Block-based Learned Image Compression without Blocking Artifacts

Jong Wook Kim[1] Suyong Bahk[1] TaeHwa Lee[1] HyunDong Cho[1]
Donghyun Kim[2] Sung-Chang Lim[2] Jin Soo Choi[2] Hui Yong Kim[1,*]

[1]Kyung Hee University
[2]Electronics and Telecommunications Research Institute (ETRI)
{jong1334[1], clapd10[1], worrospeed[1], gusehd1113[1], hykim.v[1]}@khu.ac.kr
{kimddng[2], sclim[2], jschoi[2]}@etri.re.kr

Abstract

Learned Image Compression (LIC) outperforms traditional codecs but suffers from excessive peak memory usage when handling high-resolution images. Consequently, block-based LIC has been studied to reduce peak memory and peak computational cost but often introduces blocking artifacts that degrade visual quality. To mitigate this, the JPEG-AI standard introduced a patch-based scheme where overlapped blocks are coded independently using empirically determined overlap sizes. However, the experimental search for optimal overlaps is time-consuming and does not guarantee blocking-free reconstruction.

In this paper, we propose an analytic framework modeling overlap propagation through convolution and transposed convolution layers to precisely determine the minimal overlaps for blocking-free reconstruction. Based on the calculated minimum overlaps, we provide the block-based implementation methodology that could be applied to most CNN-based LIC models. Applied to four CNN-based LIC models on 4K images partitioned into various block sizes (256×256, 512×512), our method achieves rate–distortion performance identical to full-image coding while reducing average peak memory usage to 13.94% (encoder) and 13.33% (decoder), and average peak computational cost to 2.6% and 1.24%, respectively. Notably, the proposed block-based framework does not require any re-training of the original model. Furthermore, it can also be applied to most CNN-based image processing neural networks without worrying about any performance degradation.

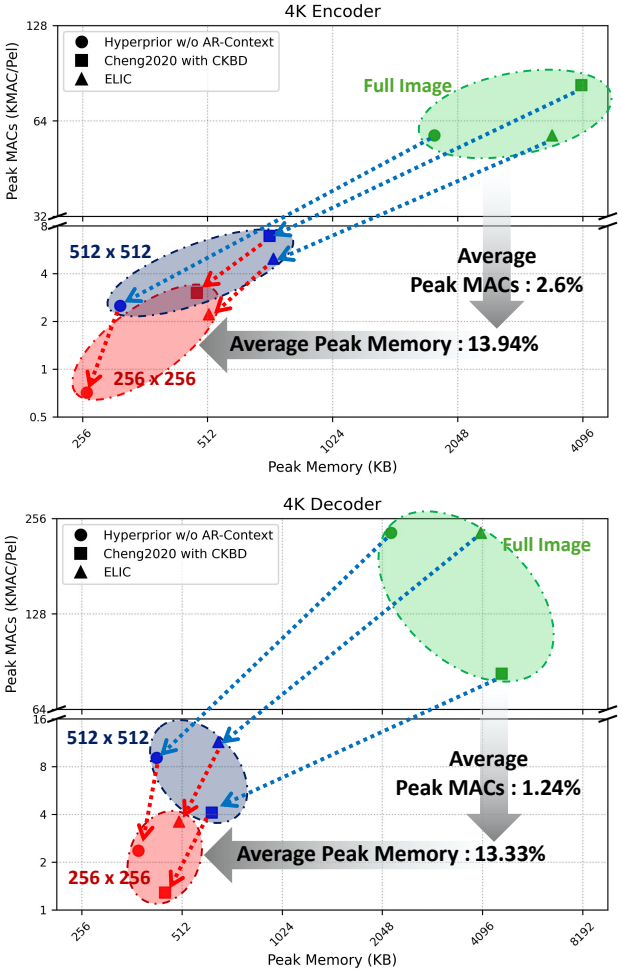

Figure 1. Compared to full-image LIC coding, the proposed block-based method significantly reduces peak memory and peak computations in various LIC models. The curve for Hyperprior with CKBD is omitted because it overlaps with the others.

1. Introduction

Convolutional Neural Network (CNN)-based Learned Image Compression (LIC) has rapidly advanced in recent

*Corresponding author

years [4, 5, 8, 12, 13, 17, 19, 20], with several CNN-based LIC models now achieving performance comparable to, or surpassing, state-of-the-art traditional codecs [8, 13]. However, such models typically require substantial peak memory and computational resources when decoding high-resolution images [15]. For example, ELIC [13] demands approximately 3.95 GB of decoder memory for a single 4K input image, making deployment on resource-constrained platforms, such as mobile or embedded devices, particularly challenging. Traditional image and video codecs address memory constraints by processing images block-by-block [7, 21–23]. JPEG applies the DCT on fixed 8×8 blocks [22], and modern standards such as AVC [23], HEVC [21], and VVC [7] also rely on block-based architectures. While block-based processing ensures low peak memory usage and high computational efficiency, it often produces blocking artifacts at low bitrates due to the absence of cross-block contextual information. To bring block-based efficiency to learned codecs, hybrid LIC approaches have been proposed [24, 25, 27]. Although they reduce memory consumption, additional post-processing networks are typically required to suppress blocking artifacts, introducing non-negligible computational overhead. This issue arises because CNN-based LIC models depend on a wide receptive field; when an image is processed block-by-block, information outside the current block is lost, inevitably causing artifacts at block boundaries.

JPEG-AI recently adopted a patch-based processing strategy [10], where each block is coded using an enlarged input patch beyond block boundaries. This overlap leverages contextual information, effectively eliminating blocking artifacts while reducing peak memory. However, this approach requires empirical finding of a proper overlap size through iterative search, demanding repeated evaluations whenever the architecture changes. Moreover, such empirical methods do not guarantee finding the optimal overlap size; insufficient overlap leads to boundary artifacts, whereas excessive overlap wastes memory and computation. In this paper, we propose a block-based framework that enables existing LIC models to operate block-wise ***without any retraining***, while preserving both ***blocking-free reconstruction*** and ***rate–distortion performance***. In contrast to empirical search, our method guarantees ***a one-shot analytic derivation*** of the precise minimum overlap required at each convolution and transposed convolution layer. By ensuring simplicity and reconstruction integrity, this approach makes block-wise decoding equivalent to full-image inference for any CNN architecture based solely on architectural parameters, removing the need for empirical tuning.

The main contributions are summarized as follows:

- We derive an analytic formulation to compute the minimum overlap required to guarantee blocking-free reconstruction across convolution and transposed convolution operations for arbitrary CNN architectures.
- Based on this formulation, we develop an implementation method that converts existing pre-trained CNN-based LIC models into block-based LIC models without any retraining. The results achieve substantial reductions in peak memory usage and peak computational cost while preserving rate–distortion performance and avoiding blocking artifacts, as demonstrated in Fig. 1 and Tab. 3.

2. Related Works

2.1. Evolutions of CNN-Based LIC

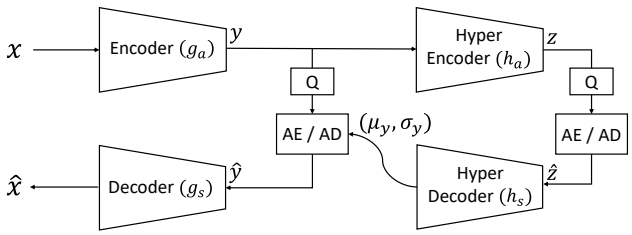

Figure 2. Basic Architectures of LIC. Q denotes Quantization, AE and AD denote arithmetic encoding and arithmetic decoding respectively.

The evolution of CNN-based LIC began with the VAE-based model by Ballé et al. [4, 5], who introduced the core LIC architecture consisting of an encoder (g_a), decoder (g_s), hyper-encoder (h_a), and hyper-decoder (h_s), as shown in Fig. 2. In this architecture, the encoder transforms the input x into a latent representation y, and the hyper-encoder produces a hyperprior z that summarizes the statistics of y. The hyperprior z is encoded using a fixed entropy model, and the hyper-decoder uses the quantized $\hat{z}$ to estimate distribution parameters of y for arithmetic coding. The decoder then reconstructs the image x from the quantized latent $\hat{y}$.

Early advancements further improved compression performance through more expressive entropy models, such as autoregressive (AR) context models [17, 20] and Gaussian Mixture Models (GMMs) [8], as well as more powerful network encoder and decoder architectures [8]. Subsequent research shifted to addressing the computational bottleneck of sequential AR models. Innovations included parallelizable context models like the checkerboard (CKBD) for spatial context [12] and channel-wise autoregressive (ChARM) models for inter-channel dependencies [19], which were later combined [13]. More recently, non-autoregressive approaches have emerged, such as using a correlation loss to directly reduce latent dependencies, eliminating the need for an explicit context model entirely [3]. Despite significant progress in both compression performance and decoding speed, the excessive memory consumption and computational load for processing high-resolution images remains a critical, unresolved challenge in CNN-based LIC.

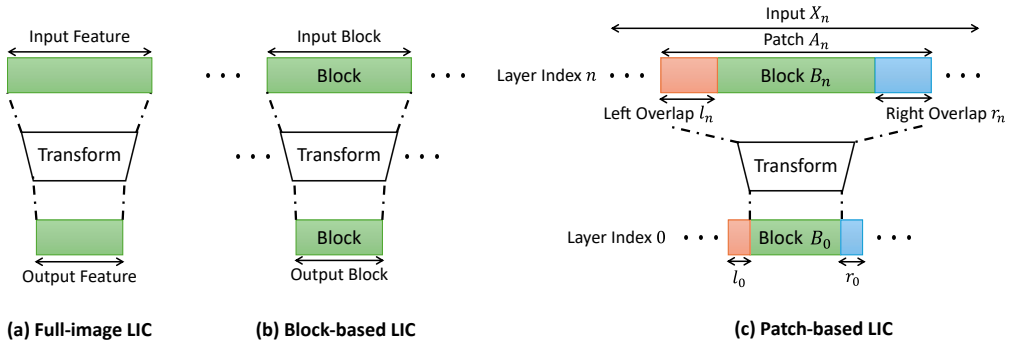

Figure 3. Comparison of different image partitioning strategies in LIC

2.2. Image Partitioning Strategies in LIC

A primary challenge in LIC is the significant increase in memory and computational demands associated with high-resolution images. To address this, recent LIC models have adopted memory-efficient partitioning strategies, which process input features in smaller, more manageable units. Fig. 3 illustrates these strategies in contrast to the conventional full-image approach. The conventional method, depicted in Fig. 3 (a), processes the entire image in a single pass. While this approach yields high reconstruction quality, it is highly resource-intensive. To mitigate this, the block-based strategy, shown in Fig. 3 (b), partitions the feature map into non-overlapping blocks that are processed independently. This method effectively reduces resource consumption but often introduces blocking artifacts at the boundaries. Hybrid approaches attempt to mitigate these artifacts using post-processing methods [24, 25, 27]; however, they introduce additional model complexity and cannot fully eliminate artifacts for every possible input image, as they rely on post-filters designed to minimize average artifacts across some datasets.

The patch-based scheme, shown in Fig. 3 (c), adopted in standards like JPEG-AI [10], resolves these issues. It partitions the feature map into blocks with overlapped features. After independent processing, the overlaps are discarded, and the valid blocks are stitched together. This yields a reconstruction that is similar to the full-image method with significantly lowered resource consumption. However, its reliance on empirical tuning to find the overlap size could result in sub-optimal performance; an insufficient overlap could result in artifacts, while an excessive one leads to computational redundancy. In addition, this tuning must be repeated for any architectural change, making the process inflexible. Zhang et al. [26] proposed a new LIC architecture that guarantees inter-block continuity. However, the applicability of this method is very limited; it cannot handle odd-sized kernels in downsampling and upsampling operations, which are widely used in most LIC models, and is not directly applicable to other LIC architectures because it depends on dedicated architectural components.

2.3. Standard Convolution Arithmetic

This section briefly reviews the arithmetic of standard convolutions and transposed convolutions, based on the work of Dumoulin et al. [9], using 1D operations for simplicity. The forward pass of a transposed convolution is mathematically equivalent to the backward pass of an associated standard convolution that shares the same kernel size (k), stride (s), and padding (p), but with its input and output dimensions swapped. For a standard convolution, the output size o is calculated from an input of size i as follows:

$$o = \lfloor (i + 2p - k)/s \rfloor + 1 \tag{1}$$

This operation can be expressed as a matrix multiplication, where $\mathbf{x}$ is the input vector, $\mathbf{W}$ is the weight matrix, and $\mathbf{y}$ is the output vector.

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{2}$$

Conversely, a transposed convolution upsamples the feature map, effectively reversing the spatial transformation. Its operation is represented by the transpose of the weight matrix, $\mathbf{W}^T$:

$$\mathbf{y}' = \mathbf{W}^T\mathbf{x}' \tag{3}$$

where $\mathbf{x}'$ is the input and $\mathbf{y}'$ is the output. As shown in Eq. (3), this operation can be modeled as a standard convolution with a stride of 1 by following these steps:

- Insert $(s-1)$ zeros between the elements of the input.
- Pad the interpolated input with p' on both sides and an additional padding of a on the right side. The values for p' and a are given by Eq. (4).

$$p' = (k - p - 1), \quad a = (i + 2p - k) \mod s \tag{4}$$

- Perform a standard convolution with a stride of 1 on the newly padded input.

This process ensures the output dimensions match the input dimensions of the associated standard convolution. However, it is crucial to note that during training, the weights of

a transposed convolution layer are learned as independent parameters and are not simply the transpose of an associated convolution's weight matrix. A detailed breakdown of the matrix formulation for Eq. (2)–Eq. (3) can be found in the supplementary material.

3. Notations and Assumptions

For clarity in this paper, the blocks can be categorized into three types based on their spatial location: those positioned at the left boundary, those at the right boundary, and those in the central region. We refer to them as the *left block*, *center block*, and *right block*, respectively. The left and right blocks are collectively referred to as *boundary blocks*.

3.1. Notations

This section defines the notation used to derive the overlap recurrence relations. As illustrated in Fig. 3 (c), the patch processed at each layer consists of a central data block and its surrounding left and right overlaps. The layers are indexed from N (initial input) down to 0 (final output), where layer n takes the feature at level n as input and produces the feature at level $n-1$. The following symbols apply to both convolution and transposed convolution operations.

Symbol	Meaning
N	Total number of layers in the network
n	Layer index, where $n \in \{0, 1, \ldots, N\}$
$\mathbb{N}_L$	The set of layer indices for recurrence relations, defined as $\{1, 2, \ldots, N\}$
X_n	Size of the entire feature map at layer n
A_n	Size of the patch at layer n
B_n	Size of the block within the patch at layer n
l_n, r_n	Sizes of the left and right overlaps at layer n
k_n	Kernel size of the n-th layer
s_n	Stride of the n-th layer
p_n	Symmetric padding size of the n-th layer
a_n	Asymmetric padding size of the n-th transposed convolution layer

3.2. Convolution Assumptions and Padding

To ensure predictable spatial scaling, we adopt two core assumptions for all convolution and transposed convolution operations.

Assumption 1 *The input size is an integer multiple of the stride s_n. Consequently, a convolution downsamples its input by a factor of exactly s_n, and its corresponding transposed convolution upsamples its input by the same factor.*

For convolution, this relationship is expressed as:

$$\begin{aligned} X_n \mod s_n = 0 \quad \text{and} \quad X_{n-1} = X_n / s_n, \\ B_n \mod s_n = 0 \quad \text{and} \quad B_{n-1} = B_n / s_n, \end{aligned} \quad n \in \mathbb{N}_L \tag{5}$$

For transposed convolution, the relationship is:

$$\begin{aligned} X_{n-1} = s_n \cdot X_n, \\ B_{n-1} = s_n \cdot B_n, \end{aligned} \quad n \in \mathbb{N}_L \tag{6}$$

Assumption 1 is not restrictive, as most CNN architectures are designed for integer-factor downsampling and upsampling. To consistently apply this assumption, we define a specific kernel alignment rule.

Assumption 2 *The convolution begins with the kernel's center aligned with the first element of the unpadded input. The kernel then shifts by s_n elements for each subsequent operation.*

To formalize this, we first define the kernel's center index, c_n, for a kernel of size k_n with indices $[0, \ldots, k_n - 1]$:

$$c_n = \lfloor (k_n - 1)/2 \rfloor \tag{7}$$

Based on Assumption 2, the number of kernel elements overhanging the input's left boundary during the first operation, denoted k_n^l, must equal the center index c_n.

$$k_n^l = c_n = \lfloor (k_n - 1)/2 \rfloor \tag{8}$$

Similarly, k_n^r is the number of kernel elements overhanging the right boundary during the final operation:

$$k_n^r = \max(0, (k_n - 1 - k_n^l) - (s_n - 1)) \tag{9}$$

These principles extend to transposed convolutions. A transposed convolution is equivalent to a standard convolution with a stride of 1, but its kernel is spatially flipped. The left and right overhangs, ${k'}_n^l$ and ${k'}_n^r$, are thus swapped:

$$\begin{aligned} {k'}_n^r &= k_n^l = \lfloor (k_n - 1)/2 \rfloor \\ {k'}_n^l &= k_n - 1 - {k'}_n^r \end{aligned} \tag{10}$$

Under our assumptions, the required symmetric padding p_n for a standard convolution is:

$$p_n = k_n^l, \quad n \in \mathbb{N}_L \tag{11}$$

A transposed convolution requires an asymmetric output padding term, a_n, to precisely reverse the spatial transformation of its corresponding convolution. This term is defined as:

$$a_n = (X_n + 2p_n - k_n) \mod s_n, \quad n \in \mathbb{N}_L \tag{12}$$

The use of p_n and a_n ensures exact spatial alignment between downsampling and upsampling operations. For a detailed explanation of Eq. (7)–Eq. (11), please refer to the supplementary material.

4. Minimum Overlap Calculation

This section details a method to recursively calculate the minimum required overlap (l_n, r_n) for an input block. The calculation proceeds layer by layer, accounting for kernel sizes and strides, in the direction of overlap growth. The methods presented here focus on center blocks (Sec. 4.1, Sec. 4.2), with boundary blocks addressed in Sec. 4.4.

4.1. Minimal Overlap Calculation: Conv

Since convolution with stride larger than one reduces spatial resolution, the required overlap naturally grows when moving from lower to higher layer indices. Therefore, aligning with the methodology from Sec. 4, the calculation must follow this direction of increasing layer indices.

$$\begin{aligned} l_n &= s_n \cdot l_{n-1} + k_n^l, \\ r_n &= s_n \cdot r_{n-1} + k_n^r, \end{aligned} \quad n \in \mathbb{N}_L \tag{13}$$

To obtain the minimum required integer overlap, we set the initial overlap to zero and iteratively apply the above recurrence Eq. (13) in the direction of increasing layer indices. This enables sequential computation of the minimum necessary overlap at each layer. The process is illustrated in Algorithm 1.

Algorithm 1 Minimal Overlap Calculation: Conv

Input: Total layers N, per-layer kernel sizes $\{k_n\}_{n=1}^N$, per-layer strides $\{s_n\}_{n=1}^N$
Output: Per-layer minimal overlaps $\{(l_n, r_n)\}_{n=0}^N$
$l_0 \leftarrow 0,\ r_0 \leftarrow 0$ ▷ Initialization
for $n \leftarrow 1$ to N **do**
  $k_n^l \leftarrow \lfloor (k_n - 1)/2 \rfloor$ ▷ Eq. (8)-Eq. (9)
  $k_n^r \leftarrow \max(0,\ (k_n - 1 - k_n^l) - (s_n - 1))$
  $l_n \leftarrow s_n \cdot l_{n-1} + k_n^l, \quad r_n \leftarrow s_n \cdot r_{n-1} + k_n^r$ ▷ Eq. (13)
end for
return $\{(l_n, r_n)\}_{n=0}^N$

For a detailed derivation of Eq. (13) and an example of Algorithm 1, please refer to the supplementary material.

4.2. Minimal Overlap Calculation: T.Conv

For transposed convolution, an upsampling operation, the required overlap increases toward lower layer indices. Thus, the calculation must proceed from higher to lower layer indices. We define the recurrence relation in Eq. (14) to compute the overlap of the preceding layer, (l_{n-1}, r_{n-1}), from that of the subsequent layer, (l_n, r_n).

$$\begin{aligned} l_{n-1} &= s_n \cdot l_n - {k'}_n^l \\ r_{n-1} &= s_n \cdot r_n - {k'}_n^r - (s_n - 1) \end{aligned} \tag{14}$$

The calculation consists of two passes. First, we recursively apply Eq. (14) from higher to lower indices to determine the smallest non-negative integer overlap (l_N, r_N) for which $l_0 \geq 0$ and $r_0 \geq 0$ hold. Second, using this initial overlap, we recursively apply the same update from higher to lower indices to materialize the minimum required overlap for all layers. Algorithm 2 summarizes this procedure.

Algorithm 2 Minimal Overlap Calculation: T.Conv

procedure PROPAGATEOVERLAP($l, r, record$)
  if $record$ **then**
    $l_N \leftarrow l, \quad r_N \leftarrow r$ ▷ record overlap
  end if
  for $n \leftarrow N$ **down to** 1 **do**
    ${k'}^r \leftarrow \lfloor (k_n - 1)/2 \rfloor, \quad {k'}^l \leftarrow k_n - 1 - {k'}^r$
    $l \leftarrow s_n \cdot l - {k'}^l$
    $r \leftarrow s_n \cdot r - {k'}^r - (s_n - 1)$
    if $record$ **then**
      $l_{n-1} \leftarrow l, \quad r_{n-1} \leftarrow r$ ▷ record overlap
    end if
  end for
  return (l, r) ▷ equals (l_0, r_0)
end procedure
Pass 1 (determine top-layer overlap):
Find smallest non-negative integers $(l_N^\star, r_N^\star)$ s.t.
  $(l_0, r_0) \leftarrow$ PROPAGATEOVERLAP$(l_N^\star, r_N^\star, false)$
  $l_0 \geq 0,\ r_0 \geq 0$
Pass 2 (materialize all layers):
$(l_N, r_N) \leftarrow (l_N^\star, r_N^\star)$
$(l_0, r_0) \leftarrow$ PROPAGATEOVERLAP$(l_N^\star, r_N^\star, true)$
return $\{(l_n, r_n)\}_{n=0}^N$

For detailed example and explanation of Eq. (14), please refer to the supplementary material.

4.3. Minimal Overlap Calculation: PixelShuffle

The PixelShuffle operation performs spatial upsampling. Thus, given an upscaling factor u, the output overlap increases by a factor of u compared to the input overlap, as expressed in Eq. (15).

$$l_{n-1} = u \cdot l_n, \quad r_{n-1} = u \cdot r_n \tag{15}$$

4.4. Minimal Overlap Calculation: Boundary Block

For boundary blocks, the overlap on the side adjacent to the image edge is zero. Specifically, for the left block, we set the left overlap $l_n = 0$, and for the right block, we set the right overlap $r_n = 0$ for all layers. The remaining non-zero overlap (r_n for the Left Block and l_n for the Right Block) is calculated using the methods described in Sec. 4.1 and Sec. 4.2. The processing of these boundary blocks is detailed in Sec. 5.

4.5. Minimal Overlap Calculation in LIC Models

In typical CNN-based LIC architectures, the overall minimum overlap is obtained by combining the layer-wise rules

in Sec. 4.1–Sec. 4.4. Since the encoder (g_a) and hyper encoder (h_a) consist only of convolution layers, their overlaps are computed by recursively applying the convolution rule in Sec. 4.1. For the decoder (g_s) and hyper decoder (h_s), which employ a combination of convolutions, transposed convolutions, and PixelShuffle, the overlaps are derived using the formulations in Sec. 4.1–Sec. 4.3. Sequentially applying these rules yields the minimum required overlap for the entire LIC model.

Tab. 1 shows the initial overlap in various LIC models. The overlap is represented in the format (l_n, r_n), and the overlaps in the vertical direction (top and bottom) use the same values as l_n and r_n, respectively. For the results of other models and their calculation processes, please refer to the supplementary material.

Table 1. Initial overlap for various LIC models

Component	Hyperprior w/o AR-Context	Hyperprior With CKBD	Cheng With CKBD	ELIC
h_a	(7, 4)	(7, 4)	(7, 4)	(7, 4)
h_s	(2, 3)	(2, 3)	(4, 4)	(2, 3)
g_a	(30, 15)	(30, 15)	(117, 102)	(132, 117)
g_s	(2, 3)	(2, 3)	(11, 11)	(9, 10)
CKBD	-	(2, 2)	(2, 2)	(2, 2)
ChARM	-	-	-	(6, 6)

5. Blocking-free Block-based LIC

This section presents implementation details for block-based LIC without blocking artifacts using the optimal overlaps l_n and r_n. For single-path CNNs, applying the padding rules in Sec. 5.1 makes the patch-based output ***identical*** to that of full-image LIC. Two cases require special handling: boundary blocks and multi-path networks (e.g., ResNet [14]). We describe how to handle both cases while maintaining identical reconstruction.

5.1. Padding Strategies

For standard convolutions, patch overlap substitutes for padding:

$$p_n = 0, \quad n \in \mathbb{N}_L. \tag{16}$$

For transposed convolutions, overlap also replaces padding. Thus we set $p'_n = 0$ and $a_n = 0$, giving:

$$p_n = k_n - 1, \quad a_n = 0, \quad n \in \mathbb{N}_L. \tag{17}$$

5.2. Handling Boundary Blocks

For boundary blocks, the padding size can also be applied by Eq. (11), where zero-padding of size p_n is applied along the boundary direction to reproduce the behavior of full-image convolution. However, in this case the minimum overlaps and padding sizes of boundary blocks differ from those of center blocks, resulting in different convolution operations and increased implementation complexity due to branching. To apply the *same convolution operation* to boundary blocks as to center blocks, we align the input shape of boundary blocks with that of center blocks. Specifically, for a left (or right) boundary block at layer n, we apply additional zero-padding of size l_n (or r_n) to the boundary side so that the input block shape matches that of a center block.

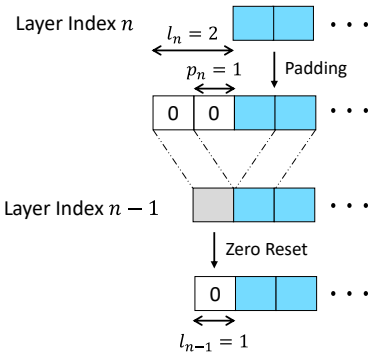

Figure 4. Left boundary block handling process. The center block has $l_n = 2$, $B_n = 2$, and $r_n = 2$ with a convolution layer ($k_n = 3$, $s_n = 1$). The output features contaminated by the kernel bias due to this additional padding are shown in gray.

As shown in Fig. 4, the original padding size p_n is sufficient to emulate full-image convolution. However, the additional padding of size l_n (or r_n) introduces an unintended side effect. Because these extra padded regions contain zeros, a portion of the output l_{n-1} (or r_{n-1}) becomes influenced by the kernel bias. This *bias-contaminated* region would otherwise propagate to deeper layers and distort the block-based reconstruction. To mitigate this, after each convolution layer, we apply a *Zero Reset* step that resets the contaminated output region of size l_{n-1} (or r_{n-1}) back to zero. This ensures that only valid features are propagated to subsequent layers, while maintaining identical convolution operations between center and boundary blocks.

5.3. Handling Multi-Path Networks

Multi-path networks such as residual [14] and gating networks are prevalent in many CNN architectures. They merge outputs from two parallel paths through element-wise operations. Since such element-wise operations are valid only when both paths share the same spatial support, cropping is needed before merging to match their spatial alignment. The required cropping sizes are given by:

$$\Delta l = |l_P^{out} - l_S^{out}|, \quad \Delta r = |r_P^{out} - r_S^{out}|. \tag{18}$$

Here, P and S denote the Primary and Secondary paths, where the Primary path contains more layers. Fig. 5 illustrates this rule in residual and gating networks. In Fig. 5(a), the input to the skip path is cropped by two units on both sides before addition. In Fig. 5(b), the gating map is cropped by one unit before multiplication to match the overlap difference.

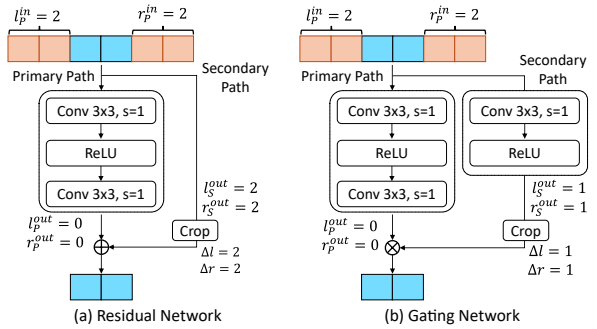

Figure 5. (a) Residual Network. (b) Gating Network. Both inputs have $l_P^{in} = r_P^{in} = 2$.

6. Experiments with Various LIC Models

6.1. Experimental Details

Training To ensure reproducibility, we trained four full-image-based LIC models (Hyperprior [5] without AR, Hyperprior with CKBD [5, 12], Cheng with CKBD [8, 12], and ELIC [13]) on the COCO2017 [18] dataset. Each model was trained for 3 million steps using the Adam optimizer [16] and the rate-distortion loss function:

$$\mathcal{L} = R + \lambda D \tag{19}$$

Training utilized $\lambda \in \{16, 32, 75, 150, 300, 450\} \times 10^{-5}$, an initial learning rate of 1×10^{-4} (halved at 2M and 2.5M steps), and 256×256 random crops, which were increased to 512×512 for the final 500k steps.

Evaluation We evaluated the Rate-Distortion (RD) performance, peak memory, and peak multiply-accumulate (MAC) operations on the DIV2K [2] and DIV8K [11] datasets. Notably, the same model weights were used for both full-image inference and our proposed block-based method. The evaluation was performed using block sizes of 256×256 and 512×512, with results averaged over five randomly selected images for each resolution.

6.2. Minimal Overlap Validation

This experiment evaluates the impact of violating the required minimal overlap. Specifically, for each transform network of the Hyperprior model without AR context [5] (h_a, h_s, g_a, g_s), we decrease the minimal overlap by exactly one pixel in all four directions (left, right, top, bottom) only for that network. The evaluation is conducted on 2K images with block size 256×256.

As shown in Tab. 2, insufficient overlap leads to severe RD performance (BD-rate, BD-PSNR [6]) degradation. This is caused by incomplete feature coverage in the encoder and inaccurate probability modeling in the hyperprior (h_a,h_s), which increases bpp. Visually, as seen in Fig. 6, insufficient overlap in both the encoder (g_a) and decoder (g_s) leads to reconstruction artifacts, with the former causing loss of latent information and the latter causing loss of structural information in the reconstructed image.

Table 2. BD-rate and BD-PSNR results compared with full-image reconstruction. "–" indicates BD-rate computation impossible because of severe PSNR degradation.

Transform	BD-rate (%)	BD-PSNR (dB)
h_a	+76.34	-2.99
h_s	+230.52	-5.91
g_a	–	-14.36
g_s	–	-20.31

Figure 6. Visual artifacts caused by insufficient minimal overlap in g_a, g_s. (a) Full-image reconstruction. (b) insufficient overlap only in g_a. (c) insufficient overlap only in g_s

Table 3. BD-rate comparison of sufficient overlap with Full Image Reconstruction

Resolution	Hyperprior w/o AR-Context	Hyperprior With CKBD	Cheng With CKBD	ELIC
2K	+0.0013%	+0.0021%	+0.0081%	-0.0000%
4K	+0.0022%	-0.0018%	-0.004%	+0.0050%

Tab. 3 shows that the minimal overlap is indeed the necessary condition for stable block-based processing: even a one-pixel reduction in any transform network causes substantial RD degradation and visible artifacts, while the minor remaining discrepancies arise only from numerical precision, confirming that our calculation yields the minimal and sufficient overlap.

6.3. Peak MACs & Peak Memory

We compared the peak memory and peak MACs of our method against full-image reconstruction on 2K and 4K images. As shown in Fig. 1, our proposed block-based processing significantly reduces both metrics. For 4K images with block sizes of 256×256 and 512×512, our method reduces average peak memory to 13.94% (encoder) and 13.33% (decoder) of full-image inference, while peak MACs are reduced to 2.6% and 1.24%, respectively. Similar trends are observed for 2K resolution, with full results provided in the supplementary material.

7. Experiments on the JPEG-AI

In this section, we verify that the proposed method produces results equivalent to full-image reconstruction.

7.1. Experimental Setup

We used the JPEG-AI reference software [1] configured for the Base Operating Point (BOP) with the official pre-

trained weights. While the baseline follows official overlap configuration and utilizes the native tiling mechanism, our method employs a custom implementation to accommodate the asymmetric overlaps required by our framework, which are not natively supported. Our proposed method is applied to the synthesis and analysis transforms (h_a, h_s, g_a, g_s). In both the baseline and the proposed method, the Multistage Context Model (MCM) operates on the full latent representation without patch-based processing. Because the latent-space inputs have much smaller spatial dimensions, full-latent MCM does not introduce peak memory or peak computation issues. Experiments were conducted using block sizes of 192×192 and 448×448.

7.2. Overlap Calculation in JPEG-AI

In the baseline, each network applies the padding rule of Eq. (11) to every patch under assumptions 1 and 2.

Table 4. Overlap propagation in g_a and g_s within the JPEG-AI reference software (Baseline vs. Proposed). PS denotes PixelShuffle.

g_a **(Y Encoder, Proposed)**					g_a **(Y Encoder, Baseline)**				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
7	Conv	3	2	**(29, 14)**	7	Conv	3	2	**(32, 32)**
6	Conv	3	1	(14, 7)	6	Conv	3	1	(16, 16)
...	...	...	...	...	...	...	...	...	...
1	Conv	1	1	(0, 0)	1	Conv	1	1	(2, 2)
0	-	-	-	(0, 0)	0	-	-	-	(2, 2)
g_s **(Y Decoder, Proposed)**					g_s **(Y Decoder, Baseline)**				
n	Op.	k_n	s_n	(l_n, r_n)	n	Op.	k_n	s_n	(l_n, r_n)
8	Conv	3	1	**(4, 4)**	8	Conv	3	1	**(2, 2)**
7	T.Conv	4	2	(3, 3)	7	T.Conv	4	2	(2, 2)
...	...	...	...	...	...	...	...	...	...
1	PS ($u = 4$)	-	-	(1, 1)	1	PS ($u = 4$)	-	-	(8, 8)
0	-	-	-	**(4, 4)**	0	-	-	-	**(32, 32)**

In Tab. 4, our proposed method yields a smaller overlap in the encoder compared to the baseline. In the decoder, it also mitigates the overlap growth observed in the baseline.

7.3. Results

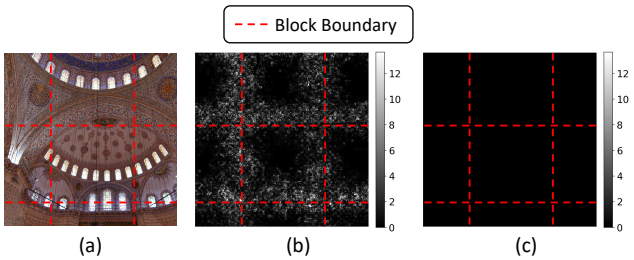

Figure 7. MSE comparison with full-image reconstruction between the baseline and proposed methods. (a) full-image reconstruction. (b) MSE in baseline. (c) MSE in proposed.

Fig. 7 shows the mean squared error (MSE) maps of the proposed method and the baseline, computed with respect to full-image reconstruction. The baseline clearly shows artifacts along block boundaries. In the baseline, every patch, including center blocks, uses both overlap and additional zero-padding. As explained in Sec. 5.2, this oversized zero-padded region activates the kernel bias, producing bias-contaminated feature values even in valid features. These features accumulate through the network and finally appear as visible artifacts along block boundaries.

Table 5. Complexity and BD-rate Comparison. BD-rate values are measured with respect to full-image inference. M. denotes method, B denotes Baseline and P denotes Proposed. Peak MAC/pel values correspond to the Y branch. Total kMAC/pel values for Y/U/V and full-image are provided in the supplementary.

Res.	**Block**	**M.**	**Peak MAC/pel**		**BD-rate (%)**			**Dec. T. (s)**
			Enc.	**Dec.**	**Y**	**U**	**V**	
2K	192×192	B	218.3	97.0	0.79	1.36	1.27	**1.24**
		P	**176.2**	**74.3**	**≤0.01**	**≤0.01**	**≤0.01**	1.45
	448×448	B	873.3	388.1	0.26	0.44	0.23	**0.99**
		P	**786.9**	**341.2**	**≤0.01**	**≤0.01**	**≤0.01**	1.06
4K	192×192	B	87.6	38.9	0.60	2.59	2.36	**1.98**
		P	**70.7**	**29.8**	**≤0.01**	**≤0.01**	**≤0.01**	2.46
	448×448	B	350.5	155.8	0.22	1.00	0.75	**1.59**
		P	**315.8**	**136.9**	**≤0.01**	**≤0.01**	**≤0.01**	1.63

As shown in Tab. 5, our method yields bit-identical reconstruction to full-image inference with lower peak computational cost than the baseline. This is because the baseline suffers from error propagation rooted in its padding strategy. Our method also reduces the peak computational cost by using smaller overlaps in the encoder and preventing their excessive growth in the decoder. Although the proposed method incurs a slight increase in decoding time, this is compensated by its lower peak computation and superior reconstruction quality.

8. Conclusion

We introduced a retraining-free, block-based LIC framework that analytically determines minimal overlaps to ensure equivalence to full-image inference. Applying our method to existing LIC models yields significant reductions in peak memory and peak computational costs.

While our method applies broadly to CNN-based architectures and standards such as JPEG-AI [10], extending it to Transformer-based models still requires further study. Nevertheless, CNN-based models remain highly attractive in practice due to their computational efficiency, which is consistent with the primary focus of this work.

9. Acknowledgements

This work was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00072, Development of Audio/Video Coding and Light Field Media Fundamental Technologies for Ultra Realistic Tera-media). This work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2025-02216217).

References

[1] JPEG-AI reference software. https://gitlab.com/wg1/jpeg-ai/jpeg-ai-reference-software. 7

[2] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017. 7

[3] Muhammad Salman Ali, Yeongwoong Kim, Maryam Qamar, Sung-Chang Lim, Donghyun Kim, Chaoning Zhang, Sung-Ho Bae, and Hui Yong Kim. Towards efficient image compression without autoregressive models. *Advances in Neural Information Processing Systems*, 36:7392–7404, 2023. 2

[4] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. In *5th International Conference on Learning Representations, ICLR 2017*, 2017. 2

[5] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations (ICLR)*, 2018. 2, 7

[6] Gisle Bjontegaard. Calculation of average psnr differences between rd-curves. *ITU SG16 Doc. VCEG-M33*, 2001. 7

[7] Benjamin Bross, Jianle Chen, Shan Liu, and Ye-Kui Wang. Overview of the versatile video coding (VVC) standard and its applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3736–3764, 2021. 2

[8] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7939–7948, 2020. 2, 7

[9] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018. 3

[10] Semih Esenlik, Yaojun Wu, Zhaobin Zhang, Ye-Kui Wang, Kai Zhang, Li Zhang, João Ascenso, and Shan Liu. An overview of the jpeg ai learning-based image coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, page 1–1, 2025. 2, 3, 8

[11] Shuhang Gu, Andreas Lugmayr, Martin Danelljan, Manuel Fritsche, Julien Lamour, and Radu Timofte. Div8k: Diverse 8k resolution image dataset. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3512–3516, 2019. 7

[12] Dailan He, Yaoyan Zheng, Baocheng Sun, Yan Wang, and Hongwei Qin. Checkerboard context model for efficient learned image compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14771–14780, 2021. 2, 7

[13] Dailan He, Ziming Yang, Weikun Peng, Rui Ma, Hongwei Qin, and Yan Wang. Elic: Efficient learned image compression with unevenly grouped space-channel contextual adaptive coding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5718–5727, 2022. 2, 7

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[15] Sonain Jamil, Md Jalil Piran, MuhibUr Rahman, and Oh-Jin Kwon. Learning-driven lossy image compression: A comprehensive survey. *Engineering Applications of Artificial Intelligence*, 123:106361, 2023. 2

[16] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

[17] Jooyoung Lee, Seunghyun Cho, and Seung-Kwon Beack. Context-adaptive entropy model for end-to-end optimized image compression. In *International Conference on Learning Representations*, 2018. 2

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 7

[19] David Minnen and Saurabh Singh. Channel-wise autoregressive entropy models for learned image compression. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3339–3343. IEEE, 2020. 2

[20] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 10771–10780, 2018. 2

[21] Gary J Sullivan, Jens Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012. 2

[22] Gregory K Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38 (1):18–34, 1992. 2

[23] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003. 2

[24] Yaojun Wu, Xin Li, Zhizheng Zhang, Xin Jin, and Zhibo Chen. Learned block-based hybrid image compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(6):3978–3990, 2021. 2, 3

[25] Zhongzheng Yuan, Haojie Liu, Debargha Mukherjee, Balu Adsumilli, and Yao Wang. Block-based learned image coding with convolutional autoencoder and intra-prediction aided entropy coding. In *2021 Picture Coding Symposium (PCS)*, pages 1–5, 2021. 2, 3

[26] Zifu Zhang, Shengxi Li, Henan Liu, Mai Xu, and Ce Zhu. Continuous patch stitching for block-wise image compression. *IEEE Signal Processing Letters*, 2025. 3

[27] Zhenghui Zhao, Chuanmin Jia, Shanshe Wang, Siwei Ma, and Jiansheng Yang. Learned image compression using adaptive block-wise encoding and reconstruction network. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021. 2, 3