Enhanced Binary MQ Arithmetic Coder with Look-Up Table

Hyung-Hwa Ko

Abstract: Binary MQ arithmetic coding is widely used as a basic entropy coder in multimedia coding system. MQ coder esteems high in compression efficiency to be used in JBIG2 and JPEG2000. The importance of arithmetic coding is increasing after it is adopted as a unique entropy coder in HEVC standard. In the binary MQ coder, arithmetic approximation without multiplication is used in the process of recursive subdivision of range interval. Because of the MPS/LPS exchange activity that happens in the MQ coder, the output byte tends to increase. This paper proposes an enhanced binary MQ arithmetic coder to make use of look-up table (LUT) for (A × Q) using quantization skill to improve the coding efficiency. Multi-level quantization using 2-level, 4-level and 8-level look-up tables is proposed in this paper. Experimental results applying to binary documents show about 3% improvement for basic context-free binary arithmetic coding. In the case of JBIG2 bi-level image compression standard, compression efficiency improved about 0.9%. In addition, in the case of lossless JPEG2000 compression, compressed byte decreases 1.5% using 8-level LUT. For the lossy JPEG2000 coding, this figure is a little lower, about 0.3% improvement of PSNR at the same rate.

Keywords: binary MQ arithmetic coder; probability estimation lookup table; JBIG2; JPEG2000

1. Introduction

Since Shannon announced in 1948 that a message can be expressed with the smallest bits based on the probability of occurrence, many studies on entropy coding have been conducted. A typical example is Huffman coding, which is simple and has relatively good performance, so it has been widely used as an entropy coding method in document and image compression [1–3].

In theory, close to optimal compression rate can be obtained using Huffman coding. To do this, we need to enlarge the alphabet, i.e., encode the whole message as a one letter of huge alphabet. However, it cannot be achieved because of the very high complexity of such Huffman code tree building. This was the reason why the arithmetic coding (AC) was developed. AC can build the codeword for a whole message with an acceptable complexity. AC procedure has its roots in the Shannon–Fano code developed by Elias (unpublished), which was analyzed by Jelinek [4]. The procedure for the construction of a prefix-free code is due to Gilbert and Moore [5]. The extension of the Shannon–Fano–Elias code to sequences is based on the enumerative methods in Cover [6] and was described with finite-precision arithmetic by Pasco [7] and Rissanen [1,8].

Arithmetic coding has the disadvantage of increasing precision by continuously dividing the probability interval of [0,1] and expressing it as a decimal value. To circumvent the precision problem posed by the partitioning of the real interval [0,1] in AC, one uses instead the set of integers [0, N]. To handle the precision issue, a renormalization is performed and output bits are used to indicate that normalization has occurred.

The multi-symbol arithmetic coding was implemented in hardware and software by Witten et al. [9] and Langdon [10] in the 1980s. Afterwards, the arithmetic code was applied to binary data to reduce computational complexity. In the 1990s, binary arithmetic coding began to be applied to still-image coding such as JPEG and JBIG.
There have been studies on solution on arithmetic coders and many adaptive methods have been published [10–13]. Many researchers pursued a probability estimation technique which provides a simple yet robust mechanism for adaptive estimation of probabilities during coding process [14–16]. The quite early successful adaptive binary arithmetic coder (BAC) was the Q-coder proposed by IBM [17]. In the BAC, the binary symbols are classified into the Most Probable Symbol (MPS) and the Least Probable Symbol (LPS). These symbols are assigned to 0 and 1. The Q-coder has a mechanism for adaptive encoding based on the probability estimated from the sequence input during the encoding process. The size of the hardware was reduced by limiting the precision of the probability interval to 16-bit length. However, various problems may occur due to the limited register size. Among them, the carry propagation problem in the renormalization process is solved by bit stuffing. That is, when eight consecutive 1s occur, 0s are inserted. Above all, in the process of continuously dividing the probability interval, a renormalization was performed to maintain the probability of occurrence of LPS and the value of the previous interval within [0.75, 1.50) and to avoid multiplication under these conditions. By adopting the 16-bit length precision, the hardware size was reduced by limiting the precision.

BAC has many advantages of high accuracy and good compression performance. However, interval subdivision requires the multiplication operation, which is accompanied by a severe challenge in implementation. Moreover, the probabilities of input symbols are often unknown. The appropriate method of accurately estimating the probabilities is another important issue. In the process of standardizing process of multimedia coding, the MQ-coder [18,19] and M-Coder [20–23] showed the best performance. These coders are table-based coder, in which multiplication is avoided by restricting the interval within a certain range.

As another multiplication-free arithmetic coding approach, Mitchell et al. [24] developed log arithmetic encoder/decoder which has a scheme of substituting multiplication with addition by using a logarithmic domain. However, the scheme fails to address the relationship between the original domain and the logarithmic domain. The domain-switching and probability-estimation processes require large amounts of memory and more computation with minor improvement in coding efficiency [14].

To address the above-mentioned problem of logarithmic encoder, new logarithmic BAC with adaptive probability estimation process was developed to improve coding performance compared with existing MQ-coder and M-coder. However, encoder–decoder complexity is quite increasing with a little bit savings [14,25].

As another adaptive BAC, the adaptive binary range coder (ABRC) [15] uses virtual sliding window (VSW) [26] for probability estimation, which does not require look-up tables. The VSW estimation provides a faster probability adaptation at the initial encoding/decoding stage and especially more precise probability estimation for very low entropy binary sources. However, it needs multiplication operations and more complex encoder—decoder processing.

In the 1990s, standardization of image coding was established. The arithmetic encoder was adopted as entropy encoding along with Huffman encoding in the compression method of still images known as JPEG [27]. Huffman coding was adopted as the baseline profile and arithmetic coding was selected as the extended profile. The JPEG committee chose the QM arithmetic coding method jointly proposed by IBM, Lucent, and Mitsubishi [2]. The QM arithmetic coding method applied MPS/LPS conditional exchange to increase the coding efficiency of Q-coder. The probability estimation was improved to enable fast initial learning. Q-coder is suitable for hardware implementation, while QM coder is suitable for implementation in software. Q-coder solved carry propagation in the decoder; however, the QM coder solved it in the encoder. Probability estimation is performed using the next estimated probability table and when the probability of the MPS is less than 0.5, MPS/LPS exchange is performed.

Subsequently, a compression standard named JBIG was established as a new standard for compressing binary documents [28]. In the JBIG compression standard, unlike
JPEG, arithmetic coding was adopted as the baseline profile. JBIG is based on Q-coder that also uses a relatively minor refinement by Mitsubishi, resulting in what became known as the QM coder. In 2000, the JBIG2 binary coding standard was established and the MQ arithmetic coder instead of QM arithmetic coder was adopted [18]. The MQ coder has changed the carry propagation method. One 0 bit is inserted only when byte 0xFF is output. The bit stuffing of MQ coder is efficient in bandwidth and execution time. In 2004, a new still image coding method, so called JPEG2000, was established as a lossless/lossy image compression standard [29]. In JPEG2000 standard, MQ arithmetic coding was selected as a baseline profile, replacing Huffman coding. H.264/AVC, which is a video compression standard, adopted the so-called context adaptive binary arithmetic coding (CABAC) method as the standard entropy coding. High Efficiency Video Coding (HEVC), also known as H.265 and MPEG-H Part 2, is a video compression standard designed as a successor to the widely used H.264/AVC. In comparison to H.264/AVC, HEVC offers quite better data compression at the same level of video quality. In addition, HEVC chose M-coder as a unique entropy coding [30].

Recently, to increase the coding efficiency for M-coder, a new convolutional neural network (CNN), which was successfully operated in object classification, was applied to compress the syntax elements of the intra-predicted residues in HEVC. Instead of manually designing the binarization process and context models, the probability distributions of the syntax elements by training CNN are estimated directly. The BD-rate reductions are obtained in almost test sequences. However, the encoding and decoding complexity is quite high and AC structure suitable for implementation should be developed [31]. However, in this paper, the probability estimation issue like VSW (Virtual Sliding Window) and CNN-based estimation is not dealt.

In this paper, enhanced MQ-coder is proposed to obtain better compression performance with maintaining multiplication-free property. The existing binary MQ arithmetic coder approximates \((A \times Q_e)\) to \(Q_e\) in order to remove multiplication. When the Q-coder was proposed, however, the computing power was not satisfactory, and this approximation method was reasonable at that time. Because computing power has enormously increased afterward, there are lots of room for improvement of the approximation method. By this motivation, in this paper, using the existent probability estimation table \(Q_e\), multi-level quantization tables of \((A \times Q_e)\) are proposed to be used as look-up table (LUT). The technical details are described in Section 3 and verified the performance by coding experiments of JBIG2 and JPEG2000.

2. Related Works

2.1. Shannon–Fano–Elias Arithmetic Coder

When trying to efficiently compress an input sequence composed of symbols of 0 and 1, the probability of occurrence of each symbol is used. According to Shannon’s entropy theory and lossless coding law, a smaller codeword is assigned to a symbol with a high probability of occurrence and a longer codeword is assigned to a symbol with a low probability of occurrence. When compressing a true document, it can be compressed very efficiently.

Let us say that the current range is represented by the interval \((\text{Lower}, \text{Upper})\). Let the initial range be \((\text{Lower}, \text{Upper}) = [0,1]\). When one bit is input, the current range is divided into two by the following Equations (1) and (2).

\[
\text{Lower} \leftarrow \text{Lower} + (\text{Upper} - \text{Lower}) \times \text{CDF}(n - 1) \tag{1}
\]

\[
\text{Upper} \leftarrow \text{Lower} + (\text{Upper} - \text{Lower}) \times \text{CDF}(n) \tag{2}
\]

Here, CDF(n) is the cumulative distribution function of the \(n\)-th input symbol. Output occurs when all sequence inputs come in. The length of the last range is equal to the product of the probability of occurrence of all symbols. As each input bit enters, the range
continuously decreases and the decimal point representing the accumulated range after the last input bit is expressed in binary. In this case, as the scale of the input sequence increases, it is difficult to increase the accuracy of the range. If the number of input bits is increased, the divided probability interval (range) value continuously decreases, and many bits are rapidly required to express the interval.

To solve the problem of increasing the precision of the arithmetic encoder, if the range value is reduced to less than 1/2, the probability interval is doubled through renormalization or scaling and output bits are generated during the renormalization process.

2.2. Binary MQ Arithmetic Coder [18]

As shown in Figure 1, the binary MQ arithmetic encoder generates binary output (CD) by inputting the continuous bit input (D) and the corresponding context (CX). Bit 0 or 1 is applied as input. The MQ arithmetic coder first determines whether the MPS and the LPS are 0 or 1, respectively, and compares the current input bit with the MPS value to determine whether it is MPS. Encoding is performed using a probabilistic model for each context of the symbol.

![Figure 1. Arithmetic coder inputs and outputs.](image)

The current range value is divided into two ranges by the input bit value. In the process of dividing the range, the range of the MPS is positioned above the LPS. The input result is displayed as base (stored in C register) and interval (stored in A register) value. As a result of the encoding, the output code string is changed to indicate the base (C) value. At the beginning of encoding, the base C is initialized to 0 and the range A is initialized to 0.75 in decimal and 0 × 8000 in hex. Renormalization is performed when necessary to keep the range A within 0.75 ≤ A < 1.5.

That is, if the value of A is less than 0.75, the value of A is doubled and the value of C is also doubled each time the value of A is doubled. This operation is repeated until the A value is greater than 0.75. The A register is composed of 16 bits and the C register is set to have a length of 32 bits.

The lower 16 bits of the C register are shifted to the LSB by 1 byte from the A register and if the count value becomes 0 while performing renormalization, the 1-byte value at the b position is output. It is removed from the C register and placed in the output data column buffer. If the predicted value of the LPS occurrence probability is \( Q_e \), the divided range A is obtained by the following Equations (3) and (4).

\[
\text{Split interval for MPS} = A - (A \times Q_e) \quad (3)
\]

\[
\text{Split interval for LPS} = A \times Q_e \quad (4)
\]

In Equations (3) and (4), one multiplication is required for each input bit, so the calculation is large. Therefore, it is assumed that the value of A is close to 1 and that \((A \times Q_e)\) is approximated by \( Q_e \) [14]. Therefore, after removing the multiplication in Equations (3) and (4), it is approximated as Equations (5) and (6).

\[
\text{Split interval for MPS} = A - Q_e \quad (5)
\]

\[
\text{Split interval for LPS} = Q_e \quad (6)
\]

Whenever the MPS is coded, the value of \( Q_e \) is added to the A register and the interval is reduced to \( A - Q_e \). Whenever the LPS is coded, the A register is left unchanged, and the interval is reduced to \( Q_e \). The precision range required for A is then restored, if necessary,
by renormalization of both A and C. With the process illustrated above, the approximations in the interval subdivision process can sometimes make the LPS sub-interval larger than the MPS sub-interval. If, for example, the value of $Q_e$ is 0.5 and A is at the minimum allowed value of 0.75, the approximate scaling gives 1/3 of the interval to the MPS and 2/3 to the LPS.

To avoid this size inversion, the MPS and LPS intervals are exchanged whenever the LPS interval is larger than the MPS interval. This MPS/LPS conditional exchange can only occur when a renormalization is needed. Whenever renormalization occurs, a probability estimation process is invoked which determines a new probability estimate for the context currently being coded. No explicit symbol counts are needed for the estimation. The relative probabilities of renormalization after coding an LPS and MPS provide an approximate symbol counting mechanism which is used to directly estimate the probabilities.

2.2.1. MPS Encoder in Binary MQ Coder (CODEMPS)

The CODEMPS procedure shown in Figure 2a usually reduces the size of the interval A into $A - Q_e$ to the MPS sub-interval. If A register value which stores the interval size becomes less than 0.75 (=0x8000) adjusts the code register C so that it points to the base of the MPS sub-interval.

If the divided interval A value is less than 0.75, i.e., if $A - Q_e$ is greater than $Q_e$, the base C value increases by $Q_e$, and otherwise, if $A - Q_e$ is less than $Q_e$, the A value is changed to $Q_e$. However, if the interval sizes are inverted, the LPS sub-interval is coded instead. Note that the size inversion cannot occur unless a renormalization (RENORME) is required after the coding of the symbol. The probability estimate update changes the Index according to the next MPS (NMPS) column in Table 1. In addition, renormalization occurs.

2.2.2. LPS Encoder in Binary MQ Coder (CODELPS)

The CODELPS procedure shown in Figure 2b usually consists of a scaling of the interval to $Q_e$, the probability estimates of the LPS determined from the Index. The upper interval is first calculated, so it can be compared to the lower interval to confirm that $Q_e$ has the smaller size. It is always followed by a renormalization (RENORME). If the interval sizes are inverted, however, the conditional MPS/LPS exchange occurs and the upper interval is coded. In either case, the probability estimate is updated. If the SWITCH flag for the Index is set, then the MPS is inverted. A new Index is saved as determined from the next LPS (NLPS) column in Table 1.

Figure 2. (a) CODEMPS procedure, (b) CODELPS procedure.
2.2.3. Renormalization (RENORME) and Byte-out (BYTEOUT) in Binary MQ Coder

The RENORME procedure for the encoder is illustrated in Figure 3a. Both the interval register A and the code register C are shifted, one bit at a time. The number of shifts is counted in the counter CT and when CT is counted down to zero, a byte of compressed data is removed from C by the procedure BYTEOUT shown in Figure 3b. Renormalization continues until A is no longer less than 0x8000.

2.2.4. Adaptive Coding with Probability Estimation Table

The probability estimation state machine consists of several sequences of probability estimates. These sequences are interlinked in a manner which provides probability estimates based on approximate symbol counts derived from the arithmetic coder renormalization [16].

Table 1 shows the Qe value associated with each index. The Qe values are expressed as hexadecimal integers and decimal fractions. To convert the 15-bit integer representation of Qe to the decimal probability, the Qe values are calculated by \((4/3) \times (0x8000)\).

![Figure 3. (a) RENORME procedure, (b) BYTEOUT procedure.](image)

The estimator can be envisioned as a finite-state machine—a table of Qe indexes and associated next states for each type of renormalization (i.e., new table positions). The change in state occurs only when the arithmetic coder interval register is renormalized. This is always done after coding the LPS and whenever the interval register is less than 0x8000 (0.75 in decimal notation) after coding the MPS. After an LPS renormalization, NLPS gives the new index for the LPS probability estimate. After an MPS renormalization, NMPS gives the new index for the LPS probability estimate. If Switch is 1, the MPS symbol sense is reversed [18].

The LPS probability estimation table of the MQ arithmetic coder is different from that of the QM arithmetic coder [16], which is adopted in JPEG with 112 probability estimation values, but MQ coder has much smaller 46 probability estimation values. Features include three probability models that have an initial value (0x5601) in index 0, 6 and 14. When 0 or 1 occurs continuously, it is determined how quickly the probability of LPS occurrence converges to zero.
Table 1. LPS probability estimation table in MQ coder.

| Index | Qe (Hex) | Qe (Decimal) | NMPS | NLPS | Switch |
|-------|----------|--------------|------|------|--------|
| 0     | 0x5601   | 0.503960     | 1    | 1    | 1      |
| 1     | 0x3401   | 0.304729     | 2    | 6    | 0      |
| 2     | 0x1801   | 0.140656     | 3    | 9    | 0      |
| 3     | 0x0ac1   | 0.063015     | 4    | 12   | 0      |
| 4     | 0x0521   | 0.030054     | 5    | 29   | 0      |
| 5     | 0x0221   | 0.012475     | 38   | 33   | 0      |
| 6     | 0x5601   | 0.503960     | 7    | 6    | 1      |
| 7     | 0x5401   | 0.492240     | 8    | 14   | 0      |
| 8     | 0x4801   | 0.421924     | 9    | 14   | 0      |
| 9     | 0x3801   | 0.328168     | 10   | 14   | 0      |
| 10    | 0x3001   | 0.281290     | 11   | 17   | 0      |
| 11    | 0x2401   | 0.210973     | 12   | 18   | 0      |
| 12    | 0x1c01   | 0.164095     | 13   | 20   | 0      |
| 13    | 0x1601   | 0.128937     | 29   | 21   | 0      |
| 14    | 0x5601   | 0.503960     | 15   | 14   | 0      |
| 15    | 0x5401   | 0.492240     | 16   | 14   | 0      |
| 16    | 0x5101   | 0.474661     | 17   | 15   | 0      |
| 17    | 0x4801   | 0.421924     | 18   | 16   | 0      |
| 18    | 0x3801   | 0.328168     | 19   | 17   | 0      |
| 19    | 0x3401   | 0.304729     | 20   | 18   | 0      |
| 20    | 0x3001   | 0.281290     | 21   | 19   | 0      |
| 21    | 0x2801   | 0.234412     | 22   | 19   | 0      |
| 22    | 0x2401   | 0.210973     | 23   | 20   | 0      |
| 23    | 0x2201   | 0.199254     | 24   | 21   | 0      |
| 24    | 0x1c01   | 0.164095     | 25   | 22   | 0      |
| 25    | 0x1801   | 0.140656     | 26   | 23   | 0      |
| 26    | 0x1601   | 0.128937     | 27   | 24   | 0      |
| 27    | 0x1401   | 0.117218     | 28   | 25   | 0      |
| 28    | 0x1201   | 0.105498     | 29   | 26   | 0      |
| 29    | 0x1101   | 0.099638     | 30   | 27   | 0      |
| 30    | 0x0ac1   | 0.063015     | 31   | 28   | 0      |
| 31    | 0x09c1   | 0.057155     | 32   | 29   | 0      |
| 32    | 0x08a1   | 0.050563     | 33   | 30   | 0      |
| 33    | 0x0521   | 0.030054     | 34   | 31   | 0      |
| 34    | 0x0441   | 0.024927     | 35   | 32   | 0      |
| 35    | 0x02a1   | 0.015405     | 36   | 33   | 0      |
| 36    | 0x0221   | 0.012475     | 37   | 34   | 0      |
| 37    | 0x0141   | 0.007348     | 38   | 35   | 0      |
| 38    | 0x0111   | 0.006249     | 39   | 36   | 0      |
| 39    | 0x0085   | 0.003044     | 40   | 37   | 0      |
| 40    | 0x0049   | 0.001671     | 41   | 38   | 0      |
| 41    | 0x0025   | 0.000847     | 42   | 39   | 0      |
| 42    | 0x0015   | 0.000481     | 43   | 40   | 0      |
| 43    | 0x0009   | 0.000206     | 44   | 41   | 0      |
| 44    | 0x0005   | 0.000114     | 45   | 42   | 0      |
| 45    | 0x0001   | 0.000023     | 46   | 43   | 0      |
| 46    | 0x5601   | 0.503960     | 46   | 46   | 0      |
2.3. M-Coder [25]

A new design of a family of multiplication-free binary arithmetic coders has been proposed for H.264/AVC coding [22,23]. Its main innovative features are given by a table-based interval subdivision coupled with probability estimation based on a finite-state machine (FSM) as well as a fast bypass coding mode. This so-called modulo (M) coder family of binary arithmetic coding schemes offers a parameterizable trade-off between coding efficiency and memory requirements for the underlying lookup tables.

The basic idea of the low-complexity M-coder approach of interval subdivision is to quantize the admissible domain \( D = [2^{b-2}, 2^{b-1}) \) for the range register \( R \) induced by renormalization into a small number of \( K \) different cells. To further simplify matters, we assume a uniform quantization of \( D \) to be applied, resulting in a set of representative equispaced range values \( Q = \{Q_0, Q_1, \ldots, Q_{K-1}\} \), where \( K \) is further constrained to be a power of 2, i.e., \( K = 2^k \) for a given integer \( k \geq 0 \). By a suitable discretization of the range of LPS-related probability values \( p_{\text{LPS}} \in (0, 0.5] \), a representative set \( P = \{p_0, p_1, \ldots, p_{M-1}\} \) of probabilities can be constructed together with a set of corresponding transition rules for FSM-based probability estimation.

Both discrete sets \( P \) and \( Q \) together enable an approximation of the multiplication operation \( p_{\text{LPS}} \times R \) for interval subdivision by means of a 2-D table \( RTAB \) that contains all \( M \times K \) pre-calculated product values \( \{p_m \times Q_k\} | 0 \leq m < M; 0 \leq k < K \} \) in a suitably chosen integer precision. The entries of the \( RTAB \) table can be easily addressed by using the probability state index \( m \) and the quantization cell index \( k \) related to the given value of \( R \). The \( RTAB \) example is partially shown in Table 2 when \( M \) equals 64 and \( K \) is 4 with \( b \) being 10.

H.264/AVC is standardized to use M-coder [24] and HEVC adopted it as a default entropy coding to remove Variable length codes (VLC) based on Golomb coding [26].

| \( m \) | \( Q_0 \) | \( Q_1 \) | \( Q_2 \) | \( Q_3 \) |
|---|---|---|---|---|
| 0 | 128 | 176 | 208 | 240 |
| 1 | 128 | 167 | 197 | 227 |
| 2 | 128 | 158 | 187 | 216 |
| 3 | 123 | 150 | 178 | 205 |
| 4 | 116 | 142 | 169 | 195 |
| 5 | 111 | 135 | 160 | 185 |
| 6 | 105 | 128 | 152 | 175 |
| 7 | 100 | 122 | 144 | 166 |
| 8 | 95 | 116 | 137 | 158 |
| 9 | ... | ... | ... | ... |
| 62 | 6 | 7 | 8 | 9 |
| 63 | 2 | 2 | 2 | 2 |

3. Proposal of Enhanced Binary MQ Coder

3.1. Problem of Existing Binary MQ Coder

As explained in related works, binary MQ coding is divided into MPS and LPS and interval segmentation is performed by approximating \( (A \times Q) \) to \( Q \) under the assumption that \( A \) is close to 1. The disadvantage of this method is that if \( Q \) becomes larger than \( A/2 \) after encoding, the MPS and LPS may be inverted. The error decreases as \( A \) gets closer to 1 through the renormalization process. In this paper, focusing on this problem, we intend to devise a method to increase the coding efficiency while maintaining multiplication-free operation by the table lookup method for \( (A \times Q) \) value. In this approach, we can make more accurate estimation of \( (A \times Q) \) instead of just approximation of \( (A \times Q) \) to \( Q \).
will show the improvement through experiment for compression system where MQ coder is applied as a standard entropy coder.

3.2. Proposed LPS Probability Estimation Look up Table Method

We will propose a table lookup method that improves coding efficiency without direct multiplication in current MQ coder. If we want to arithmetically code like Elias coding in Equations (3) and (4), multiplication is required to encode binary symbols. In this paper, we implemented the idea of applying the quantization method to obtain the efficiency of the original Elias method while maintaining the current MQ coder. In the current MQ arithmetic coder, instead of \((A \times Q_e)\) as shown in Equations (3) and (4), it is substituted with \(Q_e\) like Equations (5) and (6) on the assumption that \(A\) is close to 1. However, this is approximation method, there might be loss of coding efficiency. To reduce shortcoming, as shown in Equations (7) and (8), we propose a method to select \((A \times Q_e)\) value from look-up table (LUT) that quantizes \((A \times Q_e)\) into 2-level, 4-level and 8-level according to the range of \(A\) values instead of \(Q_e\).

\[
\text{MPS: } A = A - \text{LUT}(A \times Q_e), C = C + \text{LUT}(A \times Q_e) \tag{7}
\]

\[
\text{LPS: } A = \text{LUT}(A \times Q_e) \tag{8}
\]

To encode new (MPS, LPS) symbol values, after renormalization is performed, the value \(A\) should be located between 0.75 and 1.5. When the index of the MQ arithmetic code is 0, the probability of occurrence \(Q_e\) of LPS is 0.50395 (=0x5601) shown in original probability estimation Table 1, so \((A \times Q_e)\) value is calculated between 0.75 * 0.50395 (=0x4082) and 1.5 * 0.50395 (=0x8104). In this paper, we propose a LUT method in which the quantization level consists of 2, 4 and 8.

3.2.1. 2-Level LUT

As shown in Figure 4, when the quantization levels are two, the entire range of \(A\) is divided into two, but if the value falls from 0x8000 to 0xC000, it is said to be quantized to \(A_1\). On the other hand, when it falls within 0xC000 to 0x10000, it is said to be quantized to \(A_2\). The quantization value is set according to two modes as shown in Table 3. Mode-1 is the mode set the quantization levels to the interval mid value. The quantized value in mode-2 is determined to divide equally entire interval. Parameters \(\alpha\) and \(\beta\) are set to find the optimal quantization value to be determined experimentally according to compression methods such as non-contextual MQ, context-based JBIG2 and JPEG2000. We will seek the best compression in different modes. In addition, if possible, we can select the optimum LPS look-up table in MQ coder. The 2-level lookup tables for \((A \times Q_e)\) are shown in Table 4. Adjusting the \(\alpha\) and \(\beta\) values, the compression ratio is expected to change.

\[
\begin{align*}
\text{A} & \quad \text{A}_1 & \quad \text{A}_2 \\
\text{Mode-1} & \quad \alpha \times 0.75 \times (1 + 1/4) & \quad \beta \times 0.75 \times (1 + 3/4) \\
\text{Mode-2} & \quad \alpha \times 0.75 \times (1 + 1/3) & \quad \beta \times 0.75 \times (1 + 2/3)
\end{align*}
\]

![Figure 4. 2-level quantization method.](image-url)

**Table 3.** Two modes of 2-level quantization of A-register value.
Table 4. Two modes of 2-level lookup tables for $(A_i \times Q_i)$ (47x2, $\alpha = \beta = 1$).

| Index | $Q_i$ | $A_i \times Q_i$ | $A_i \times Q_i$ | $A_i \times Q_i$ | $A_i \times Q_i$ |
|-------|-------|-------------------|-------------------|-------------------|-------------------|
| 0     | 0x5601| 0x50A2            | 0x70E3            | 0x5602            | 0x68B3            |
| 1     | 0x3401| 0x30C2            | 0x4442            | 0x3402            | 0x4102            |
| 2     | 0x1801| 0x1681            | 0x1F82            | 0x1801            | 0x1E02            |
| 3     | 0x0A1C| 0x0A15            | 0x0E1E            | 0x0D71            | 0x0669            |
| 4     | 0x0521| 0x04CF            | 0x06BB            | 0x0521            | 0x0669            |
| 5     | 0x0221| 0x01FF            | 0x02CB            | 0x0221            | 0x02A9            |
| 6     | 0x5601| 0x50A2            | 0x70E3            | 0x5602            | 0x68B3            |
| 7     | 0x5401| 0x4EC2            | 0x6E43            | 0x5402            | 0x6903            |
| 8     | 0x4801| 0x4382            | 0x5683            | 0x4802            | 0x5A03            |
| 9     | 0x3801| 0x3482            | 0x4982            | 0x3802            | 0x4602            |
| 10    | 0x3001| 0x2D02            | 0x3F02            | 0x3002            | 0x3C02            |
| 11    | 0x2401| 0x21C1            | 0x2F42            | 0x2D02            |                  |
| 12    | 0x1C01| 0x1A41            | 0x24C2            | 0x1C01            | 0x2302            |
| 13    | 0x1601| 0x14A1            | 0x1CE2            | 0x1601            | 0x18B2            |
| 14    | 0x5601| 0x50A2            | 0x70E3            | 0x5602            | 0x68B3            |
| 15    | 0x5401| 0x4EC2            | 0x6E43            | 0x5402            | 0x6903            |
| 16    | 0x5101H| 0x4BF2          | 0x6A53            | 0x5101            | 0x6543            |
| 17    | 0x4801| 0x4382            | 0x5683            | 0x4802            | 0x5A03            |
| 18    | 0x3801| 0x3482            | 0x4982            | 0x3802            | 0x4602            |
| 19    | 0x3401| 0x30C2            | 0x4442            | 0x3402            | 0x4102            |
| 20    | 0x3001| 0x2D02            | 0x3F02            | 0x3002            | 0x3C02            |
| 21    | 0x2801| 0x2582            | 0x3482            | 0x2802            | 0x3202            |
| 22    | 0x2401| 0x21C1            | 0x2F42            | 0x2402            | 0x2D02            |
| 23    | 0x2201| 0x1FE1            | 0x2CA2            | 0x2202            | 0x2A82            |
| 24    | 0x1C01| 0x1A41            | 0x24C2            | 0x2302            |                  |
| 25    | 0x1801| 0x1681            | 0x1F82            | 0x1801            | 0x1E02            |
| 26    | 0x1601| 0x14A1            | 0x1CE2            | 0x1601            | 0x18B2            |
| 27    | 0x1401| 0x12C1            | 0x1A42            | 0x1401            | 0x1902            |
| 28    | 0x1201| 0x10E1            | 0x17A2            | 0x1201            | 0x1682            |
| 29    | 0x1301| 0x0FF1            | 0x1552            | 0x1301            | 0x1542            |
| 30    | 0x0A1C| 0x0A15            | 0x0E1E            | 0x0A1C            | 0x0D71            |
| 31    | 0x09C1| 0x0925            | 0x0CCE            | 0x09C1            | 0x0C31            |
| 32    | 0x08C1| 0x0817            | 0x0B53            | 0x08A1            | 0x0AC9            |
| 33    | 0x0521| 0x04CF            | 0x06B8            | 0x0521            | 0x0669            |
| 34    | 0x0441| 0x03FD            | 0x0595            | 0x0441            | 0x0551            |
| 35    | 0x02A1| 0x0277            | 0x0373            | 0x02A1            | 0x0349            |
| 36    | 0x0221| 0x01FF            | 0x02CB            | 0x0221            | 0x02A9            |
| 37    | 0x0141| 0x012D            | 0x01A5            | 0x0141            | 0x0191            |
| 38    | 0x0111| 0x0100            | 0x0166            | 0x0111            | 0x0155            |
| 39    | 0x0085| 0x007D            | 0x00AF            | 0x0085            | 0x00A6            |
| 40    | 0x0049| 0x0044            | 0x0060            | 0x0049            | 0x005B            |
| 41    | 0x0025| 0x0023            | 0x0031            | 0x0025            | 0x002E            |
| 42    | 0x0015| 0x0014            | 0x001A            | 0x0015            | 0x001A            |
| 43    | 0x0009| 0x0008            | 0x000C            | 0x0009            | 0x000B            |
| 44    | 0x0005| 0x0005            | 0x0007            | 0x0005            | 0x0006            |
3.2.2. 4-Level LUT

When the quantization level is set to 4, the value of interval A is divided into 4 sections as shown in Table 5 and Figure 5.

**Table 5.** MPS and LPS interval subdivision quantization level in case of 4-level.

| Condition                                      | Equation          |
|------------------------------------------------|-------------------|
| if (0x8000 ≤ A < 0xA000) then A = A₁ (shown in Table 6) | 
| else if (0xA000 ≤ A < 0xC000) then A = A₂ (shown in Table 6) | 
| else if (0xC000 ≤ A < 0xE000) then A = A₃ (shown in Table 6) | 
| else then A = A₄ (shown in Table 6)                 |

**Figure 5.** 4-level Quantization method.

Because the quantization level is set to 4, the interval A is divided into four, and A values variably are set by 4-modes. The quantization values of A₁–A₄ for each divided interval are determined shown in Table 6. The quantization LUT values are assigned to [A₁ × Qₑ, A₂ × Qₑ, A₃ × Qₑ, A₄ × Qₑ] and the parameters α and β are determined to experimentally find the optimal quantization value. Table 7 shows the LUT examples for 4-level quantized values of A₁ × Qₑ.

**Table 6.** Four modes of 4-level quantization.

| Mode   | A₁ × Qₑ | A₂ × Qₑ | A₃ × Qₑ | A₄ × Qₑ |
|--------|---------|---------|---------|---------|
| Mode-1 | α × 0.75 × (1 + 1/8)× Qₑ | β × 0.75 × (1 + 3/8)× Qₑ | β × 0.75 × (1 + 5/8)× Qₑ | α × 0.75 × (1 + 7/8)× Qₑ |
| Mode-2 | α × 0.75 × (1 + 1/5)× Qₑ | β × 0.75 × (1 + 3/5)× Qₑ | β × 0.75 × (1 + 5/5)× Qₑ | α × 0.75 × (1 + 1/5)× Qₑ |
| Mode-3 | α × 0.75 × (1 + 1/5)× Qₑ | β × 0.75 × (1 + 3/5)× Qₑ | β × 0.75 × (1 + 5/5)× Qₑ | α × 0.75 × (1 + 1/5)× Qₑ |
| Mode-4 | α × 0.75 × (1 + 1/8)× Qₑ | β × 0.75 × (1 + 2/5)× Qₑ | β × 0.75 × (1 + 3/5)× Qₑ | α × 0.75 × (1 + 1/8)× Qₑ |

**Table 7.** (Aᵢ × Qₑ) lookup table (47x4, α = 1, β = 1).

| Mode | Index | A₁ × Qₑ | A₂ × Qₑ | A₃ × Qₑ | A₄ × Qₑ | NMPS | NLPS | Switch |
|------|-------|---------|---------|---------|---------|------|------|--------|
| 1    | 0     | 0x4892  | 0x58B2  | 0x68D3  | 0x78F3  | 1    | 1    | 1      |
|      | 1     | 0x2B2E2 | 0x35A2  | 0x3F62  | 0x4923  | 2    | 6    | 0      |
|      | 2     | 0x1AB1 | 0x18C1  | 0x1D42  | 0x21C2  | 3    | 9    | 0      |
|      | 3     | 0x0913  | 0x0B17  | 0x0D1B  | 0x0F20  | 4    | 12   | 0      |
|      | 4     | 0x0454  | 0x05A4  | 0x0640  | 0x0737  | 5    | 29   | 0      |
|      | 5     | 0x01CC  | 0x0232  | 0x0298  | 0x02FE  | 38   | 33   | 0      |
|      | 6     | 0x0A92  | 0x08B2  | 0x06D3  | 0x78F3  | 7    | 6    | 1      |
|      | ...   | ...     | ...     | ...     | ...     | ...  | ...  | ...    |
| 2    | 0     | 0x4D69  | 0x5AD4  | 0x6736  | 0x741D  | 1    | 1    | 1      |
|      | 1     | 0x2ECE  | 0x369C  | 0x3E69  | 0x4636  | 2    | 9    | 0      |
|      | 2     | 0x159B  | 0x1935  | 0x1CCE  | 0x2068  | 3    | 9    | 0      |
3.2.3. 8-Level LUT

When the quantization level is set to 8, interval A is divided into 8 in Table 8 and Figure 6 and the quantization value of each divided interval is used to obtain a lookup table with the center value of the divided interval as the distinct value (A1~A8). The quantization values of A1~A8 for each divided interval are determined shown in Table 9.

Table 8. MPS and LPS interval subdivision quantization level in case of 8-level.

| Level | Quantization Values |
|-------|---------------------|
| 3     | 0x09AE 0xB4B 0xCE8 0xE85 |
| 4     | 0x049E 0x0563 0x0628 0x06ED | 5 |
| 5     | 0x01EB 0x023C 0x028E 0x02E0 | 6 |
| 6     | 0x4D69 0x5A4F 0x6736 0x741D | ... |

| Level | Quantization Values |
|-------|---------------------|
| 3     | 0x4D69 0x58B2 0x68D3 0x741D |
| 4     | 0x4892 0x5A4F 0x6736 0x78F3 | 5 |
| 5     | 0x4892 0x5A4F 0x6736 0x78F3 | 6 |

Figure 6. 8-level quantization method.
Table 9. Two-modes of 8-level quantization.

| Mode | \( A_1 \) | \( A_2 \) | \( A_3 \) | \( A_4 \) | \( A_5 \) | \( A_6 \) | \( A_7 \) | \( A_8 \) |
|------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 1    | \( \alpha \times 0.75 \times \) | \( \alpha \times 0.75 \times \) | \( \alpha \times 0.75 \times \) | \( \alpha \times 0.75 \times \) | \( \beta \times 0.75 \times \) | \( \beta \times 0.75 \times \) | \( \beta \times 0.75 \times \) | \( \beta \times 0.75 \times \) |
|      | \( (1 + 1/9) \) | \( (1 + 1/9) \) | \( (1 + 1/9) \) | \( (1 + 1/9) \) | \( (1 + 4/9) \) | \( (1 + 4/9) \) | \( (1 + 4/9) \) | \( (1 + 4/9) \) |
| 2    | \( \alpha \times 0.75 \times \) | \( \alpha \times 0.75 \times \) | \( \alpha \times 0.75 \times \) | \( \alpha \times 0.75 \times \) | \( \beta \times 0.75 \times \) | \( \beta \times 0.75 \times \) | \( \beta \times 0.75 \times \) | \( \beta \times 0.75 \times \) |
|      | \( (1 + 1/9) \) | \( (1 + 1/9) \) | \( (1 + 1/9) \) | \( (1 + 1/9) \) | \( (1 + 5/16) \) | \( (1 + 5/16) \) | \( (1 + 5/16) \) | \( (1 + 5/16) \) |

The quantization values are assigned to \( [A_1 \times Q_e, A_2 \times Q_e, A_3 \times Q_e, A_4 \times Q_e, A_5 \times Q_e, A_6 \times Q_e, A_7 \times Q_e, A_8 \times Q_e] \) shown in Table 10 and the parameter \( \alpha \) and \( \beta \) should be set to find the optimal quantization value through experiment. For example, in Mode-1, assuming that the \( Q_e \) value is 0x5601 and \( \alpha \) and \( \beta \) equal to 1, when the index is 0, \( A_1 \times Q_e \) is 0x47AC, \( A_2 \times Q_e \) is 0x4ED7, \( A_3 \times Q_e \) is 0x5602, \( A_4 \times Q_e \) is 0x5D2D, \( A_5 \times Q_e \) is 0x6457, \( A_6 \times Q_e \) is 0x6B82, \( A_7 \times Q_e \) is 0x72AD and \( A_8 \times Q_e \) is 0x79D8. Table 10 shows part of the \( (A_1 \times Q_e) \) lookup tables of 47x8 sizes when \( \alpha \) and \( \beta \) is 1.

Table 10. AQc lookup table (Mode-1, 2, 47x8, \( \alpha = \beta = 1 \)).

| Mode | Idx | \( A_1 \times Q_e \) | \( A_2 \times Q_e \) | \( A_3 \times Q_e \) | \( A_4 \times Q_e \) | \( A_5 \times Q_e \) | \( A_6 \times Q_e \) | \( A_7 \times Q_e \) | \( A_8 \times Q_e \) |
|------|-----|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 1    | 0   | 0x47AC          | 0x4ED7          | 0x5602          | 0x5D2D          | 0x6457          | 0x6B82          | 0x72AD          | 0x79D8          |
|      | 1   | 0x2B56          | 0x2FAC          | 0x3401          | 0x3857          | 0x3CAC          | 0x4102          | 0x4557          | 0x49AD          |
|      | 2   | 0x1401          | 0x1601          | 0x1A01          | 0x1C01          | 0x1E01          | 0x2001          | 0x2201          |                 |
|      | 3   | 0x08F6          | 0x9DB           | 0xAC1           | 0xBA6           | 0xC8C           | 0xD71           | 0xE56           | 0xF93           |
|      | 4   | 0x0446          | 0x0B3           | 0x0521          | 0x058E          | 0x05FB          | 0x0669          | 0x06D6          | 0x0744          |
|      | 5   | 0x1C6           | 0x01F3          | 0x0221          | 0x024E          | 0x027B          | 0x02A9          | 0x02D6          | 0x0304          |
|      | 6   | 0x47AC          | 0x4ED7          | 0x5602          | 0x5D2D          | 0x6457          | 0x6B82          | 0x72AD          | 0x79D8          |
|      |     |                 |                 |                 |                 |                 |                 |                 |                 |
|      | 2   | 0   | 0x47AC          | 0x4ED7          | 0x54AA          | 0x5CBA          | 0x64CA          | 0x6CDA          | 0x72AD          | 0x79D8          |
|      |      | 1   | 0x2B56          | 0x2FAC          | 0x3331          | 0x3811          | 0x3CF2          | 0x41D2          | 0x4557          | 0x49AD          |
|      |      | 2   | 0x1401          | 0x1601          | 0x17A1          | 0x19E1          | 0x1CE1          | 0x1E61          | 0x2001          | 0x2201          |
|      |      | 3   | 0x08F6          | 0x09DB          | 0x0A96          | 0x0B98          | 0x0C9A          | 0x0D9C          | 0x0E56          | 0x0F93          |
|      |      | 4   | 0x0446          | 0x0B3           | 0x050C          | 0x0587          | 0x0602          | 0x067D          | 0x06D6          | 0x0744          |
|      |      | 5   | 0x1C6           | 0x01F3          | 0x0218          | 0x024B          | 0x027E          | 0x02B1          | 0x02D6          | 0x0304          |
|      |      | 6   | 0x47AC          | 0x4ED7          | 0x54AA          | 0x5CBA          | 0x64CA          | 0x6CDA          | 0x72AD          | 0x79D8          |

4. Experimental Results

4.1. Experimental Environment

To verify the proposed algorithm, experiment was done by Visual Studio 2019 with binary document image, gray and color images in various resolutions. For binary document compression, a total of 12 documents were used. Eight 200(dpi) and 300(dpi) standard binary documents presented by the JBIG committee are used in Figure 7. Two Korean documents created by scanning at 200(dpi) and 300(dpi) were used in the experiment as shown in Figure 8. The color and gray images used for JPEG2000 experiment are shown in Figure 9. The images in Figure 9a–f are widely used in still image compression at SD resolution. An additional two HD images by capturing test video for H.264 are shown in Figure 9g–h.
Figure 7. Standard binary images used for simulation. (a) CCITT1 document, (b) CCITT4 document, (c) CCITT5 document, (d) CCITT7 document.
Three experiments were executed to evaluate the compression performance of the proposed method. The first experiment is to compress standard binary documents with context free MQ arithmetic coding. The second experiment is JBIG2 lossless compression for binary document with two-dimensional context. The third experiment is executed for JPEG2000 lossless and lossy compression with proposed method by Jasper program [32].
4.2. Basic Context-Free MQ Coding Experimental Results

Using binary document images shown in Figures 7 and 8, we executed binary MQ encoding without context. Compared to the standard MQ coding, the degree of improvement for the proposed method of 2, 4 and 8 level LUTs were measured. Parameter $\alpha$ and $\beta$ values were changed by values such as 0.9, 0.95, 0.97, 1.0, 1.02, 1.03, 1.05, 1.1, etc., and the optimal parameter values were attempted to obtain experimentally.

The experimental results of the 2-level MQ encoder are shown in Table 11. For 12 documents, after varying the parameters $\alpha$ and $\beta$, the best result in mode-1 was obtained when $\alpha$ equals 1.02 and $\beta$ equals 1.02. In this case, the average degree of improvement was 2.99%. The best overall improvement in mode-2 was 2.36%. Mode-1 shows the better results than mode-2. In terms of difference of resolution at 200 dpi and 300 dpi, the lower the resolution, the better the improvement.

Table 11. Experimental results for each mode using 2-level LUT for binary images.

| Document | Resolution (dpi) | Original MQ Byte | Proposed MQ Byte | Improvement(%) |
|----------|-----------------|-----------------|-----------------|----------------|
|          | (\(\alpha, \beta\)) | mode-1 | mode-2 | mode-1 | mode-2 |
| CCITT1   | 200             | 50,784        | 48,814         | 49,084         | 3.88%  | 3.35% |
| CCITT4   | 200             | 156,129       | 149,394        | 150,249        | 4.31%  | 3.77% |
| CCITT5   | 200             | 99,719        | 97,802         | 98,187         | 1.92%  | 1.54% |
| CCITT7   | 200             | 169,154       | 167,048        | 168,072        | 1.25%  | 0.64% |
| Hangueul-1 | 200         | 72,416       | 70,225         | 70,698         | 3.03%  | 2.37% |
| Hangueul-2 | 200         | 87,902       | 84,763         | 85,482         | 3.57%  | 2.73% |
| Avr. Improvement in 200 |               |                |                |                | 2.29%  | 2.40% |
| CCITT1   | 300             | 92,705        | 89,516         | 90,052         | 3.44%  | 2.86% |
| CCITT4   | 300             | 332,320       | 316,943        | 319,075        | 4.69%  | 3.99% |
| CCITT5   | 300             | 180,281       | 175,457        | 176,546        | 2.68%  | 2.07% |
| CCITT7   | 300             | 334,128       | 331,311        | 332,923        | 0.84%  | 0.36% |
| Hangueul-1 | 300         | 148,594       | 144,376        | 145,673        | 2.84%  | 1.97% |
| Hangueul-2 | 300         | 183,611       | 177,237        | 178,824        | 3.47%  | 2.61% |
| Avr. Improvement in 300 |               |                |                |                | 2.98%  | 2.31% |
| Overall Improvement |             |                |                |                | 2.99%  | 2.36% |

In the 4-level MQ coding experiment, four modes were simulated and results are shown in Table 12. Though experimental data are not included when $\alpha$ and $\beta$ are 1, there are improvements of 1.6–1.8% compared with original MQ coder. The mode-4 showed the best improvement of 2.16% when alpha and beta is 1.03 after various attempts for parameters $\alpha$ and $\beta$. In terms of the resolution, the case with 200 dpi documents showed better performance than that with 300 dpi documents. It was found that the performance was lower than that of 2-level MQ encoding, which showed that 2-level LUT MQ may be sufficient if context information is not considered.

Table 12. Experimental Results for each mode using 4-level LUT for binary images.

| Document | Resol. (dpi) | Original MQ Byte | Proposed MQ Improvement(%) |
|----------|--------------|-----------------|---------------------------|
|          | \(\alpha, \beta\) | mode-1 | mode-2 | mode-3 | mode-4 |
| CCITT1   | 200          | 50,784        | 2.96%         | 3.01%  | 3.01%  | 3.03%  |
| CCITT4   | 200          | 156,129       | 3.59%         | 3.44%  | 3.46%  | 3.61%  |
| CCITT5   | 200          | 99,719        | 1.23%         | 1.21%  | 1.19%  | 1.16%  |
In the 8-level MQ coding experiment, in mode-1, it showed the best performance when \( \alpha \) and \( \beta \) are 1.0 and the improvement was averaged 2.13% for all documents shown in Table 13. In the case of mode-2, the best improvement was 2.08% when \( \alpha \) and \( \beta \) are 1.03. In addition, the lower the resolution, the better the improvement.

### Table 13. Experimental results for each mode using 8-level LUT for binary images.

| Documents | Resol. (dpi) | Original MQ Byte | Proposed MQ Byte | Improvement (%) |
|-----------|--------------|------------------|------------------|-----------------|
|           | Mode (\( \alpha, \beta \)) | (1.0, 1.0) | (1.03, 1.03) | (1.0, 1.0) | (1.03, 1.03) |
| CCITT1    | 200          | 50,784           | 49,342           | 49,298          | 2.84% | 2.93% |
| CCITT4    | 200          | 156,129          | 150,970          | 150,860         | 3.30% | 3.37% |
| CCITT5    | 200          | 99,719           | 98,598           | 98,565          | 1.12% | 1.16% |
| Hangeul-1 | 200          | 148,594          | 145,605          | 145,974         | 2.35% | 2.15% |
| Hangeul-2 | 200          | 183,611          | 178,919          | 179,237         | 2.01% | 2.38% |
| Avg Improvement in 200 |  | 2.18% | 2.14% |

Comparing the 2-level, 4-level and 8-level LUT MQ coder, 2-level MQ coder showed the best results. However, in the case of the JBIG2 and JPEG2000 experiments to be described below, different results can be obtained. It will be comprehensively examined later.

### 4.3 JBIG2 Experimental Results

For the experiment applying the proposed method to the JBIG2 standard compression at binary documents, a lossless refinement method was executed. First, in the case of the two-level LUT MQ method, as we saw in Table 14, there was no significant difference in performance between 200 dpi and 300 dpi documents. In the case of mode-1 when \( \alpha \) and \( \beta \) equal 1, the average improvement was 0.58%. In addition, for 300 dpi document the
best performance happens when $\alpha$ and $\beta$ equal 1.03. In the case of mode-2, the best improvement of 0.52% is obtained when $\alpha$ and $\beta$ equal 1.

Table 14. 2-level LUT experimental results for JBIG2. ($\alpha = \beta = 1.0$).

| Documents | Resol. (dpi) | Original JB2 Byte | Proposed JB2 Byte | Improvement (%) |
|-----------|-------------|-------------------|-------------------|----------------|
|           | Mode        |                   | mode-1 | mode-2 | mode-1 | mode-2 |
| CCITT1    | 200         | 16,116            | 16,032 | 16,026 | 0.52%  | 0.56%  |
| CCITT4    | 200         | 60,737            | 60,268 | 60,292 | 0.78%  | 0.73%  |
| CCITT5    | 200         | 28,109            | 27,962 | 27,980 | 0.53%  | 0.46%  |
| CCITT7    | 200         | 60,456            | 60,143 | 60,184 | 0.52%  | 0.45%  |
| Hanguel-1 | 200         | 19,681            | 19,591 | 19,602 | 0.46%  | 0.40%  |
| Hanguel-2 | 200         | 25,998            | 25,862 | 25,858 | 0.53%  | 0.54%  |
| Avg improvement in 200 |               |                   | 0.56%  | 0.52%  |
| CCITT1    | 300         | 22,026            | 21,895 | 21,893 | 0.60%  | 0.60%  |
| CCITT4    | 300         | 85,167            | 84,748 | 84,726 | 0.49%  | 0.52%  |
| CCITT5    | 300         | 39,049            | 38,793 | 38,867 | 0.66%  | 0.47%  |
| CCITT7    | 300         | 87,132            | 86,620 | 86,693 | 0.59%  | 0.50%  |
| Hanguel-1 | 300         | 27,938            | 27,782 | 27,788 | 0.56%  | 0.54%  |
| Hanguel-2 | 300         | 37,471            | 37,220 | 37,226 | 0.67%  | 0.65%  |
| Avg improvement in 300 |               |                   | 0.60%  | 0.52%  |
| Overall improvement |               |                   | 0.58%  | 0.52%  |

As shown in Table 15, when applying the 4-level LUT MQ coder to JBIG2, mode-3 provided the best performance compared with other modes. Compared to the original MQ coding, the best improvement percentage was about 0.8%. With varying $\alpha$ and $\beta$, there was no significant improvement. In addition, there was no significant difference in performance with different resolutions. It was found to be 0.22% better than the 2-level LUT MQ coder.

Table 15. 4-level LUT experimental results for JBIG2. ($\alpha = \beta = 1.0$).

| Document | Resol. (dpi) | Original JB2 Byte | Improvement (%) |
|----------|-------------|-------------------|----------------|
|          | Mode        |                   | mode-1 | mode-2 | mode-3 | mode-4 |
| CCITT1   | 200         | 16,116            | 0.81%  | 0.79%  | 0.94%  | 0.77%  |
| CCITT4   | 200         | 60,737            | 0.96%  | 0.98%  | 1.06%  | 0.94%  |
| CCITT5   | 200         | 28,109            | 0.76%  | 0.70%  | 0.73%  | 0.74%  |
| CCITT7   | 200         | 60,456            | 0.69%  | 0.72%  | 0.65%  | 0.70%  |
| Hanguel-1| 200         | 19,681            | 0.68%  | 0.71%  | 0.69%  | 0.68%  |
| Hanguel-2| 200         | 25,998            | 0.68%  | 0.77%  | 0.77%  | 0.76%  |
| Avg improvement in 200 |               |                   | 0.76%  | 0.76%  | 0.76%  |
| CCITT1   | 300         | 22,026            | 0.70%  | 0.74%  | 0.78%  | 0.74%  |
| CCITT4   | 300         | 85,167            | 0.74%  | 0.68%  | 0.74%  | 0.75%  |
| CCITT5   | 300         | 39,049            | 0.78%  | 0.78%  | 0.76%  | 0.87%  |
| CCITT7   | 300         | 87,132            | 0.77%  | 0.82%  | 0.85%  | 0.77%  |
| Hanguel-1| 300         | 27,938            | 0.72%  | 0.72%  | 0.82%  | 0.71%  |
| Hanguel-2| 300         | 37,471            | 0.83%  | 0.81%  | 0.81%  | 0.78%  |
| Avg improvement in 300 |               |                   | 0.76%  | 0.76%  | 0.79%  |
| Overall improvement |               |                   | 0.76%  | 0.77%  | 0.80%  | 0.77%  |
As shown in Table 16, in the case of 8-level LUT MQ coder, slightly better results are obtained in mode-1. Improvement of compression ratio was measured 0.9% compared with original JBIG2 coder. As the quantization level increased, it was found to be less sensitive to parameter $\alpha$ and $\beta$.

Table 16. 8-level LUT experimental results for JBIG2. ($\alpha = \beta = 1.0$).

| Document | Resolution (dpi) | Original JPEG2 Byte | Proposed JPEG2 Byte | Improvement (%) |
|----------|------------------|---------------------|---------------------|-----------------|
|          |                  | mode-1              | mode-2              | mode-1          | mode-2          |
| CCITT1   | 200              | 16,116              | 15,970              | 15,955          | 0.91%           | 1.00%           |
| CCITT4   | 200              | 60,737              | 60,071              | 60,096          | 1.10%           | 1.06%           |
| CCITT5   | 200              | 28,109              | 27,857              | 27,861          | 0.90%           | 0.88%           |
| CCITT7   | 200              | 60,456              | 59,954              | 59,961          | 0.83%           | 0.82%           |
| Hangeul-1| 200              | 19,681              | 19,524              | 19,537          | 0.80%           | 0.73%           |
| Hangeul-2| 200              | 25,998              | 25,764              | 25,779          | 0.90%           | 0.84%           |
| Avg improvement in 300 |                  |                     |                     | 0.90%           | 0.89%           |
| CCITT1   | 300              | 22,026              | 21,823              | 21,839          | 0.92%           | 0.85%           |
| CCITT4   | 300              | 85,167              | 84,442              | 84,428          | 0.85%           | 0.87%           |
| CCITT5   | 300              | 39,049              | 38,689              | 38,687          | 0.92%           | 0.93%           |
| CCITT7   | 300              | 87,132              | 86,359              | 86,377          | 0.89%           | 0.87%           |
| Hangeul-1| 300              | 27,938              | 27,698              | 27,724          | 0.86%           | 0.77%           |
| Hangeul-2| 300              | 37,471              | 37,112              | 37,116          | 0.96%           | 0.95%           |
| Avg improvement in 300 |                  |                     |                     | 0.90%           | 0.87%           |
| Overall improvement |                  |                     |                     | 0.90%           | 0.88%           |

4.4. JPEG2000 Experimental Results

JPEG2000 coding, which is the new still image compression standard, is applied on SD and HD color and gray images shown in Figure 9. Experimental results of lossless coding about 2-level, 4-level and 8-level LUT MQ coder are presented in Table 17 and Table 18. This experiment was carried out by modifying Jasper program [32]. In the case of the color image, only the results of green color are shown because the results of red and blue color are similar and in the case of MAN image, the result of the gray values is given.

Compared to the existing JPEG2000 lossless compression, there were performance improvement of 1.2% for 2-level LUT and 1.38-1.45% for 4-level LUT and 1.5% for 8-level LUT MQ coder. At 2-level, the best results of 1.21% are obtained in mode-2, when $\alpha$ and $\beta$ equal to 1.05. In the case of 4-level LUT, mode-2 showed the lowest performance and in the other modes, the similar improvement was obtained. In the case of 8-level LUT, there is no significant difference in between two modes. The maximum improvement percentage was 1.51%. In all modes, the best improvements are obtained when $\alpha$ and $\beta$ is 1.05.

Table 17. Improvement percentage of compressed byte with proposed algorithm applying multi-level MQ in JPEG2000 (lossless compression, 2-level and 8-level LUT MQ coder) ($\alpha = \beta = 1.05$).

| Image | Image Size | Original JP2k Byte | 2-Level | 2-Level | 8-Level | 8-Level |
|-------|------------|---------------------|---------|---------|---------|---------|
|       | mode-1     | mode-2              | mode-1  | mode-2  | mode-1  | mode-2  |
| Baboon| 500        | 570,089             | 1.18%   | 1.40%   | 1.60%   | 1.61%   |
| Lena  | 512        | 446,062             | 1.23%   | 1.27%   | 1.55%   | 1.55%   |
| Monarch| 768       | 442,830             | 1.19%   | 1.13%   | 1.44%   | 1.43%   |
| Barbara| 720       | 464,509             | 1.22%   | 1.26%   | 1.54%   | 1.53%   |
| Zelda  | 780        | 564,745             | 0.98%   | 0.93%   | 1.25%   | 1.24%   |
| Man    | 1024       | 633,199             | 1.21%   | 1.30%   | 1.55%   | 1.54%   |
Table 18. Improvement percentage of compressed byte with proposed algorithm applying multi-level MQ in JPEG2000 (lossless compression, 4-level LUT MQ coder) ($\alpha = \beta = 1.05$).

| Image | Image Size | Original JP2k Byte | 4-Level | 4-Level | 4-Level | 4-Level |
|-------|------------|---------------------|---------|---------|---------|---------|
| Mode  | W  | H                  | mode-1 | mode-2 | mode-3 | mode-4 |
| Baboon| 500 | 480                | 570,089 | 1.55% | 1.45% | 1.56% | 1.49% |
| Lena  | 512 | 512                | 446,062 | 1.47% | 1.41% | 1.47% | 1.49% |
| Monarch| 768 | 512                | 442,830 | 1.36% | 1.35% | 1.38% | 1.38% |
| Barbara| 720 | 576                | 646,509 | 1.47% | 1.42% | 1.48% | 1.48% |
| Zelda | 780 | 576                | 564,745 | 1.18% | 1.12% | 1.17% | 1.19% |
| Man   | 1024| 1024               | 633,199 | 1.47% | 1.42% | 1.48% | 1.48% |
| Four people| 1280 | 720 | 823,889 | 1.51% | 1.43% | 1.51% | 1.54% |
| Jockey| 1920| 1080               | 1,851,880 | 1.51% | 1.43% | 1.50% | 1.52% |
| Average Improvement | | | 1.44% | 1.38% | 1.44% | 1.45% |

Table 19 and Table 20 show the experimental results of lossy compression with compression ratio of 20:1 in JPEG2000. In the case of lossy compression, performance improvement is lower than that of lossless compression. PSNR is compared at the same compression ratio. In the table at the compression ratio of 0.05, PSNR results using original JPEG2000 algorithm with existing MQ coding are shown and remaining values are the improvement ratio in PSNR at the same rate. In the case of 2-level, improvement of 0.26-0.28% is obtained and in the case of 4-level and 8-level, the performance improvement was achieved by 0.30-0.34%. In the case of 2-level, the best is when $\alpha$ and $\beta$ equal to 1 in mode-2. In lossy compression, because there is no significant difference in performance, 4-level LUT at mode-1 is adequate.

Table 19. Improvement percentage of PSNR with proposed algorithm applying multi-level MQ in JPEG2000 (lossy compression (compression ratio = 20:1), 2-level and 8-level LUT MQ coder).

| Image | Color | Original JPEG2000 PSNR (dB) | 2-Level | 2-Level | 8 Level | 8-Level |
|-------|-------|-----------------------------|---------|---------|---------|---------|
| mode  | Color | (\(\alpha, \beta\))       | mode-1 | mode-2 | model-1 | mode-2 |
| Baboon| G     | 26.025                      | 0.21%   | 0.31% | 0.20% | 0.38% |
| Lena  | G     | 35.943                      | 0.38%   | 0.39% | 0.44% | 0.38% |
| Monarch| G    | 40.269                      | 0.34%   | 0.33% | 0.38% | 0.39% |
| Barbara| G    | 35.953                      | 0.34%   | 0.33% | 0.40% | 0.42% |
| Zelda | G     | 41.754                      | 0.12%   | 0.13% | 0.12% | 0.15% |
| Man   | Gray  | 32.441                      | 0.42%   | 0.42% | 0.48% | 0.44% |
| Four people| G    | 44.364                      | 0.28%   | 0.28% | 0.29% | 0.31% |
| Jockey| G     | 45.473                      | 0.16%   | 0.16% | 0.19% | 0.19% |
| Average improvement | | | 0.28% | 0.29% | 0.31% | 0.34% |

Table 20. Improvement percentage of PSNR (peak signal-to-noise ratio) with proposed algorithm applying multi-level MQ in JPEG2000 (lossy compression (compression ratio = 20:1), 4-level LUT MQ coder).

| Image | Color | Original JPEG2000 | 4-Level | 4-Level | 4-Level | 4-Level |
|-------|-------|-------------------|---------|---------|---------|---------|
| Mode  | Color | (\(\alpha, \beta\)) | mode-1 | mode-2 | mode-3 | mode-4 |

JPEG2000 (lossy compression, compression ratio

Table 22. Improvement percentage of PSNR with proposed algorithm applying multi-level MQ in JPEG2000 (lossy compression, compression ratio = 50:1, 2-level and 8-level LUT MQ coder).

| (α, β) | PSNR (dB) | Original | 2-Level | 2-Level | 8-Level | 8-Level |
|--------|-----------|----------|---------|---------|---------|---------|
| Baboon | G         | 26.025   | 0.33%   | 0.21%   | 0.23%   | 0.36%   |
| Lena   | G         | 35.943   | 0.44%   | 0.44%   | 0.44%   | 0.43%   |
| Monarch| G         | 40.269   | 0.38%   | 0.37%   | 0.38%   | 0.37%   |
| Barbara| G         | 35.953   | 0.37%   | 0.41%   | 0.37%   | 0.40%   |
| Zelda  | G         | 41.754   | 0.18%   | 0.13%   | 0.13%   | 0.15%   |
| Man    | Gray      | 32.441   | 0.45%   | 0.40%   | 0.44%   | 0.48%   |
| Four people | G | 44.364   | 0.29%   | 0.29%   | 0.29%   | 0.31%   |
| Jockey | G         | 45.473   | 0.18%   | 0.17%   | 0.18%   | 0.18%   |
| Average improvement |           | 0.32%   | 0.31%   | 0.31%   | 0.34%   |

When the compression ratio is set to 50:1, the PSNR comparison results are shown in Table 21 and Table 22. The performance improvement ratio was 0.25–0.27% for 2-level LUT and 0.26–0.31% for 4-level LUT and 0.28–0.31% for 8-level LUT. 4-level and 8-level show similar performance to compare the best mode.

Table 21. Improvement percentage of PSNR with proposed algorithm applying multi-level MQ in JPEG2000 (lossy compression, compression ratio = 50:1, 2-level and 8-level LUT MQ coder).

| PSNR Improve | Color | Original | 2-Level | 2-Level | 8-Level | 8-Level |
|--------------|-------|----------|---------|---------|---------|---------|
| Mode (α, β)  | JPEG2000 | (1.0, 1.0) | (1.0, 1.0) | (1.0, 1.0) | (1.05, 1.05) | (1.0, 1.0) |
| Baboon       | G     | 22.874   | 0.29%   | 0.33%   | 0.36%   | 0.32%   |
| Lena         | G     | 32.894   | 0.26%   | 0.27%   | 0.27%   | 0.26%   |
| Monarch      | G     | 33.868   | 0.54%   | 0.51%   | 0.58%   | 0.56%   |
| Barbara      | G     | 30.601   | 0.32%   | 0.31%   | 0.39%   | 0.40%   |
| Zelda        | G     | 39.060   | 0.13%   | 0.13%   | 0.19%   | 0.21%   |
| Man          | Gray  | 29.040   | 0.21%   | 0.17%   | 0.22%   | 0.21%   |
| Four people  | G     | 39.398   | 0.25%   | 0.22%   | 0.20%   | 0.28%   |
| Jockey       | G     | 42.456   | 0.12%   | 0.11%   | 0.16%   | 0.15%   |
| Average improvement |       | 0.27%   | 0.26%   | 0.30%   | 0.30%   |

Table 22. Improvement percentage of PSNR with proposed algorithm applying multi-level MQ in JPEG2000 (lossy compression, compression ratio = 50:1, 4-level LUT MQ coder).

| PSNR Improve | Color | Original | 4-Level | 4-Level | 4-Level | 4-Level |
|--------------|-------|----------|---------|---------|---------|---------|
| Mode (α, β)  | JPEG2000 | (1.0, 1.0) | (1.0, 1.0) | (1.05, 1.05) | (1.0, 1.0) |
| Baboon       | G     | 22.874   | 0.29%   | 0.30%   | 0.41%   | 0.30%   |
| Lena         | G     | 32.894   | 0.27%   | 0.25%   | 0.25%   | 0.26%   |
| Monarch      | G     | 33.868   | 0.59%   | 0.51%   | 0.56%   | 0.56%   |
| Barbara      | G     | 30.601   | 0.31%   | 0.34%   | 0.39%   | 0.39%   |
| Zelda        | G     | 39.060   | 0.19%   | 0.18%   | 0.19%   | 0.22%   |
| Man          | Gray  | 29.040   | 0.19%   | 0.23%   | 0.20%   | 0.25%   |
| Four people  | G     | 39.398   | 0.27%   | 0.24%   | 0.27%   | 0.25%   |
| Jockey       | G     | 42.456   | 0.15%   | 0.14%   | 0.16%   | 0.15%   |
| Average improvement |       | 0.28%   | 0.27%   | 0.31%   | 0.29%   |

5. Discussion and Conclusions

A problem of the existing MQ coder to compute only addition and subtraction without multiplication is that encoding efficiency is degraded due to frequent MPS/LPS conditional exchange. In this paper, a method to reduce the artifacts caused by approximation
is proposed while retaining the probability estimation table of the binary MQ arithmetic coder. By using a pre-calculated 2-level, 4-level and 8-level quantized ($A \times Q_e$) look-up table instead of approximation value of $Q_e$, coding performance is improved. Instead of using uniform quantized value of ($A \times Q_e$), experiments were executed with varying the parameter $\alpha$ and $\beta$ at each level of 2, 4 and 8. Nonuniform quantization of ($A \times Q_e$) is applied with changing $\alpha$, $\beta$. When applied to the JBIG2 and JPEG2000 coding standards, we obtained affirmative results. By performing lots of experiments, optimal LUT was derived. In the case of JBIG2, the more quantized level, the better compressed performance. At 4-level or 8 levels, the best-chosen parameters $\alpha$ and $\beta$ are 1.0. Meanwhile, in the case of JPEG2000, the best compression performance was obtained at a value of 1.05 at most quantization levels.

Compressing a binary document without context, compression rate was improved by up to 3% compared to the existing method. In addition, according to the JBIG2 coding with 2-dimesional context, it was confirmed that the proposed method improved the compression rate of 0.9% compared to the conventional method. In the case of the JPEG2000 still image lossless coding, we obtained around 1.0–1.5% depending on the number of levels. In case of JPEG2000 lossy image compression, the improvement of PSNR was 0.3%, which is a little disappointing compared to lossless compression. From the experimental results, the performance is slightly lowered at 2-level LUT and the 8-level LUT is slightly better than the 4-level LUT, but the degree of improvement is reduced.

It is not easy to select the optimal level and mode in common suitable to every coding method, rather it will be possible to variably select the optimal quantization level in rate-distortion optimization. This optimization is not easily done because there are many different binary input sources with different probability distributions. If optimization is needed, we pre-code the sample binary sequence and achieve an optimal quantization level and parameters $\alpha$ and $\beta$ experimentally. However, if the optimization is not necessary, the best selection strategy for still image coding will be 4-level LUT with $\alpha$ and $\beta$ equal to 1.05 according to my experimental results. In case of binary document compression, 8-level quantization with $\alpha$ and $\beta$ equal to 1.0 is the best choice.

Finally, because probability estimation table in MQ coder was developed in 1980s, there is reason to change or modify the estimation table suitable to high resolution images such as HD and super HD images. The methods to be considered for this purpose are machine learning or deep learning approaches to train large data sets of binary sources.

**Funding:** The research was conducted starting with the 2013 sabbatical support at Kwangwoon University.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The author declares no conflict of interest.

**Acknowledgments:** The author would like to thank the reviewers for their helpful comments and suggestions.

**References**

1. Rissanen, J.J. Generalized Kraft Inequality and Arithmetic Coding. *IBM J. Res. Dev.* 1976, 20, 198–203.
2. Abramson, N. *Information Theory and Coding*; McGraw-Hill: New York, NY, USA, 1963.
3. Tian, X.; Le, T.M.; Lian, M. Chap. 2 Review of CAVLC, Arithmetic Coding, and CABAC. In *Entropy Coders of the H.264/AVC Standard, Signals and Communication Technology*; Springer, Berlin, Germany, 2011, pp. 29–39.
4. Jelinek, F. *Probabilistic Information Theory*; McGraw-Hill: New York, NY, USA, 1968.
5. Gilbert, E.N.; Moore, E.F. Variable-Length Binary Encodings. *Bell Syst. Tech. J.* 1959, 7, 932–967.
6. Cover, T.; Thomas, J. *Elements of Information Theory*; Wiley-InterScience: New York, NY, USA, 2005.
7. Pasco, R. Source Coding Algorithms for Fast Data Compression. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 1976.
8. Cover, T.M. Enumerative Source Encoding. *IEEE Trans. Inf. Theory* 1973, 19, 73–77.
9. Witten, I.H.; Neal, R.M.; Cleary, J.G. Arithmetic Coding for Data Compression. *Communication of the ACM* 1987, 30, 520–540.
10. Langdon, G., Jr. An Introduction to Arithmetic Coding. *IBM J. Res. Dev.* 1984, 28, 135–149.
11. Howard, P.; Vitter, J. Arithmetic Coding for Data Compression. *Proc. IEEE* 1994, 82, 856–865.
12. Moffat, A.; Neal, R.M.; Witten, H. Arithmetic Coding. Revisited. *ACM Trans. Inf. Syst.* 1998, 16, 256–294.
13. Said, A. *Introduction to Arithmetic Coding: Theory and Practice, Technical Report*; HPL-2004-76; HP Laboratories: Palo Alto, CA, USA, 2004.
14. Yu, Q.; Yu, W.; Yang, P.; Zheng, J.; Zheng, X.; He, Y. An Efficient Adaptive Binary Arithmetic Coder Based on Logarithmic Domain. *IEEE Trans. Image Process.* 2015, 24, 4225–4239.
15. Belyaev, E.; Liu, K.; Gabbouj, M.; Li, Y. An Efficient Adaptive Binary Range Coder and Its VLSI Architecture. *IEEE Trans. Circuits Syst. Video Technol.* 2015, 225, 1435–1446.
16. Belyaev, E.; Turlakov, A.; Egiazarian, K.; Gabbouj, M. An Efficient Adaptive Binary Arithmetic Coder with Low Memory Requirement. *IEEE J. Sel. Top. Signal Process Special Issue on Video Coding HEVC Beyond 2013*, 7, 1053–1061.
17. Pennebaker, W.; Mitchell, J.; Langdon, G., Jr.; Arps, R. An Overview of the Basic Principles of the Q-coder Adaptive Binary Arithmetic Coder. *IBM J. Res. Dev.* 1988, 32, 717–726.
18. ITU-T Recommendation T.88. *Information Technology—Lossy/Lossless Coding of Bi-Level Images*; February 2000. Available online: https://www.itu.int/rec/T-REC-T.88/en (accessed on 5 January 2021)
19. ITU-T Recommendation T.89. *Application Profiles for Recommendation T.88—Lossy/Lossless Coding of Bi-Level Images for Facsimile*; September 2001. Available online: https://www.itu.int/rec/T-REC-T.89/en (accessed on 5 January 2021)
20. Marpe, D. Fast Adaptive Binary Arithmetic Coder (M Coder). Available online: http://iphome.hhi.de/marpe/mcoder.htm (accessed on 5 January 2021)
21. Marpe, D.; Wiegand, T. A Highly Efficient Multiplication-free Binary Arithmetic Coder and Its Application in Video Coding. In Proceedings of the IEEE International Conference Image Processing (ICIP), Barcelona, Spain, 14–17 September 2003.
22. Marpe, D.; Schwartz, H.; Wigland, T. Context-based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. *IEEE Trans. Circuits Syst. Video Technol.* 2003, 13, 620–636.
23. Marpe, D.; Marten, G.; Cycon, H.L. A Fast Renormalization Technique for H.264/MPEG-4 AVC Arithmetic Coding. In Proceedings of the 51st Internationales Wissenschaftliches Kolloquium (IWK), Ilmenau, Deutschland, 11–15 September 2006; Ilmenau University of Technology: Ilmenau, Germany, 2006.
24. Mitchell, L.; Pennebaker, W.; Goertzel, G. Log Encoder/Decoder System. U.S. Patent 4,791,403, 13 December 1988. Available online: https://patents.google.com/patent/US4791403A/ (accessed on 5 January 2021)
25. Niu, B.; Cao, X.; Wang, L.; Wei, Z.; Yu, Q.; Zheng, J.; He, Y. An Efficient Probability Estimation Design for Logarithmic Binary Arithmetic Coding. In Proceedings of the Picture Coding Symposium (PCS), Ningbo, China, 12–15 November 2019.
26. Belyaev, E.; Gilmutdinov, M.; Turlakov, A. Binary Arithmetic Coding System with Adaptive Probability Estimation by Virtual Sliding Window. In Proceedings of the 10th IEEE International Symposium on Consumer Electronics, St Petersburg, Russia, 28 June–1 July 2006; 194–198.
27. ITU-T Recommendation T.81. *Information Technology—Digital Compression and Coding of Continuous-tone Still Images—Requirements and Guidelines*; September 1992. Available online: https://www.itu.int/rec/T-REC-T.81 (accessed on 5 January 2021)
28. ITU-T Recommendation T.82. *Information Technology—Coded Representation of Picture and Audio Information—Progressive Bi-Level Image Compression*; March 1993. Available online: https://www.itu.int/rec/T-REC-T.82 (accessed on 5 January 2021)
29. ISO/IEC 15444-1:2004. *Information Technology—JPEG 2000 Image Coding System: Part 1: Core Coding System*; September 2004. Available online: https://www.iso.org/standard/37674.html (accessed on 5 January 2021)
30. ITU-T Recommendation H.265. *High Efficiency Video Coding*; April 2013. Available online: https://www.itu.int/rec/T-REC-H.265 (accessed on 5 January 2021)
31. Ma, C.D.; Liu, D.; Peng, X.; Li, L.; Wu, F. Convolutional Neural Network-Based Arithmetic Coding for HEVC Intra-Predicted Residues. *IEEE Trans. Circuits Syst. Video Technol.* 2020, 30, 1901–1916.
32. Michael Adams, JPEG2000 Jasper Project. Available online: http://www.ece.uvic.ca/~frodo/jasper/ (accessed on 5 January 2021).