Appendix – AdaInt: Learning Adaptive Intervals for 3D Lookup Tables on Real-time Image Enhancement

Canqian Yang[1,4*], Meiguang Jin[2*], Xu Jia[3], Yi Xu[1,4†], Ying Chen[2]
[1]MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University [2]Alibaba Group
[3]Dalian University of Technology [4]Chongqing Research Institute, Shanghai Jiao Tong University
{charles.young, xuyi}@sjtu.edu.cn
xjia@dlut.edu.cn {meiguang.jmg, chenying.ailab}@alibaba-inc.com

A. Motivation of Using 3D LUT

Inspired by the practice in the image signal processor (ISP) [5], we adopt the 3D LUT instead of a more general mapping function (*e.g.*, a multi-layer perceptron (MLP)) to model the $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ color transform function due to the consideration of *efficiency and expressiveness*. Specifically, if a *static* MLP is used, it is easy to learn an over smooth transform since the network needs to adapt to all images in the dataset. Therefore, the model might suffer from its limited expressiveness and diversity among images. Adopting an additional network for predicting the parameters of an MLP to enable image-adaptiveness is another alternative and has been investigated in CSRNet [3]. However, since an MLP requires a cascade of several linear and nonlinear sub-operations to increase the model capability, it suffers from a higher computational burden, especially on high-resolution inputs, as shown in Table 2 in the paper (**48 times** over our AdaInt on 4K resolution). Instead, our 3D LUT with learned adaptive intervals is able to achieve enhanced expressiveness and image-adaptiveness while still presenting high efficiency by directly recording and retrieving a complex transform via simple lookup and interpolation operations.

B. Details of AiLUT-Transform

B.1. Forward

Given an input image denoted as $X \in [0,1]^{3\times H\times W}$, suppose an image-adaptive 3D LUT is learned, its output values and sampling coordinates are abbreviated as $T \in [0,1]^{3\times N_s\times N_s\times N_s}$ and $\hat{P} \in [0,1]^{3\times N_s}$, respectively. N_s is the number of sampling coordinates along each lattice dimension. The AiLUT-Transform takes all of X, T, and $\hat{P}$ as inputs, and produces the transformed output image $\hat{Y}$:

$$\hat{Y} = \text{AiLUT-Transform}(X, T, \hat{P}). \tag{1}$$

The transform consists of a lookup step and an interpolation step. The former locates each input pixel of X in the LUT, whereas the latter computes the corresponding output using the values of eight nearest neighborhood sampling points in T. Suppose an input query pixel x is given, which is composed of three color components $\{x_r, x_g, x_b\}$, the transform on x is performed in the following steps.

The Lookup Step The transform first locates the query pixel x into a lattice cell closed by 8 nearest neighborhood sampling points. These 8 points (or called vertices) are determined by 6 coordinates obtained by finding both the left and right neighbors $x_c^0, x_c^1 \in \hat{P}$ $(c = r, g, b)$ along each lattice dimension, and their corresponding indices in the LUT $e_c^0, e_c^1 \in \mathbb{I}_0^{N_s-1}$. In a standard LUT transform where the 3D lattice is constructed with equal intervals $\Delta = 1/(N_s - 1)$, this can be easily achieved:

$$e_c^0 = \lfloor \frac{x_c}{\Delta} \rfloor, e_c^1 = e_c^0 + 1,$$
$$x_c^0 = e_c^0 \Delta, x_c^1 = e_c^1 \Delta,$$

where $\lfloor \cdot \rfloor$ is the floor function. It is obvious that the $\hat{P}$ is not needed in the standard LUT transform, so it is always omitted in existing implementations. However, in AiLUT-Transform, since the intervals are no longer a constant value, a searching algorithm on $\hat{P}$ is required to find x_c^0, x_c^1 and e_c^0, e_c^1, which can be formulated as:

$$\begin{aligned} x_c^0, e_c^0 &= \max\{\hat{P}_{c,s}, s | \hat{P}_{c,s} \leq x_c, s \in \mathbb{I}_0^{N_s-1}\}, \\ x_c^1, e_c^1 &= \min\{\hat{P}_{c,s}, s | \hat{P}_{c,s} > x_c, s \in \mathbb{I}_0^{N_s-1}\}. \end{aligned} \tag{2}$$

Thanks to the bounded and monotone increasing properties of the learned sampling coordinates stored in $\hat{P}$ (see Section 3.2 in the paper), Equation (2) can be easily achieved by introducing a binary search.

Once the 6 indices $(e_r^0, e_r^1; e_g^0, e_g^1; e_b^0, e_b^1)$ for the nearest neighborhood vertices in the LUT are determined, the corresponding output color values of the 8 neighbors in T can

*Equal Contribution †Corresponding Author
Work partially done during an internship of C. Yang at Alibaba Group.

be defined as:

$$\tilde{T}_{:,i,j,k} = T[:, e_r^i, e_g^j, e_b^k],\ i,j,k \in \{0,1\}. \tag{3}$$

The Interpolation Step After querying 8 adjacent vertices in the LUT for query pixel x, the transformed result can be obtained by applying trilinear interpolation, including the following steps:

- Compute the normalized offsets between the input color and the 8 neighbors:

$$x_r^d = \frac{x_r - x_r^0}{x_r^1 - x_r^0}, x_g^d = \frac{x_g - x_g^0}{x_g^1 - x_g^0}, x_b^d = \frac{x_b - x_b^0}{x_b^1 - x_b^0}. \tag{4}$$

- Compute the interpolation weights, which are defined as the partial volume diagonally opposite the vertices:

$$\begin{aligned} V_{i,j,k} =& (x_r^d)^i (1 - x_r^d)^{1-i} \\ & \cdot (x_g^d)^j (1 - x_g^d)^{1-j} \\ & \cdot (x_b^d)^k (1 - x_b^d)^{1-k}. \end{aligned} \tag{5}$$

- Compute the linear combination of $V_{i,j,k}$ and $\tilde{T}_{:,i,j,k}$:

$$\hat{y} = \sum_{i,j,k \in \{0,1\}} V_{i,j,k} \cdot \tilde{T}_{:,i,j,k}. \tag{6}$$

B.2. Backpropagation

In our method, both T and $\hat{P}$ are learned by neural networks to enable more flexible 3D LUT with adaption to different image content. The challenge exists in the absence of interpolation implementation in popular deep learning libraries (such as PyTorch [9]) that can provide gradients to $\hat{P}$. To this end, we provide a new implementation that can compute the gradients with respect to $\hat{P}$ during backpropagation to enable the end-to-end learning of AdaInt. The core is to define the partial derivatives of x_c^0, x_c^1:

$$\begin{aligned} \frac{\partial \hat{y}}{\partial x_c^0} &= \sum_{i,j,k \in \{0,1\}} \tilde{T}_{:,i,j,k} \frac{\partial V_{i,j,k}}{\partial x_c^d} \frac{\partial x_c^d}{\partial x_c^0}, \\ \frac{\partial \hat{y}}{\partial x_c^1} &= \sum_{i,j,k \in \{0,1\}} \tilde{T}_{:,i,j,k} \frac{\partial V_{i,j,k}}{\partial x_c^d} \frac{\partial x_c^d}{\partial x_c^1}. \end{aligned} \tag{7}$$

where,

$$\begin{aligned} \frac{\partial V_{i,j,k}}{\partial x_r^d} &= (-1)^{1-i} \cdot (x_g^d)^j (1 - x_g^d)^{1-j} \cdot (x_b^d)^k (1 - x_b^d)^{1-k}, \\ \frac{\partial V_{i,j,k}}{\partial x_g^d} &= (x_r^d)^i (1 - x_r^d)^{1-i} \cdot (-1)^{1-j} \cdot (x_b^d)^k (1 - x_b^d)^{1-k}, \\ \frac{\partial V_{i,j,k}}{\partial x_b^d} &= (x_r^d)^i (1 - x_r^d)^{1-i} \cdot (x_g^d)^j (1 - x_g^d)^{1-j} \cdot (-1)^{1-k}, \end{aligned} \tag{8}$$

and,

$$\begin{aligned} \frac{\partial x_r^d}{\partial x_r^0} &= -\frac{1 - x_r^d}{x_r^1 - x_r^0}, \frac{\partial x_r^d}{\partial x_r^1} = -\frac{x_r^d}{x_r^1 - x_r^0}, \\ \frac{\partial x_g^d}{\partial x_g^0} &= -\frac{1 - x_g^d}{x_g^1 - x_g^0}, \frac{\partial x_g^d}{\partial x_g^1} = -\frac{x_g^d}{x_g^1 - x_g^0}, \\ \frac{\partial x_b^d}{\partial x_b^0} &= -\frac{1 - x_b^d}{x_b^1 - x_b^0}, \frac{\partial x_b^d}{\partial x_b^1} = -\frac{x_b^d}{x_b^1 - x_b^0}. \end{aligned} \tag{9}$$

Note that since x_c^0, x_c^1 are elements in $\hat{P}$, Equations (7) to (9) indeed define the partial derivatives with respect to $\hat{P}$, allowing the loss gradients to flow back to the sampling coordinates, and therefore back to the sampling intervals and any preceding neural network modules. The AiLUT-Transform works on each pixel independently, so it is highly parallelizable, especially on GPUs.

C. Experimental Details

Both datasets used in our work, i.e., the MIT-Adobe FiveK [1] and PPR10K [7], are publicly released, free of charge for research use, and contain no personally identifiable information. We conducted our experiments based on PyTorch 1.8.1 [9] and the MMEditing toolbox (v0.11.0, under Apache 2.0 license) [1]. The proposed AiLUT-Transform is complied as a PyTorch CUDA extension using CUDA 10.2. In the following section, we provide more experimental details about these two datasets.

C.1. FiveK

Datasets Preprocessing Experiments on the FiveK dataset are conducted on two different resolutions (480p and 4K) and two different applications (retouching and tone mapping). For retouching and tone mapping on 480p, we directly download the dataset released by [11] [2] for model training and testing. To obtain the original 4K images, we follow [11] to download the original dataset [3] and use the Adobe Lightroom software to pre-process the RAW images. We use the same training/testing splits as [11] to divide the dataset into 4500 training pairs and 500 testing pairs. For the sake of research reproducibility, we provide the split files and the detailed instruction of data pre-processing at https://github.com/ImCharlesY/AdaInt.

We train all our models on the 480p resolution. During training, input images (8-bit sRGB for retouching and 16-bit CIE XYZ for tone mapping) are normalized to $[0, 1]$ for unified processing and are randomly augmented to mitigate overfitting and facilitate performance. The augmentations include random ratio cropping, random horizontal flipping,

[1] https://github.com/open-mmlab/mmediting
[2] https://github.com/HuiZeng/Image-Adaptive-3DLUT
[3] https://data.csail.mit.edu/graphics/fivek

Id	Layer	Output Feature Shape
0	Bilinear Resize	$3 \times 256 \times 256$
1	Conv3x3, LeakyReLU	$16 \times 128 \times 128$
2	InstanceNorm	$16 \times 128 \times 128$
3	Conv3x3, LeakyReLU	$32 \times 64 \times 64$
4	InstanceNorm	$32 \times 64 \times 64$
5	Conv3x3, LeakyReLU	$64 \times 32 \times 32$
6	InstanceNorm	$64 \times 32 \times 32$
7	Conv3x3, LeakyReLU	$128 \times 16 \times 16$
8	InstanceNorm	$128 \times 16 \times 16$
9	Conv3x3, LeakyReLU	$128 \times 8 \times 8$
10	Dropout (0.5)	$128 \times 8 \times 8$
11	AveragePooling	$128 \times 2 \times 2$
12	Reshape	512

Table 1. Network architecture of the backbone module (the mapping f in the paper) on the FiveK [1] dataset.

and random color jittering. The trained models can be directly applied to the original 4K resolution without performance drop (see Table 2 in the paper). This protocol significantly speeds up the training stage and also demonstrates the flexibility and scalability of our method.

Model Architectures The model architectures employed on the FiveK experiments are listed in Tables 1 to 3. For the backbone network (mapping f), we directly adopt the 5-layer [4] backbone in [11]. The mapping h uses a cascade of two fully-connected (FC) layers, which reformulates the implementation in [11], *i.e.*, the first FC layer (mapping h_0) is responsible for predicting M input-dependent fusing weights, whereas the second FC layer (mapping h_1) encodes the parameters of M basis 3D LUT. For the instantiation of mapping g, a single FC layer is adopted. We follow [11] to initialize the parameters of f and h. As for the proposed AdaInt module (mapping g), we initialize its weights and bias to all 0s and 1s, which makes the predicted sampling intervals start from a uniform state, hence stabilizing the training of AdaInt.

Training Details We train our models for 400 epochs with a fixed learning rate of 1×10^{-4} using the standard Adam optimizer [6]. The mini-batch size is set to 1. To further stabilize the AdaInt learning and facilitate the learning of more output color values in the 3D lattice, we decay the learning rate of g by a factor of 0.1 and freeze its parameters in the first 5 training epochs.

C.2. PPR10K

Datasets Preprocessing Experiments on the PPR10K dataset are conducted on the 360p resolution for the photo retouching application. The official training/testing $(8,875 : 2,286)$ splits are adopted. During training, images augmented by the dataset creator are used as our inputs. Please refer to [7] for more details. Besides the pre-augmentations, we further apply some commonly-used data augmentation methods such as random ratio cropping and random horizontal flipping. Since the dataset is relatively larger-scale than the FiveK dataset, images are also resized into a unified 448×448 resolution to enable mini-batch processing for speeding up the training stage.

[4] The term "5-layer" corresponds to 5 convolutional layers.

Id	Layer	Output Feature Shape
0	FC	M
1	FC	$3N_s^3$
2	Reshape	$3 \times N_s \times N_s \times N_s$

Table 2. Network architecture of the mapping h in the paper. "FC" denotes the fully-connected layer.

Id	Layer	Output Feature Shape
0	FC	$3(N_s - 1)$
1	Reshape	$3 \times (N_s - 1)$
2	Softmax	$3 \times (N_s - 1)$
3	Cumsum	$3 \times (N_s - 1)$
4	ZeroPad	$3 \times N_s$

Table 3. Network architecture of the mapping g in the paper. "FC" denotes the fully-connected layer, and "Cumsum" is the operation to conduct cumulative summation.

Model Architectures For experiments on the PPR10K dataset, the mapping g and h share identical architectures (Tables 2 and 3) with those on the FiveK dataset. For the backbone network (the mapping f), we follow [7] to adopt the ResNet-18 [4] (with a preceding bilinear layer resizing images into 224×224 resolution) for a fair comparison. The initializations of g and h are the same as those in the FiveK experiments, whereas the ResNet-18 backbone is initialized using ImageNet [2] pretrained weights.

Training Details We train our models for 200 epochs with a mini-batch size of 16 to speed up the training stage on such a large-scale dataset. For other settings, they are kept consistent with those in the FiveK experiments.

D. More Visualization of AdaInt

To better investigate the behavior of AdaInt, inspired by [8], we introduce the per-color-channel *Accumulative Error Histogram (AEH)*, which, to some extent, can indicate locations in the color ranges where more dense sampling points are required.

The AEH bins the intensity changes between input and target images according to the values of the input pixels.

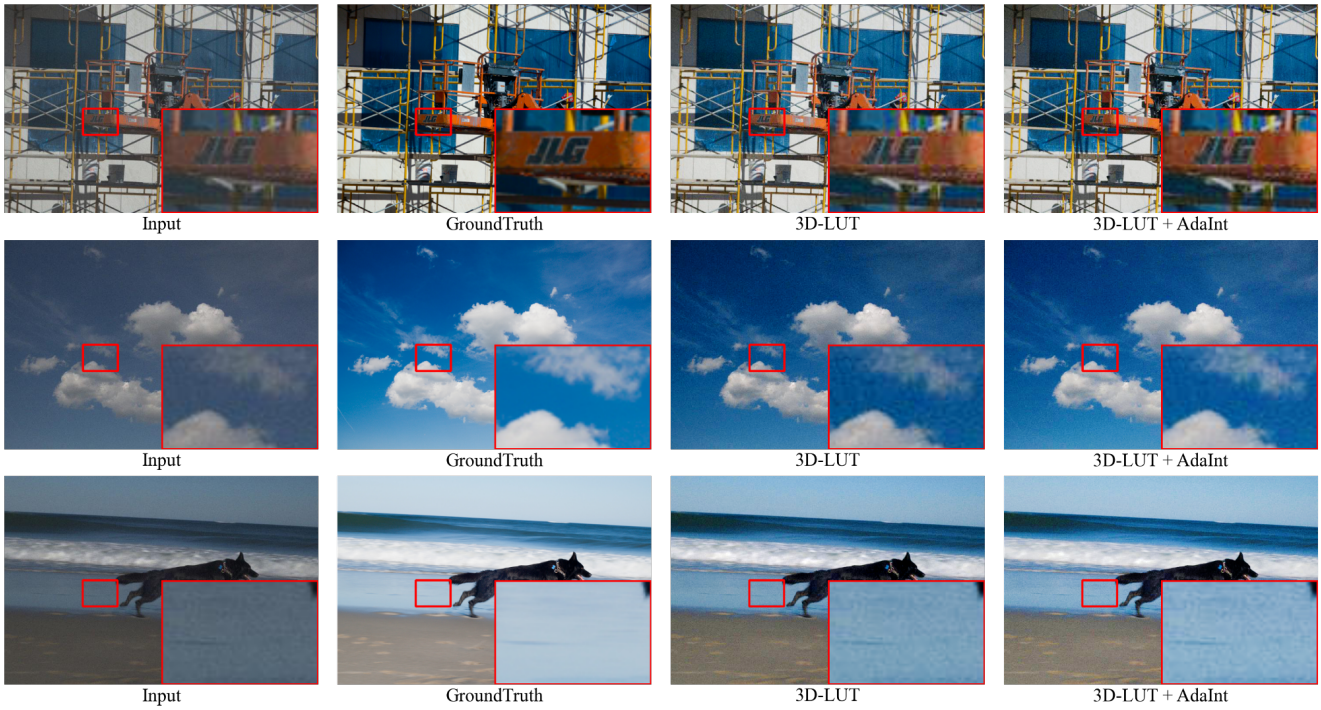

Figure 1. Qualitative results on noisy images for **photo retouching** on the **FiveK** dataset (480p) [11]. Best viewed on screen.

Therefore, it indicates the intensity change required for each input pixel value to transform the image. Given an input image $X \in [0,1]^{3\times H\times W}$ and the corresponding groundtruth image $Y \in [0,1]^{3\times H\times W}$, the AEH is constructed by first calculating the error map D between X and Y:

$$D_{c,h,w} = (X_{c,h,w} - Y_{c,h,w})^2, \tag{10}$$

where $c = \{r, g, b\}$, $h \in \mathbb{I}_0^{H-1}$, and $w \in \mathbb{I}_0^{W-1}$. Before computing the histogram, we divide the input color range $[0, 1]$ for a single channel uniformly into N_{bin} bins and assign each input pixel into one of them according to its value at the corresponding color channel:

$$\Omega_c^k = \{X_{c,h,w} | k\Delta \leq X_{c,h,w} < (k+1)\Delta\}, \tag{11}$$

where $\Delta = 1/N_{\text{bin}}$ and Ω_c^k is the k-th bin ($k \in \mathbb{I}_0^{N_{\text{bin}}-1}$) for channel c. The normalized error histogram H_c for channel c is then computed as:

$$H_c[k] = \frac{1}{\Theta_c} \sum_{X_{c,h,w} \in \Omega_c^k} D_{c,h,w}, \tag{12}$$

where Θ_c is a normalization term such that $\sum_k H_c[k] = 1$. The final per-color-channel AEH can be easily derived by applying accumulative summation on H_c. In this work, we set $N_{\text{bin}} = 1000$. The AEH can be visualized as a 1D curve as shown in Figure 2. The partial curve exhibiting higher curvature shows the color range requires a more substantial intensity change. Though not precise, the AEH indicates regions in the color range where the non-linearity is most prominent, and hence more sampling points are required.

We provide more visualization in Figure 2 of the learned sampling coordinates and the corresponding 3D LUTs for different images from the PPR10K dataset. It is clearly shown that the learned sampling coordinates are inclined to locate densely in the color range where the AEH exhibits high curvature, which suggests that our AdaInt is able to capture the complexity of the underlying optimal color transform with the adaption to the image content and achieve a more optimal sampling point allocation.

E. Additional Qualitative Comparisons

In this section, we provide additional visual comparisons on the FiveK (4K) dataset in Figure 3 and on the PPR10K (360p) dataset in Figure 4.

F. Real-time Performance on 8K Images

We also follow the protocol in [11] to resize the images into 8K (7680×4320) resolution and measure the GPU inference time of both the baseline [11] and our method on a V100 GPU. The baseline can execute at a speed of 2.36 ms per 8K image, while AdaInt only increases the runtime to 2.54 ms, which still exceeds the requirement of real-time processing by a large margin. Note that the high efficiency of our method mainly owns to the characteristic that the computational cost of the CNN network is fixed as it operates on a downsampled, fixed-resolution version of the

input, and the proposed AiLUT-Transform can be highly parallelized via customized CUDA code.

G. Limitation

As discussed in Section 5 in the paper, the proposed method is based on pixel-wise mapping and thus may fail to tackle heavy noise existing in the input. For verification, we randomly select three images from the FiveK dataset [1] and add Gaussian noise (with a standard deviation of $\sigma = 0.02$) to them. As shown in Figure 1, both the baseline [11] and our method could not eliminate the added noise due to the lack of filtering capability. Constructing pixel-wise LUTs [10] might be a possible solution but still requires careful investigation to prevent substantial increases in memory and computational costs.

H. Broader Impacts

The proposed method mimics the retouching style of human annotators based on learned statistics of the training dataset and as such will reflect biases in those data, including ones with negative societal impacts. Besides, there is no guarantee that the transformed colors by the method will please everybody due to the subjectiveness of the image/photo enhancement task. A possible mitigation strategy is to concern the preference of specific downstream users and retrains the method accordingly.

References

[1] Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. In *The Twenty-Fourth IEEE Conference on Computer Vision and Pattern Recognition*, 2011. 2, 3, 5, 7, 8

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 3

[3] Jingwen He, Yihao Liu, Yu Qiao, and Chao Dong. Conditional sequential modulation for efficient global image retouching. *arXiv preprint arXiv:2009.10390*, 2020. 1

[4] 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. 3

[5] Hakki Can Karaimer and Michael S Brown. A software platform for manipulating the camera imaging pipeline. In *European Conference on Computer Vision*, pages 429–444. Springer, 2016. 1

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 3

[7] Jie Liang, Hui Zeng, Miaomiao Cui, Xuansong Xie, and Lei Zhang. Ppr10k: A large-scale portrait photo retouching dataset with human-region mask and group-level consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021. 2, 3, 6

[8] Hai Ting Lin, Zheng Lu, Seon Joo Kim, and Michael S Brown. Nonuniform lattice regression for modeling the camera imaging pipeline. In *European Conference on Computer Vision*, pages 556–568. Springer, 2012. 3, 6

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 2

[10] Tao Wang, Yong Li, Jingyang Peng, Yipeng Ma, Xian Wang, Fenglong Song, and Youliang Yan. Real-time image enhancer via learnable spatial-aware 3d lookup tables. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2471–2480, 2021. 5

[11] Hui Zeng, Jianrui Cai, Lida Li, Zisheng Cao, and Lei Zhang. Learning image-adaptive 3d lookup tables for high performance photo enhancement in real-time. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 2, 3, 4, 5

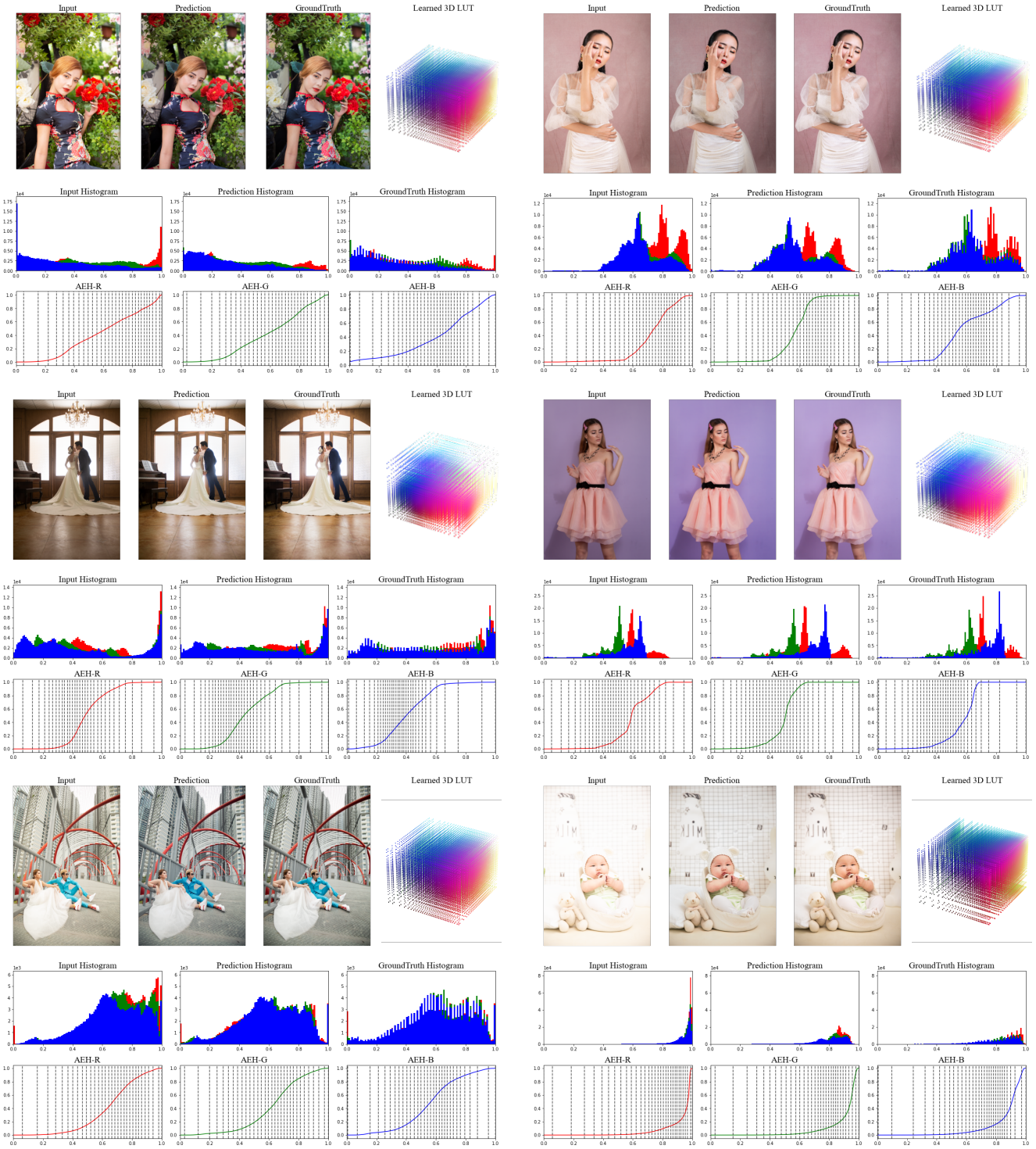

Figure 2. Illustration of the learned sampling coordinates and the corresponding 3D LUTs for **photo retouching** on the **PPR10K** dataset (360p) [7]. The bottom row visualizes the learned sampling coordinates on the so-called per-color-channel *Accumulative Error Histogram (AEH)* [8]. The regions in the AEH exhibiting high curvature indicate wherein more sampling points are needed. Best viewed on screen.

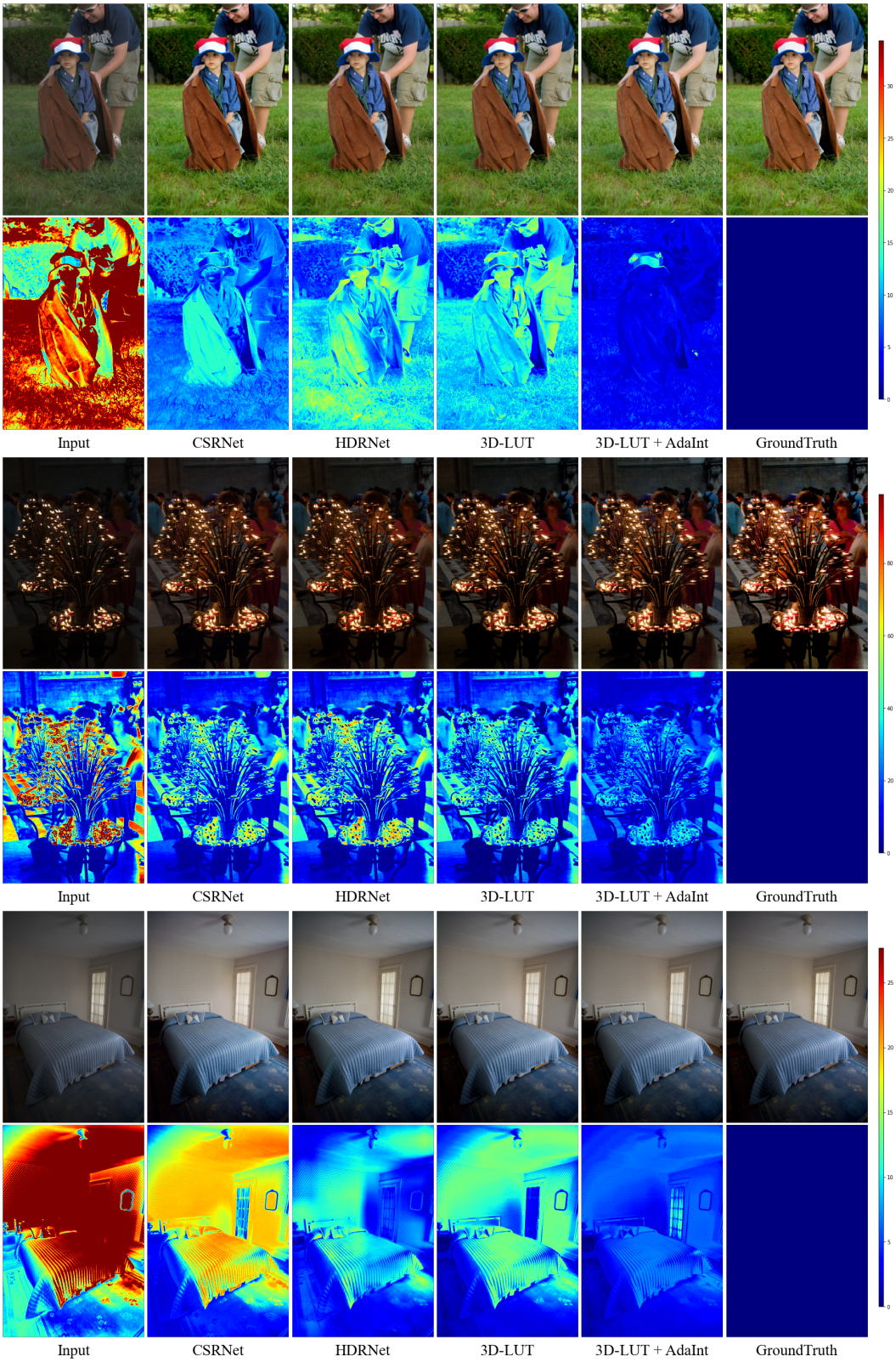

Figure 3. Additional qualitative comparisons of different methods for **photo retouching** on the **FiveK** dataset (4K) [1], and the corresponding error maps. Best viewed in zoom in.

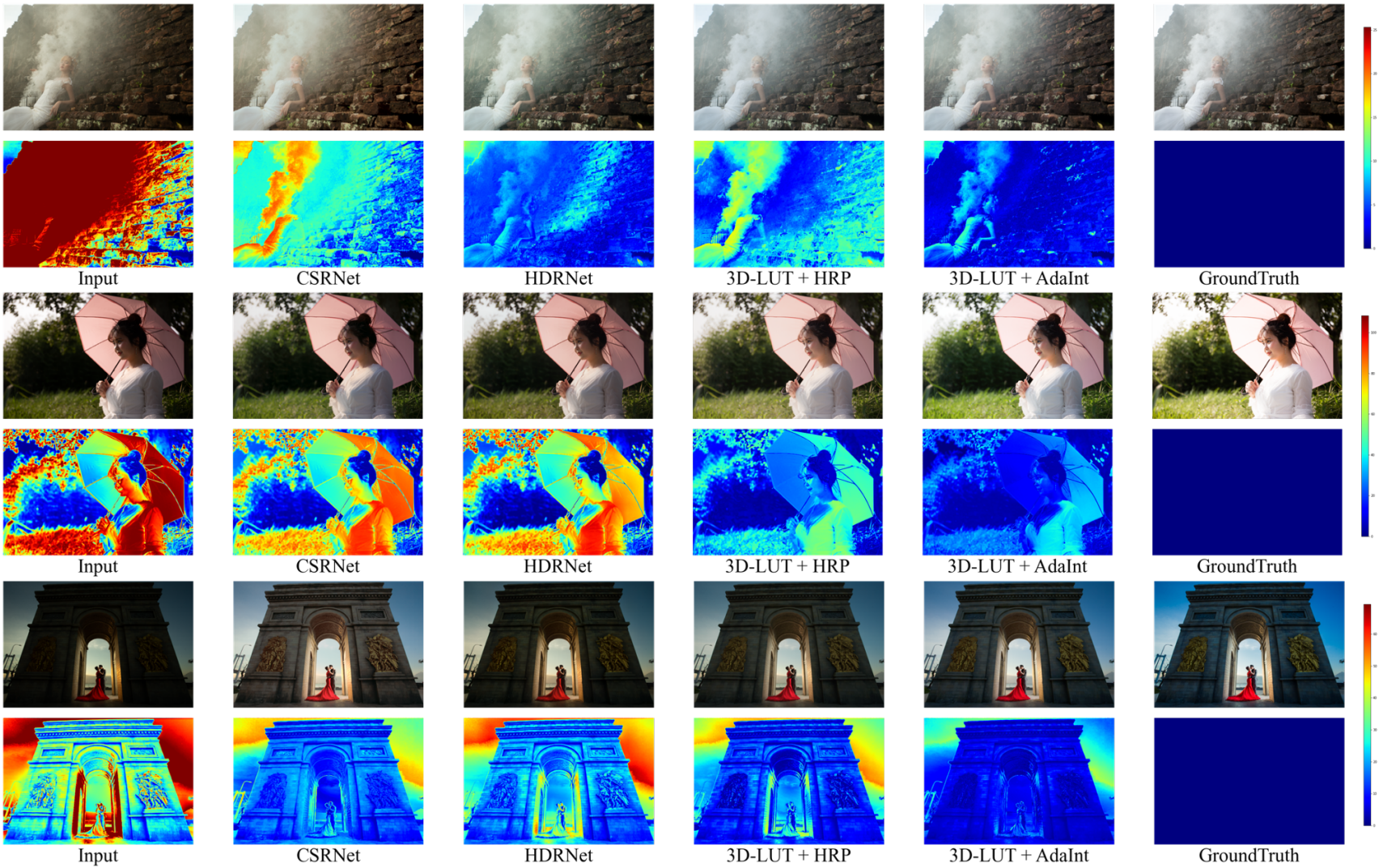

Figure 4. Additional qualitative comparisons of different methods for **photo retouching** on the **PPR10K** dataset (360p) [1], and the corresponding error maps. Best viewed in zoom in.