx 2, 7-ctxMMAE and Blend-28) was presented. It is based on a two-stage approach, where in the first step, the values of prediction errors are encoded by adaptive Golomb encoder, and then its output stream is compressed using a CABAC. A separate encoder for the prediction error sign bit and an encoder for predictor coefficients, which sometimes take a lot of space in header data, have also been introduced. The key role in the adaptive coder is a proper way of contextual division, where data with different characteristics of nearest neighborhood are grouped into separate classes to compress data among every group in the most efficient way. An original method Conditional Move To Front for improving the efficiency of prediction, which can be useful in compressing images with high noise level (high variance of input data), was proposed. For images with low variance, it was proven that MMAE has a noticeable advantage over MMSE for determining linear predictive models.

Owing to the proposed solutions, a further increase in compression efficiency has been achieved. Two-stage encoding Golomb-CABAC allows for fast adaptation to the variability of data in certain image regions, which in combination with highly efficient prediction method implemented in our Blend-28 codec, enabled to reduce bit average by 11% on average in comparison to the modern WebP codec.

REFERENCES

1. Scharcanski J. Lossless and Near-Lossless Compression for Mammographic Digital Images. Proceedings of International Conference on Image Processing ICIP'06, Atlanta, GA, USA, 8–11 October 2006, 2253–2256. https://doi.org/10.1109/ICIP.2006.312811
2. Xie X., Li G., Wang Z. A near-lossless image compression algorithm suitable for hardware design in wireless endoscopy system. EURASIP Journal on Advances in Signal Processing, 2007, 1–13. https://doi.org/10.1155/2007/82160
3. Kassim A.A., Yan P., Lee W.S., Sengupta K. Motion compensated lossy-to-lossless compression of 4-D medical images using integer wavelet transforms. IEEE Transactions on Information Technology in Biomedicine, March 2005, 9(1), 132–138. https://doi.org/10.1109/TITB.2004.838376
4. Sanchez V., Nasiopoulos P., Abugharbieh R. Efficient 4D Motion Compensated Lossless Compression of Dynamic Volumetric Medical Image Data. Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2008, Las Vegas, Nevada, USA, 31 March - 4 April 2008, 549–552. https://doi.org/10.1109/ICASSP.2008.4517668

5. Chen X., Canagarajah C.N., Vitulli R., Nunez-Yanez J.L. Lossless Compression for Space Imagery in a Dynamically Reconfigurable Architecture. in: Proceedings of International Workshop on Applied Reconfigurable Computing (ARC2008). March 2008, LNCS 4943, 336–341. https://doi.org/10.1007/978-3-540-78610-8_38
6. Lossless Data Compression. Recommendation for Space Data System Standards, CCSDS 120.1-G-1. Green Book. 1. Washington, D.C.: CCSDS, June 2007.
7. Andriani S., Calvagno G., Erseghe T., Mian G.A., Durigon M., Rinaldo R., Knee M., Walland P., Koppetz M. Comparison of Lossy to Lossless Compression Techniques for Digital Cinema. Proceedings of International Conference on Image Processing ICIP'04, Singapore, 24–27 October 2004, 1, 513–516. https://doi.org/10.1109/ICIP.2004.1418803
8. Sayood K. Introduction to Data Compression. Fifth Edition, Morgan Kaufmann Publ./Elsevier Inc., US, Cambrdge, 2018.
9. Weinberger M.J., Seroussi G. Sapiro G., The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. IEEE Transactions on Image Processing, August 2000, 9(8), 1309–1324. https://doi.org/10.1109/83.855427
10. Wu X., Memon N.D. Context-based, adaptive, lossless image coding. IEEE Trans. on Communications, May 1997, 45(7), 437–444. https://doi.org/10.1109/26.585919
11. Codec WebP 1.3, https://storage.googleapis.com/downloads.webmproject.org/releases/webp/libwebp-1.3.0-windows-x64.zip (Accessed: 08.04.2023).
12. Meyer B., Tischer P. TMW – a new method for lossless image compression. Proceedings of International Picture Coding Symposium (PCS97), Berlin, Germany, September 1997, 533–538.
13. Meyer B., Tischer P. TMW^{Lego} - An Object Oriented Image Modelling Framework. Proceedings of Data Compression Conference 2001, Snowbird, Utah, 504. https://doi.ieeecomputersociety.org/10.1109/DCC.2001.10044
14. Matsuda I., Ozaki N., Umezu Y., Itoh S. Lossless Coding Using Variable Blok-Size Adaptive Prediction Optimized for Each Image. Proceedings of 13th European Signal Processing Conference EUSIPCO 2005, 4–8 September 2005.
15. Ulacha G., Stasiński R. Context Based Lossless Coder Based on RLS Predictor Adaptation Scheme. Proceedings of International Conference on Image Processing ICIP 2009, Egypt, Cairo, 7–11 November 2009, 1917–1920. https://doi.org/10.1109/ICIP.2009.5413680
16. Ulacha G., Stasiński R., Wernik C. Extended Multi WLS Method for Lossless Image Coding. Entropy 2020, 22(9), 919. https://doi.org/10.3390/e22090919
17. Marcellin M., Gormish M., Bilgin A., Boliek M. An Overview of JPEG2000. Proceedings Data Compression Conference, Snowbird, Utah, March 2000, 523–541. https://doi.org/10.1109/DCC.2000.838192
18. Kiely A., Klimesh M. The ICER Progressive Wavelet Image Compressor. The Interplanetary Network Progress Report 42, November 15, 2003, 155.
19. 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. https://doi.org/10.1109/ACCESS.2021.3088936
20. 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 - CS - Computer Vision and Pattern Recognition, September 2022. https://doi.org/10.48550/arXiv.2209.04847
21. Carpentieri B., Weinberger M.J., Seroussi G. Lossless compression of continuous-tone images. Proceedings of the IEEE, November 2000, 88(11), 1797–1809.
22. Deng G. Transform domain LMS-based adaptive prediction for lossless image coding. Signal Processing Image Communication, February 2002, 17(20), 219–229. https://doi.org/10.1016/S0923-5965(01)00019-4
23. Hashidume Y., Morikawa Y. Lossless Image Coding Based on Minimum Mean Absolute Error Predictors. Proceedings of SICE Annual Conference 2007, Kagawa University, Japan, September 2007, 17–20, 2832–2836. https://doi.org/10.1109/SICE.2007.4421471
24. Wang X., Wu X. On Design of Linear Minimum-Entropy Predictor. Proceedings of IEEE 9th Workshop on Multimedia Signal Processing, MMSP 2007, Crete, October 2007, 1–3, 199–202. https://doi.org/10.1109/MMSP.2007.4412852
25. Ulacha G., Stasiński R. A New Fast Multi-Context Method for Lossless Image Coding. Proceedings of the 2018 International Conference on Sensors, Signal and Image, Prague, Czech Republic, 12–14 October 2018, 69–72, https://doi.org/10.1145/3290589.3290600
26. Pinho A.J. On the impact of histogram sparseness on some lossless image compression techniques. Proceedings of International Conference on Image Processing, Thessaloniki, Greece, 2001, 3, 442–445. https://doi.org/10.1109/ICIP.2001.958146
27. Ulacha G., Łazoryszczak M. Lossless image compression using context-dependent linear prediction based on absolute error minimization. Entropy 2024. In review.

28. Wu X., Zhai G., Yang X., Zhang W. Adaptive Sequential Prediction of Multidimensional Signals With Applications to Lossless Image Coding. IEEE Trans. on Image Processing, 2011, 20(1), 36–42. https://doi.org/10.1109/TIP.2010.2061860
29. Liu J., Zhai G., Yang X., Chen L. Lossless Predictive Coding for Images With Bayesian Treatment. IEEE Transactions on Image Processing, December 2014, 23(12), 5519–5530. https://doi.org/10.1109/TIP.2014.2365698
30. Ueno H., Morikawa Y. A New Distribution Modeling for Lossless Image Coding Using MMAE Predictors. Proceedings of the 6th International Conference on Information Technology and Applications, Hanoi, Vietnam, 9–12 November 2009, 249–254.
31. Deng G., Ye H. Maximum likelihood based framework for second-level adaptive prediction, IEE Proceedings - Vision, Image and Signal Processing, June 2003, 150(3), 193–197. https://doi.org/10.1049/ip-vis:20030381
32. Masmoudi A., Puech W., Masmoudi A. An improved lossless image compression based arithmetic coding using mixture of non-parametric distributions. Multimed Tools Appl 2015, 74, 10605–10619. https://doi.org/10.1007/s11042-014-2195-8
33. Kau L.J. Lin Y.P. Least-Squares-Based Switching Structure for Lossless Image Coding. IEEE Transactions on Circuits and Systems I: Regular Papers, July 2007, 54(7), 1529–1541. https://doi.org/10.1109/TCSI.2007.899608
34. Matsuda I., Ishikawa T., Kameda Y., Itoh S. A Lossless Image Coding Method Based on Probability Model Optimization. Proceedings 26th European Signal Processing Conference EUSIPCO 2018, Rome, Italy, 3–7 September 2018, 156–160. https://doi.org/10.23919/EUSIPCO.2018.8553404
35. Meyer B., Tischer P. GLICBAWLS - Grey Level Image Compression by Adaptive Weighted Least Squares. Proceedings of Data Compression Conference 2001, Snowbird, Utah, 503. https://doi.ieeecomputersociety.org/10.1109/DCC.2001.10020
36. Gallager R.G. Variations on a theme by Huffman. IEEE Transactions on Information Theory, November 1978, 24(6), 668–674. https://doi.org/10.1109/TIT.1978.1055959
37. Kau L.J. Lossless image coding using adaptive predictor with automatic context modelling. 10th IEEE International Conference on Electronics, Circuits and Systems, 2003. ICECS 2003. Proceedings of the 2003, Sharjah, United Arab Emirates, 2003, 1, 116–119. https://doi.org/10.1109/ICECS.2003.1301990
38. Matsumura S., Maezawa T., Takago D., Kato K., Takebe T. Least-square-based block adaptive prediction approach for lossless image coding. 2007 18th European Conference on Circuit Theory and Design, Seville, Spain, 27–30 August 2007, 188–191. http://doi.org/10.1109/ECCTD.2007.4529568
39. Deng G. Context-based adaptive arithmetic coding for lossy plus lossless image compression. The IX European Signal Processing Conference, Island of Rhodes, Greece, September 1998, 2317–2320.
40. Matsuda I., Shirai N., Itoh S. Lossless Coding Using Predictors and Arithmetic Code Optimized for Each Image. Lecture Notes in Computer Science, 2003, 2849, 199–207. http://doi.org/10.1007/978-3-540-39798-4_26
41. Deng G., Ye H. Lossless image compression using adaptive predictor combination, symbol mapping and context filtering. Proceedings of IEEE 1999 International Conference on Image Processing, Kobe, Japan, October 1999, 4, 63–67. https://doi.org/10.1109/ICIP.1999.819520
42. Aiazzi B., Alparone L., Baronti S. Context modeling for near-lossless image coding. IEEE Signal Processing Letters, March 2002, 9(3), 77–80. https://doi.org/10.1109/97.995822
43. Motta G., Storer J.A., Carpentieri B. Adaptive linear prediction lossless image coding. Proceedings DCC'99 Data Compression Conference (Cat. No. PR00096), Snowbird, UT, USA, 1999, 491–500. https://doi.org/10.1109/DCC.1999.755699
44. Ye H., Deng G., Devlin J.C. Parametric Probability Models for Lossless Coding of Natural Image. Proceedings of 11th European Signal Processing Conference EUSIPCO-02, Toulouse, France, 3-6 September 2002, 2, 514–517.
45. Ye H., Deng G., Devlin J.C. A weighted least squares method for adaptive prediction in lossless image compression. Proceedings Picture Coding Symposium, Saint-Malo, France, 2003, 489–493.
46. Avcibas I., Memon N., Sankur B., Sayood K. A progressive Lossless/Near-Lossless image compression algorithm. IEEE Signal Processing Letters, 2002, 9(10), 312–314 (extended version). https://doi.org/10.1109/LSP.2002.804129
47. Krivoulets A. A method for progressive near-lossless image compression. Proceedings of International Conference on Image Processing ICIP'03, Barcelona, Catalonia, Spain, 14–18 September 2003, 2, 185–188. https://doi.org/10.1109/ICIP.2003.1246647
48. Bhaskaran V., Konstantinides K. Image and video compression standards – algorithms and architectures. Second Edition, Norwell, MA, USA, Kluwer Academic Publishers, 1997.
49. Salomon D. Data compression. The complete

reference, Fourth Edition, New York, Springer-Verlag, 2006.

50. Sugiura R., Kamamoto Y., Harada N., Moriya T. Optimal Golomb-Rice Code Extension for Lossless Coding of Low-Entropy Exponentially Distributed Sources. IEEE Transactions on Information Theory, April 2018, 64(4), 3153–3161. https://doi.org/10.1109/tit.2018.2799629
51. Xu M., Wu X., Franti P. Context quantization by kernel Fisher discriminant. IEEE Transactions on Image Processing, 15(1), January 2006, 169–177. https://doi.org/10.1109/TIP.2005.860357
52. Dataset of 45 Images. 2022. Available online: https://kakit.zut.edu.pl/fileadmin/Test_Images.zip (Accessed: 03.05.2024).
53. Ulacha G., Wernik C. A High Efficiency Multistage Coder for Lossless Audio Compression using OLS+ and CDCCR Method. Journal: Applied Sciences-Basel 2019, 9(23), 1–22, 5218. https://doi.org/10.3390/app9235218
54. Ulacha G., Stasinski R. High performance predictor blending lossless image coder. 2023 Data Compression Conference (DCC), Snowbird, UT, USA, 21–24 March 2023, 366–366. https://doi.org/10.1109/DCC55655.2023.00066
55. Hsieh F.Y., Wang C.M., Lee C.C., Fan K.C. A Lossless Image Coder Integrating Predictors and Block-Adaptive Prediction. Journal of Information Science and Engineering. September 2008, 24(5), 1579–1591.
56. Dai W., Xiong H. Gaussian Process Regression Based Prediction for Lossless Image Coding. Proceedings of Data Compression Conference, Snowbird, Utah, USA, March 2014, 93–102. https://doi.org/10.1109/DCC.2014.72
57. Dai W., Xiong H., Wang J., Zheng Y.F., Large Discriminative Structured Set Prediction Modeling With Max-Margin Markov Network for Lossless Image Coding. IEEE Transactions on Image Processing. February 2014, 23(2), 541–554. https://doi.org/10.1109/tip.2013.2293429
58. Weinlich A., Amon P., Hutter A., Kaup A. Probability Distribution Estimation for Autoregressive Pixel-Predictive Image Coding. IEEE Transactions on Image Processing, March 2016, 25(3), 1382–1395. https://doi.org/10.1109/TIP.2016.2522339
59. Unno K., Kameda Y., Matsuda I., Itoh S., Naito S. Lossless Image Coding Exploiting Local and Non-local Information via Probability Model Optimization. 27th European Signal Processing Conference (EUSIPCO), A Coruna, Spain, 2–6 September 2019, 1–5. https://doi.org/10.23919/EUSIPCO.2019.8903128
60. 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, Proceedings of SPIE 12177, International Workshop on Advanced Imaging Technology (IWAIT 2022), April 2022. http://doi.org/10.1117/12.2625972
61. Kaji K., Kita Y., Matsuda I., Itoh S. Kameda Y. Enhancement of CNN-based Probability Modeling by Locally Trained Adaptive Prediction for Efficient Lossless Image Coding. 2022 Picture Coding Symposium (PCS), San Jose, CA, USA, 2022, 79–83. https://doi.org/10.1109/PCS56426.2022.10018003
62. Salimans T., Karpathy A., Chen X., Kingma D.P. PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications. 5th International Conference on Learning Representations (ICLR 2017), April 2017. https://doi.org/10.48550/arXiv.1701.05517
63. Kojima H., Kameda Y., Kita Y., Matsuda I., Itoh S. Probability model adjustment for the CNN-based lossless image coding method. Proceedings of SPIE 11766, International Workshop on Advanced Imaging Technology (IWAIT 2021), March 2021, 1176605, 13 https://doi.org/10.1117/12.2590982
64. Kau L.J., Lin Y.P., Lin C.T. Lossless image coding using adaptive, switching algorithm with automatic fuzzy context modelling. IEE Proceeding Vision, Image and Signal Processing, October 2006, 153(5), 684–694. https://doi.org/10.1049/ip-vis:20045256
65. Kau L.J., Lin Y.P. Least squares-adapted edge-look-ahead prediction with run-length encodings for lossless compression of images. Proceedings of International Conference on Acoustics. Speech and Signal Processing ICASSP 2008. Las Vegas. Nevada. U.S.A. 31 March – 4 April 2008, 1185–1188. https://doi.org/10.1109/ICASSP.2008.4517827