Location-aware Single Image Reflection Removal
Supplementary Material

Zheng Dong[1] Ke Xu[2,4] Yin Yang[3] Hujun Bao[1] Weiwei Xu*[1] Rynson W.H. Lau[4]

[1]State Key Lab of CAD&CG, Zhejiang University [2]Shanghai Jiao Tong University
[3]Clemson University [4]City University of Hong Kong

zhengdong@zju.edu.cn, kkangwing@gmail.com, yin5@clemson.edu, {bao, xww}@cad.zju.edu.cn, Rynson.Lau@cityu.edu.hk

1. Network Architecture.

In Tab. 3 and Tab. 4, we report the detailed parameters of the SE-ResBlock and CBAM-ResBlock used in our network respectively. Moreover, Tab. 5 illustrates the overall structure of our network model. Our network can work well when the width and height of the input images $\mathbf{I}, \hat{\mathbf{T}}_i, \hat{\mathbf{R}}_i, \hat{\mathbf{C}}_i$ are an integer multiple of 8 since the minimal resolution of the features that appear in our network is $1/8$ of input images. Setting the resolution to an integer multiple of 8 can avoid the misalignment solutions caused by convolution and down-sample operations since our network is based on a recurrent structure. Besides, images of resolution up to 1576 in width and height can be input into our network when inferring on an Nvidia Geforce RTX 2080 Ti GPU.

2. Data Augmentation

In this section, we describe the details of three types of operations to augment synthetic training images, which include (1) adding gray-scale training images (Gray); (2) adding ghosting effects (Ghost); (3) increasing the range of Gaussian kernel size (IKR). Such operations can generate training images that cover more real-world reflection types, as shown in Fig. 1. For clarity, we denote the ground-truth (GT) transmission and reflection layer images before gamma correction by $\tilde{\mathbf{T}}$, $\tilde{\mathbf{R}}$, and their corresponding gamma-corrected layer images by $\mathbf{T}, \mathbf{R}$.

For sampled transmission and reflection layer images used to synthesize $\mathbf{I}$, the data augmentation procedure first applies inverse gamma correction to $\mathbf{T}$, $\mathbf{R}$ and get $\tilde{\mathbf{T}}$, $\tilde{\mathbf{R}}$, then chooses whether to add ghosting effects or use a gray-scale version of $\mathbf{I}$ with a probability of 0.3 or 0.2 respectively. The usage of gray-scale reflection images is an efficient acceleration strategy in the early training of reflection removal networks. The ghosting effect is used to simulate the reflective effect of thick glasses [1, 6], which can be formulated as: $\tilde{\mathbf{R}} = \beta \cdot \mathbf{H} \otimes \mathbf{K} \otimes \tilde{\mathbf{R}}$, where $\mathbf{K}$ is a Gaussian kernel; $\mathbf{H}$ is a two-pulse kernel(size=13, $peak_1 = 1 - \sqrt{\alpha}$ $peak_2 = \sqrt{\alpha} - \alpha$), $\alpha \in [0.8, 1]$; β is a reflection rate map of size $256 \times 256 \times 3$ cropped from a $560 \times 560 \times 3$ Gaussian map (The standard deviation is set to 0.3) [5]; $\otimes$ is the convolution operation. If the ghosting effect is not chosen in the first step, all the images will be blurred with a Gaussian filter [5, 9, 13]. The filter kernel size is in the range of $[2, 5]$, and we linearly increase the kernel range from $[2, 5]$ to $[0.9, 6.0]$ to cover more variations of the blurring degree of the reflection images (IKR). Specifically, the augmented reflection layer $\tilde{\mathbf{R}}$ is generated by $\tilde{\mathbf{R}} = \beta \cdot \mathbf{K} \otimes \tilde{\mathbf{R}}$, the reflection rate β is the same parameter as in adding a ghosting effect. We compose the processed reflection images with transmission images to synthesize $\tilde{\mathbf{I}}$ by the alpha blending model, *i.e.*, $\tilde{\mathbf{I}} = \alpha \cdot \tilde{\mathbf{T}} + \tilde{\mathbf{R}}$ in [13], and then perform gamma correction for $\tilde{\mathbf{T}}, \tilde{\mathbf{R}}$ and the composited image $\tilde{\mathbf{I}}$ to get triples $\{\mathbf{T}, \mathbf{R}, \mathbf{I}\}$. For sampled real-world images, the above operations are not performed.

Finally, we apply random rotation ($90°, 180°, 270°$) and flipping to all the loaded images to obtain the final training images as the network's inputs. After training for 90 epochs, we reduce the Gaussian kernel size to $[0.5, 3.5]$ and the learning rate to $3e^{-5}$ for the fine-tuning of our network on the synthesized images with strong reflections.

In Tab. 1, we show the ablation study results for our data augmentation by retraining the model with different combinations of data augmentation operations. It can be seen that our data augmentation (IKR + Gray + Ghost) can enhance the performance of the proposed model. It achieves better average PSNR/SSIM scores, 24.058/0.894, for three datasets, compared to the results without data augmentation (PSNR/SSIM: 23.767/0.882).

Data Augmentation	*SIR*^{2} [7]		Zhang *et al.*[13]		Li *et al.*[5]	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
w/o IKR, Ghost, and Gray	23.851	0.889	22.776	0.806	22.850	0.801
+ IKR	24.018	0.896	22.803	0.808	22.247	0.798
IKR + Gray	23.949	0.889	22.167	0.802	22.403	**0.833**
IKR + Gray + Ghost	**24.117**	**0.901**	**23.338**	**0.812**	**23.451**	0.808

Table 1. Data augmentation ablation study.

*Corresponding author.

Figure 1. Examples of our synthetic images. Top: the transmission layers. Bottom: the synthetic images with reflections. First two columns: adding ghosting effect. Middle two columns: the blurred reflections. Last two columns: adding reflected highlights.

3. Kernels learned by LKI and RKI.

Tab. 6 and Tab. 7 list the kernel weights learned by Laplacian kernel initialization (LKI) and random kernel initialization (RKI). Fig. 7 and Fig. 8 show the maps computed with the two learned kernels. It can be seen that the learned Laplacian kernel weights with LKI is more efficient to suppress low-frequency reflections, compared to the learned weights with RKI. Our paper's ablation study also verifies that the fine-tuned Laplacian weights can improve the SIRR performance since they are updated according to the reflection dataset.

4. Evaluation of the β parameter for loss $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$.

When using $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$ to supervise the learning of RCMap in our model, β is the parameter that is used to compute the GT binary masks, $\mathbf{C}_{gt}$, to indicate reflection-dominated regions. That is, pixels in $\mathbf{C}_{gt}$ whose values are greater than β will be regarded as reflection-dominated pixels. Therefore, their mask value will be set to 1, otherwise 0. To determine the appropriate value for β that leads to the best performance for $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$, we test the choices of β on SIR^2 [7], Zhang *et al.* [13] and Li *et al.* [5] datasets with a set of values, *i.e.*, $\beta = \{0.15, 0.2, 0.25, 0.3, 0.35, 0.4\}$. Fig. 2 illustrates the results. According to the PSNR/SSIM scores, we choose the parameter β to be 0.3.

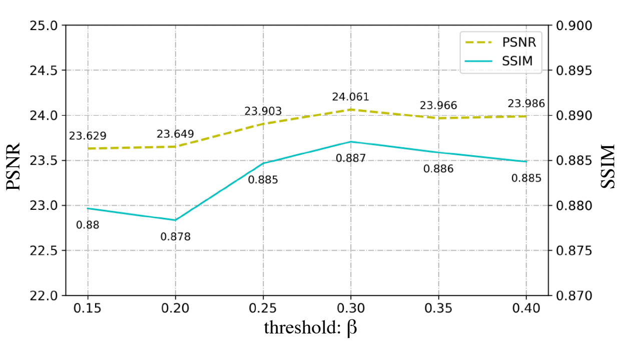


Figure 2. Evaluation of β for $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$.

5. Evaluation of the detection accuracy and the protection of non-reflection regions.

Detection accuracy. To evaluate the detection accuracy of the RDM, we first convert a predicted RCMap into a binary mask $\mathbf{M}$ using a threshold $\gamma = \mathbf{C}_{\min} + 0.5 * (\mathbf{C}_{\max} - \mathbf{C}_{\min})$, and then calculate the pixel-wise accuracy between the mask $\mathbf{M}$ and the GT RCMap $\mathbf{C}_{gt}$. The $\mathbf{C}_{gt}$ is computed in the same manner as defined in the Sec. 5 of our main paper, where the threshold is set to be $\max\{\bar{\mathbf{R}}^{gray}_{\min} + \beta * (\bar{\mathbf{R}}^{gray}_{\max} - \bar{\mathbf{R}}^{gray}_{\min}), \beta)$, and the parameter β is chosen to be 0.3 according to Sec. 4. The mean accuracy obtained is 0.818 on the SIR^2 dataset. Note that we compute the accuracy for both reflection-dominated regions (pixels with value 1 in the binary masks) and transmission-dominated regions (pixels with value 0 in the binary masks).

As shown in Fig. 3, $\mathbf{C}_{gt}$ can be used to indicate the reflection-dominated regions. That is why we use it to calculate the detection accuracy. However, we also observe that $\mathbf{C}_{gt}$ might sometimes mislabel the regions with clearly visible reflections as transmission-dominated regions (the third row in Fig. 3). We hypothesize that it is why the detection accuracy calculated above is not high. We treat how to figure out a good way to define GT reflection-dominated regions that match reflection-dominated regions perceived by human eyes as future work.

(a) Input (b) Ground-truth $\mathbf{T}$ (c) $\mathbf{C}_{gt}$: $\|\mathbf{I} - \mathbf{T}\|$

Figure 3. Calculated GT RCMaps used in detection accuracy evaluation.

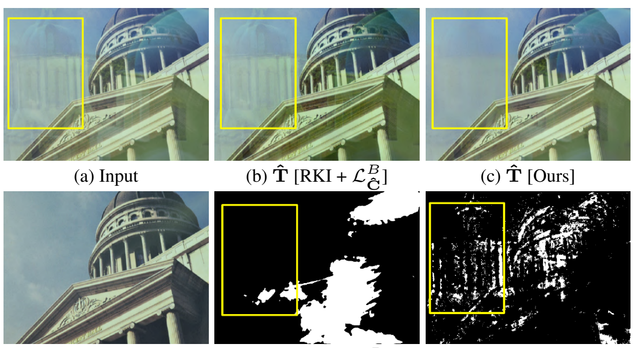
Protection of non-reflection regions. We observe that our predicted RCMaps can help to protect the non-reflection regions. We also use the $\mathbf{C}_{gt}$ computed above to evaluate the protection ability of the proposed method for non-reflection regions. Specifically, we first apply $\mathbf{C}_{gt}$ to the GT transmission layer image $\mathbf{T}$ and the predicted transmission image $\hat{\mathbf{T}}$ to obtain non-reflection regions separately, *i.e.* the regions that contain pixels with mask value 0, and then calculate the PNSR/SSIM scores there. The results are shown in the Tab. 2. It can be seen that our network can achieve higher PSNR/SSIM scores for the image contents in non-reflected regions.

Index(↑)	ERRNet [9]	Kim *et al.* [4]	IBCLN [5]	Ours
PSNR	26.617	26.840	26.418	**27.707**
SSIM	0.935	0.931	0.941	**0.955**

Table 2. Quantitative results on non-reflection protection.

6. Comparison with the variant RKI & $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$.

The qualitative comparison result is shown in Fig. 4. In this testing case, our network trained using the composition loss $\mathcal{L}_{\hat{\mathbf{C}}}$ can detect the reflection regions and remove reflections (yellow box) better than the network trained with the variant RKI & $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$.



(a) Input (b) $\hat{\mathbf{T}}$ [RKI + $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$] (c) $\hat{\mathbf{T}}$ [Ours]

(d) Ground-truth $\mathbf{T}$ (e) $\mathbf{M}$ [RKI + $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$] (f) $\mathbf{M}$ [Ours]

Figure 4. Comparison with the variant using RKI & $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$. $\mathbf{M}$: The binary mask generated according to the RCMap generated by our network trained with RKI & $\mathcal{L}^{B}_{\hat{\mathbf{C}}}$ and $\mathcal{L}_{\hat{\mathbf{C}}}$ respectively.

7. Evaluate the iteration number N.

To investigate whether increasing N will lead to better results, we conduct an experiment on N using our network. On SIR^2, the PSNR/SSIM values are 23.662/0.891 (N=1), 23.780/0.894 (N=2), 24.095/0.893 (N=4), 24.005/0.895 (N=5), 23.504/0.889 (N=6), respectively. These values are lower than (24.117/0.901) when N=3. We hypothesize that the false removal of $\mathbf{T}$ may accumulate when increasing N (*i.e.*, $N > 3$) and affects the performance. We would like to explore how to design a metric that is more consistent with perceptions to supervise our network in the future.

8. Evaluate network performance when R covers entire image.

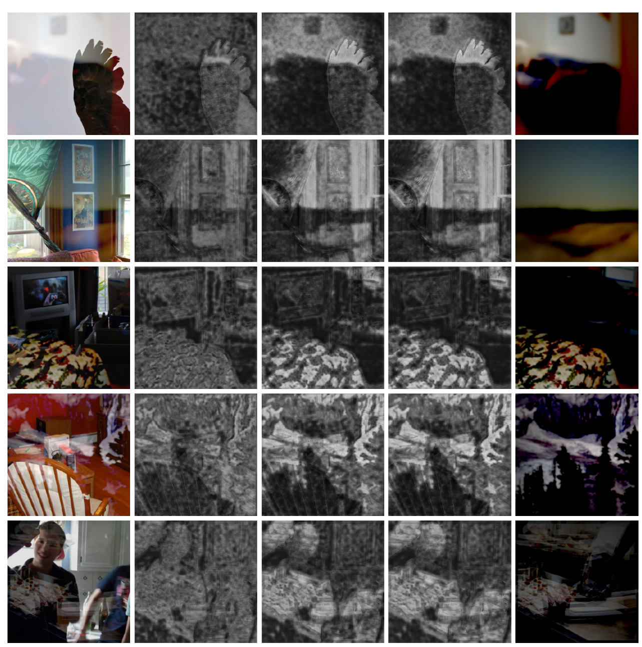
As shown in Fig. 5, we randomly generate three images with reflections spreading almost all over the images. As shown in the first two rows, our predicted $\mathbf{T}$ contains most details from GT $\mathbf{T}$, even if reflection-dominated regions are not clearly labeled in RCMaps. It verifies that the second stage of our network can efficiently remove weak reflections without accurate RCMaps as input. The last row of Fig. 5 shows an extreme case where the reflection dominates the content of the image. Our model can still remove most reflections from the original image when $\mathbf{R}$ is clearly visible in $\mathbf{I}$. However, reflections from the regions (yellow box) where RCMap fails to detect are not removed.

(a) $\mathbf{I}$ (b) Ours $\mathbf{R}$ (c) Ours $\mathbf{T}$ (d) $\mathbf{T}$ (e) RCMap

Figure 5. SIRR results when $\mathbf{R}$ covers the whole image. (a) Images with reflections. (b) Predicted Reflections. (c) Predicted Transmissions. (d) Ground truth transmissions are shown in column. (e) Predicted RCMaps

9. More RCMap Results.

How the the predicted RCMaps is improved along with network iterations for 5 synthesized images is illustrated in Fig. 6.



(a) $\mathbf{I}$ (b) Iter = 1 (c) Iter = 2 (d) Iter = 3 (e) $\mathbf{R}$

Figure 6. More predicted RCMaps and the GT reflection layers.

10. More Iterative Refinement Results.

In Fig. 9 and Fig. 10, we show four groups of iterative refinement results of our recurrent network. The improved transmission layers: $\hat{\mathbf{T}}_i$, reflection layers: $\hat{\mathbf{R}}_i$, and reflection confidence maps: $\hat{\mathbf{C}}_i$ in iteration i are shown in the two figures respectively.

11. More Results on Benchmark Datasets.

From Fig. 11 to Fig. 13, we show more results of our method on the benchmark datasets. We also compare our method with the state-of-art SIRR methods, including Zhang *et al*. [13], BDN [12], RMNet [10], ERRNet [9], CoRRN [8], Kim *et al*. [4], and IBCLN [5].

12. Results on Internet Images.

Fig. 14 shows the SIRR results on the images photographed through glasses collected from Internet. While these pictures are taken in a variety of scenes, our method can efficiently reconstruct high-quality transmission layers.

References

[1] Zhixiang Chi, Xiaolin Wu, Xiao Shu, and Jinjin Gu. Single image reflection removal using deep encoder-decoder network. *arXiv preprint arXiv:1802.00094*, 2018.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[3] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7132–7141, 2018.

[4] Soomin Kim, Yuchi Huo, and Sung-Eui Yoon. Single image reflection removal with physically-based training images. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5164–5173, 2020.

[5] Chao Li, Yixiao Yang, Kun He, Stephen Lin, and John E Hopcroft. Single image reflection removal through cascaded refinement. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3565–3574, 2020.

[6] YiChang Shih, Dilip Krishnan, Fredo Durand, and William T Freeman. Reflection removal using ghosting cues. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3193–3201, 2015.

[7] R. Wan, B. Shi, L. Duan, A. Tan, and A. C. Kot. Benchmarking single-image reflection removal algorithms. In *Int. Conf. Comput. Vis.*, pages 3942–3950, 2017.

[8] Renjie Wan, Boxin Shi, Haoliang Li, Ling-Yu Duan, Ah-Hwee Tan, and Alex Kot Chichung. Corrn: Cooperative reflection removal network. *PAMI*, 42(12):2969–2982, 2020.

[9] Kaixuan Wei, Jiaolong Yang, Ying Fu, David Wipf, and Hua Huang. Single image reflection removal exploiting misaligned training data and network enhancements. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8178–8187, 2019.

[10] Qiang Wen, Yinjie Tan, Jing Qin, Wenxi Liu, Guoqiang Han, and Shengfeng He. Single image reflection removal beyond linearity. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3771–3779, 2019.

[11] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Eur. Conf. Comput. Vis.*, pages 3–19, 2018.

[12] Jie Yang, Dong Gong, Lingqiao Liu, and Qinfeng Shi. Seeing deeply and bidirectionally: A deep learning approach for single image reflection removal. In *Eur. Conf. Comput. Vis.*, pages 654–669, 2018.

[13] Xuaner Zhang, Ren Ng, and Qifeng Chen. Single image reflection separation with perceptual losses. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4786–4794, 2018.

Block name	Output size	Filter size or Setting
SE-ResBlock [3]		
Conv_1 + PReLU / ReLU	$H \times W \times C$	$3 \times 3, C$, stride=1
Conv_2	$H \times W \times C$	$3 \times 3, C$, stride=1
SE-Layer		
AdaptiveAvgPool2d + Reshape	C	Pooling output size: 1×1
Linear_1 + PReLU / ReLU	C / reduction	$C \times C/2$ (reduction=2)
Linear_2 + Sigmoid	C	$C/2 \times C$ (reduction=2)
Reshape	$1 \times 1 \times C$	None
Multiplication_1	$H \times W \times C$	Input: output of conv_2
Multiplication_2 + Add	$H \times W \times C$	multiply by res_scale (0.1) Input: feature before Conv_1

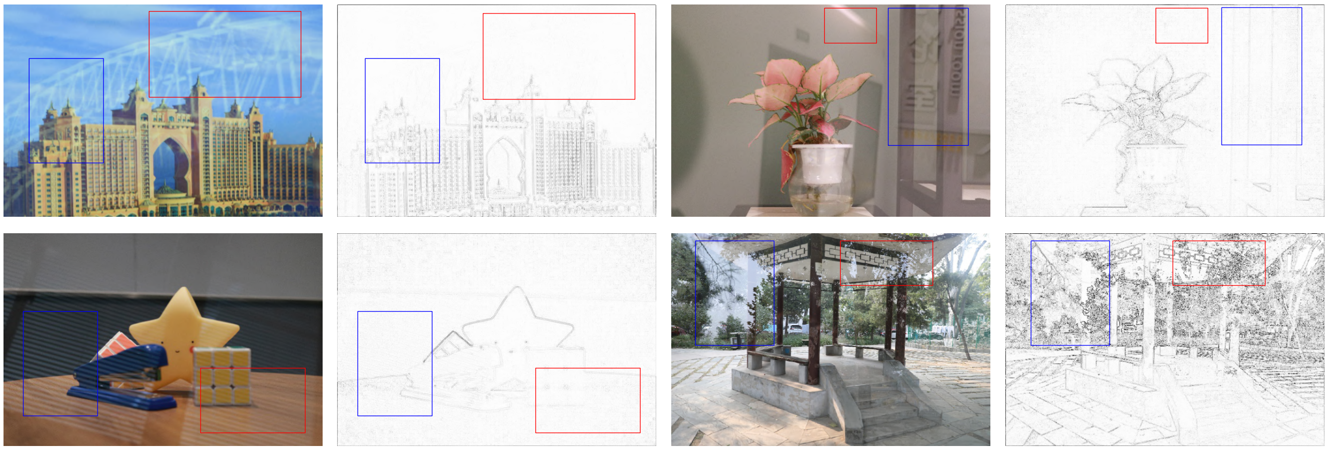
Table 3. The architecture and detailed parameters of SE-ResBlocks [3]. The dimension of the input feature map is $H \times W \times C$, where H and W are the height and width of the feature map, and C is the channel number. The activation function PReLU is used in RDM & TSM modules, while ReLU is used in the Image feature branch. The variable reduction is set to 2 in this table.

Block name	Output size	Filter size or Setting
CBAM-ResBlock [11]		
Conv_1 + ReLU	$H \times W \times C$	$3 \times 3, C$, stride=1
Conv_2	$H \times W \times C$	$3 \times 3, C$, stride=1
CBAM-Layer		
Channel-attention (Shared MLP)		
AdaptiveAvgPool2d	$1 \times 1 \times C$	Input: the output of Conv_2 Pooling output size: 1×1
Conv_2 + ReLU	$1 \times 1 \times (C/2)$	$1 \times 1, C/2$ (reduction=2), stride=1
Conv_3	$1 \times 1 \times C$	$1 \times 1, C$, stride=1
AdaptiveMaxPool2d	$1 \times 1 \times C$	Input: the output of Conv_2 Pooling output size: 1×1
Conv_2 + ReLU	$1 \times 1 \times (C/2)$	$1 \times 1, C/2$ (reduction=2), stride=1
Conv_3	$1 \times 1 \times C$	$1 \times 1, C$, stride=1
Add + Sigmoid_1	$1 \times 1 \times C$	Input: the two outputs of Conv_3
Spatial-attention		
Mean	$H \times W \times 1$	Input: the output of Conv_2 keep_dim=True
Max	$H \times W \times 1$	Input: the output of Conv_2 keep_dim=True
Concat + Conv_4 + Sigmoid_2	$H \times W \times 1$	Input: the output feature of Mean and Max 7×7, 1, stride=1
Multiplication_1	$H \times W \times C$	Input: the output feature of Sigmoid_1 Input: the feature before Conv_1
Multiplication_2	$H \times W \times C$	Input: the output feature of Multiplication_1 Input: the output feature of Sigmoid_2
Add	$H \times W \times C$	Input: the feature before Conv_1

Table 4. The architecture and detailed parameters of CBAM-ResBlocks [11]. The dimension of the input feature map is $H \times W \times C$. The variable reduction is set to 2 in this table.

Block name	**Output size**	**Filter size** or **Setting**
Concat	$H \times W \times 6$	Input: the original image $\mathbf{I}$ and the transmission layer $\hat{\mathbf{T}}_{i-1}$
Stage 1:		
Image feature branch:		
Conv_0 + ReLU	$H \times W \times 32$	$3 \times 3, 32$, stride=1
SE-ResBlock (ReLU, reduction=2) ×6, (Tab. 3)		
Reflection detection module (RDM):		
Multi-scale Laplacian submodule (MLSM):		
Laplacian_conv_0 + Concat	$H \times W \times 24$	Input: the Concat results $3 \times 3, 6$, stride=1
Conv_1 + PReLU	$H \times W \times 32$	$3 \times 3, 32$, stride=1
SE-ResBlock (PReLU, reduction=2) ×3, (Tab. 3)		
Conv_2 + ReLU	$H \times W \times 32$	$3 \times 3, 32$, stride=1
Conv_3 + Sigmoid_0	$H \times W \times 1$	$3 \times 3, 1$, stride=1; Output: RCMap $\hat{\mathbf{C}}_i$
Transmission-feature suppression module (TSM):		
SE-ResBlock (PReLU, reduction=2) ×3, (Tab. 3)		
Multiplication	$H \times W \times 32$	Input: $\hat{\mathbf{C}}_i$ from Sigmoid_0
Concat	$H \times W \times 64$	Input: output of Image feature branch
Conv2D LSTM (Input feature size: $H \times W \times 64$, output feature size: $H \times W \times 32$) [2]		
Conv_4 + ReLU	$H \times W \times 32$	$3 \times 3, 32$, stride=1
Conv_5 + ReLU_0	$H \times W \times 3$	$3 \times 3, 3$, stride=1; Output: $\hat{\mathbf{R}}_i$
Stage 2:		
Encoder:		
Concat	$H \times W \times 10$	Input: $\mathbf{I}$, $\hat{\mathbf{T}}_{i-1}$, $\hat{\mathbf{R}}_i$ from ReLU_0, $\hat{\mathbf{C}}_i$ from Sigmoid_0
Conv_6 + ReLU	$H \times W \times 64$	$3 \times 3, 64$, stride=1
CBAM-ResBlock (reduction=2) ×1, (Tab. 4)		
Conv_7 + ReLU	$(H/2) \times (W/2) \times 128$	$3 \times 3, 128$, stride=2
Conv_8 + ReLU	$(H/2) \times (W/2) \times 128$	$3 \times 3, 128$, stride=1
CBAM-ResBlock (reduction=4) ×2, (Tab. 4)		
Conv_9 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=2
Conv_10 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1
Conv_11 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1
CBAM-ResBlock (reduction=8) ×3, (Tab. 4)		
diConv_0 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1, dilation=2
diConv_1 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1, dilation=4
diConv_2 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1, dilation=8
diConv_3 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1, dilation=16
Conv_12 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1
Conv_13 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1
Decoder:		
Conv_15 + ReLU	$(H/4) \times (W/4) \times 3$	$3 \times 3, 3$, stride=1; Output: $\hat{\mathbf{T}}_i^{1/4}$
deConv_0 + AvgPool2d + ReLU	$(H/2) \times (W/2) \times 128$	$4 \times 4, 128$, stride=2 Input: the input of Conv_15
Add + Conv_16 + ReLU	$(H/2) \times (W/2) \times 128$	Input: skip connection from the input of Conv_9 $3 \times 3, 128$, stride=1
Conv_17 + ReLU	$(H/2) \times (W/2) \times 3$	$3 \times 3, 3$, stride=1; Output: $\hat{\mathbf{T}}_i^{1/2}$
deConv_1 + AvgPool2d + ReLU	$H \times W \times 64$	$4 \times 4, 64$, stride=2 Input: the input of Conv_17
Add + Conv_18 + ReLU	$H \times W \times 32$	Input: skip connection from the input of Conv_7 $3 \times 3, 32$, stride=1
Conv_19 + ReLU	$H \times W \times 3$	$3 \times 3, 3$, stride=1; Output: $\hat{\mathbf{T}}_i$

Table 5. The architecture and detailed parameters of our network. The dimension of the input RGB image is denoted as $H \times W \times 3$. Our Laplacian_conv_0 block operates with four scales(original scale's 1/1, 1/2, 1/4, 1/8) on the concatenation of images $\{\mathbf{I}, \hat{\mathbf{T}}_{i-1}\}$, and then upsamples their Laplacian maps to the original resolution. The learned kernel weights are described in Tab. 6.



Input | Inverse Laplacian map | Input | Inverse Laplacian map

Figure 7. The original input images and their inverse Laplacian maps computed with the fine-tuned Laplacian kernel $\mathcal{L}ap$ in Tab. 6. The inverse Laplacian maps are obtained in the same way as described in Fig.4 in the paper. We use the first three channels of $\mathcal{L}ap$ to process the original image $\mathbf{I}$, since the last three channels correspond to the processing of the transmission layer $\hat{\mathbf{T}}_i$. It can be seen that the fine-tuned Laplacian kernel $\mathcal{L}ap$ can also suppress the low-frequency reflection signals while preserving the high-frequency reflection signals. (blue and red boxes).

(a) Input | (b) M_1 | (c) M_2 | (d) M_3

Figure 8. Comparison between RKI and LKI on low-frequency reflection suppression. To ease the comparison, we use the same input image (a) in the second row in Fig. 4 in our paper. M_1: inverse RKI map after processing the input image by $\hat{k}_R$ in Tab. 7. M_2: inverse Laplacian map without gradient clipping (GC). M_3: inverse Laplacian map using both GC and LKI. Matrices in the left bottom of inverse maps show the learned kernel weights using RKI and LKI at channel index 0. Compared with the map M_1, the inverse Laplacian map M_2 and M_3 can efficiently suppress the low-frequency reflection signals.

Kernel name	Channels: Num / Index	Kernel weights
Original kernel: k_L		$[0,\ -1,\ 0;\ -1,\ 4,\ -1;\ 0,\ -1,\ 0]$
Fine-tuned Laplacian kernel: $\mathcal{L}ap$	6 / 0	$[\,1.811e^{-3},\ -1.0049,\ 6.135e^{-3};$ $-1.0111,\ 4.0027,\ -1.0060;$ $4.766e^{-3},\ -1.0013,\ 0.0102\,]$
	6 / 1	$[\,9.178e^{-4},\ -1.0004,\ 6.982e^{-3};$ $-1.0086,\ 4.0024,\ -0.9997;$ $-1.998e^{-3},\ -1.0023,\ 5.798e^{-3}\,]$
	6 / 2	$[\,-2.500e^{-3},\ -1.0017,\ 5.400e^{-3};$ $-1.0121,\ 4.0074,\ -1.0034;$ $2.136e^{-4},\ -0.9975,\ 8.423e^{-3}\,]$
	6 / 3	$[\,2.398e^{-4},\ -1.0142,\ -4.881e^{-4};$ $-1.0128,\ 4.0073,\ -1.0161;$ $6.679e^{-3},\ -1.0092,\ 3.787e^{-3}\,]$
	6 / 4	$[\,2.976e^{-3},\ -1.0024,\ 0.0127;$ $-1.0073,\ 3.9949,\ -1.0033;$ $9.467e^{-3},\ -1.0018,\ 0.0106\,]$
	6 / 5	$[\,3.622e^{-3},\ -1.0026,\ 8.311e^{-3};$ $-1.0078,\ 3.9986,\ -1.0030;$ $1.199e^{-3},\ -1.0036,\ 7.030e^{-3}\,]$

Table 6. Fine-tuned Laplacian kernel weights for $\mathcal{L}ap$. The Laplacian kernel weights are shared across scales but fine-tuned separately for R,G,B channels. Since we concatenate the original image and transmission layer as inputs, there are six sets of fine-tuned Laplacian kernel weights. It can be seen that the fine-tuned Laplacian kernel weights at each channel are close to the original Laplacian kernel weights.

Kernel name	Channels: Num / Index	Kernel weights
Kernel: $\hat{k}_R$ learned with RKI	6 / 0	$[\,0.0198,\ 0.0091,\ -0.0086;$ $0.0141,\ 0.0254,\ 0.0014;$ $0.0316,\ 0.0368,\ 0.0303\,]$
	6 / 1	$[\,-0.0219,\ 0.0577,\ 0.0752;$ $0.0240,\ 0.0390,\ 0.0793;$ $-0.0599,\ -0.0073,\ 0.0128\,]$
	6 / 2	$[\,-0.0833,\ -0.0414,\ -0.0029;$ $-0.0161,\ 0.0309,\ 0.0325;$ $-0.0874,\ -0.0299,\ -0.0195\,]$
	6 / 3	$[\,0.0049,\ 0.0211,\ 0.0490;$ $-0.0138,\ 0.0207,\ 0.0627;$ $-0.0162,\ -0.0022,\ 0.0141\,]$
	6 / 4	$[\,-0.0411,\ -0.362,\ -0.0024;$ $-0.0350,\ -0.0152,\ -0.0193;$ $-0.0573,\ -0.0310,\ 0.0391\,]$
	6 / 5	$[\,0.0105,\ 0.0300,\ 0.0244;$ $0.0012,\ -0.152,\ 0.0149;$ $-0.0080,\ 0.0190,\ 0.0597\,]$

Table 7. Kernel weights of $\hat{k}_R$ after the convergence of the training with RKI. The k_R has the same kernel shape with $\mathcal{L}ap$. Note that the learned $\hat{k}_R$ parameters are different from the parameters of Laplacian kernel k_L.

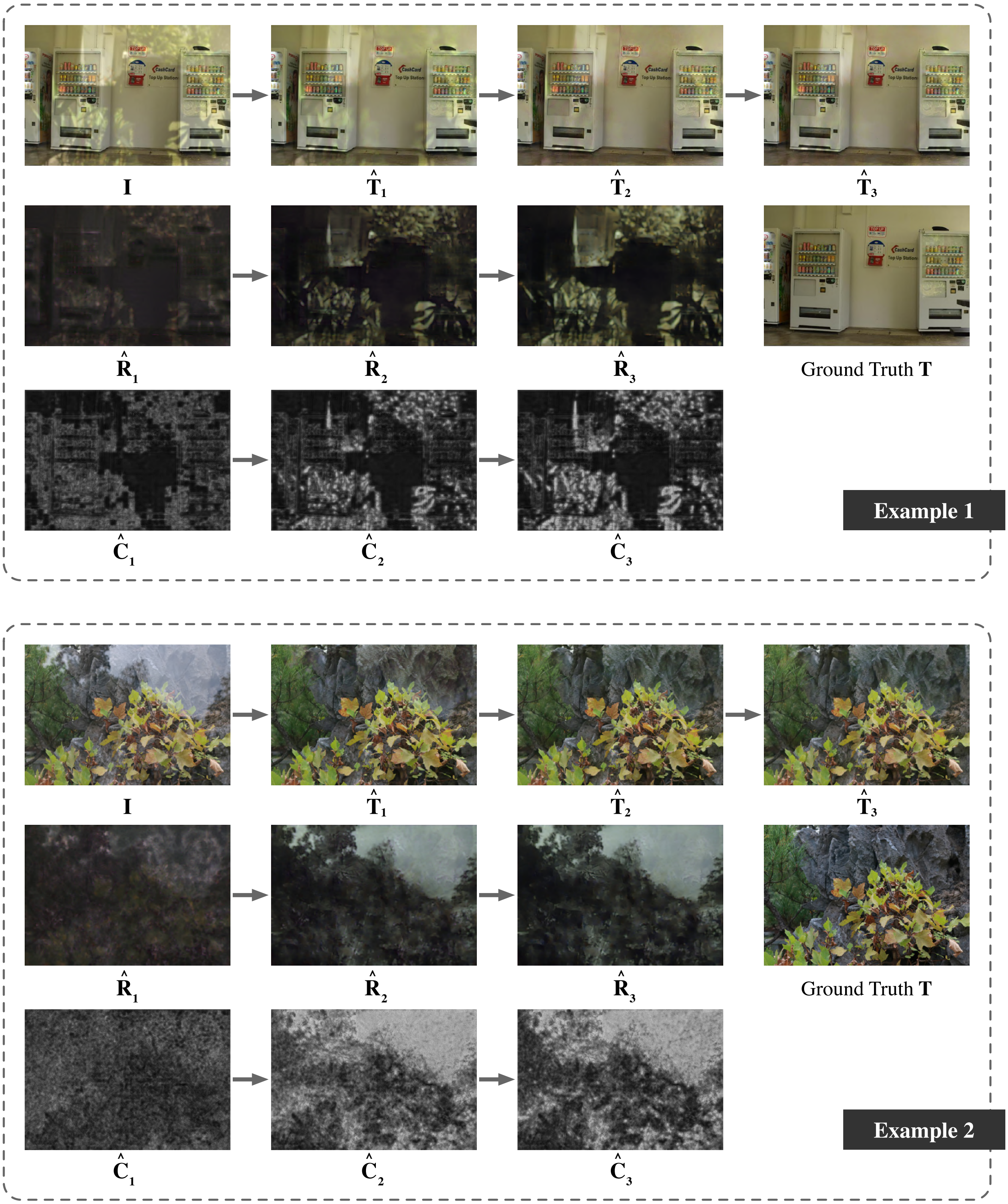


Figure 9. Illustration of the iterative refinement of transmission layer $\hat{\mathbf{T}}_i$, reflection layer $\hat{\mathbf{R}}_i$ and RCMap $\hat{\mathbf{C}}_i$, Part I. The iteration number is indexed by the subscript i.

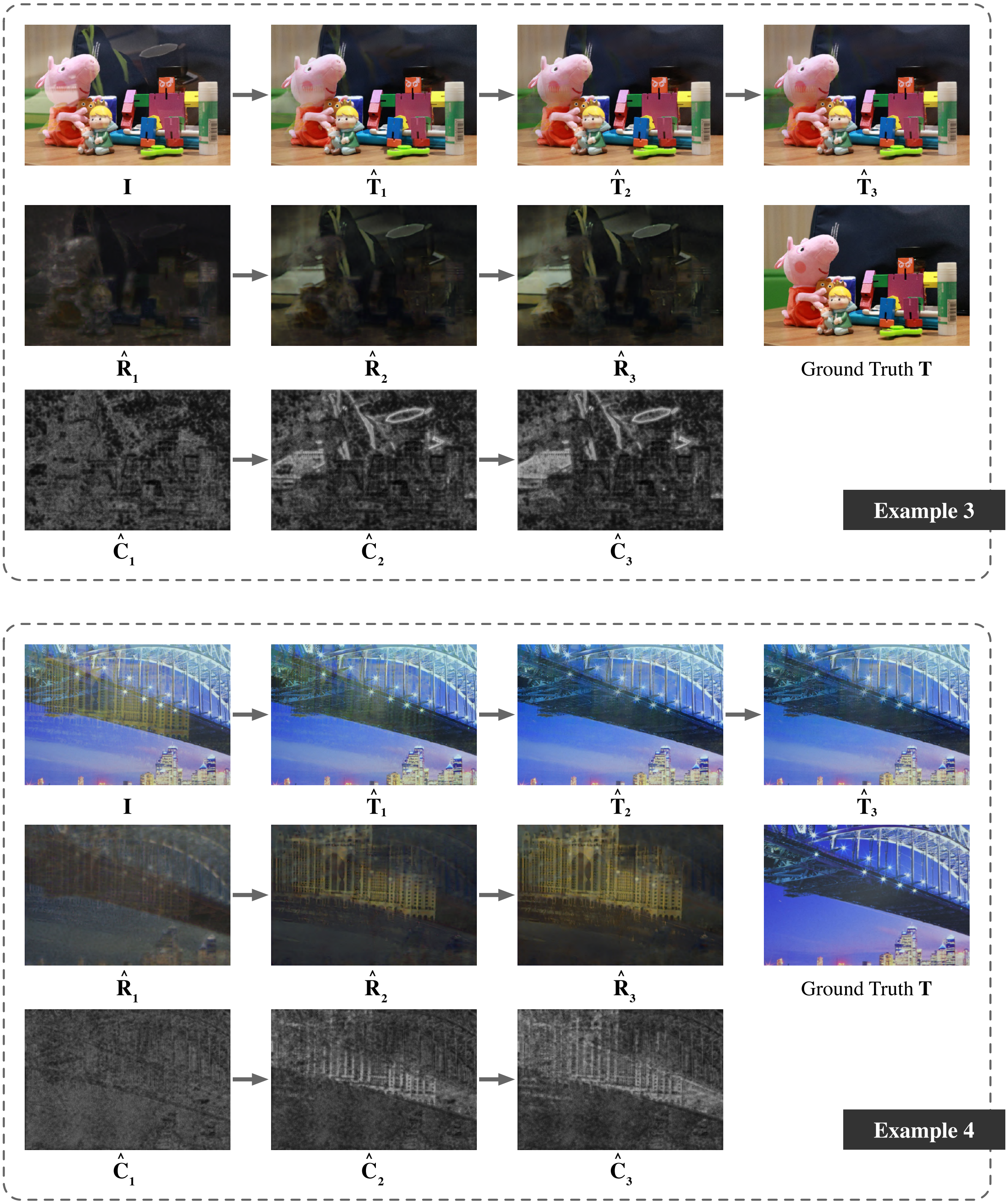


Figure 10. Illustration of the iterative refinement for transmission layer $\hat{\mathbf{T}}_i$, reflection layer $\hat{\mathbf{R}}_i$ and RCMap $\hat{\mathbf{C}}_i$, Part II. The iteration number is indexed by the subscript i.

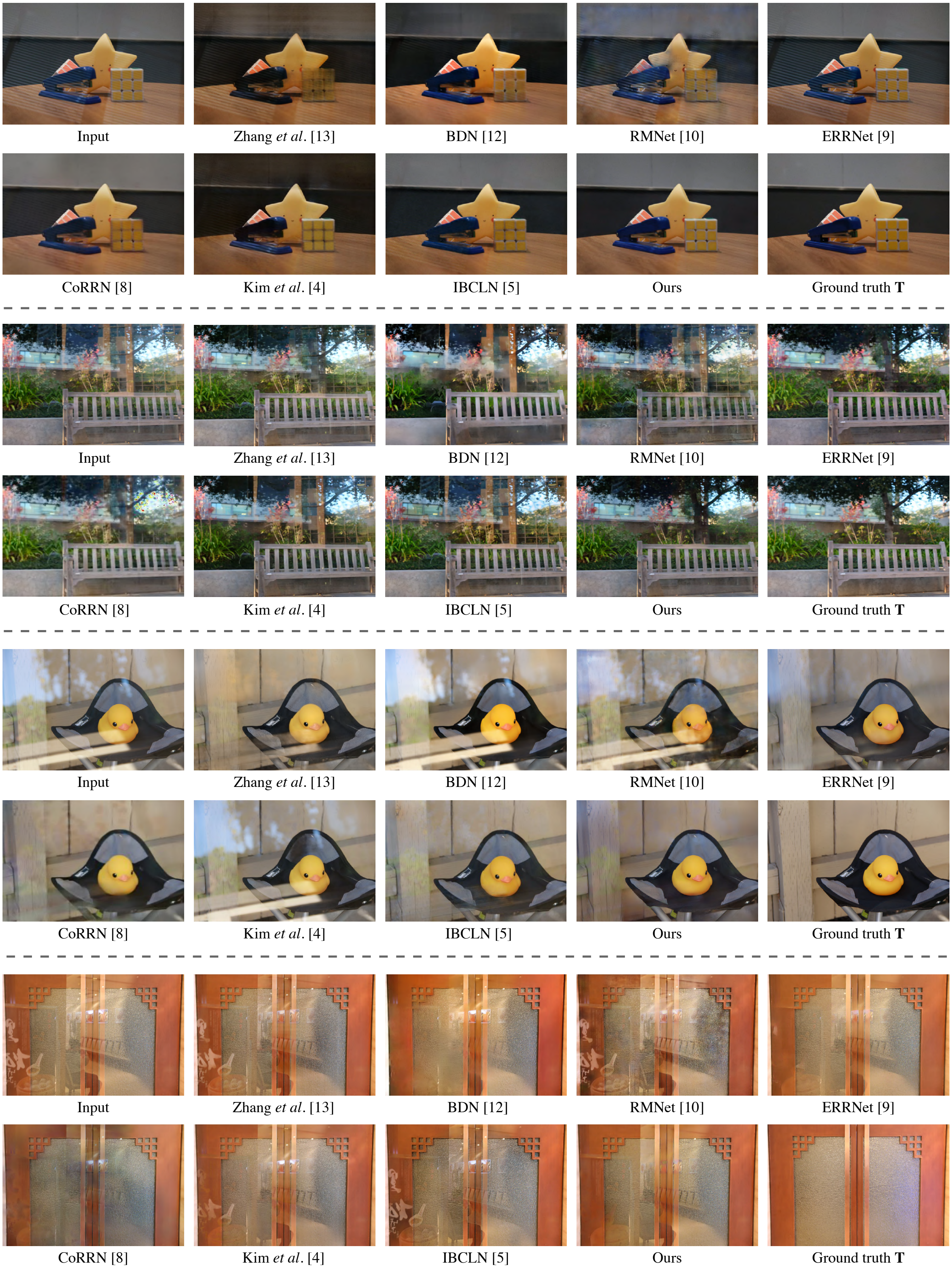


Figure 11. Qualitative comparisons between our method and five state-of-the-art SIRR methods, Part I.

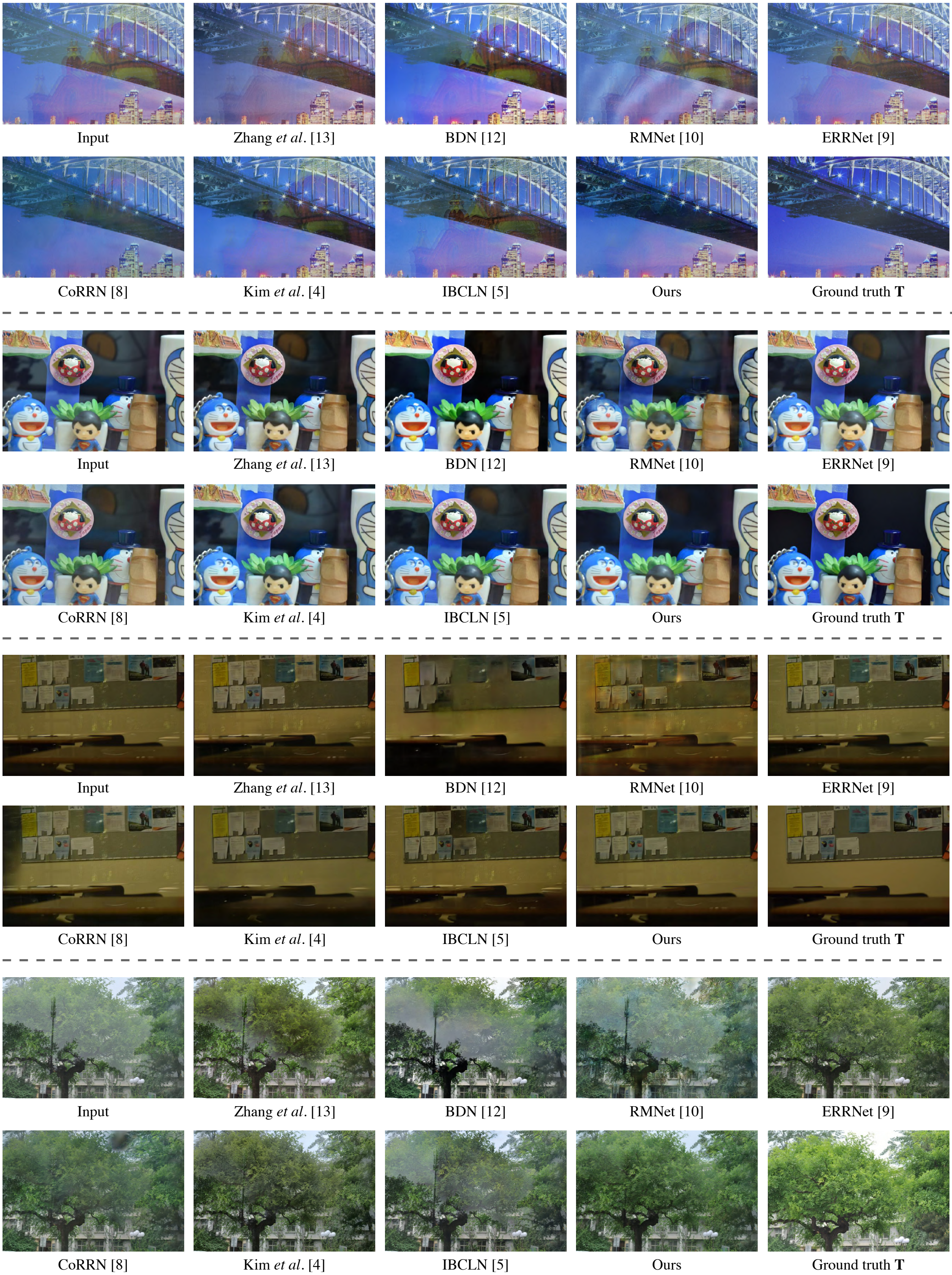


Figure 12. Qualitative comparisons between our method and five state-of-the-art SIRR methods, Part II.

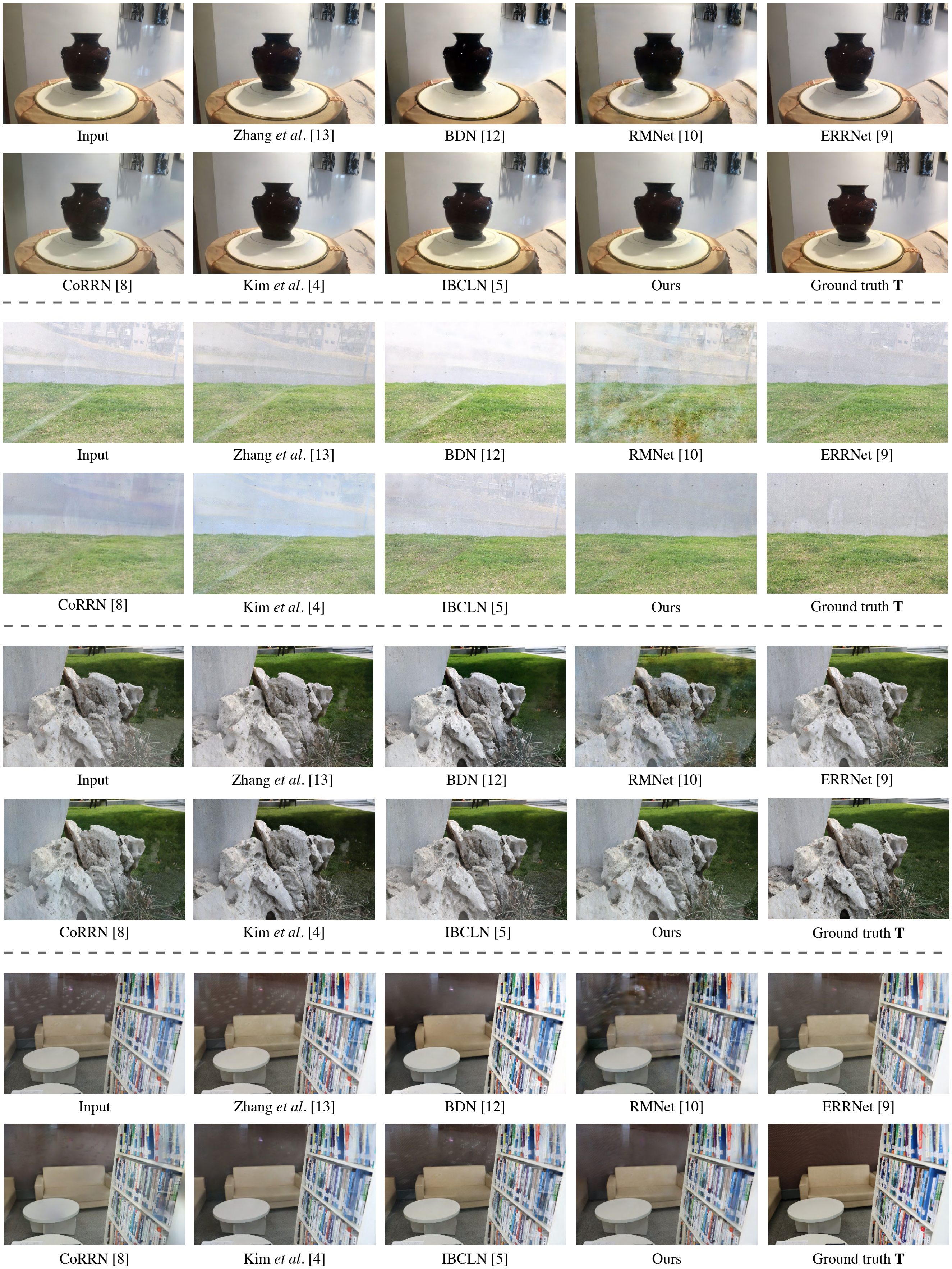


Figure 13. Qualitative comparisons between our method and five state-of-the-art SIRR methods, Part III.

Input Ours Input Ours

Figure 14. Our SIRR results on Internet images.