XYDeblur: Divide and Conquer for Single Image Deblurring
-Supplementary Material-

In this *supplementary material*, we provide detailed explanations for the experimental settings and additional results on the GoPro [3], RealBlur [4] and REDS [2] datasets.

1. Implementation details

XYDeblur. For reproducing the proposed method, we uploaded the source code of XYDeblur with sample data.[1]

To test XYDeblur, run the command below:

- python main.py –mode "test"

Resultant data including result image, $\hat{\mathbf{r}}_1$, and $\hat{\mathbf{r}}_2$ will be saved in the following directory:

- results/XYDeblur/eval

For $\hat{\mathbf{r}}_1$ and $\hat{\mathbf{r}}_2$, two types of files will be stored ($*$.png and $*$.mat). The first $*$.png file contains the normalized data in the range of [0,255]. The second $*$.mat file contains the original tensor data.

To train XYDeblur, run the command below:

- python main.py –mode "train"

During the training process, the training log and intermediate weights will be stored in the following directories:

- runs/XYDeblur
- results/XYDeblur/weights

Since we only provide a sample training dataset, whole blur and sharp images should be copied to the following directories to further train the network with entire training dataset:

- sample_data/train/blur
- sample_data/train/sharp

[1] https://drive.google.com/file/d/1yVmxIqUXHicr62PQu83QvNy90z3I_mPq/view?usp=sharing

Applying the proposed method to the top-performing networks. Detailed descriptions for the networks [1, 3, 5] and their variants used in the experiment are listed in Tab 1. All three networks adopt a multi-stage approach, and each network architecture listed in Tab 1 describes a single stage network. The rest stages are constructed in a similar manner. Unlike the other two methods, the architecture of MSCNN cannot be clearly divided into encoder and decoder. Therefore, the left half and the right half of the architecture are considered as encoder and decoder, respectively.

Table 1. Detailed architecture of the baseline networks and their variants.

	PSS-NSC [1] and its variants											
	PSS-NSC & PSS-NSC$^{\text{Ours}}$				PSS-NSC$^{\text{Channel}\uparrow}$				PSS-NSC$^{\text{Layer}\uparrow}$			
	Operation	C_{in}*	C_{out}†	S‡	Operation	C_{in}	C_{out}	S	Operation	C_{in}	C_{out}	S
$\mathcal{E}$	$\text{Conv}_{\times 1}$	6	32	-	$\text{Conv}_{\times 1}$	6	32	-	$\text{Conv}_{\times 1}$	6	32	-
	$\text{ResBlock}_{\times 8}$	32	32	-	$\text{ResBlock}_{\times 8}$	32	32	-	$\text{ResBlock}_{\times 8}$	32	32	-
	$\text{Conv}_{\times 1}$	32	64	0.5	$\text{Conv}_{\times 1}$	32	64	0.5	$\text{Conv}_{\times 1}$	32	64	0.5
	$\text{ResBlock}_{\times 8}$	64	64	-	$\text{ResBlock}_{\times 8}$	64	64	-	$\text{ResBlock}_{\times 8}$	64	64	-
	$\text{Conv}_{\times 1}$	64	128	0.5	$\text{Conv}_{\times 1}$	64	128	0.5	$\text{Conv}_{\times 1}$	64	128	0.5
	$\text{ResBlock}_{\times 8}$	128	128	-	$\text{ResBlock}_{\times 8}$	128	128	-	$\text{ResBlock}_{\times 8}$	128	128	-
					$\text{Conv}_{\times 1}$	128	256	-				
$\mathcal{D}$	$\text{ResBlock}_{\times 8}$	128	128	-	$\text{ResBlock}_{\times 8}$	256	256	-	$\text{ResBlock}_{\times 16}$	128	128	-
	$\text{Deconv}_{\times 1}$	128	64	2	$\text{Deconv}_{\times 1}$	256	128	2	$\text{Deconv}_{\times 1}$	128	64	2
	$\text{ResBlock}_{\times 8}$	64	64	-	$\text{ResBlock}_{\times 8}$	128	128	-	$\text{ResBlock}_{\times 16}$	64	64	-
	$\text{Deconv}_{\times 1}$	64	32	2	$\text{Deconv}_{\times 1}$	128	64	2	$\text{Deconv}_{\times 1}$	64	32	2
	$\text{ResBlock}_{\times 8}$	32	32	-	$\text{ResBlock}_{\times 8}$	64	64	-	$\text{ResBlock}_{\times 16}$	32	32	-
	$\text{Conv}_{\times 1}$	32	3	-	$\text{Conv}_{\times 1}$	64	3	-	$\text{Conv}_{\times 1}$	32	3	-
	DMPHN [5] and its variants											
	DMPHN & DMPHN$^{\text{Ours}}$				DMPHN$^{\text{Channel}\uparrow}$				DMPHN$^{\text{Layer}\uparrow}$			
	Operation	C_{in}	C_{out}	S	Operation	C_{in}	C_{out}	S	Operation	C_{in}	C_{out}	S
$\mathcal{E}$	$\text{Conv}_{\times 1}$	3	32	-	$\text{Conv}_{\times 1}$	3	32	-	$\text{Conv}_{\times 1}$	3	32	-
	$\text{ResBlock}_{\times 2}$	32	32	-	$\text{ResBlock}_{\times 2}$	32	32	-	$\text{ResBlock}_{\times 2}$	32	32	-
	$\text{Conv}_{\times 1}$	32	64	0.5	$\text{Conv}_{\times 1}$	32	64	0.5	$\text{Conv}_{\times 1}$	32	64	0.5
	$\text{ResBlock}_{\times 2}$	64	64	-	$\text{ResBlock}_{\times 2}$	64	64	-	$\text{ResBlock}_{\times 2}$	64	64	-
	$\text{Conv}_{\times 1}$	64	128	0.5	$\text{Conv}_{\times 1}$	64	128	0.5	$\text{Conv}_{\times 1}$	64	128	0.5
	$\text{ResBlock}_{\times 2}$	128	128	-	$\text{ResBlock}_{\times 2}$	128	128	-	$\text{ResBlock}_{\times 2}$	128	128	-
					$\text{Conv}_{\times 1}$	128	256	-				
$\mathcal{D}$	$\text{ResBlock}_{\times 2}$	128	128	-	$\text{ResBlock}_{\times 2}$	256	256	-	$\text{ResBlock}_{\times 4}$	128	128	-
	$\text{Deconv}_{\times 1}$	128	64	2	$\text{Deconv}_{\times 1}$	256	128	2	$\text{Deconv}_{\times 1}$	128	64	2
	$\text{ResBlock}_{\times 2}$	64	64	-	$\text{ResBlock}_{\times 2}$	128	128	-	$\text{ResBlock}_{\times 4}$	64	64	-
	$\text{Deconv}_{\times 1}$	64	32	2	$\text{Deconv}_{\times 1}$	128	64	2	$\text{Deconv}_{\times 1}$	64	32	2
	$\text{ResBlock}_{\times 2}$	32	32	-	$\text{ResBlock}_{\times 2}$	64	64	-	$\text{ResBlock}_{\times 4}$	32	32	-
	$\text{Conv}_{\times 1}$	32	3	-	$\text{Conv}_{\times 1}$	64	3	-	$\text{Conv}_{\times 1}$	32	3	-
	MSCNN [3] and its variants											
	MSCNN & MSCNN$^{\text{Ours}}$				MSCNN$^{\text{Channel}\uparrow}$				MSCNN$^{\text{Layer}\uparrow}$			
	Operation	C_{in}	C_{out}	S	Operation	C_{in}	C_{out}	S	Operation	C_{in}	C_{out}	S
$\mathcal{E}$	$\text{Conv}_{\times 1}$	6	64	-	$\text{Conv}_{\times 1}$	6	64	-	$\text{Conv}_{\times 1}$	6	64	-
	$\text{ResBlock}_{\times 10}$	64	64	-	$\text{ResBlock}_{\times 10}$	64	64	-	$\text{ResBlock}_{\times 10}$	64	64	-
					$\text{Conv}_{\times 1}$	64	128	-				
$\mathcal{D}$	$\text{ResBlock}_{\times 9}$	64	64	-	$\text{ResBlock}_{\times 9}$	128	128	-	$\text{ResBlock}_{\times 18}$	64	64	-
	$\text{Conv}_{\times 1}$	64	3	-	$\text{Conv}_{\times 1}$	128	3	-	$\text{Conv}_{\times 1}$	64	3	-

* The input channel dimension of the feature map.
† The output channel dimension of the feature map.
‡ Scaling factor for the resolution of the feature map.

2. Additional validation for separated decoders

We tested XYDeblur using a blurred Checkerboard image shown in Fig. 1 (a), which contains equally distributed signals along both horizontal and vertical axes. The sub-solution estimated by the first decoder ($\hat{\mathbf{r}}_1$) is shown in Fig. 1 (b), and its 1D representation on the two cross-sections parallel to the x- and y- axis are plotted in Fig. 1 (d). As seen in Fig. 1 (d), $\hat{\mathbf{r}}_1$ contains high level of x-axis frequency component (yellow colored signal) and low level of y-axis frequency component (blue colored signal). In terms of the signal power, the 1D signal in x-axis has the signal power of 289.37 and the 1D signal in y-axis has the signal power of 0.81. On the contrary, the sub-solution from the second decoder ($\hat{\mathbf{r}}_2$) contains the x-axis signal of power 0.51 and the y-axis signal of power 353.31 as shown in Figs. 1 (c) and (e).

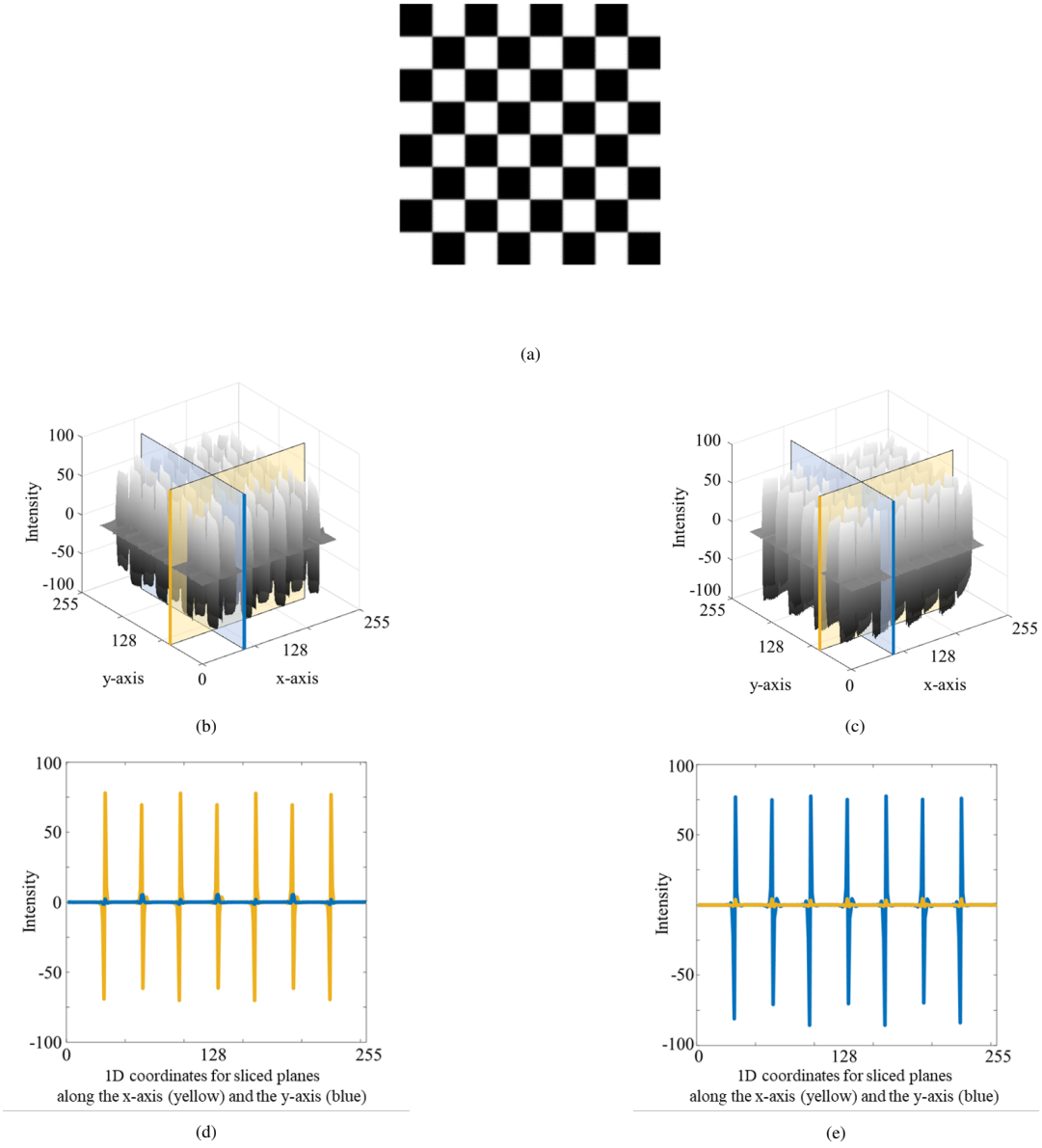

Figure 1. Input Checkerboard image and sub-solutions of Y-Net: (a) Blurred Checkerboard image; (b) $\hat{\mathbf{r}}_1$ plotted in 3D-space with two slice planes; (c) $\hat{\mathbf{r}}_2$ plotted in 3D-space with two slice planes; (d) 1D representation of sliced planes along the x-axis (yellow) and the y-axis (blue) for $\hat{\mathbf{r}}_1$; (e) 1D representation of sliced planes along the x-axis (yellow) and the y-axis (blue) for $\hat{\mathbf{r}}_2$.

3. Training of the three network variations in Table

We extended the training epochs of the three network variations in Table 2 from the main manuscript. Table 2 shows the performance obtained after 1,800, 1,600, and 1,300 epochs for PSS-NSC, DMPHN, and MSCNN, respectively. Note that all three network variations were sufficiently trained and found to have converged after approximately 1,800, 1,600, and 1,300 epochs, respectively, and further training of Channel$^{\uparrow}$ and Layer$^{\uparrow}$ did not show any noticeable differences. All the variations showed consistent improvements over Baseline. Ours still outperforms the other two variations by a large margin for MSCNN and shows comparable performance for PSS-NSC and DMPHN at the expense of approximately 60 $\%$ (w.r.t. Channel$^{\uparrow}$) and 30 $\%$ (w.r.t. Layer$^{\uparrow}$) fewer network parameters.

Table 2. Performance evaluation on GoPro with further training

Method	Variations	PSNR$_{\text{(training epoch)}}$	
PSS-NSC	Baseline	30.94$_{(1,100)}$	
	Channel$^{\uparrow}$	31.06$_{(1,100)}$	31.31$_{(1,800)}$
	Layer$^{\uparrow}$	31.18$_{(1,100)}$	31.53$_{(1,800)}$
	Ours	31.27$_{(1,100)}$	31.55$_{(1,800)}$
DMPHN	Baseline	30.28$_{(1,000)}$	
	Channel$^{\uparrow}$	30.23$_{(1,000)}$	30.67$_{(1,600)}$
	Layer$^{\uparrow}$	30.48$_{(1,000)}$	30.98$_{(1,600)}$
	Ours	30.63$_{(1,000)}$	30.97$_{(1,600)}$
MSCNN	Baseline	29.22$_{(600)}$	
	Channel$^{\uparrow}$	29.34$_{(600)}$	29.92$_{(1,300)}$
	Layer$^{\uparrow}$	28.97$_{(600)}$	29.90$_{(1,300)}$
	Ours	29.98$_{(600)}$	30.63$_{(1,300)}$

4. More results on the prevention of color shift artifacts

As described in the main manuscript, spatial kernel rotation for parameter sharing between the two decoders prevents undesired division of complementary features and also improves the efficiency of the network. In this section, we provide more results on the GoPro and RealBlur test datasets. Figs. 2- 5 shows the input, GT, and resultant images of U-Net^{2D} and the proposed XYDeblur. It can be clearly seen that the two decoders of U-Net^{2D} unnecessarily decrease or increase the color temperature of the resultant images. On the contrary, the two decoders of the proposed XYDeblur suppress estimation of undesired complementary sub-solutions, resulting images with the balanced color temperature.

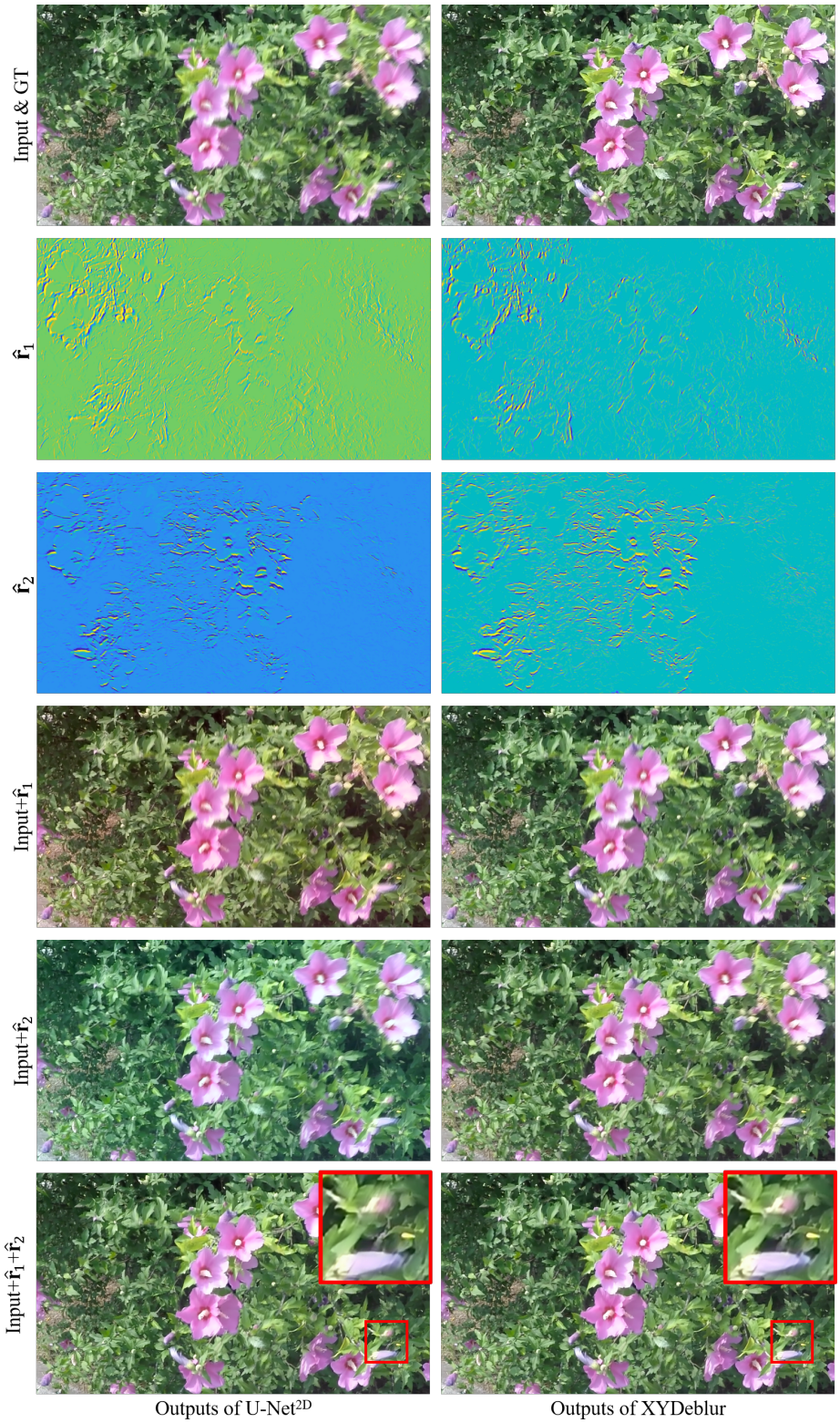

Figure 2. Experimental data of U-Net^{2D} and XYDeblur from the GoPro test dataset to demonstrate the color shift artifact.

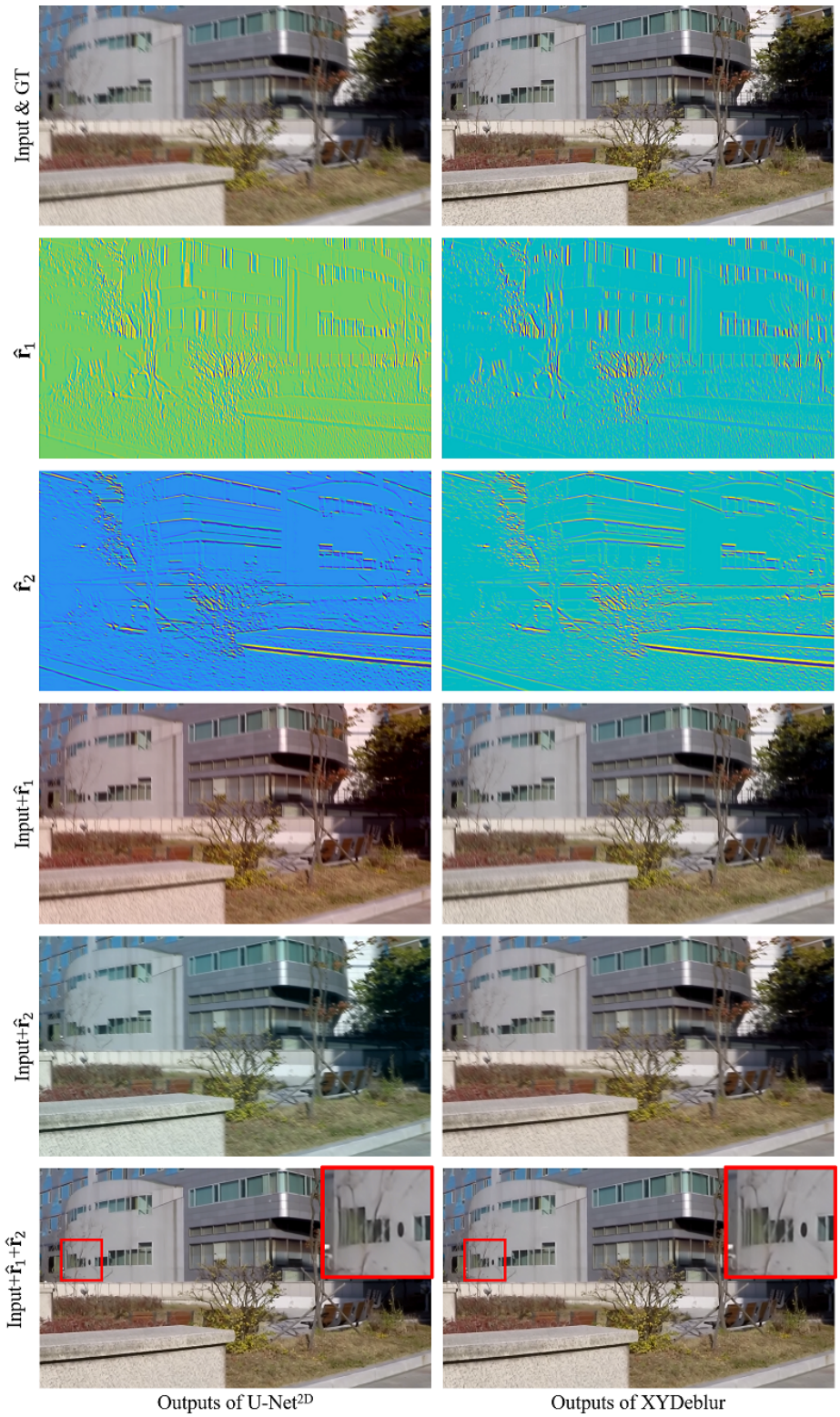

Figure 3. Experimental data of U-Net^{2D} and XYDeblur from the GoPro test dataset to demonstrate the color shift artifact.

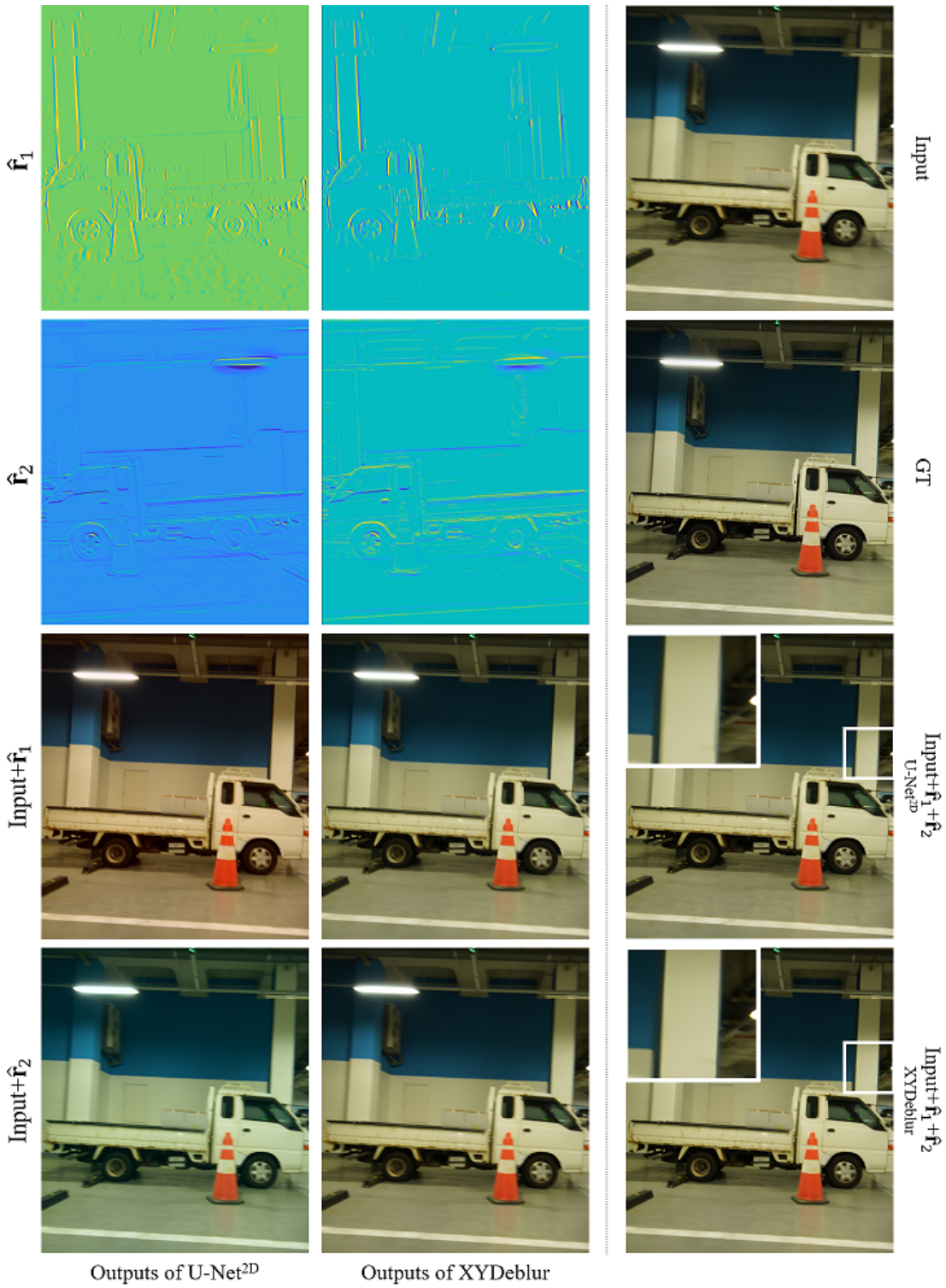

Figure 4. Experimental data of U-Net^{2D} and XYDeblur from the RealBlur test dataset to demonstrate the color shift artifact.

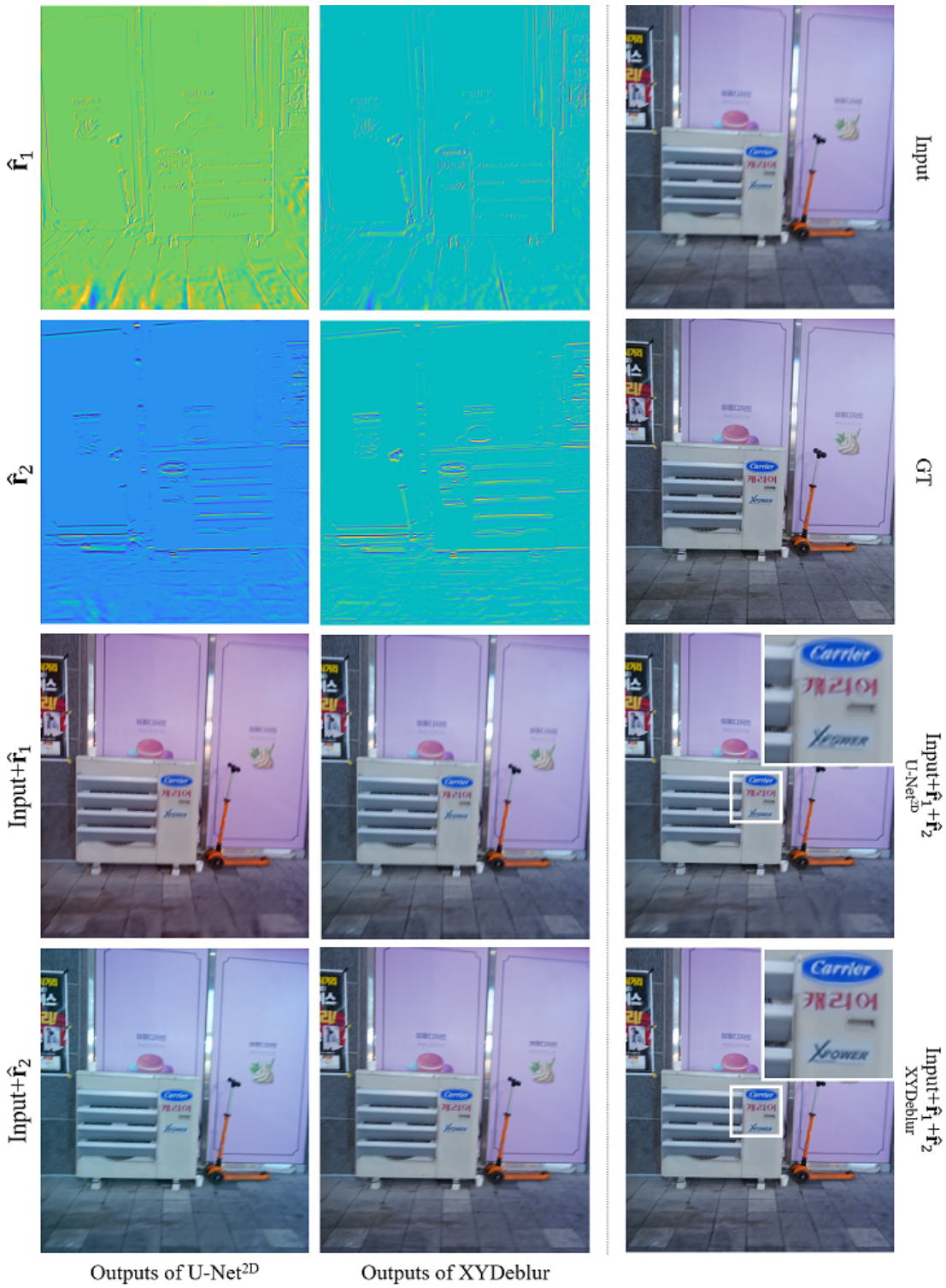

Figure 5. Experimental data of U-Net^{2D} and XYDeblur from the RealBlur test dataset to demonstrate the color shift artifact.

5. Resultant images of the state-of-the-art networks and their variations

In Section 4.3 of the main manuscript, we performed an experiment to validate the applicability and extensibility of XYDeblur. In this section, we visualize the resultant images of the RealBlur test dataset in Figs. 6 & 7. In these figures, the second rows show the resultant images obtained by the baseline network, i.e., MSCNN, DMPHN, and PSS-NSC. The third rows represent the results of the Network$^{\text{Ours}}$, and the last two rows are the residual images generated by the two decoders of Network$^{\text{Ours}}$. These figures show that Network$^{\text{Ours}}$ produce better deblurred results than the baseline networks.

To further verify that the property of X-Y separation is not acquired by choice of one specific training dataset, we trained PSS-NSC with the proposed approach on the REDS dataset. The resultant images of the REDS dataset can be found in Fig. 8, where the trained model exhibits a clear separation of X-Y components. Also, this validation shows a similar trend to the results of the model trained on the GoPro dataset.

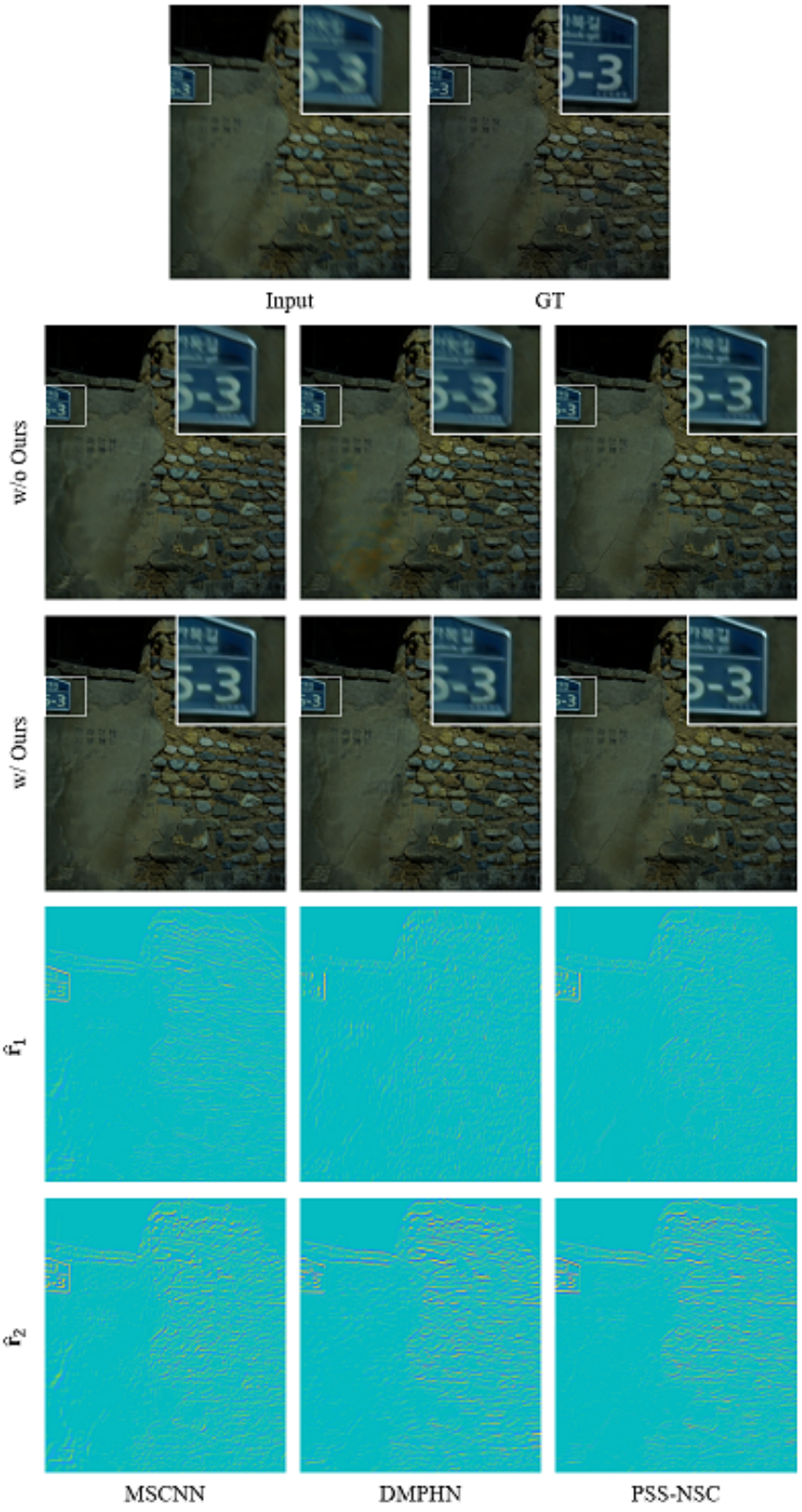

Figure 6. Experimental results for MSCNN, DMPHN, PSS-NSC with and without the proposed approach from the RealBlur dataset.

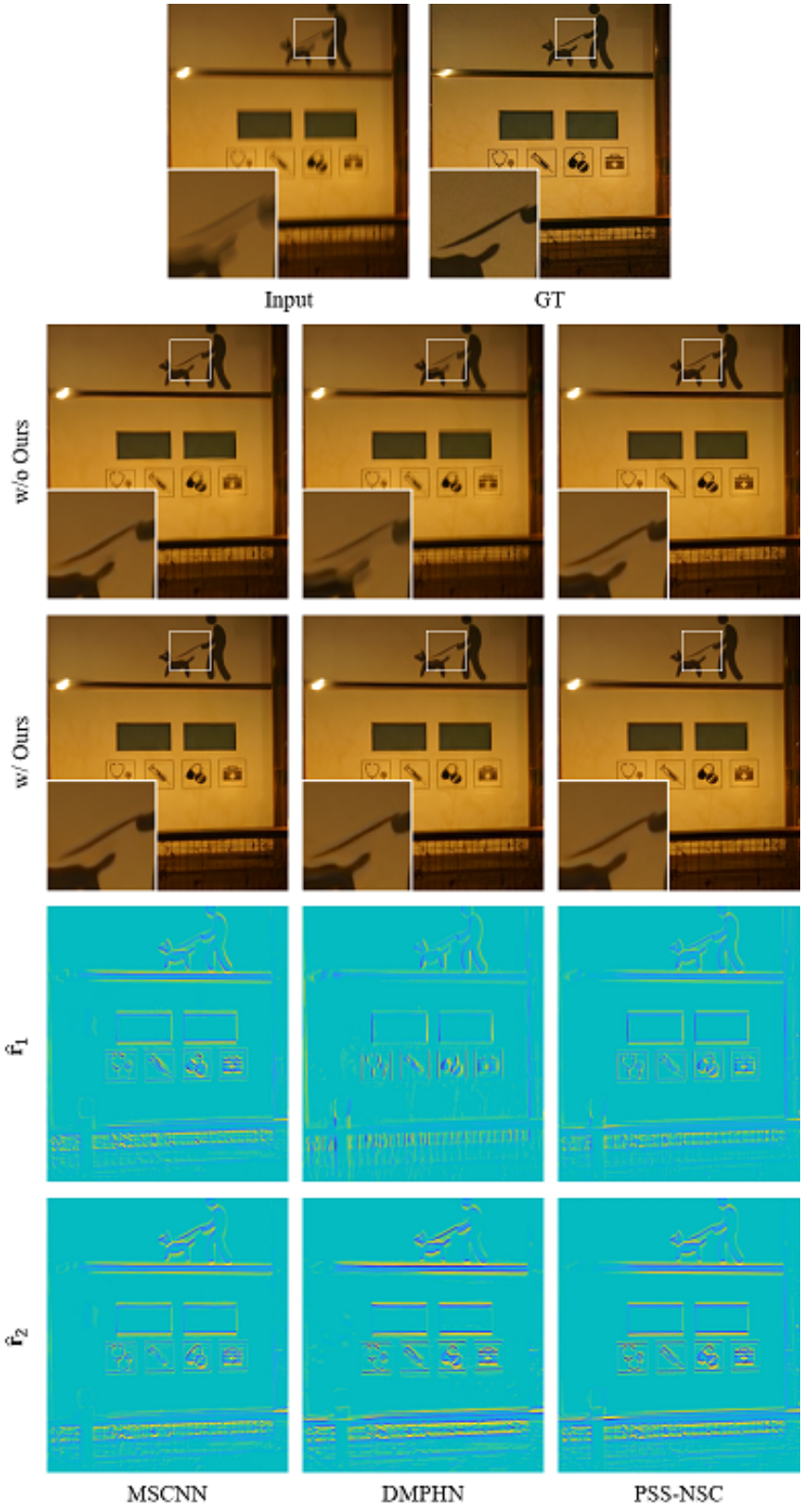

Figure 7. Experimental results for MSCNN, DMPHN, PSS-NSC with and without the proposed approach from the RealBlur dataset.

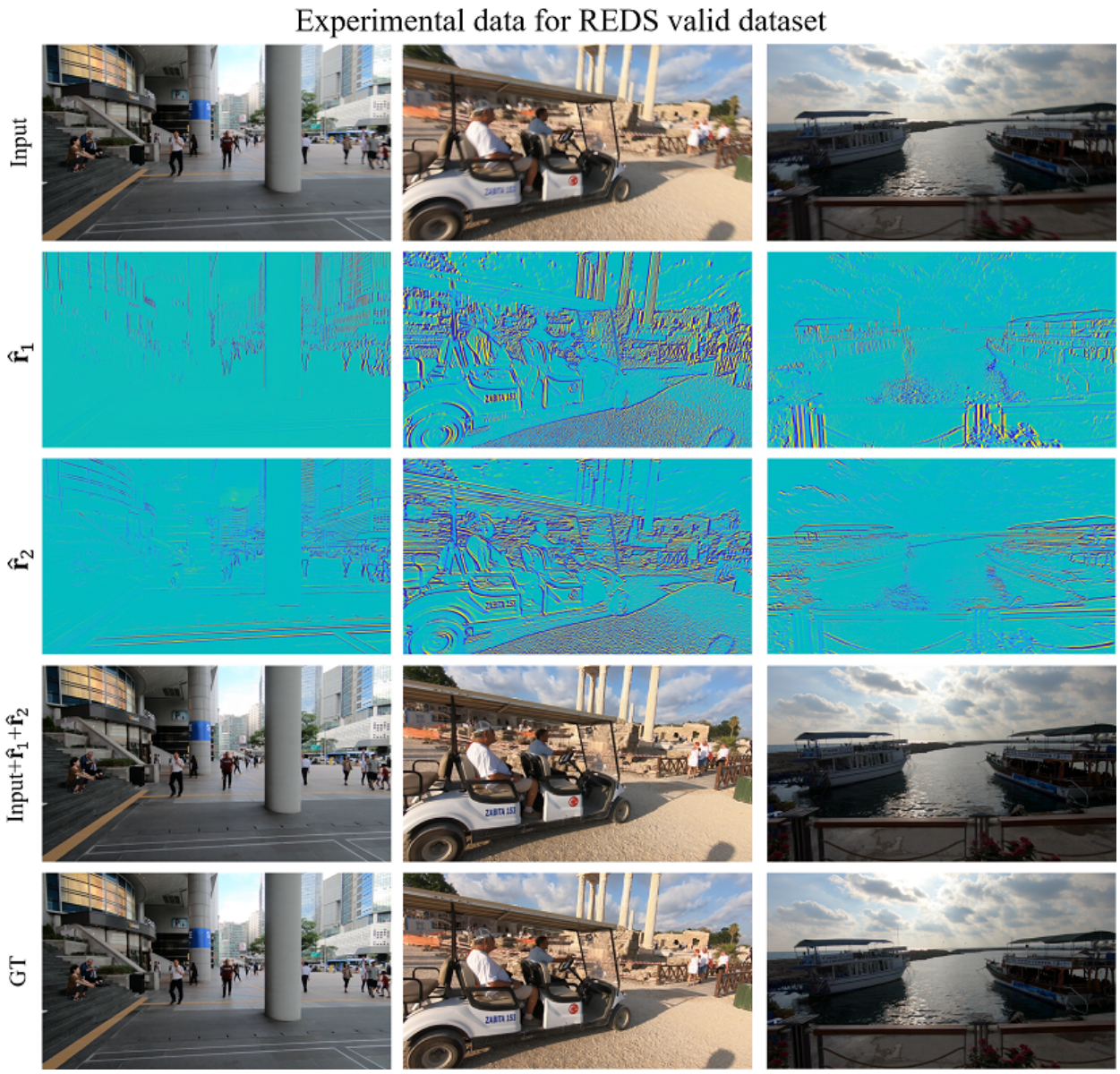

Figure 8. Experimental results for PSS-NSC with the proposed approach from the REDS dataset.

References

[1] Hongyun Gao, Xin Tao, Xiaoyong Shen, and Jiaya Jia. Dynamic scene deblurring with parameter selective sharing and nested skip connections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3848–3856, 2019. 2

[2] Seungjun Nah, Sungyong Baik, Seokil Hong, Gyeongsik Moon, Sanghyun Son, Radu Timofte, and Kyoung Mu Lee. Ntire 2019 challenge on video deblurring and super-resolution: Dataset and study. In *CVPR Workshops*, June 2019. 1

[3] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dynamic scene deblurring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3883–3891, 2017. 1, 2

[4] Jaesung Rim, Haeyun Lee, Jucheol Won, and Sunghyun Cho. Real-world blur dataset for learning and benchmarking deblurring algorithms. In *Proceedings of the European Conference on Computer Vision*, pages 184–201, 2020. 1

[5] Hongguang Zhang, Yuchao Dai, Hongdong Li, and Piotr Koniusz. Deep stacked hierarchical multi-patch network for image deblurring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5978–5986, 2019. 2