Supplementary Material

1. Analysis of of bit rate distribution

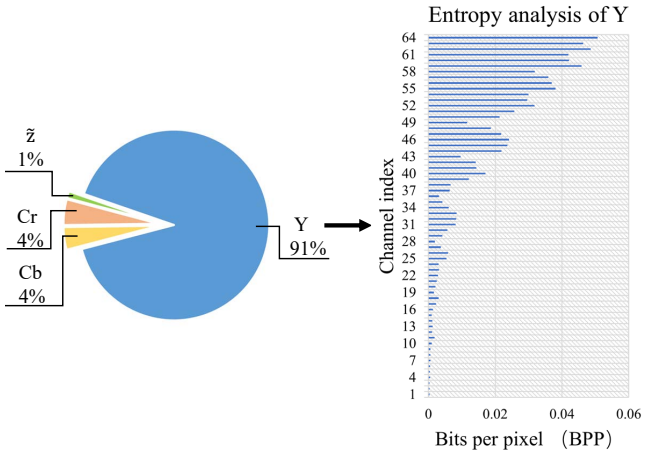


Figure 1. The distribution of bit rate for our proposed model.

As shown in the left of Fig. 1, the Y component occupies much more bitrates than Cb and Cr components. Then we analyze the entropy of each channel in the Y component and find that channels with smaller indexes (corresponding to higher frequency) occupy less entropy, which demonstrates the information asymmetry mentioned in Sec. 3.4 in the main text.

2. Entropy coding and probability model

Lossless compression technique is essentially an entropy coder along with a probability model. JPEG algorithm adopts Huffman coding together with Huffman tables defining the probability model to compress quantized DCT coefficients losslessly. However, these coefficients are considered independently and identically distributed (i.i.d.) under this fixed probability model, resulting in a mismatch between estimated and actual data distribution, which decreases compression savings.

Our method contains two significant improvements in this aspect. First, our method uses Laplace distribution with different parameters for each coding symbol to obtain an adaptive probability model, where the two parameters (scale b and location μ) of each Laplace distribution are learned by neural networks. It is known that AC coefficients of Fourier-related transformations, like DCT coefficients of JPEG algorithm, obey Laplace distribution [2]. Second, we replace the Huffman coding with arithmetic coding, which is a more efficient technique that almost achieves the lower bound entropy for long enough symbol streams.

3. Details about performance on different quality levels

Tab. 1 is the detailed data corresponding to Figure 6 in the main text. All results at row **Ours (QP 75)** and row **Ours (QP 95)** are obtained by our models trained with QP 75 and QP 95 respectively. While results in row **Ours (QP independent)** are generated by 7 models trained with QP $35, 45, 55, 65, 75, 85$ and 95 respectively to ensure that QP value is exactly the same for training and testing. Data of these three rows are visualized by Fig. 2. It shows that the **Ours (QP 75)** has comparable performance with **Ours (QP independent)** at all QPs lower than 85, which further demonstrates that our model trained for $quality = 75$ can generalize well to different quality levels except very high quality like 95.

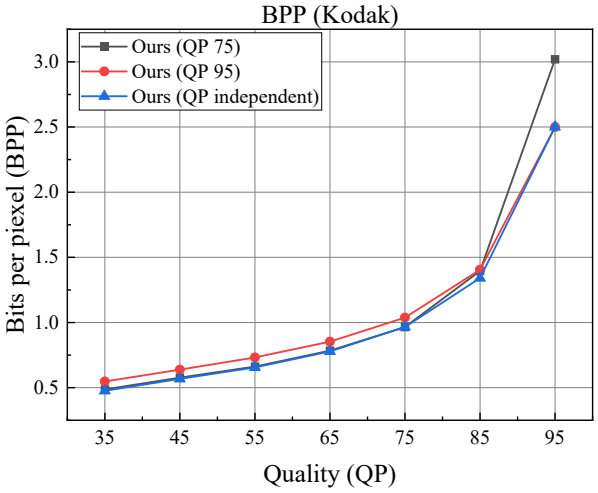


Figure 2. Comparison of bits per pixel (BPP) on Kodak dataset when recompressing JPEG images of different quality levels (QP).

Method	QP35	QP45	QP55	QP65	QP75	QP85	QP95
JPEG	0.729	0.850	0.964	1.127	1.369	1.859	3.401
Lepton	0.549	0.655	0.755	0.896	1.102	1.520	2.786
JPEG XL	0.599	0.710	0.815	0.960	1.173	1.595	2.849
CMIX	0.519	0.621	0.718	0.853	1.054	1.452	2.648
Ours (QP 75)	0.487	0.577	0.662	0.784	0.965	1.396	3.022
Ours (QP 95)	0.547	0.638	0.732	0.853	1.038	1.405	2.500
Ours (QP independent)	0.476	0.568	0.655	0.778	0.965	1.341	2.500

Table 1. Comparison of bits per pixel (BPP) on Kodak dataset when recompressing JPEG images of different quality levels (QP).

4. Details about comparison with learned lossless methods

We reproduce the multi-scale model following the instructions in the original paper, and our reproduced model achieves bits per sub-pixel (BPSP) of 3.942 (computed by negative log-likelihood) on ImageNet64 dataset, which is slightly better than BPSP 3.96 presented in the original paper.

We also reproduce IDF based on the source code released by the authors. Nevertheless, the original IDF can only accept input with fixed size due to the logistic mixture model (LMM) used for latent prior z_L (L is the level of flows). To make IDF resolution-adaptive, we replace the LMM with the univariate non-parametric density model used in [1]. Our reproduced model achieves BPSP of 3.879 (computed by negative log-likelihood) on ImageNet64 dataset, slightly better than BPSP of 3.90 given in the original paper.

As stated in the main text, these two methods are designed for RGB 4:4:4 input, so we convert JPEG 4:2:0 input data to RGB 4:4:4 or YCbCr 4:4:4 by upsampling Cb and Cr components. This upsampling operation increases resolution and may cause unfair comparison. Consequently, we also carry out experiments with JPEG 4:4:4 source format and convert it to RGB 4:4:4, YCbCr 4:4:4 and DCT 4:4:4 as model input, which ensures a fair comparison. We make our method suitable for JPEG 4:4:4 images by removing the downsampling and upsampling of Y component in CFM and CPSM. The full experiment settings are given in Tab. 2. We use Pillow library to read RGB values from the source JPEG images, and YCbCr values are converted from RGB values.

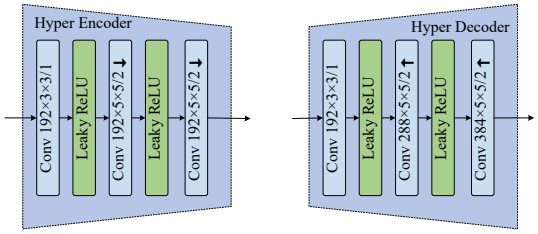
For IDF and multi-scale model using DCT coefficients as input, we do not adopt the DCT coefficients rearrangement proposed in the main text to avoid enormous architecture modification (this is because they are originally designed for thin input format like RGB values with only 3 channels). To deal with DCT 4:2:0 input, we reshape Y component to 4 channels by space-to-depth operation and then concatenate these 4 channels with Cb and Cr components to form 6-channel inputs with $\frac{1}{4}$ of their original resolutions. Meanwhile, both of the methods need to be adjusted slightly to fit this kind of input. For multi-scale model, the number of input channels is changed from 3 to 6, and autoregression over RGB channels is disabled. For IDF, we start from the original IDF and increase the number of input channels to 6.

For all of these experiments, we use Adam optimizer and MultiStepLR scheduler. We use the same training set and use Kodak for evaluation, and the quality level is 75. Other settings and experiment results are presented in Tab. 2. For both JPEG 4:2:0 and JPEG 4:4:4 images, our method outperforms other methods on all input formats by a large margin. It is worth noting that though for JPEG 4:4:4 images, our method for compressing Cb and Cr components is relatively simple, the performance is still superior. And it is obvious that lossless image compression methods designed for PNG do not perform well on recompressing JPEG 4:2:0 and JPEG 4:4:4 images.

5. More architecture details

5.1. Hyper-network

Fig. 3 shows the architecture details for the hyper-network mentioned in the main text.



(a) Hyper Encoder (b) Hyper Decoder

Figure 3. Detailed architecture of Hyper Encoder and Hyper Decoder.

5.2. Cross-color Cases

Fig. 4, Fig. 5 and Fig. 6 show the architectures for the three cases of cross-color entropy model respectively.

Source format	Model	Input format	Crop size	Learning rate	Batch size	Epoch	Milestones	Gamma	BPP
JPEG 4:2:0	IDF	RGB 4:4:4	64×64	1e-4	64	3000	250, 500, 750	0.1	6.964
		YCbCr 4:4:4							6.183
		DCT 4:2:0	256×256		32	1000	150		1.994
	Multi-scale	RGB 4:4:4	256×256	2e-4	64	6000	3000	0.5	4.398
		YCbCr 4:4:4							3.984
		DCT 4:2:0		2e-5					1.674
	Ours	DCT 4:2:0	256×256	1e-4	16	2000	1500	0.1	**0.965**
JPEG 4:4:4	IDF	RGB 4:4:4	64×64	1e-4	64	3000	250, 500, 750	0.1	7.059
		YCbCr 4:4:4							6.362
		DCT 4:4:4							5.875
	Multi-scale	RGB 4:4:4	256×256	2e-4	64	6000	3000	0.5	4.604
		YCbCr 4:4:4							4.079
		DCT 4:4:4		2e-5		3000			2.600
	Ours	DCT 4:4:4	256×256	1e-4	16	2000	1500	0.1	**1.122**

Table 2. Experiment settings and results on Kodak dataset for multi-scale and IDF models. Milestones and gamma are parameters of MultiStepLR in PyTorch. All models are trained and evaluated with QP 75.

5.3. Variants of MLCC model

This section shows architecture details of the three variants of MLCC (*i.e*. **Only Outer Channel** in Fig. 8, **Only Inner Channel** in Fig. 10, and **Column-to-row** in Fig. 12) and their corresponding context models (Fig. 7, Fig. 9, Fig. 11) mentioned in Sec. 4.3 in the main text.

References

[1] 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*, 2018. 2

[2] Julian Minguillon and Jaume Pujol. Jpeg standard uniform quantization error modeling with applications to sequential and progressive operation modes. *Journal of Electronic Imaging*, 10(2):475 – 485, 2001. 1

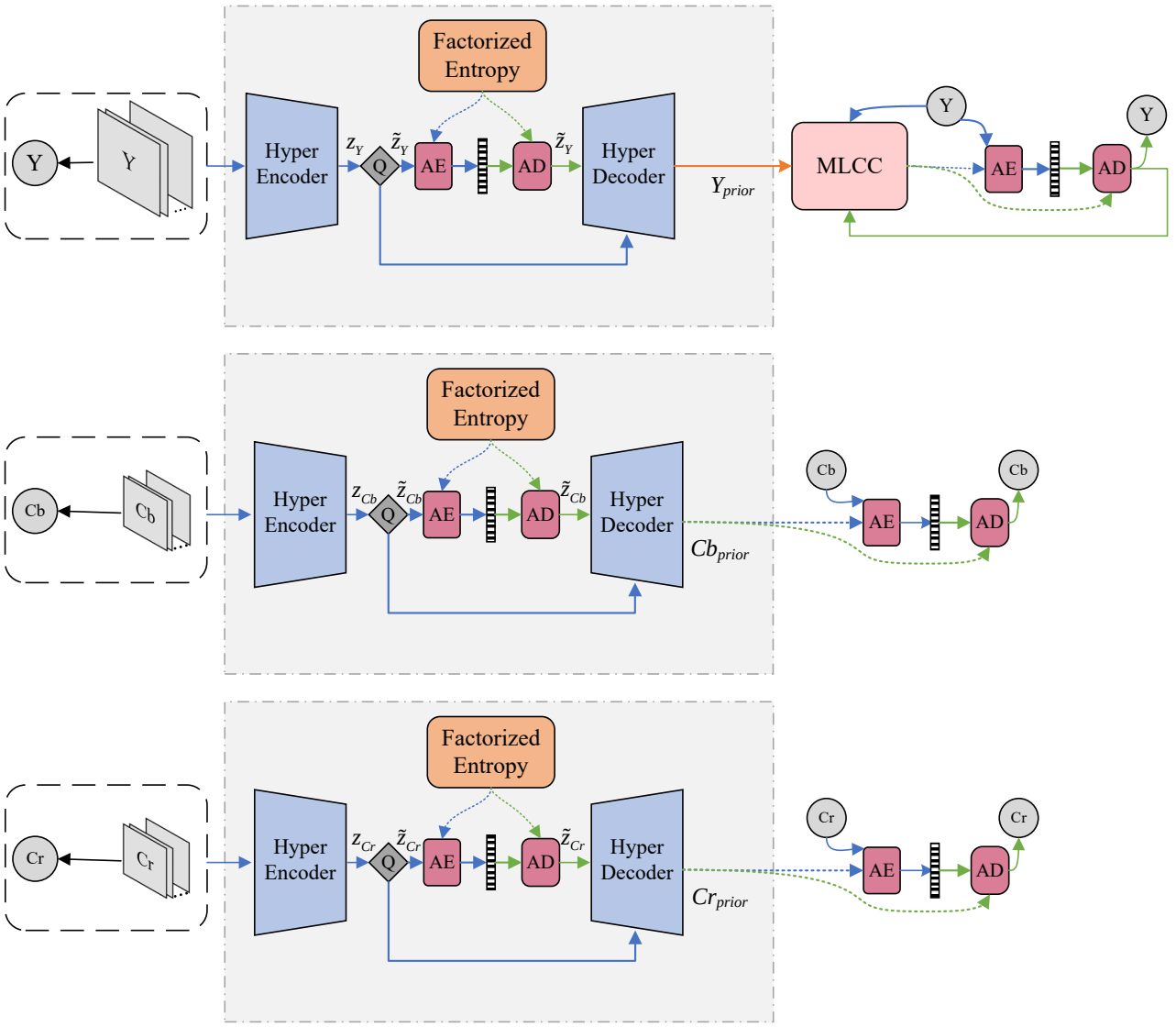


Figure 4. Architecture details of **Cross-color case 1**. Blue and green lines indicate data-flow for encoding and decoding respectively, orange lines are shared.

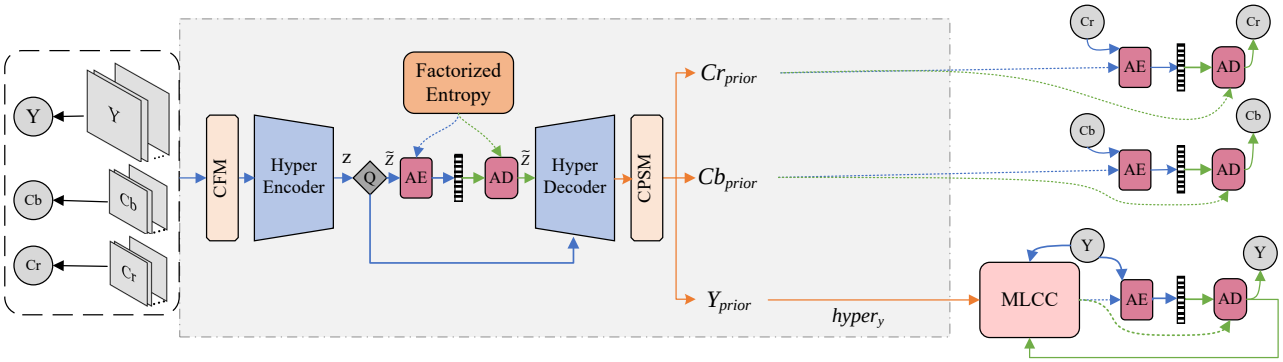


Figure 5. Architecture details of **Cross-color case 2**. Blue and green lines indicate data-flow for encoding and decoding respectively, orange lines are shared.

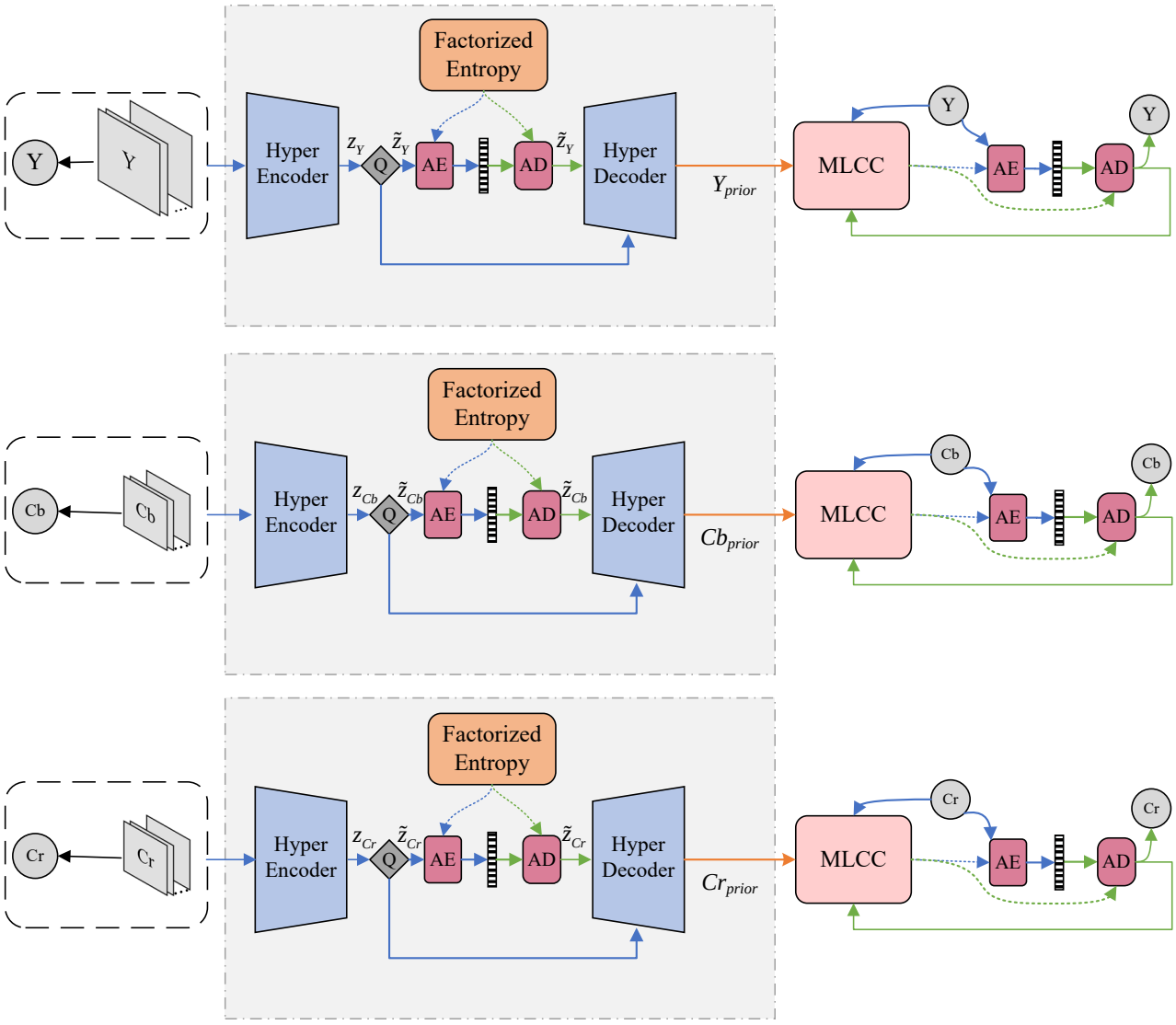


Figure 6. Architecture details of **Cross-color case 3**. Blue and green lines indicate data-flow for encoding and decoding respectively, orange lines are shared.

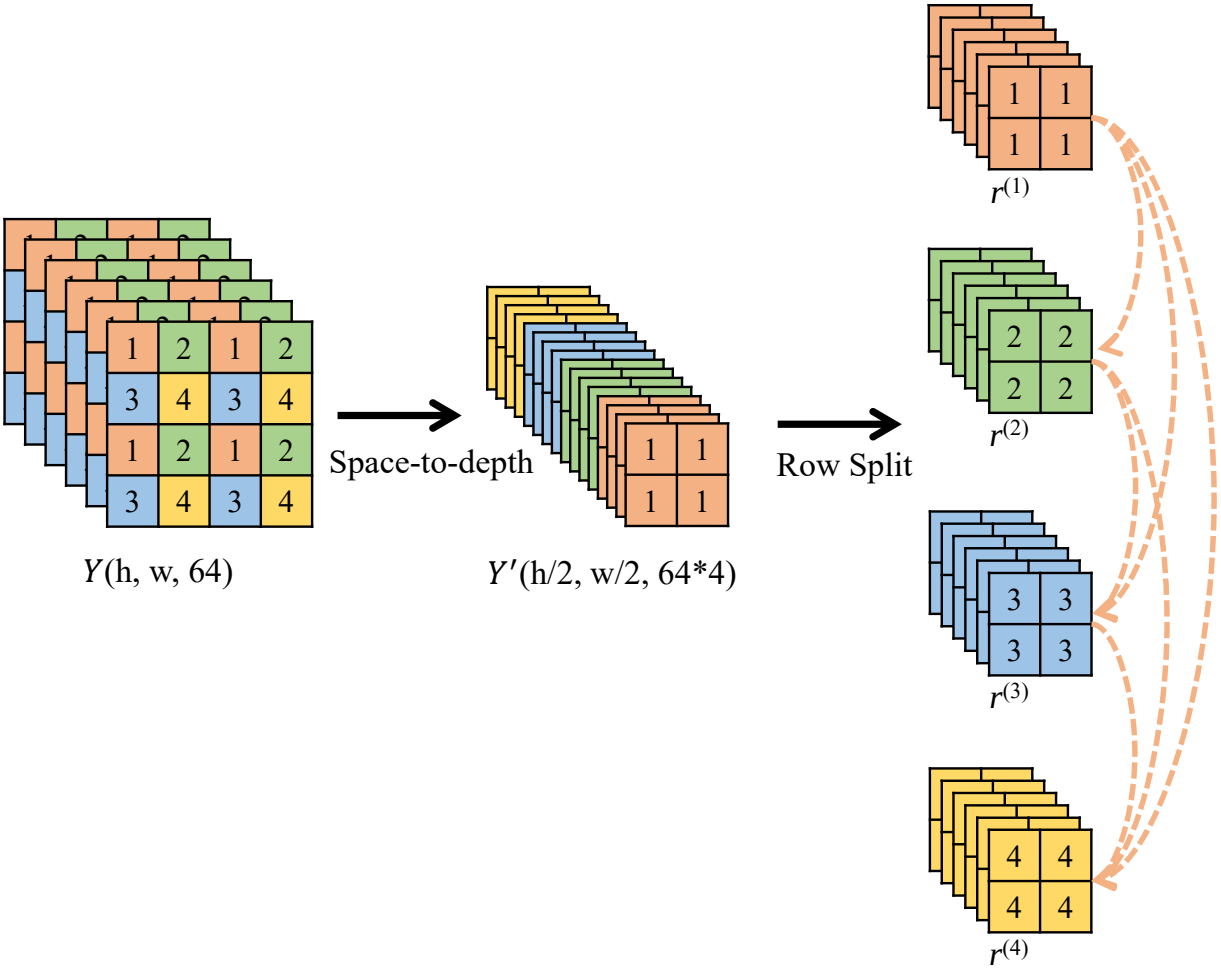


Figure 7. Matrix context modeling method with only row split. Solid arrows indicate data operation and dotted arrows denote conditional relationships.

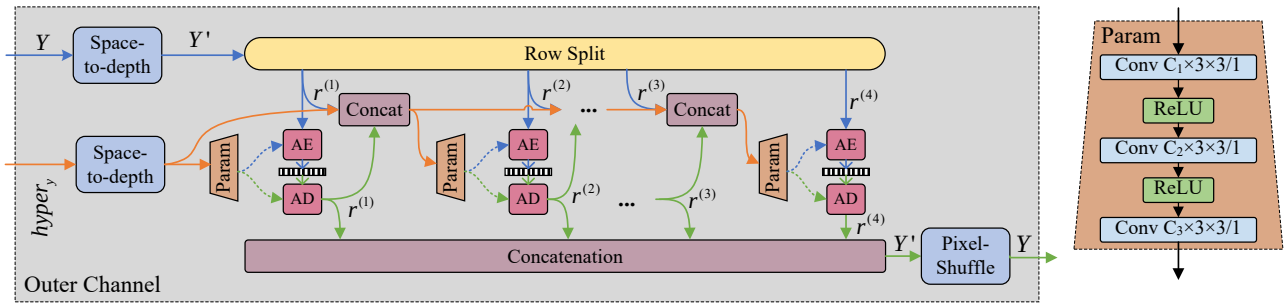


Figure 8. Architecture details of **Only Outer Channel**. Fig. 7 is its corresponding context model. Letting n, m representing the channel number of input tensor and next slice to be modeled respectively, C_1, C_2 and C_3 are decided by $C_1 = n - d, C_2 = n - 2 * d, C_3 = 2 * m, d = (n - 2 * m)//3$. Blue and green lines indicate data-flow for encoding and decoding respectively, orange lines are shared.

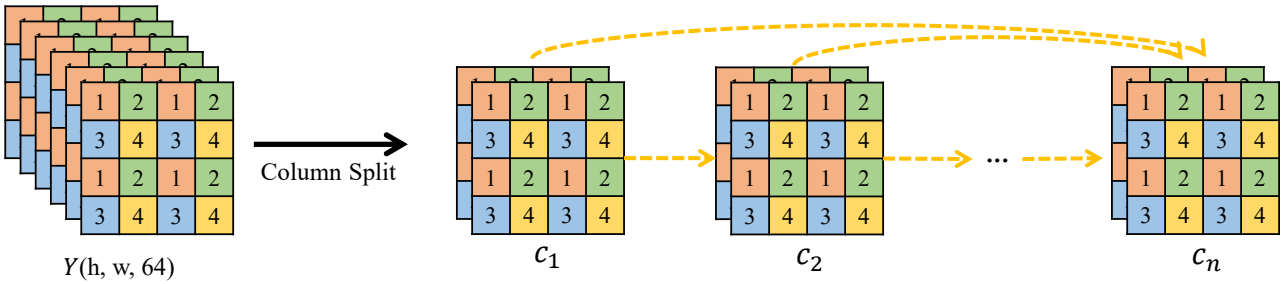


Figure 9. Matrix context modeling method with only column split.Solid arrows indicate data operation and dotted arrows denote conditional relationships.

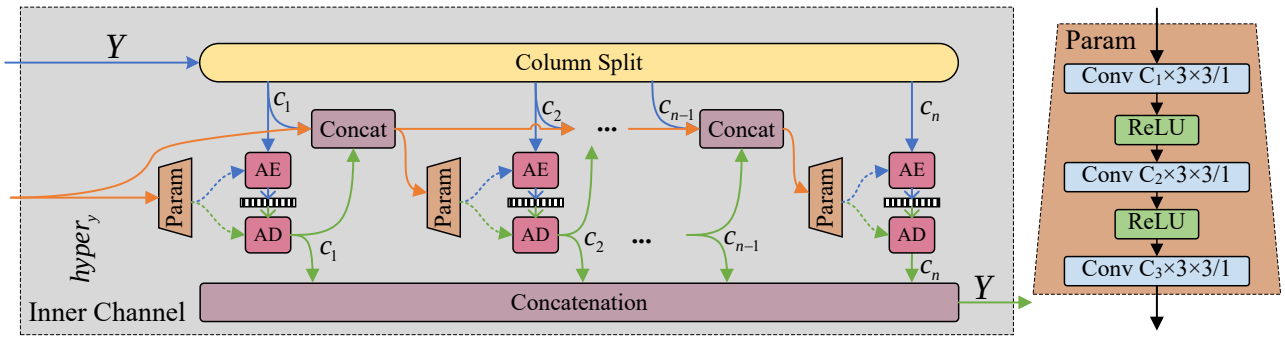


Figure 10. Architecture details of **Only Inner Channel**. Fig. 9 is its corresponding context model. Letting n, m representing the channel number of input tensor and next slice to be modeled respectively, C_1, C_2 and C_3 are decided by $C_1 = n - d, C_2 = n - 2 * d, C_3 = 2 * m, d = (n - 2 * m)//3$. Blue and green lines indicate data-flow for encoding and decoding respectively, orange lines are shared.

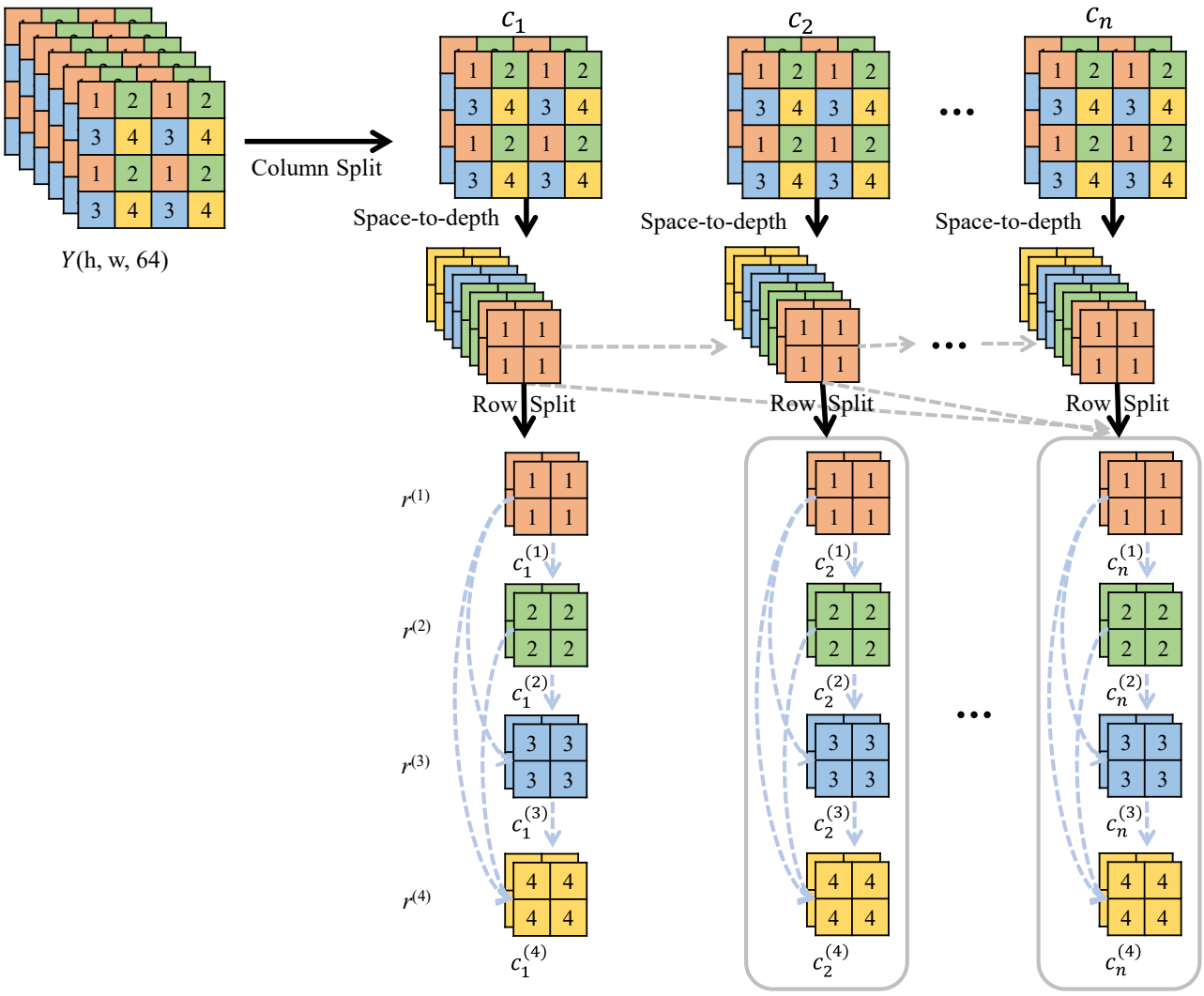


Figure 11. Matrix context modeling method in column-to-row manner.Solid arrows indicate data operation and dotted arrows denote conditional relationships. Light grey and light blue dotted arrows align with Outer Channel and Inner Channel in Fig. 12, respectively.

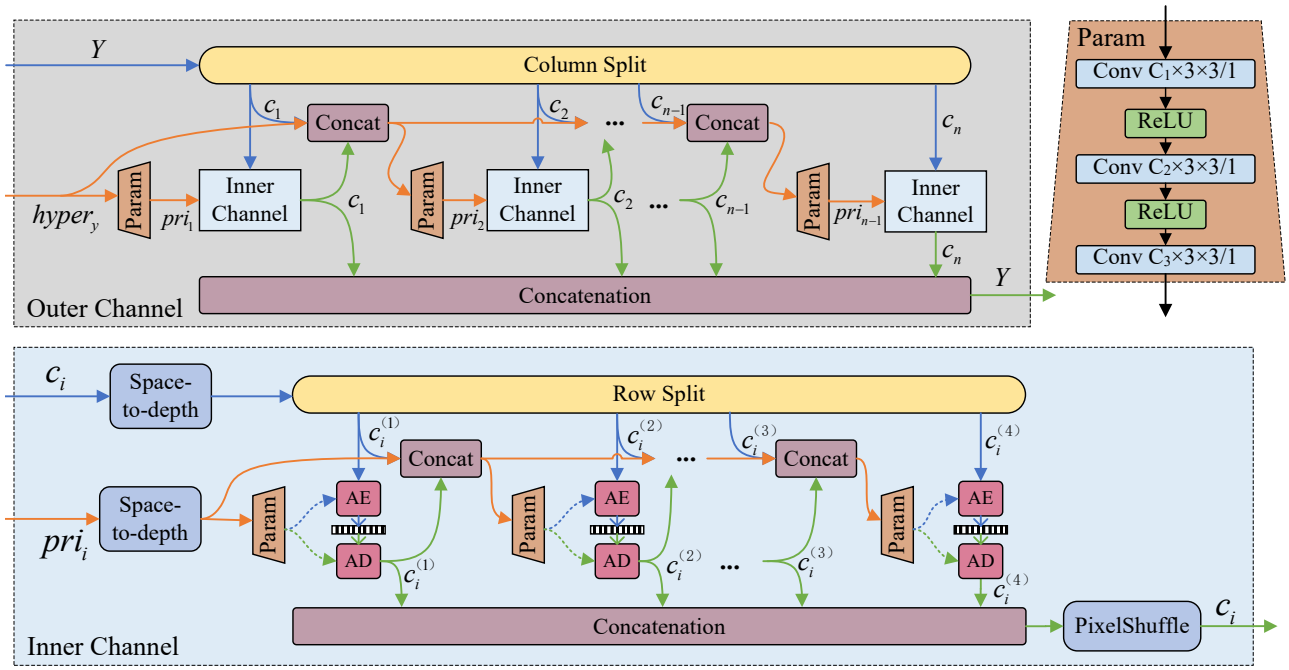


Figure 12. Architecture details of **column-to-row**. Fig. 11 is its corresponding context model. Letting n, m representing the channel number of input tensor and next slice to be modeled respectively, C_1, C_2 and C_3 are decided by $C_1 = n - d, C_2 = n - 2 * d, C_3 = 2 * m, d = (n - 2 * m)//3$. Blue and green lines indicate data-flow for encoding and decoding respectively, orange lines are shared.