CAMixerSR: Only Details Need More "Attention"

Supplementary Material

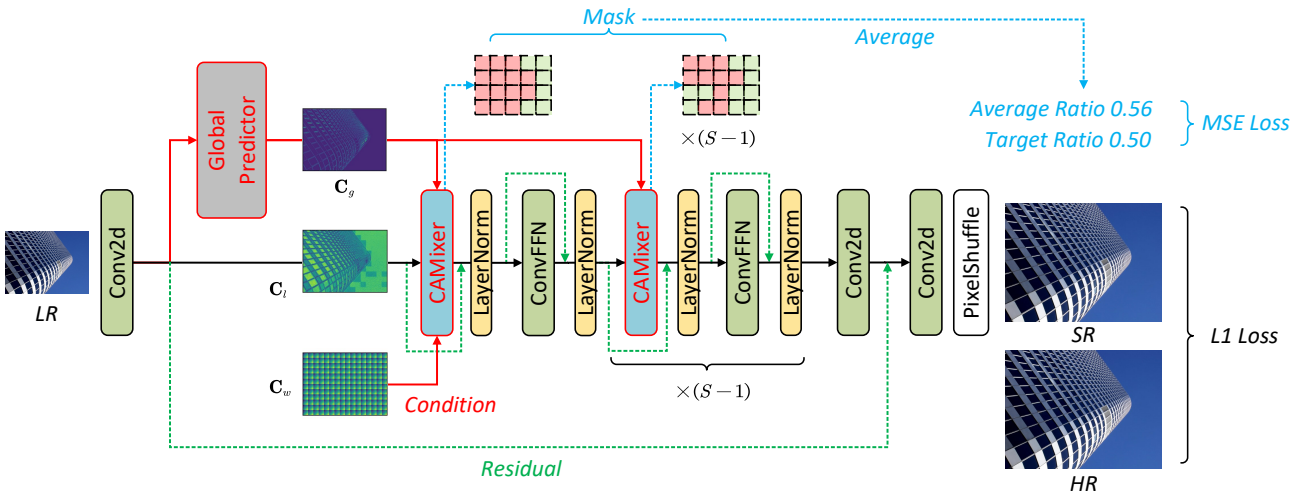


Figure 2. Overview of the proposed CAMixerSR framework. The network architecture is based on SwinIR-light [10] but replaces the window-based self-attention with the proposed CAMixer and adds an extra global predictor. (The CAMixer is stacked by group, omitted here for simplicity.) The training framework utilizes two losses, the common ℓ_1 loss for image restoration and MSE loss for predictor.

1. More Implementation Details

1.1. Network Architecture

We visualize the overall framework of the proposed CAMixerSR in Fig. 2. As discussed in the main paper, the CAMixerSR is a modified SwinIR-light [10] that uses CAMixer and a global predictor. In general, the CAMixerSR consists of four parts: shallow extractor, deep extractor, reconstruction module, and additional global predictor.

Shallow Extractor (SE). Following previous work [11, 19], given the input low-resolution (LR) image $\mathbf{I}_{LR} \in \mathbb{R}^{3\times H\times W}$, we employ a 3×3 convolution as the shallow extractor to obtain the initial feature:

$$\mathbf{F}_0 = f_{Conv}(\mathbf{I}_{LR}) \in \mathbb{R}^{C\times H\times W}. \tag{1}$$

Global Predictor (GP). Based on the $\mathbf{F}_0$, we employ a global predictor to generate global condition $\mathbf{C}_g$, which is illustrated in the main paper and stacked by two vanilla convolutions.

$$\mathbf{C}_g = f_{Global}(\mathbf{F}_0) \in \mathbb{R}^{2\times H\times W}. \tag{2}$$

Deep Extractor (DE). Similar to SwinIR [10], we stack the proposed CAMixers and Convolutional Feed-Forward Network (ConvFFN) to accomplish deep feature extraction. Specifically, our DE utilizes the Swinv2 [12] design for the basic block f_{Block}. Given the input feature $\mathbf{F}$ and corresponding condition maps $\mathbf{C}_g$ and $\mathbf{C}_w$, this process can be expressed by:

$$\begin{aligned} \mathbf{F} &= f_{LN}(f_{CAMixer}(\mathbf{F}, \mathbf{C}_g, \mathbf{C}_w) + \mathbf{F}), \\ \mathbf{F} &= f_{LN}(f_{FFN}(\mathbf{F}) + \mathbf{F}), \end{aligned} \tag{3}$$

where $f_{LN}(\cdot)$ represents layer normalization. $f_{CAMixer}(\cdot)$ and $f_{FFN}(\cdot)$ are CAMixer and ConvFFN, respectively.

Then, we stacks total S blocks by group G={4,4,6,6} to capture the immediate feature $\mathbf{F}_i$, which is formulated by:

$$\mathbf{F}_i = \begin{cases} f_{Block_i}(\mathbf{F}_{i-1}), \\ f_{Conv_j}(f_{Block_i}(\mathbf{F}_{i-1})) + \mathbf{F}_{i-G_j}, \end{cases} i = 1, 2, ..., S \tag{4}$$

where the bottom equation comes into force when it is the tail of the group, *i.e.*, i={4,10,14,20}.

Reconstruction Module (RM). Following SwinIR-light, we adopt the simplest uscale module to reconstruct the super-resolution image from the captured deep feature:

$$\mathbf{I}_{SR} = f_{RM}(\mathbf{F}_S + \mathbf{F}_0) \in \mathbb{R}^{3\times sH\times sW}, \tag{5}$$

where $f_{RM}(\cdot)$ is implemented by a 3×3 convolution to squeeze the channel number C to $3s^2$, and a pixel shuffle operator to transfer depth to space. s indicates the upscale factor.

Algorithm 1: Training/Inference of CAMixer

Data: feature $\mathbf{X}$, global condition $\mathbf{C}_g$, window condition $\mathbf{C}_w$
Result: refined feature $\mathbf{Y}$
calculate the *value*: $\mathbf{V} = \mathbf{C}_l = f_{PWConv}(\mathbf{X})$;
Predictor: use conditions ($\mathbf{C}_l$, $\mathbf{C}_g$, $\mathbf{C}_w$) to calculate metrics (mask m, offsets Δp, attentions $\mathbf{A}_c$ and $\mathbf{A}_s$) based on Eq. 2 of main paper;
calculate warped feature $\tilde{\mathbf{X}}$ by using offsets Δp and bilinear interpolation $\phi(\cdot)$:
if *training* **then**
  modulate mask by *gumble_softmax* function [16]: $\mathbf{M} = gumble\_softmax(m)$;
  hard tokens: $\tilde{\mathbf{X}}_{hard} = \tilde{\mathbf{X}} \cdot \mathbf{M}$, $\mathbf{V}_{hard} = \mathbf{V} \cdot \mathbf{M}$;
  simple tokens: $\mathbf{V}_{simple} = \mathbf{V} \cdot (1 - \mathbf{M})$;
else
  calculate K by $\sum \mathbf{M}$;
  obtain index by *argsort* function: $I = argsort(m)$, $I_{hard} = I[:K]$, $I_{simple} = I[K:]$;
  hard tokens: $\tilde{\mathbf{X}}_{hard} = \tilde{\mathbf{X}}[I_{hard}]$, $\mathbf{V}_{hard} = \mathbf{V}[I_{hard}]$;
  simple tokens: $\mathbf{V}_{simple} = \mathbf{V}[I_{simple}]$;
end
calculate *query* and *key* by: $\tilde{\mathbf{Q}} = \tilde{\mathbf{X}}_{hard}\mathbf{W}_q$, $\tilde{\mathbf{K}} = \tilde{\mathbf{X}}_{hard}\mathbf{W}_k$;
Attention: use self-attention for complex areas: $\mathbf{V}_{hard} = softmax(\frac{\tilde{\mathbf{Q}}\tilde{\mathbf{K}}^T}{\sqrt{d}})\mathbf{V}_{hard}$;
use convolutional sptial attention for simple areas: $\mathbf{V}_{simple} = \mathbf{V}_{simple} \cdot \mathbf{A}_s$;
if *training* **then**
  $\mathbf{V}_{attn} = \mathbf{V}_{hard} + \mathbf{V}_{simple}$;
else
  $\mathbf{V}_{attn}[I_{hard}] = \mathbf{V}_{hard}$, $\mathbf{V}_{attn}[I_{simple}] = \mathbf{V}_{simple}$;
end
Convolution: calculate convolution and channel attention: $\mathbf{V}_{conv} = f_{DWConv}(\mathbf{V}_{attn}) \cdot \mathbf{A}_c + \mathbf{V}_{conv}$;
project to obtain output $\mathbf{Y} = f_{PWConv}(\mathbf{V}_{attn})$.

1.2. Training and Inference of CAMixer

For the proposed CAMixer, we utilize two implementations for training and inference as shown in Algorithm 1.

Inference. For inference, as formulated in the main paper (Eq. 4), we use the *argsort* to obtain the indices, and then select the top-K tokens to calculate the self-attention. Despite directness and simplicity, this process is non-differentiable.

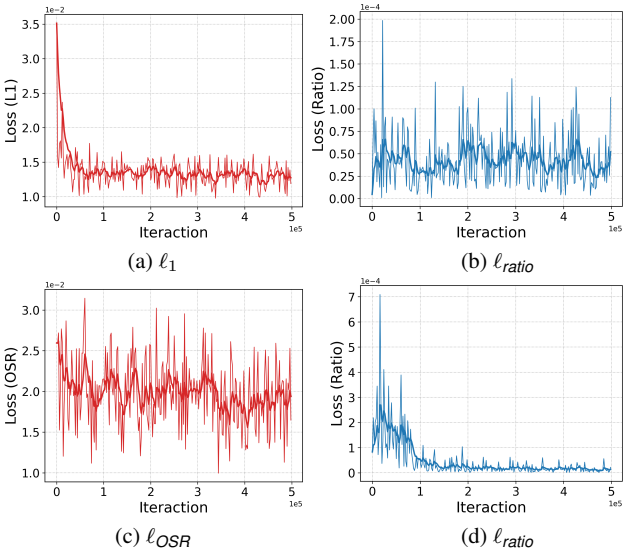
Training. Following DynamicViT [16], we leverage *gumble_softmax* function to generate differentiable 0-1 mask $\mathbf{M}$ for training, where the index "1" represents the mask of the tokens processed by self-attention. Moreover, *gumble_softmax* function generates one-hot tensor, of which the expectation equals m exactly. Specifically, to enable the dynamical adjustment of the attention ratio γ, the dimension of the softmax is 1 rather than 0 for $m' \in \mathbb{R}^{\frac{HW}{M^2} \times 2}$, where m' is the original output from the predictor.

1.3. Training Loss

Training Loss for ODI SR. Based on LAU-Net [3] and OSRT [17], we utilize the weighted ℓ_1 loss for reconstruction. Given the input LR-HR pairs $\{I_i^{LR}, I_i^{HR}\}_{i=1}^N$, this calculation can be formulated by:

$$\ell_{OSR} = \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{W}_j\left(\mathbf{I}_i^{HR} - f_{CAMixerSR}(\mathbf{I}_i^{LR})\right)\right\|_1, \quad (6)$$

where the $\mathbf{W}_j$ is the weight matrix which defines the importance of each pixel according to its latitude. Given the latitude of the p-th row in $\mathbf{W}_j$ is l, following [3], we calculate its weight by $cos(\frac{l+0.5-H/2}{H}\pi)$. Similar to the classic SR task, we simply sum the ℓ_{OSR} and ℓ_{ratio} as the overall loss to train model for ODI SR.



(a) ℓ_1 (b) ℓ_{ratio} (c) ℓ_{OSR} (d) ℓ_{ratio}

Figure 3. The loss curves for classic SR (a, b) and ODI SR (c, d).

More Discussion. We visualize the loss curves in Fig. 3 with two different tasks: classic SR and ODI SR task. Generally, for ℓ_{ratio}, the curve for ODI-SR is more reasonable and stable since 360° images have more plain area at high latitude while the density of complex area is random for images from classic SR datasets. This property also induces the difference of ℓ_1 and ℓ_{OSR}, where (a) is smoothly descended with fewer oscillations than (c).

1.4. Texture Inconsistency

The texture inconsistency may induce a huge performance drop since the CAMixer dynamically selects "details" tokens to calculate WSA. As discussed in the main paper, we resolve the texture inconsistency from two perspectives. 1) Design, we utilize the convolutional spatial/channel attention acting as a simple alignment for attentive features. 2) Training, the network itself learns to harmonize the feature by distributing CAMixer processing varied tokens. In Fig. 5, we offer heatmaps to show how these strategies work to erase the potential inconsistency. Specifically, for the model with convolutional attention, the difference between complex and simple tokens is alleviated to a large extent

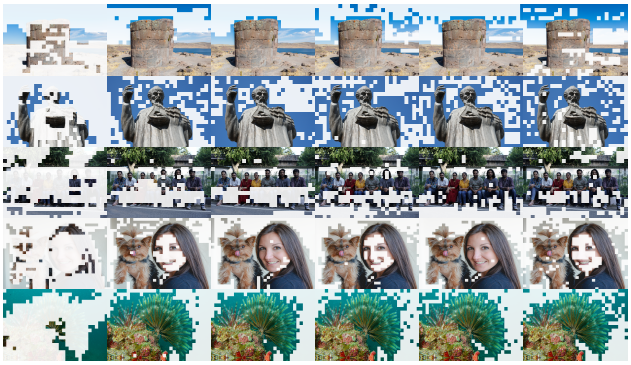
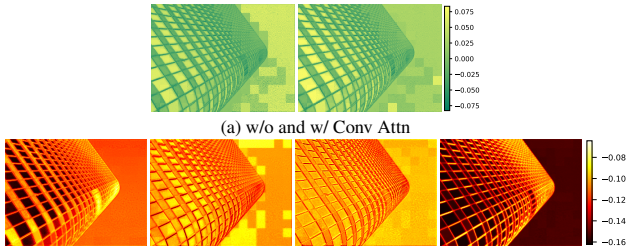


Figure 4. More visualizations of progressively classified tokens.



(a) w/o and w/ Conv Attn

(b) heatmaps for sampled {4,8,14,20}-th CAMixers

Figure 5. Effects of convolutional attention and block schedule.

but still exists for uncoordinated patches. Then, as (b) illustrates, CAMixers of different layers hierarchically erase the inconsistency.

2. More Results

2.1. More Visualization of Predicted Mask

In Fig. 4, we present more visual results of the predicted mask with γ=0.5. For images with plenty of plain areas, our CAMixer can adopt eligible partitions for SA/Conv. However, two defects remain to be solved in our future work. 1) The fixed γ is not flexible for images with excessively complex/simple textures. 2) The partition is learned from data without guidance, while some plain areas, *e.g.*, human face, deserve more "attention". In the future, we will continue to refine CAMixer with the adjustable ratio γ and the human-guiding partition.

2.2. Runtime Performance

Initially, we show the runtime percentage for components of CAMixer in Fig. 6. Obviously, the self-attention branch is the main barrier (67.8%) that constrains efficiency. Thus, CAMixer integrates content-aware routing to reduce the latency for attention. We validate the runtime performance of the proposed CAMixer on efficient SR tasks with the same setting as the NTIRE ESR Challenge[1] [9]. In Tab. 1,

[1] https://github.com/ofsoundof/NTIRE2022_ESR

Device	Training	Inference		
		$\gamma = 1.00$	$\gamma = 0.50$	$\gamma = 0.25$
#MAdds	-	77.9G	53.8G (69%)	43.0G (55%)
CPU†	11.6s	13.2s	9.6s (73%)	9.0s (68%)
Tesla T4	214.9ms	227.7ms	180.6ms (79%)	177.3ms (78%)
Tesla A10	106.3ms	113.4ms	97.4ms (86%)	91.6ms (81%)
Tesla V100	95.3ms	97.4ms	90.2ms (92%)	88.7ms (91%)

Table 1. Runtime performance for CAMixerSR with various γ on multiple devices. Mult-Adds (MAdds) are measured under the setting of upscaling the image to 1280×720. The latency is the average runtime for single image SR on Urban100 [5]. "†" uses Set5 [1] and single core of Intel Xeon Platinum 8336C@2.3GHz.

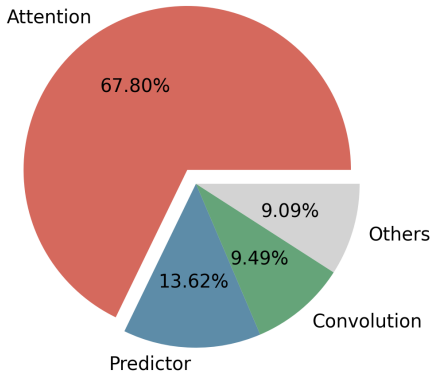


Figure 6. Runtime percentage of varied components (predictor, attention, convolution, and others) of CAMixer ($\gamma = 1.0$).

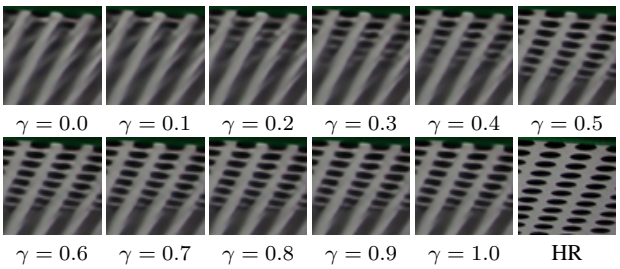


Figure 7. Visual comparison for varied attention ratio γ.

we examine our CAMixerSR with varied devices on Urban100 [5]. We can observe that the training mode is slightly faster than the inference ($\gamma = 1.00$) due to the inference executing an extra selection operation to classify tokens. For latency, when testing on CPU or weak GPU (*e.g.*, T4), the latency reduction is similar to MAdds, *i.e.*, 30% for $\gamma = 0.5$ and 40% for $\gamma = 0.25$. Due to device limitations, the consumer-grade GPUs, *e.g.* GTX and RTX series, are not included, which may attain larger improvements than T4. For more powerful GPU (*e.g.*, V100), the runtime decrease is rather limited, less than 10%. The results indicate that, on devices with low FLOPS barriers, our CAMixer can effectively save the computations and running time.

Table 2. Quantitative comparison (PSNR) for CAMixerSR-**Small/Medium/Base** with varied ratio γ on F2K, Test2K, Test4K, and Test8K.

	Ratio γ	#Params	F2K	#FLOPs	Test2K	#FLOPs	Test4K	#FLOPs	Test8K	#FLOPs
Small	1.00	351K	29.12	894M (100%)	26.26	894M (100%)	27.73	894M (100%)	33.66	894M (100%)
	0.50		29.08	652M (73%)	26.24	652M (73%)	27.70	652M (73%)	33.63	652M (73%)
	0.25		28.98	532M (59%)	26.18	532M (59%)	27.63	532M (59%)	33.55	532M (59%)
	0.00		28.83	410M (46%)	26.10	410M (46%)	27.52	410M (46%)	33.43	410M (46%)
Medium	1.00	535K	29.20	1.37G (100%)	26.32	1.37G (100%)	27.80	1.37G (100%)	33.72	1.37G (100%)
	0.50		29.18	1.03G (75%)	26.30	1.03G (75%)	27.79	1.03G (75%)	33.71	1.03G (75%)
	0.25		29.11	858M (62%)	26.26	858M (62%)	27.74	858M (62%)	33.66	858M (62%)
	0.00		28.92	686M (50%)	26.15	686M (50%)	27.59	686M (50%)	33.50	686M (50%)
Base	1.00	765K	29.35	1.96G (100%)	26.40	1.96G (100%)	27.89	1.96G (100%)	33.81	1.96G (100%)
	0.50		29.32	1.49G (76%)	26.39	1.49G (76%)	27.87	1.49G (76%)	33.81	1.49G (76%)
	0.25		29.26	1.26G (65%)	26.35	1.26G (65%)	27.83	1.26G (65%)	33.77	1.26G (65%)
	0.00		29.08	1.03G (53%)	26.23	1.03G (53%)	27.70	1.03G (53%)	33.63	1.03G (53%)

Table 3. Quantitative comparison (average PSNR/SSIM, Parameters, and Mult-Adds) with varied ratio γ for efficient image SR. Mult-Adds (MAdds) are measured under the setting of upscaling the image to 1280×720. "†" indicates using the DF2K [11] training set.

Ratio γ	Scale	#Params	#MAdds	Set5 [1]		Set14 [18]		BSD100 [14]		Urban100 [5]		Manga109 [15]	
				PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
1.00	×2	746K	294.1G	38.24	0.9614	34.00	0.9217	32.34	0.9017	32.97	0.9352	39.34	0.9782
0.50	×2		205.2G	38.23	0.9613	34.00	0.9214	32.34	0.9016	32.95	0.9348	39.32	0.9781
0.25	×2		160.7G	38.16	0.9610	33.90	0.9206	32.31	0.9010	32.78	0.9329	39.25	0.9779
1.00†	×2	746K	294.1G	**38.28**	**0.9614**	**34.04**	**0.9218**	**32.37**	**0.9021**	**33.04**	**0.9364**	**39.50**	**0.9788**
0.50†	×2		205.2G	38.27	0.9614	34.03	0.9215	32.36	0.9019	33.01	0.9357	39.49	0.9787
0.25†	×2		160.7G	38.21	0.9611	33.96	0.9208	32.33	0.9013	32.83	0.9336	39.43	0.9785
1.00	×4	765K	77.9G	32.51	0.8992	28.82	0.7873	27.73	0.7421	26.65	0.8024	31.20	0.9170
0.50	×4		53.8G	32.51	0.8988	28.82	0.7870	27.72	0.7416	26.63	0.8012	31.18	0.9166
0.25	×4		43.0G	32.45	0.8978	28.78	0.7856	27.69	0.7401	26.51	0.7966	31.06	0.9148
1.00†	×4	765K	77.9G	**32.60**	**0.9003**	**28.91**	**0.7889**	**27.78**	**0.7434**	**26.80**	**0.8068**	**31.42**	0.9168
0.50†	×4		53.8G	32.58	0.9000	28.90	0.7885	27.77	0.7430	26.77	0.8055	31.41	**0.9171**
0.25†	×4		43.0G	32.47	0.8986	28.84	0.7870	27.73	0.7413	26.63	0.8005	31.31	0.9168

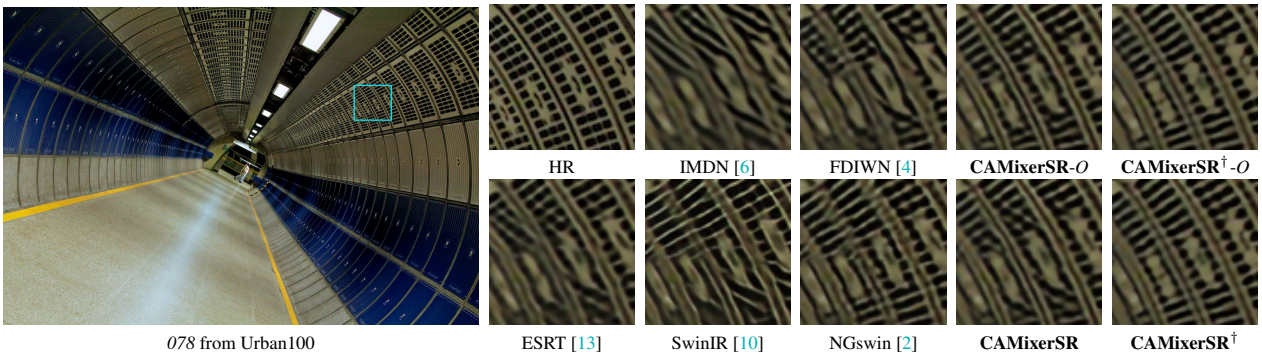


Figure 8. Visual comparison of CAMixerSR with other methods for ×4 task on Urban100 dataset.

2.3. Large-Image SR

In Tab. 2, we offer more quantitative results of CAMixerSR with varied γ on the Large-Image SR task. In the main paper, we manually set $\gamma = 0.5$ to attain the promising trade-offs for three tasks. However, for the 8K task, further decreasing the ratio to 0.25 can save an additional 115M (10%) calculations while inducing only 0.04dB drops. We also examine the models without using self-attention ($\gamma = 0$). In detail, the non-attention models suffer about 0.3dB PSNR drops. In Fig. 9, we offer more visual comparisons between our CAMixerSR with other methods, where CAMixerSR obtains better restoration quality. These results indicate that only details need more "attention", and we only need to pay "attention" to 25%-50% areas.

2.4. Lightweight SR

In Tab. 3 and Fig. 7, we supply the quantitative and qualitative results of CAMixerSR with varied γ on the Lightweight SR task. Different from Sec. 2.3, the CAMixer with $\gamma = 0.25$ encounters more extensive PSNR drops (0.06-0.14dB)

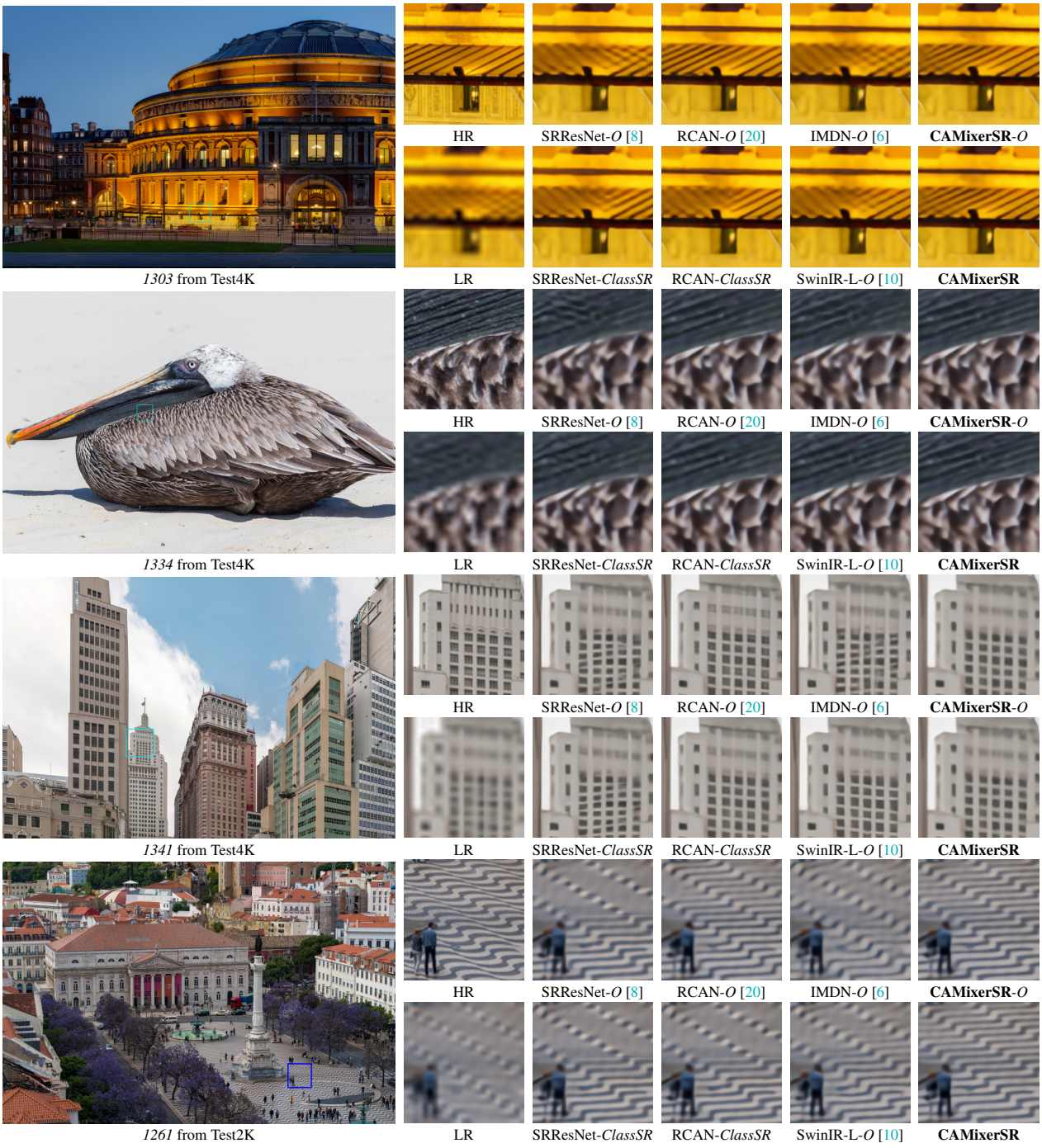


Figure 9. Visual comparison of CAMixerSR with other methods for $\times 4$ task on Test2K and Test4K dataset.

on benchmark datasets. Moreover, we train our CAMixerSR with a large-scale training set, DF2K [11] to explore and exploit the maximum representation capability. Similar to previous work [10], using DF2K significantly improves the restoration quality for the baseline model ($\gamma = 1.0$). In detail, the PSNR increases by 0.22dB on Manga109 [15]. More importantly, for models trained with large-scale sets, reducing the attention area (γ) can also maintain remarkable performance as using small-scale sets. In conclusion, improving the training schedule would not affect the CAMixer, showing its generality and robustness.

References

[1] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie-Line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *BMVC*, pages 1–10, 2012. 3, 4

[2] Haram Choi, Jeongmin Lee, and Jihoon Yang. N-gram in swin transformers for efficient lightweight image super-resolution. In *CVPR*, pages 2071–2081, 2023. 4

[3] Xin Deng, Hao Wang, Mai Xu, Yichen Guo, Yuhang Song, and Li Yang. Lau-net: Latitude adaptive upscaling network for omnidirectional image super-resolution. In *CVPR*, pages 9189–9198, 2021. 2

[4] Guangwei Gao, Wenjie Li, Juncheng Li, Fei Wu, Huimin Lu, and Yi Yu. Feature distillation interaction weighting network for lightweight image super-resolution. In *Proceedings of the AAAI conference on artificial intelligence*, pages 661–669, 2022. 4

[5] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *CVPR*, pages 5197–5206, 2015. 3, 4

[6] Zheng Hui, Xinbo Gao, Yunchu Yang, and Xiumei Wang. Lightweight image super-resolution with information multi-distillation network. In *ACM MM*, pages 2024–2032, 2019. 4, 5

[7] Xiangtao Kong, Hengyuan Zhao, Yu Qiao, and Chao Dong. Classsr: A general framework to accelerate super-resolution networks by data characteristic. In *CVPR*, pages 12016–12025, 2021. 1

[8] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, pages 4681–4690, 2017. 5

[9] Yawei Li, Kai Zhang, Luc Van Gool, Radu Timofte, et al. Ntire 2022 challenge on efficient super-resolution: Methods and results. In *CVPRW*, 2022. 3

[10] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *ICCVW*, pages 1833–1844, 2021. 1, 4, 5

[11] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *CVPRW*, pages 1132–1140, 2017. 1, 4, 5

[12] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12009–12019, 2022. 1

[13] Zhisheng Lu, Juncheng Li, Hong Liu, Chaoyan Huang, Linlin Zhang, and Tieyong Zeng. Transformer for single image super-resolution. In *CVPRW*, pages 457–466, 2022. 4

[14] David R. Martin, Charless C. Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, pages 416–425, 2001. 4

[15] Yusuke Matsui, Kota Ito, Yuji Aramaki, Azuma Fujimoto, Toru Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. Sketch-based manga retrieval using manga109 dataset. *Multim. Tools Appl.*, 76(20):21811–21838, 2017. 4, 5

[16] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *NeurIPS*, 34:13937–13949, 2021. 2

[17] Fanghua Yu, Xintao Wang, Mingdeng Cao, Gen Li, Ying Shan, and Chao Dong. Osrt: Omnidirectional image super-resolution with distortion-aware transformer. In *CVPR*, pages 13283–13292, 2023. 2

[18] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *Curves and Surfaces - 7th International Conference*, pages 711–730, 2010. 4

[19] Xindong Zhang, Hui Zeng, Shi Guo, and Lei Zhang. Efficient long-range attention network for image super-resolution. In *ECCV*, pages 649–667, 2022. 1

[20] 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 294–310, 2018. 5