Learning Texture Transformer Network for Image Super-Resolution Supplementary Material

Fuzhi Yang[1*], Huan Yang[2], Jianlong Fu[2], Hongtao Lu[1], Baining Guo[2]
[1]Department of Computer Science and Engineering,
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University,
[2]Microsoft Research, Beijing, P.R. China,
{yfzcopy0702, htlu}@sjtu.edu.cn, {huayan, jianf, bainguo}@microsoft.com

In this supplementary material, Section 1 illustrates the details of TTSR's network structure. Section 2 provides additional analyses about the texture transformers on different scales. Section 3 describes the comparison of the running time and the parameter number. Finally, more visual comparison results will be shown in Section 4.

1. Details of Network Structure

In this section, we will illustrate the detailed network structure of our approach TTSR, including the learnable texture extractor in the texture transformer, the generator with three stacked texture transformers and the discriminator. The structure of the learable texture extractor is shown in Table 1, in which the layers \$0, \$3 and \$6 are used to search and transfer texture features in the texture transformer. Table 2 shows the details of the generator, and Table 3 illustrates the discriminator.

2. Texture Transformers on Different Scales

Our proposed TTSR contains three stacked texture transformers. The texture transformer at each scale fuses HR texture features of different levels from the Ref image. Here we conduct experiments of using the texture transformers on different scales. The model here is without CSFI since CSFI is designed for multi-scale stacked texture transformers. Table 4 shows the results. The larger scale the texture transformer is applied at, the more performance it brings, which demonstrates that the texture features at a larger scale have a less loss of details. When we gradually add the texture transformers at other scales, the performance can be further improved.

3. Running Time and Model Size

In this section, the running time and the model size of TTSR will be discussed. We compare the proposed TTSR with state-of-the-art SISR and RefSR approaches, RCAN [5], RSRGAN [4], CrossNet [8] and SRNTT [7]. For running time, all approaches are run on a Tesla V100 PCIe GPU and tested on an $83 \times 125 \times 3$ LR input image with the up-sampling factor of $4\times$. Table 5 shows the comparison results. Specifically, the stacked texture transformers cost 0.037s and the other parts cost 0.059s, and TTSR takes a total time of 0.096s. The results show that TTSR achieves the best performance with a relatively small parameter number and running time.

Table 1. Network structure of the learnable texture extractor. Conv(N_{in}, N_{out}) indicates the convolutional layer with N_{in} input channels and N_{out} output channels. The kernel size is 3×3 for all convolutional layers. Pool(2×2) is the 2×2 pooling layer with stride 2.

Id	Layer Name
0	Conv(3,64), ReLU
1	Conv(64,64), ReLU
2	Pool(2×2)
3	Conv(64,128), ReLU
4	Conv(128,128), ReLU
5	Pool(2×2)
6	Conv(128, 256), ReLU

4. More Visual Comparison

In this section, we show more comparison results among the proposed TTSR and other SR methods, including RDN [6], RCAN [5], RSRGAN [4], CrossNet [8] and SRNTT [7]. RCAN has achieved state-of-the-art performance on both PSNR and SSIM in recent years and RSRGAN is considered to achieve the state-of-the-art visual quality. CrossNet and SRNTT are two state-of-the-art RefSR approaches which significantly outperform previous RefSR methods. The visual comparison results on CUFED5 [7], Sun80 [3], Urban100 [1] and Manga109 [2] are shown in Figure 1-4, Figure 5-6, Figure 7-8 and Figure 9-10, respectively.

*This work was performed when the first author was visiting Microsoft Research as a research intern.

Table 2. Network structure of the generator. Conv(N_{in}, N_{out}) indicates the convolutional layer with N_{in} input channels and N_{out} output channels. The kernel size is 3×3 for all convolutional layers except that the last convolution uses 1×1 kernel. RB denotes the residual block without batch normalization layers and the ReLU layer after the skip connection. TT represents the texture transformer and PS is the $2\times$ pixel shuffle layer. $\uparrow$ indicates bicubic up-sampling followed by a 1×1 convolution, while $\downarrow$ denotes the strided convolution.

	Id	Layer Name (scale1×)	Id	Layer Name (scale2×)	Id	Layer Name (scale4×)
Stage0	1-0	Conv(3,64), ReLU				
	1-1	RB ×16				
	1-2	Conv(64,64)				
	1-3	$1-0 + $1-2				
Stage1	1-4	TT				
	1-5	RB×16				
	1-6	Conv(64,64)				
	1-7	$1-4 + $1-6				
Stage2			2-0	Conv(64,256), PS, ReLU($1-7)		
			2-1	TT		
	1-8	Concat($1-7 || $2-1↓)	2-2	Concat($1-7↑ || $2-1)		
	1-9	Conv(128,64), ReLU	2-3	Conv(128,64), ReLU		
	1-10	RB×8	2-4	RB×8		
	1-11	Conv(64,64)	2-5	Conv(64,64)		
	1-12	$1-7 + $1-11	2-6	$2-1 + $2-5		
Stage3					4-0	Conv(64,256), PS, ReLU($2-6)
					4-1	TT
	1-13	Concat($1-12 || $2-6↓ || $4-1↓)	2-7	Concat($1-12↑ || $2-6 || $4-1↓)	4-2	Concat($1-12↑ || $2-6↑ || $4-1)
	1-14	Conv(192,64), ReLU	2-8	Conv(192,64), ReLU	4-3	Conv(192,64), ReLU
	1-15	RB×4	2-9	RB×4	4-4	RB×4
	1-16	Conv(64,64)	2-10	Conv(64,64)	4-5	Conv(64,64)
	1-17	$1-12 + $1-16	2-11	$2-6 + $2-10	4-6	$4-1 + $4-5
Stage4					4-7	Concat($1-17↑ || $2-11↑ || $4-6)
					4-8	Conv(192,64), ReLU
					4-9	Conv(64,32)
					4-10	Conv(32,3)

Table 3. Network structure of the discriminator. Conv(N_{in}, N_{out}, S) indicates the convolutional layer with N_{in} input channels, N_{out} output channels and stride S. The kernel size is 3×3 for all convolutional layers. The parameter is 0.2 for all the leaky ReLU layers. The size of HR and SR input is $160 \times 160 \times 3$

Id	Layer Name
0	Conv(3,32,1), LReLU
1	Conv(32,32,2), LReLU
2	Conv(32,64,1), LReLU
3	Conv(64,64,2), LReLU
4	Conv(64,128,1), LReLU
5	Conv(128,128,2), LReLU
6	Conv(128,256,1), LReLU
7	Conv(256,256,2), LReLU
8	Conv(256,512,1), LReLU
9	Conv(512,512,2), LReLU
10	FC(12800,1024), LReLU
11	FC(1024,1)

Table 4. Performance on CUFED5 testing set using texture transformers on different scales.

scale1×	scale2×	scale4×	PSNR/SSIM
✓			26.56 / .785
	✓		26.77 / .793
		✓	26.85 / .796
✓	✓		26.80 / .793
✓	✓	✓	26.92 / .797

Table 5. Running time and parameter number of different approaches. The last column shows the PSNR/SSIM performance on CUFED5 testing set.

Approach	Time	Param.	PSNR/SSIM
RCAN [5]	0.108s	16M	26.06 / .769
RSRGAN [4]	0.007s	1.5M	22.31 / .635
CrossNet [8]	0.229s	33.6M	25.48 / .764
SRNTT [7]	4.977s	4.2M	26.24 / .784
TTSR	0.096s	6.4M	27.09 / .804

References

[1] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *CVPR*, pages 5197–5206, 2015.

[2] Yusuke Matsui, Kota Ito, Yuji Aramaki, Azuma Fujimoto, Toru Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. Sketch-based manga retrieval using manga109 dataset. *Multimedia Tools and Applications*, 76(20):21811–21838, 2017.

[3] Libin Sun and James Hays. Super-resolution from internet-scale scene matching. In *ICCP*, pages 1–12, 2012.

[4] Wenlong Zhang, Yihao Liu, Chao Dong, and Yu Qiao. Ranksrgan: Generative adversarial networks with ranker for image super-resolution. In *ICCV*, pages 3096–3105, 2019.

[5] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. In *ECCV*, pages 286–301, 2018.

[6] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *CVPR*, pages 2472–2481, 2018.

[7] Zhifei Zhang, Zhaowen Wang, Zhe Lin, and Hairong Qi. Image super-resolution by neural texture transfer. In *CVPR*, pages 7982–7991, 2019.

[8] Haitian Zheng, Mengqi Ji, Haoqian Wang, Yebin Liu, and Lu Fang. Crossnet: An end-to-end reference-based super resolution network using cross-scale warping. In *ECCV*, pages 88–104, 2018.

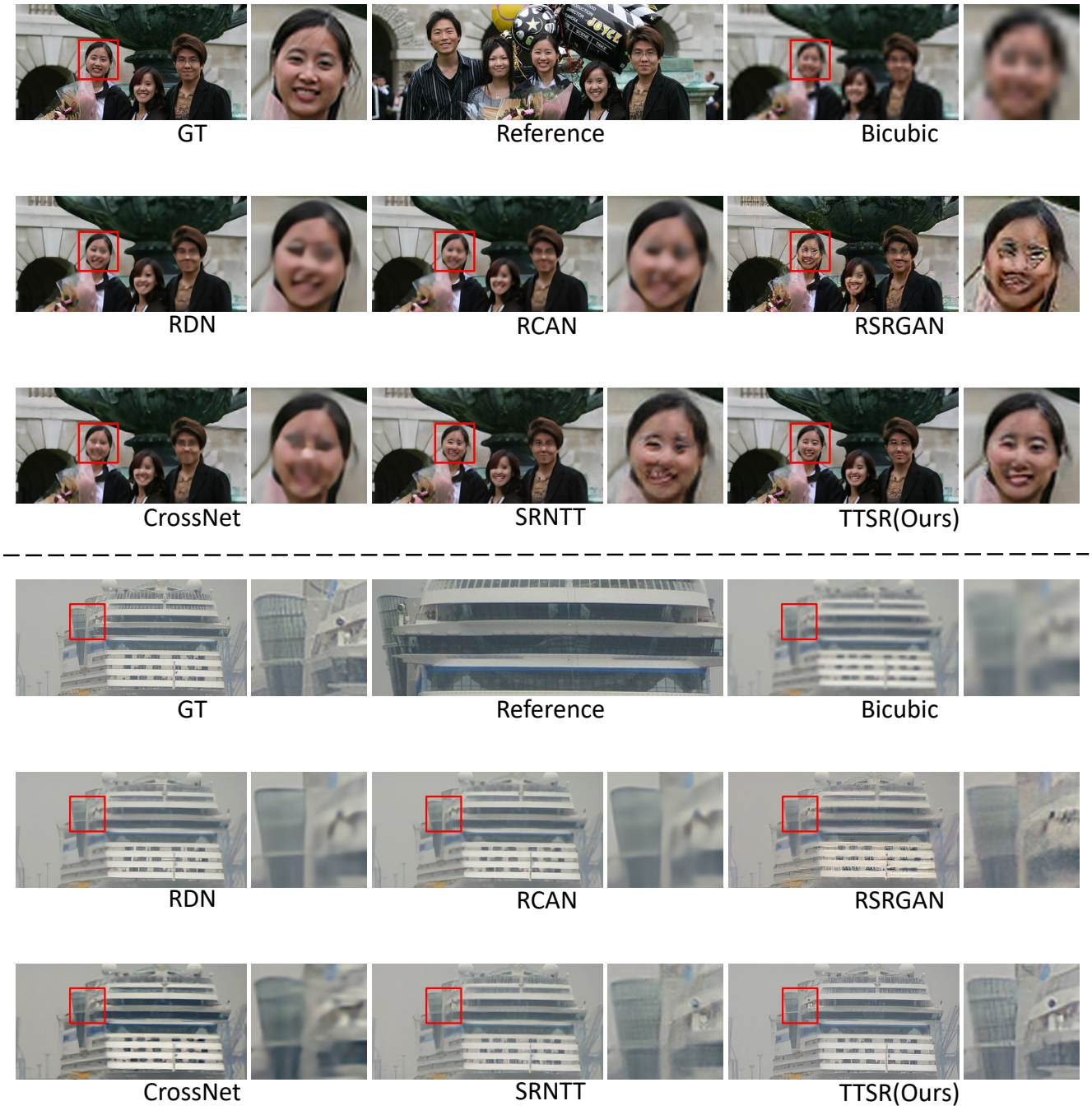

Figure 1. Visual comparison of different SR methods on CUFED5 [7] dataset.

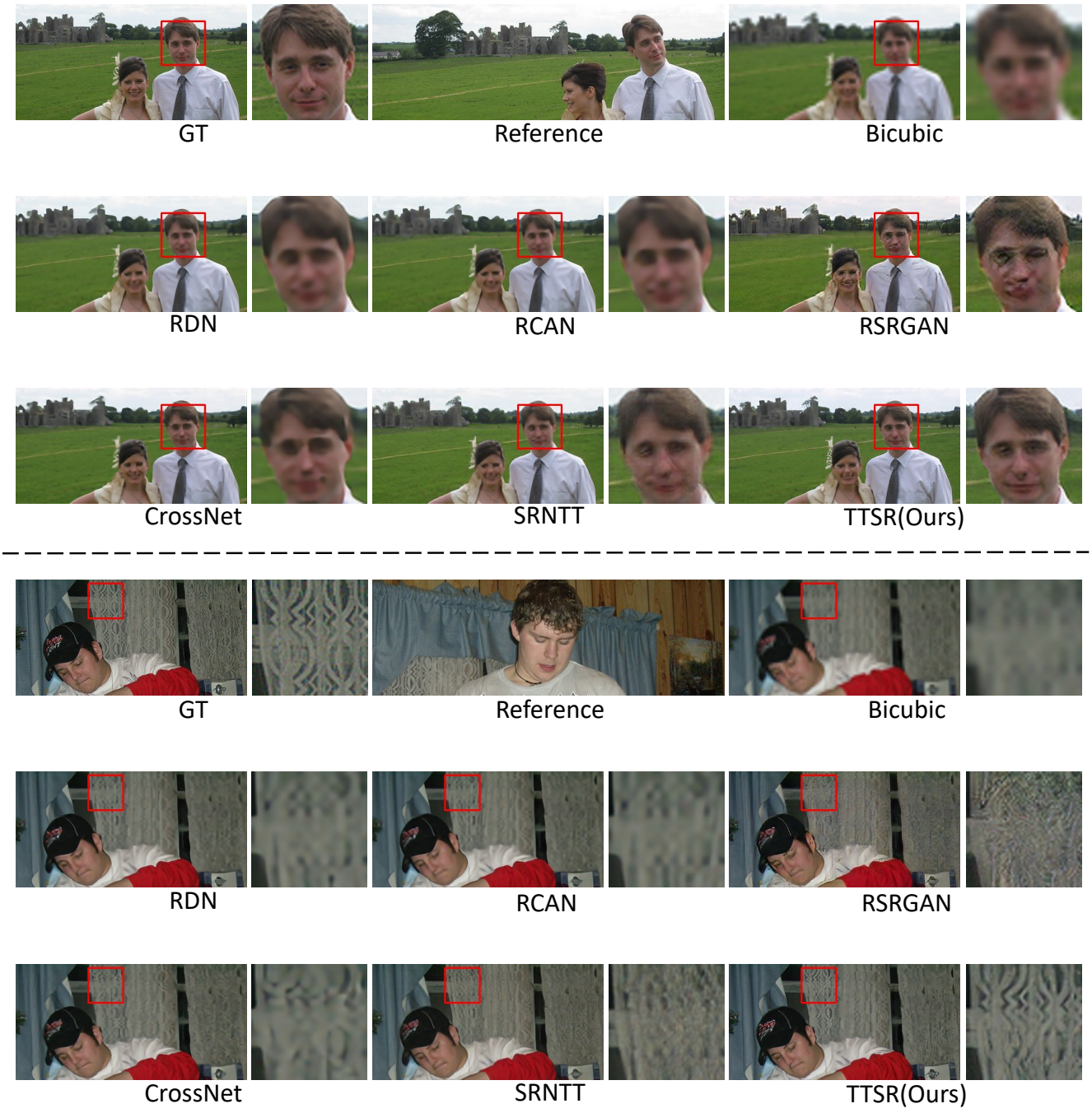

Figure 2. Visual comparison of different SR methods on CUFED5 [7] dataset.

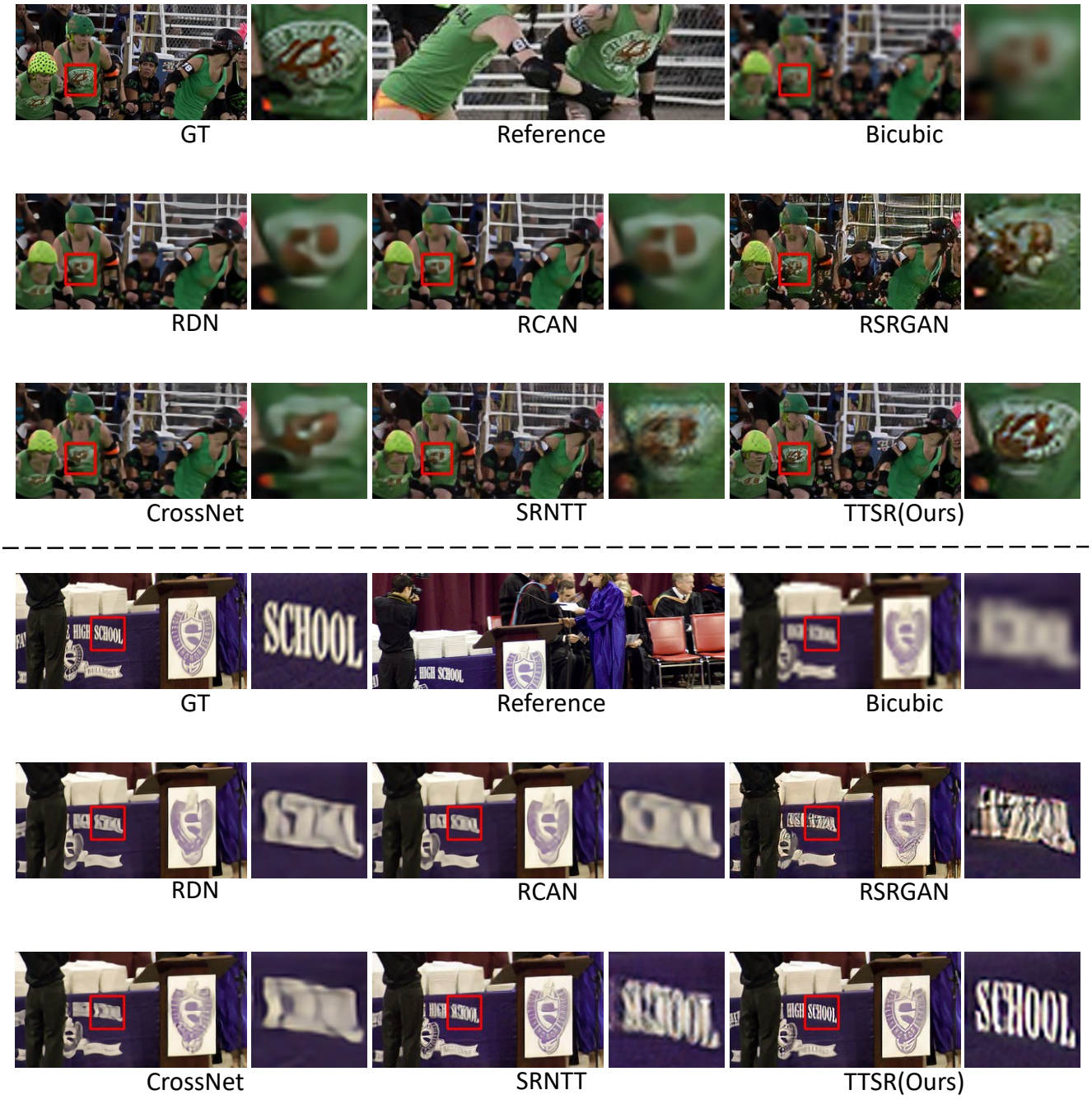

Figure 3. Visual comparison of different SR methods on CUFED5 [7] dataset.

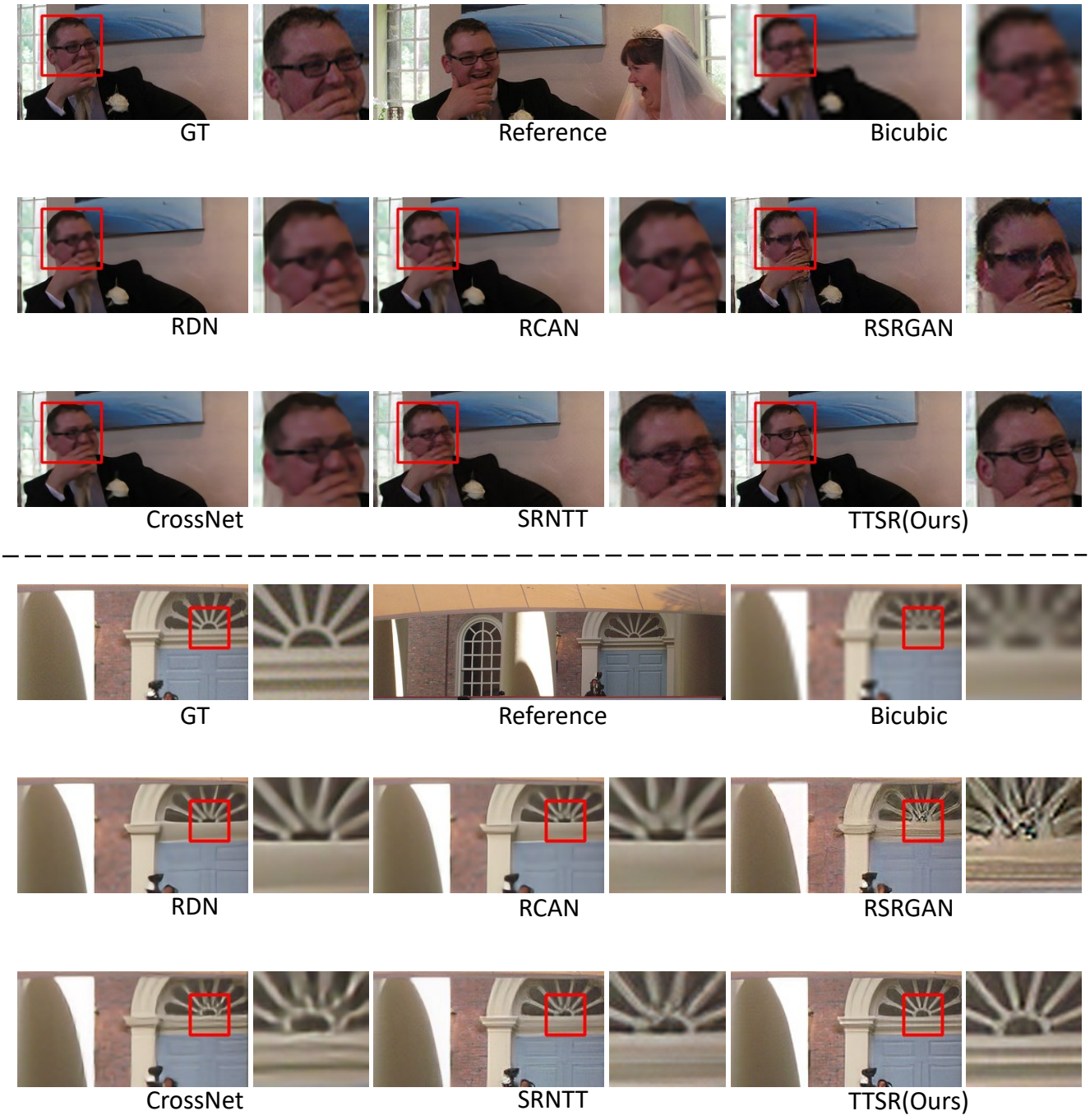

Figure 4. Visual comparison of different SR methods on CUFED5 [7] dataset.

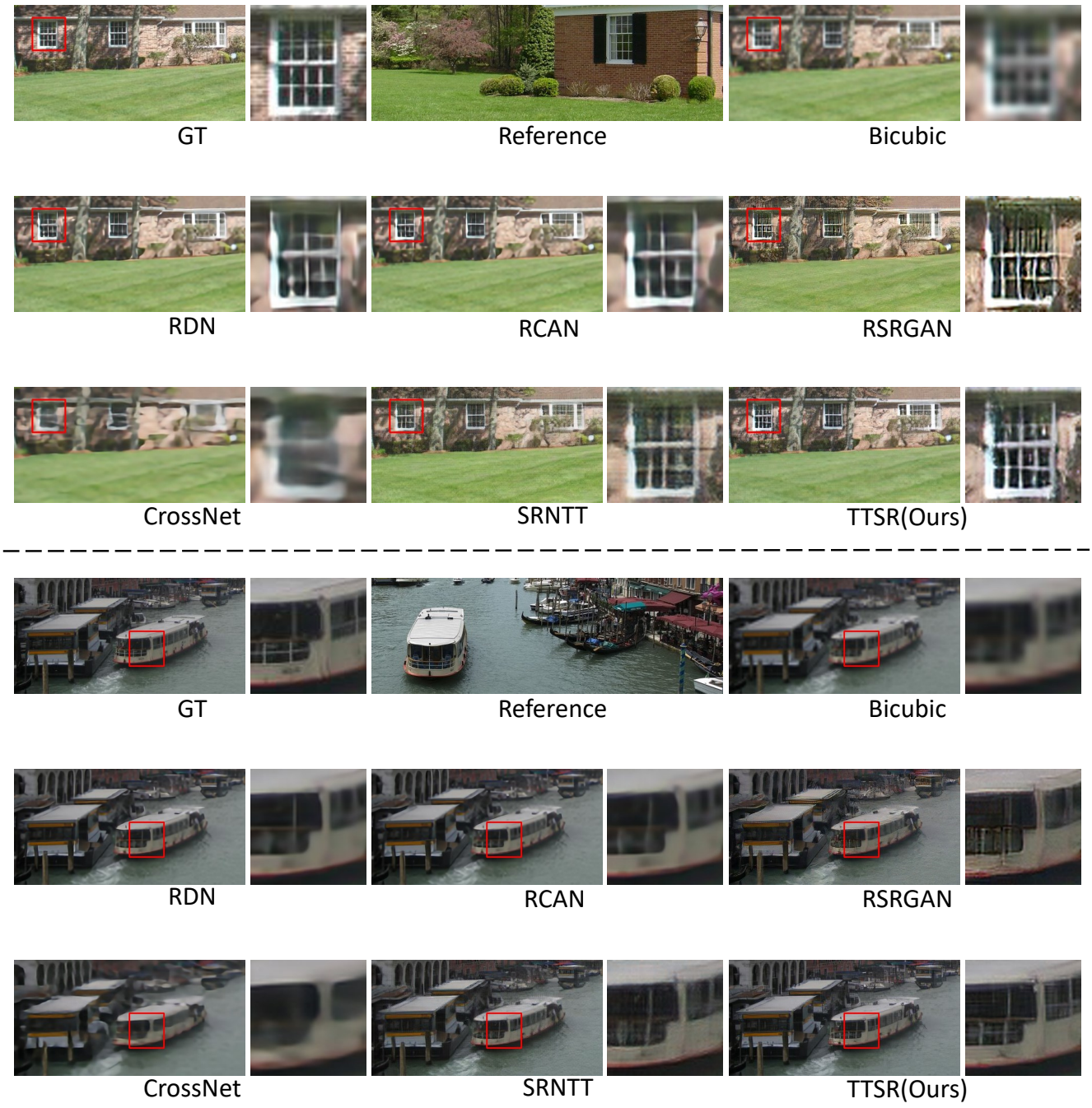

Figure 5. Visual comparison of different SR methods on Sun80 [3] dataset.

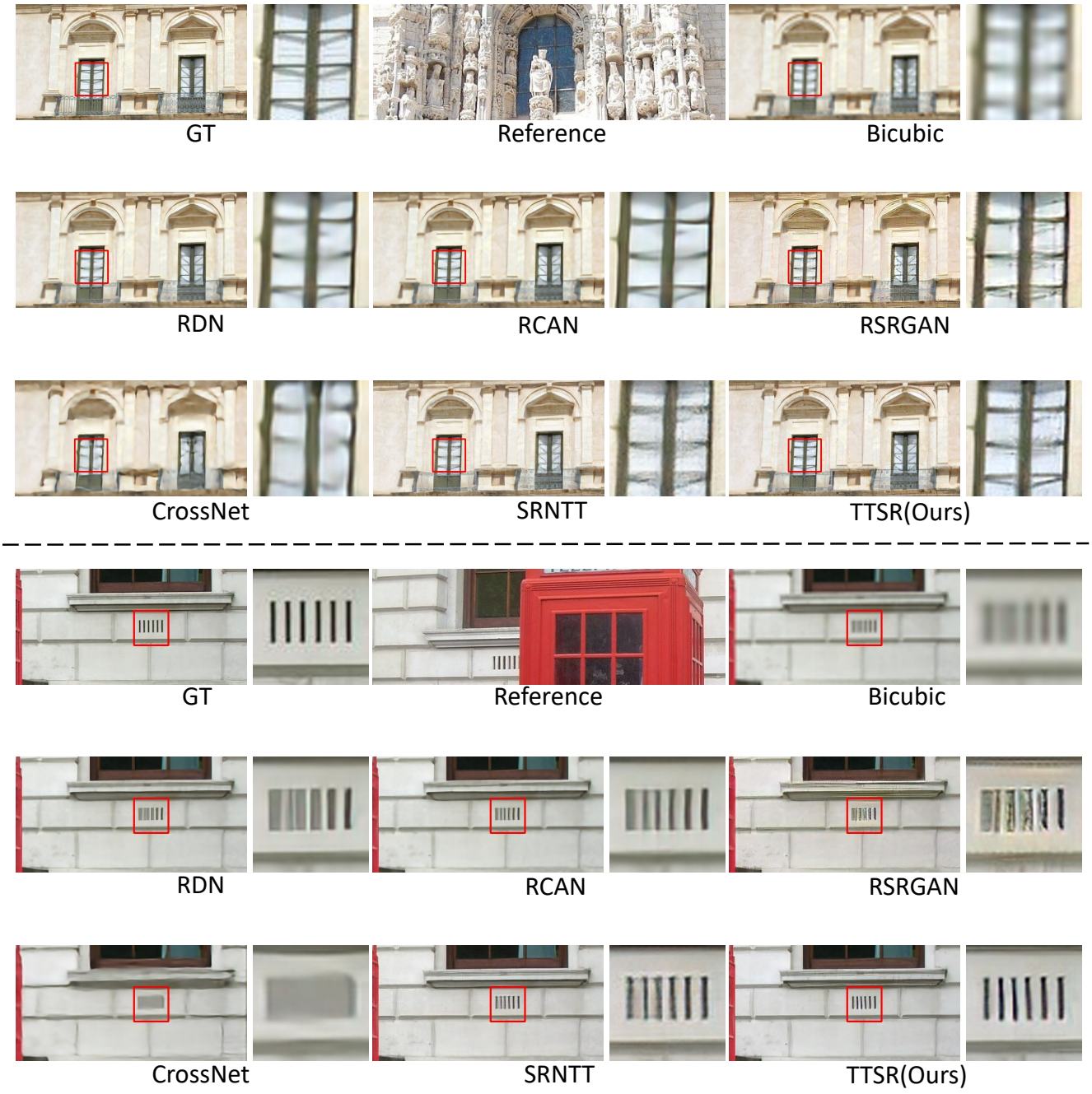

Figure 6. Visual comparison of different SR methods on Sun80 [3] dataset.

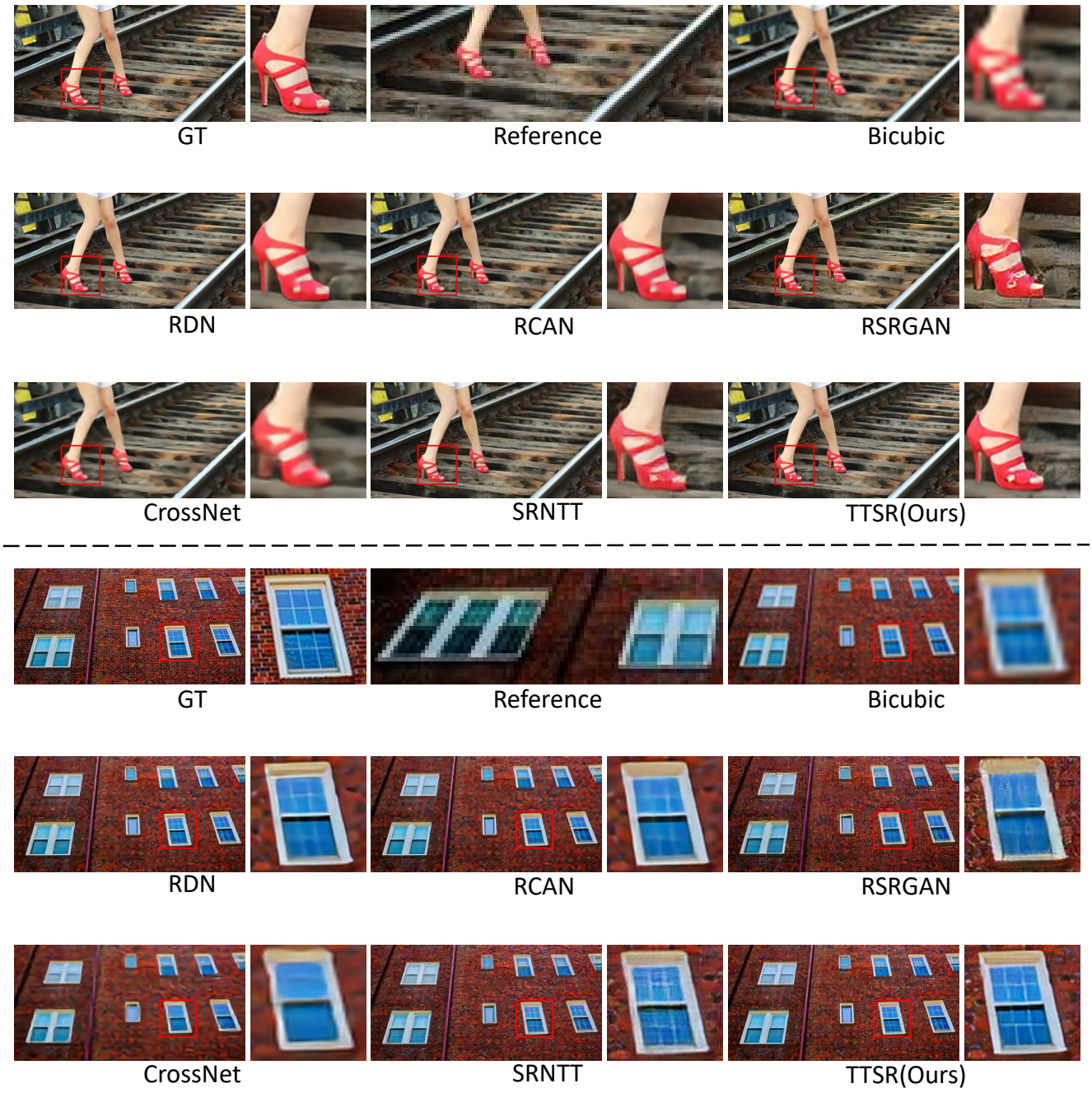

Figure 7. Visual comparison of different SR methods on Urban100 [1] dataset.

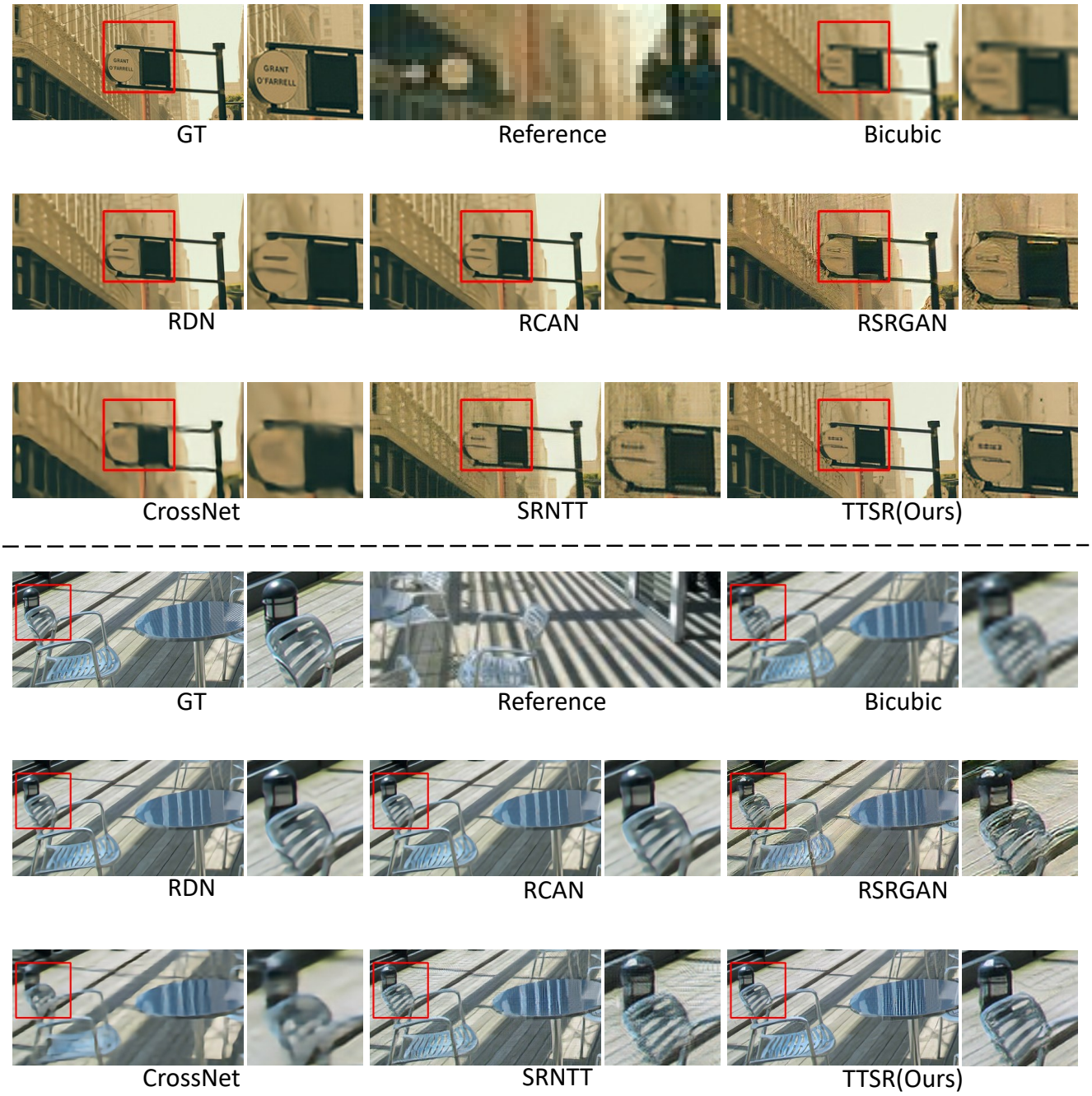

Figure 8. Visual comparison of different SR methods on Urban100 [1] dataset.

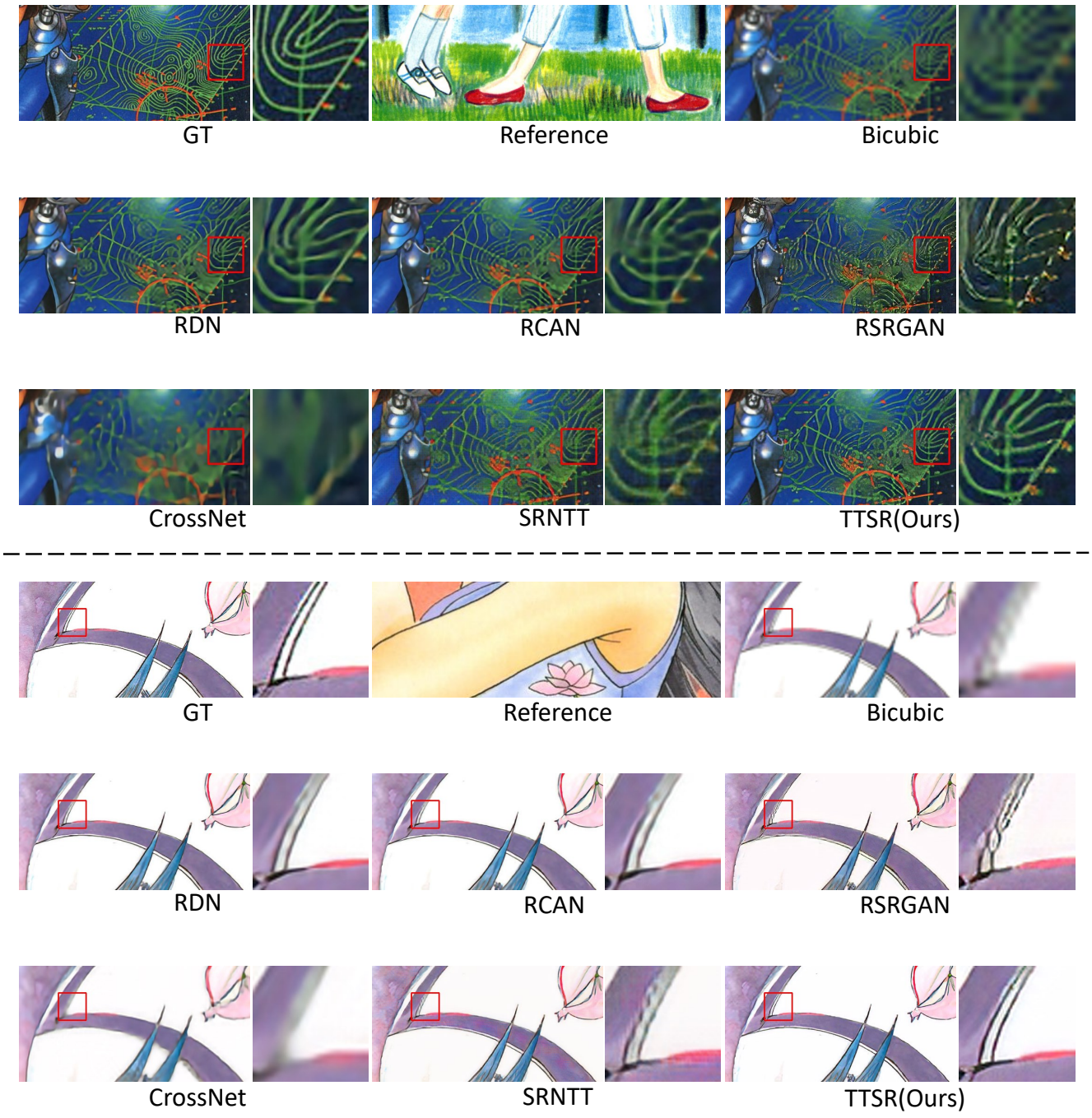

Figure 9. Visual comparison of different SR methods on Manga109 [2] dataset.

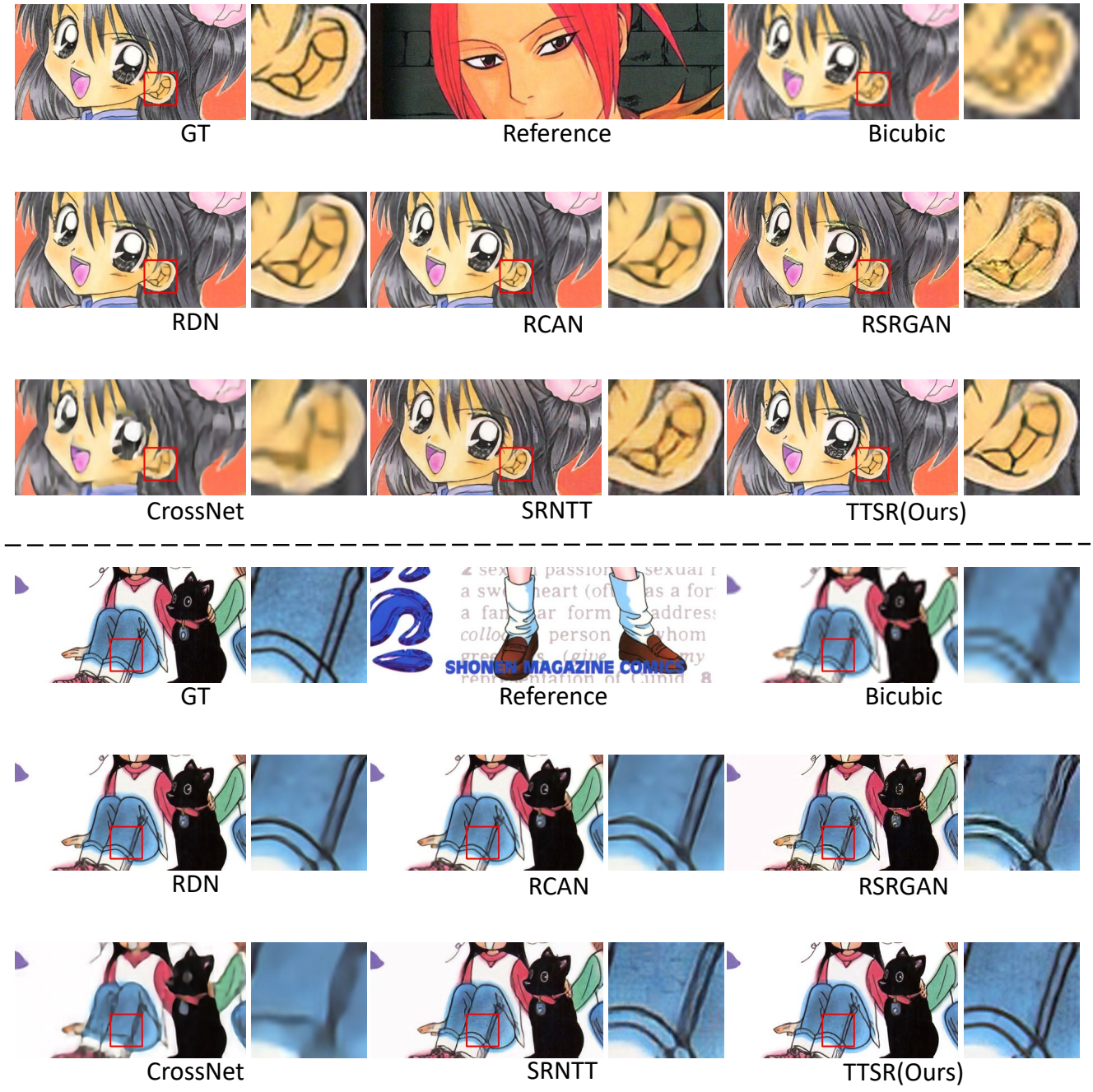

Figure 10. Visual comparison of different SR methods on Manga109 [2] dataset.