Learning A Single Network for Scale-Arbitrary Super-Resolution

Longguang Wang, Yingqian Wang, Zaiping Lin, Jungang Yang, Wei An, Yulan Guo*
National University of Defense Technology
{wanglongguang15,yulan.guo}@nudt.edu.cn

Abstract

Recently, the performance of single image super-resolution (SR) has been significantly improved with powerful networks. However, these networks are developed for image SR with specific integer scale factors (e.g., $\times 2/3/4$), and cannot handle non-integer and asymmetric SR. In this paper, we propose to learn a scale-arbitrary image SR network from scale-specific networks. Specifically, we develop a plug-in module for existing SR networks to perform scale-arbitrary SR, which consists of multiple scale-aware feature adaption blocks and a scale-aware upsampling layer. Moreover, conditional convolution is used in our plug-in module to generate dynamic scale-aware filters, which enables our network to adapt to arbitrary scale factors. Our plug-in module can be easily adapted to existing networks to realize scale-arbitrary SR with a single model. These networks plugged with our module can produce promising results for non-integer and asymmetric SR while maintaining state-of-the-art performance for SR with integer scale factors. Besides, the additional computational and memory cost of our module is very small.

1. Introduction

Single image super-resolution (SR) aims at recovering a high-resolution (HR) image from its low-resolution (LR) counterpart. As a long-standing low-level computer vision problem, single image SR has been investigated for decades [1, 2, 3, 4, 5, 6]. Recently, the rise of deep learning provides a powerful tool to solve this problem, with numerous CNN-based methods [7, 8, 9, 10, 11] being developed to improve the SR performance.

Although recent CNN-based single image SR networks [6, 10, 12, 13] have achieved promising performance, they are developed for image SR with specific integer scale factors (*e.g.*, $\times 2/3/4$). In many real-world applications like image retargeting, image editing and artworks, non-integer SR (*e.g.*, from 100×100 to 220×220) and asymmetric SR (*e.g.*, from 100×100 to 220×420) are highly demanded. However, due to the fixed filters in upscale modules, most existing networks can only zoom in an image with specific integer scales and cannot handle scale-arbitrary SR in real-world scenarios.

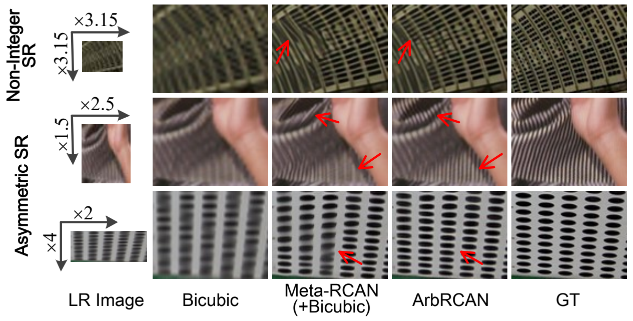

Figure 1. Visual comparison achieved by Bicubic, Meta-RCAN [14] and our ArbRCAN. "(+Bicubic)" means that the network output is further resized to the expected resolution using bicubic interpolation for asymmetric scale factors.

To address this limitation, Hu *et al.* [14] proposed a Meta-SR network to dynamically predict filters in the upscale module for different scale factors using meta-learning. As a result, Meta-SR achieves promising performance on non-integer scale factors. However, scale information is only used for upsampling in Meta-SR. That is, features in the backbone are the same for all SR tasks with different scale factors, which hinders the further improvement of performance. Moreover, Meta-SR focuses on SR with non-integer scale factors but cannot handle SR with asymmetric scale factors.

In this paper, we propose to learn a scale-arbitrary single image SR network from scale-specific networks. Specifically, we develop a plug-in module for existing SR networks to enable scale-arbitrary SR, which consists of multiple scale-aware feature adaption blocks and a scale-aware upsampling layer. The scale-aware feature adaption blocks are used to adapt features in the backbone to specific scale factors and the scale-aware upsampling layer is used for scale-arbitrary upsampling. Within our plug-in module, conditional convolutions are used to generate dynamic scale-aware filters to handle different scale factors. Our plug-in module can be easily adapted to existing networks for scale-arbitrary SR with small additional computational and memory cost. Baseline networks equipped with our module can produce promising results for non-integer and asymmetric SR (Fig. 1), while maintaining state-of-the-art performance on integer scale factors with a single model. To the best of our knowledge, our plug-in module is the first work to handle asymmetric SR.

Our main contributions can be summarized as follows: 1) We develop a plug-in module for existing SR networks to achieve scale-arbitrary SR, including multiple scale-aware feature adaption blocks and a scale-aware upsampling layer. 2) Our plug-in module uses conditional convolution to dynamically generate filters based on the input scale information, which facilitates our network to adapt to specific scale factors. 3) Experimental results show that baseline networks equipped with our module produce promising results for scale-arbitrary SR with only a single model. A video demo is provided in the supplemental material.

2. Related Work

In this section, we first briefly review several major works for CNN-based single image SR. Then, we discuss conditional convolutions that are related to our work.

Single Image Super-Resolution. Due to the powerful feature representation and model fitting capabilities of deep neural network, CNN-based single image SR methods [7, 8, 9, 6, 10] outperform traditional methods [1, 2, 15, 3, 4, 5] significantly. Dong *et al.* [7] proposed a three-layer convolutional network (namely, SRCNN) to learn the non-linear mapping between LR images and HR images. A deeper network (namely, VDSR) with 20 layers [8] was then developed to achieve better performance. Later, Lim *et al.* [16] proposed a very deep and wide network, namely EDSR. Specifically, batch normalization (BN) layers were removed and a residual scaling technique was used to enable the training of such a large model. Recently, Zhang *et al.* [12] and Dai *et al.* [13] further improved the SR performance by introducing channel attention and second-order channel attention, respectively.

Although existing single image SR networks have achieved promising results, they are trained for SR with a single specific integer scale factor. To overcome this limitation, Lim *et al.* [16] proposed a multi-scale deep super-resolution (MDSR) system to integrate modules trained for multiple integer scale factors (*i.e.*, $\times 2/3/4$). However, MDSR cannot super-resolve images with non-integer scale factors. Recently, Hu *et al.* [14] proposed a Meta-SR network to solve the scale-arbitrary upsampling problem. Specifically, they used meta-learning to predict weights of filters for different scale factors. Nevertheless, Meta-SR does not exploit the benefits of scale information during feature learning in the backbone. To make better use of scale information, Fu *et al.* [17] introduced a residual scale attention network (RSAN), where the scale information is used as a prior knowledge to learn discriminative features for superior performance.

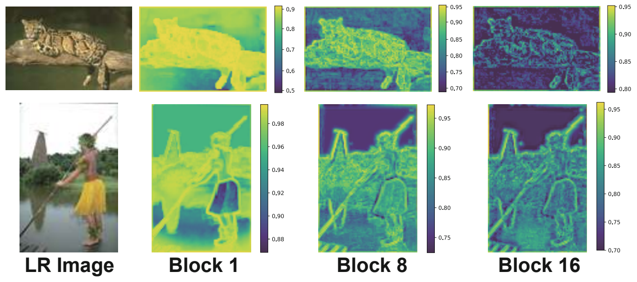

Figure 2. Visualization of feature similarity maps.

Despite Meta-SR and RSAN are able to super-resolve images with non-integer scale factors, they cannot handle asymmetric SR. In many real-world applications like image retargeting, image editing and artworks, asymmetric SR is also highly demanded. However, it is still under-investigated in literature. In this paper, we develop a plug-in module for existing SR networks to enable SR with both non-integer and asymmetric scale factors.

Conditional Convolutions. Different from traditional convolutional layers with static filters, conditional convolutions [18, 19, 20, 21] parameterize their filters conditioned on the input as linear combinations of several experts. Consequently, the capacity of the network can be efficiently improved without a significant increase in computational cost. In this paper, we extend the idea of conditional convolutions to generate dynamic scale-aware filters to handle the scale-arbitrary SR task. It is demonstrated that conditional convolutions facilitate our network to adapt to arbitrary scale factors to achieve better SR performance.

3. Methodology

3.1. Motivation

Since SR tasks with different scale factors are inter-related [16], it is non-trivial to learn a scale-arbitrary SR network from scale-specific networks (*e.g.*, $\times 2/3/4$). Early attempts [8, 22, 23] use shared features in the backbone to handle multiple scale factors without considering the scale information during feature learning. Intuitively, since the degradation is different for various scale factors, scale information can further be used to learn discriminative features to improve SR performance [17]. In this section, we investigate the relationship between $\times 2/3/4$ SR tasks to provide insights for scale-arbitrary SR.

We conduct experiments to compare the feature similarity on specific layers in pre-trained $\times 2/3/4$ SR networks. In our experiments, EDSR [16] is selected as the baseline network. First, we downsample an image to $\frac{1}{4}$ size (denoted as $I \in \mathbb{R}^{H \times W}$). Then, we feed I to EDSR networks devel-

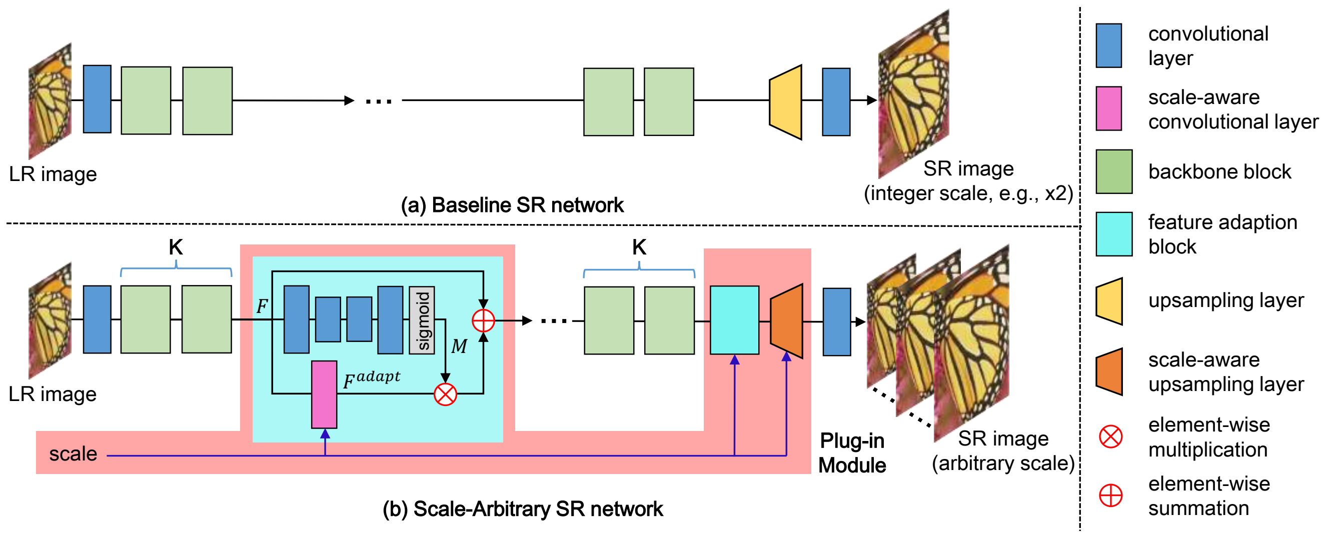

Figure 3. An overview of our plug-in module. The details of our scale-aware convolutional layer and scale-aware upsampling layer are further illustrated in Figs. 4 and 5, respectively.

oped for $\times 2/3/4$ SR. Following [24], features from the last layer in the i^{th} residual block are whitened to remove global component for more precise calculation of pairwise similarity, resulting in $F_i^{\times 2}, F_i^{\times 3}, F_i^{\times 4} \in \mathbb{R}^{256\times H\times W}$. Next, for each location p, we extract a triplet of feature samples at p to obtain $f_i^{\times 2}, f_i^{\times 3}, f_i^{\times 4} \in \mathbb{R}^{256}$ and compute the feature similarity among them:

$$S_i(p) = \frac{1}{3}\left(\frac{(f_i^{\times 2})^T f_i^{\times 3}}{\|f_i^{\times 2}\|\|f_i^{\times 3}\|} + \frac{(f_i^{\times 2})^T f_i^{\times 4}}{\|f_i^{\times 2}\|\|f_i^{\times 4}\|} + \frac{(f_i^{\times 3})^T f_i^{\times 4}}{\|f_i^{\times 3}\|\|f_i^{\times 4}\|}\right). \tag{1}$$

The feature similarity map S_i is visualized in Fig. 2. For more results, please refer to the supplemental material.

From Fig. 2, we can see that feature similarity varies for different blocks and regions. That is, the sensitivity of features to the change of scale factors is different for various blocks and regions. Consequently, we are motivated to perform pixel-wise feature adaption accordingly. For features within regions of high feature similarities, they can be directly used for SR with arbitrary scale factors. In contrast, features within regions of low feature similarities are adapted to specific scale factors. More analyses are included in the supplemental material.

3.2. Our Plug-in Module

The architecture of our plug-in module is shown in Fig. 3. Given a baseline network (*e.g.*, EDSR) developed for SR with integer scale factors, we can extend it to a scale-arbitrary SR network using our plug-in module. Specifically, scale-aware feature adaption is performed after every K backbone blocks, as shown in Fig. 3(b). Following the backbone module, a scale-aware upsampling layer is used for scale-arbitrary upsampling.

Scale-Aware Feature Adaption. Given a feature map F, it is first fed to an hourglass module with four convolutions and a sigmoid layer to generate a guidance map M with values ranging from 0 to 1, as shown in Fig. 3(b). Then, F is fed to a scale-aware convolution for feature adaption, resulting in an adapted feature map F^{adapt}. Next, the guidance map M is used to fuse F and F^{adapt} as:

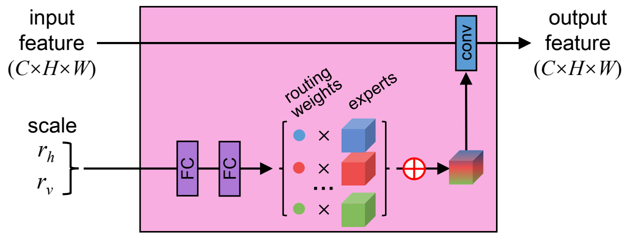

Figure 4. An illustration of our scale-aware convolutional layer.

$$F^{fuse} = F + F^{adapt} \times M. \tag{2}$$

Intuitively, in regions with high feature similarities across different scale factors, F can be directly used as F^{fuse}. In contrast, in regions with low feature similarities, F^{adapt} is added into F^{fuse} for feature adaption. That is, M serves as a gating mechanism and learns to guide pixel-wise feature adaption. It is demonstrated in Sec. 4.3 that our network benefits from the guidance maps to produce better results for scale-arbitrary SR.

The scale-aware convolutional layer within our feature adaption blocks is further illustrated in Fig. 4. First, the horizontal and vertical scale factors r_h and r_v are fed to a model controller with two fully connected (FC) layers to generate routing weights. Then, these routing weights are used to combine the experts, resulting in a scale-aware filter. Here, experts represent a set of convolutional kernels to be

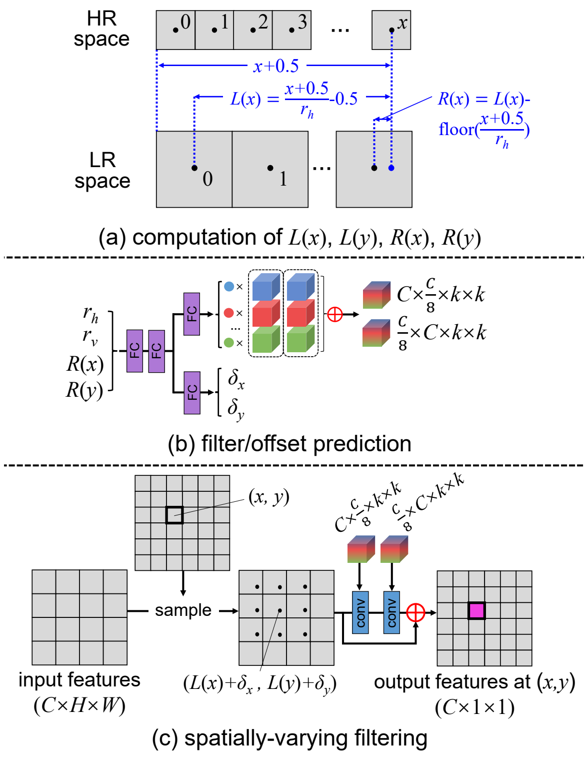
HR space
LR space
x+0.5
$L(x) = \frac{x+0.5}{r_h}$-0.5
$R(x) = L(x)$-floor($\frac{x+0.5}{r_h}$)
(a) computation of $L(x)$, $L(y)$, $R(x)$, $R(y)$
r_h
r_v
$R(x)$
$R(y)$
FC
$C\times\frac{C}{8}\times k\times k$
$\frac{C}{8}\times C\times k\times k$
δ_x
δ_y
(b) filter/offset prediction
(x, y)
sample
conv
input features ($C\times H\times W$)
$(L(x)+\delta_x, L(y)+\delta_y)$
output features at (x,y) ($C\times 1\times 1$)
(c) spatially-varying filtering

Figure 5. An illustration of our scale-aware upsampling layer.

combined based on the scale information. Finally, the predicted filter is used to process the input feature maps for feature adaption. Different from vanilla convolution with fixed filter, our scale-aware convolution dynamically customizes its filter conditioned on the scale information by combining knowledge from experts. It is demonstrated in Sec. 4.3 that scale-aware convolutions facilitate our network to adapt to specific scale factors to achieve better performance.

Scale-Aware Upsampling. Pixel shuffling layer [25] is widely used in SR networks for upsampling with integer scale factors. For $\times r (r = 2, 3, 4)$ SR, input features of size $C_{in} \times H \times W$ are first fed to a convolution to produce features of size $r^2 C_{out} \times H \times W$. Then, the resulting features are shuffled to the size of $C_{out} \times rH \times rW$. The pixel shuffling layer can be considered as a two-step pipeline, which consists of a sampling step and a spatially-varying filtering step (*i.e.*, r^2 convolutions for r^2 different sub-positions). Please refer to the supplemental material for more details.

In this paper, we generalize the pixel shuffling layer to a scale-aware upsampling layer, as shown in Fig. 5. First, each pixel (x, y) in the HR space is projected to the LR space to compute its coordinates ($L(x)$ and $L(y)$) and relative distances ($R(x)$ and $R(y)$):

$$L(x) = \frac{x + 0.5}{r_h} - 0.5, \tag{3}$$

$$R(x) = L(x) - \text{floor}(\frac{x + 0.5}{r_h}), \tag{4}$$

where $L(y)$ and $R(y)$ are calculated similar to $L(x)$ and $R(x)$. Next, $R(x)$, $R(y)$, r_h and r_v are concatenated and fed to two FC layers for feature extraction, as shown in Fig. 5(b). The resulting features are then passed to filter and offset heads to predict routing weights and offsets (δ_x and δ_y), respectively. After that, the routing weights are used to combine two groups of experts, resulting in a pair of filters for the bottleneck/expansion layers in Fig. 5(c). Finally, a $k \times k$ neighborhood centered at $(L(x) + \delta_x, L(y) + \delta_y)$ is sampled using bilinear interpolation and convolved with the predicted filters to produce the output features at (x, y), as shown in Fig. 5(c).

In the implementation, a pair of convolutional kernels ($\mathbb{R}^{C\times\frac{C}{8}\times k\times k}$ and $\mathbb{R}^{\frac{C}{8}\times C\times k\times k}$) need to be generated and stored for each location in HR space. Since the memory consumption can be very high for $k=3$ ($\sim$31.6G for a 720P HR image), k is set to 1 in our networks for memory efficiency ($\sim$3.5G).

4. Experiments

4.1. Datasets and Metrics

We used the DIV2K dataset [26] for network training and five benchmark datasets for evaluation, including Set5 [27], Set14 [28], B100 [29], Urban100 [30], and Manga109 [31]. Peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) were used as evaluation metrics. Similar to [14], we cropped borders for fair comparison. Note that, all metrics were computed in the luminance channel.

4.2. Implementation Details

Following [14], symmetric scale factors varying from 1 to 4 with a stride of 0.1 (*i.e.*, $1.1, 1.2, ..., 3.9, 4.0$) were used to generate LR training images. Moreover, asymmetric scale factors with a stride of 0.5 along horizontal and vertical axes (*i.e.*, $\frac{1.5}{2.0}, \frac{1.5}{2.5}, ..., \frac{4.0}{3.0}, \frac{4.0}{3.5}$) were also included for LR image generation. During training, a pair of horizontal/vertical scale factors was randomly selected from the above ranges for each batch and then 16 LR patches with the size of 50×50 were randomly cropped. Meanwhile, their corresponding HR patches were also cropped. Data augmentation was performed through random rotation and random flipping.

In our experiments, EDSR [16], RDN [10] and RCAN [12] were used as baseline networks to produce three scale-arbitrary networks, *i.e.*, ArbEDSR, ArbRDN and ArbRCAN. We use 4 experts in the scale-aware convolutions and set $K = 4/2/1$ for ArbEDSR/ArbRDN/ArbRCAN to control the model size. Since the available pre-trained RDN models are implemented in Torch while our networks are implemented in PyTorch [32], we re-trained RDN as our baseline network. Pre-trained $\times 4$ SR models of EDSR/RDN/RCAN were used to initialize the backbone

Table 1. PSNR results achieved by our network with different settings on Set5.

Model	Scale-Aware Feature Adaption		Scale-Aware	$\times 1.7$	$\times 2$	$\times 2.95$	$\times 3$	$\times 3.1$	$\frac{\times 1.3}{\times 3.9}$	$\frac{\times 1.9}{\times 3.5}$	$\frac{\times 2}{\times 3.3}$	$\frac{\times 3.3}{\times 1.9}$	$\frac{\times 4}{\times 1.8}$
	Scale-Aware Conv	Guidance Map	Upsampling										
EDSR (+Bicubic)*	✗	✗	✗	39.72	38.19	34.64	34.68	34.25	34.10	34.70	34.92	35.68	34.61
1	✗	✗	bicubic	39.56	37.86	33.77	33.78	33.47	33.12	33.90	34.21	34.90	33.42
2	✗	✗	✓	39.76	38.13	34.70	34.68	34.40	34.32	34.91	35.02	35.85	34.67
3	✓	✗	✓	39.81	38.15	34.70	34.70	34.42	34.38	34.98	35.10	35.91	34.72
4	✓	✓	✓	**39.87**	**38.19**	**34.75**	**34.73**	**34.48**	**34.44**	**35.03**	**35.16**	**35.95**	**34.81**

* To perform SR with non-integer and asymmetric scale factors (*e.g.*, $\times 1.7/\frac{\times 1.3}{\times 3.9}$ SR) using baseline network, we first super-resolve the LR image for $\times 2/\times 4$ SR and then downscale the result to the expected resolution using bicubic interpolation following [14].

blocks in ArbEDSR/ArbRDN/ArbRCAN, respectively. We used the Adam method [33] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for optimization. An L_1 loss between SR results and HR images was used as the loss function. Following [10], 1000 iterations of back-propagation constitute an epoch. The initial learning rate was set to 1×10^{-4} and reduced to half after every 30 epochs. To maintain training stability, we first trained our networks on integer scale factors ($r = 2, 3, 4$) for 1 epoch and then trained the networks on all scale factors. The training was stopped after 150 epochs.

4.3. Ablation Study

Ablation experiments were conducted on Set5 to test the effectiveness of our design choices. We used EDSR as the baseline network and introduced 4 variants. All variants were re-trained for 150 epochs.

Scale-Aware Upsampling. To enable scale-arbitrary SR, a naive approach is to replace the pixel shuffling layer with an interpolation layer (*e.g.*, bicubic interpolation). To demonstrate the effectiveness of our scale-aware upsampling layer, we introduced two variants. For variant 1, we replaced the pixel shuffling layer in the baseline network with a bicubic upsampling layer. For variant 2, we replaced the pixel shuffling layer with the proposed scale-aware upsampling layer. It can be observed from Table 1 that the PSNR values are relatively low when bicubic upsampling is used. With our scale-aware upsampling layer, the performance is significantly improved (*e.g.*, 39.76/38.13 vs. 39.56/37.86 for $\times 1.7/2$ SR). That is because, our scale-aware upsampling layer can learn dynamic filters conditioned on the scale factors while bicubic upsampling uses a fixed filter.

Scale-Aware Feature Adaption. Scale-aware feature adaption is used to adapt features to specific scale factors for better performance. Note that, our scale-aware feature adaption block consists of two key components: scale-aware convolution and guidance map generation. To demonstrate their effectiveness, we first added scale-aware convolutions to variant 2 to produce variant 3. Then, variant 4 is further obtained by adding guidance map generation to variant 3. It can be observed from Table 1 that the performance benefits from both scale-aware convolution and guidance map, with PSNR values being improved from 39.76/38.13/34.91 to 39.87/38.19/35.03 for $\times 1.7/2/\frac{1.9}{3.5}$ SR. Without feature adaption, model 2 uses shared features in the backbone for SR with different scale factors. This variant suffers inferior performance since the difference among features learned for various scale factors is not considered. With our scale-aware feature adaption blocks, our network can adapt features in the backbone according to the scale information. Therefore, better performance can be achieved.

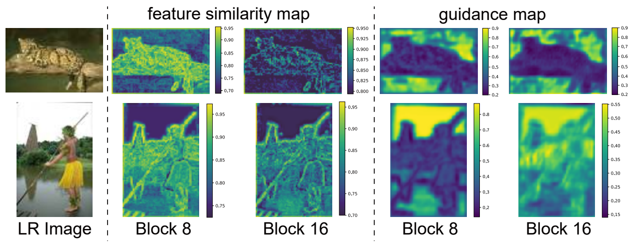

Figure 6. Visualization of guidance maps and their corresponding feature similarity maps.

Table 2. PSNR results achieved by our network with different number of experts on Set5. The running time is averaged over B100 on $\times 4$ SR.

#Experts	Params.	Time	$\times 1.7$	$\times 2$	$\times 2.55$	$\times 3.8$	$\frac{\times 1.3}{\times 3.9}$	$\frac{\times 2}{\times 3.5}$	$\frac{\times 3.3}{\times 1.8}$	$\frac{\times 4}{\times 1.2}$
1	38.4M	0.09s	39.37	37.87	35.82	32.13	33.34	34.37	35.50	33.93
2	38.6M	0.10s	39.88	38.20	36.02	32.99	34.23	34.92	36.03	34.87
4	39.2M	0.10s	39.87	38.19	36.02	33.00	34.44	34.96	36.07	35.18
8	40.4M	0.11s	39.88	38.22	36.03	32.98	34.46	34.98	36.07	35.20

We further visualize the guidance maps learned by variant 4 in Fig. 6. It can be observed that the learned maps are consistent with the feature similarity maps (also shown in Fig. 2). Specifically, regions with high values in guidance maps are consistent with those of low feature similarities. This demonstrates that our guidance maps can effectively guide the fusion of F and F^{adapt} (Eq. 2) to perform pixel-wise feature adaption accordingly.

Number of Experts in Scale-Aware Convolution. Scale-aware convolution dynamically generates scale-aware filters by combining knowledge from experts. To analyze the effect of the number of experts, we compare the performance of our network with different numbers of experts in Table 2. With only one expert, our scale-aware convolutions are degraded to vanilla ones with static filters and cannot well handle different scale factors. Therefore, variant 1 suffers a performance drop from 39.88/38.22/36.03 to 39.37/37.87/35.82 for $\times 1.7/2/2.55$ SR. As more experts are included in scale-aware convolutions, our network pro-

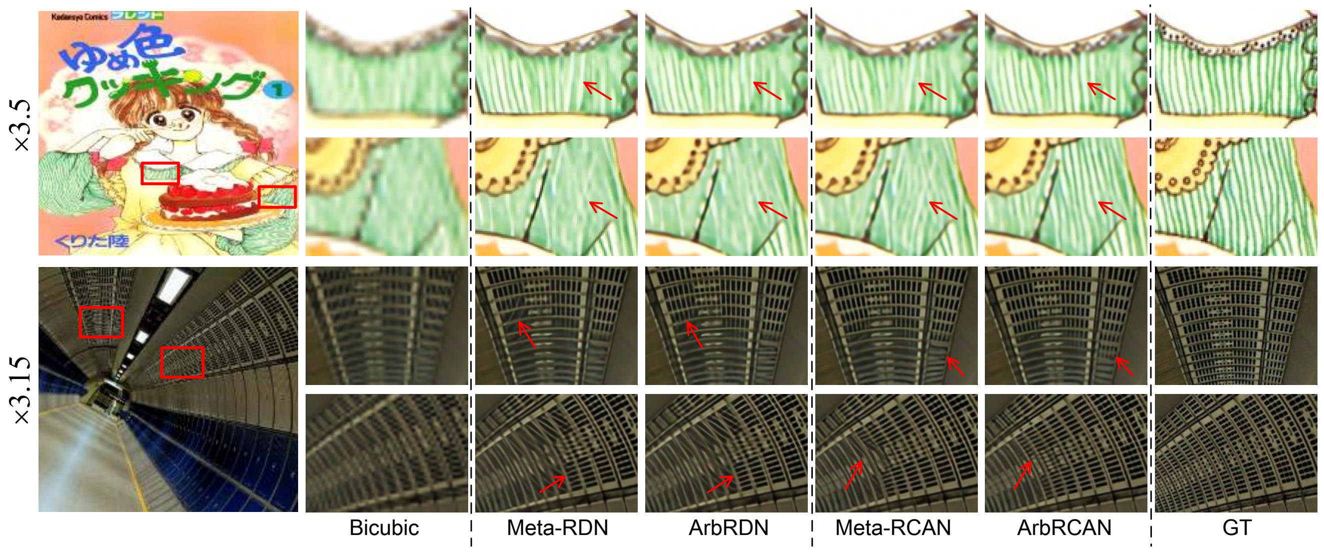

Figure 8. Visual comparison for SR with non-integer scale factors.

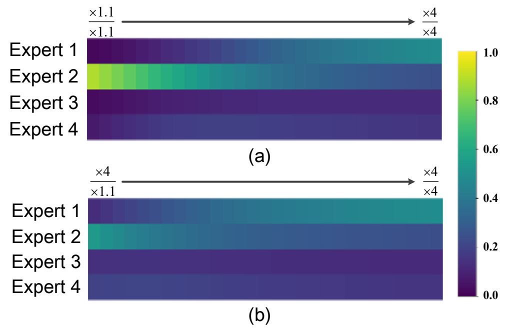

Figure 7. Visualization of routing weights in the scale-aware convolution of the first scale-aware feature adaption block. (a) and (b) show the routing weights for symmetric and asymmetric scale factors, respectively.

duces comparable results on symmetric scale factors while achieving better performance on asymmetric scale factors. This means that SR with asymmetric scale factors benefits a lot from the increase of experts. Further, we can see that the performance improvement on highly asymmetric scale factors are more significant (*e.g.,* 34.46(↑0.23)/34.98(↑0.06) vs. 34.23/34.92 for $\frac{\times 1.3}{\times 3.9}/\frac{\times 2}{\times 3.5}$ SR). Since more experts than 4 cannot introduce notable performance improvements, we use 4 experts as our default setting to achieve a better trade-off between performance and model size.

We further visualize the routing weights in our scale-aware convolution to investigate the knowledge of different experts with respect to various scale factors. As shown in Fig. 7(a), expert 2 is dominant for small scale factors while expert 1 is gradually activated for large ones. Compared to symmetric scale factors, more experts are activated for asymmetric ones. For example, experts 1, 3 and 4 are assigned with higher weights for $\frac{\times 4}{\times 1.1}$ than $\frac{\times 1.1}{\times 1.1}$. This observation is consistent with the results in Table 2 that SR with asymmetric scale factors benefits a lot from more experts.

4.4. Results for SR with Symmetric Scale Factors

In this section, we compare our ArbEDSR, ArbRDN and ArbRCAN to EDSR [16], RDN [10], RCAN [12], Meta-RDN [14] and Meta-RCAN [14] on the SR task with symmetric scale factors (both integer and non-integer scale factors)[1]. Since pre-trained models for Meta-RCAN are unavailable, we used officially released codes for re-training. Note that, we also fine-tuned Meta-RDN and Meta-RCAN on our training set for fair comparison. Comparative results are shown in Table 3 and Fig. 8.

Quantitative Results. It can be observed from Table 3 that our ArbEDSR, ArbRDN and ArbRCAN achieve comparable performance to their corresponding baseline networks on integer scale factors. For SR with non-integer scale factors, our networks significantly outperform their baseline networks. For example, our ArbEDSR is on par with EDSR for $\times 2$ SR on Set5 (38.19 vs. 38.19) while producing much better results for $\times 1.6/1.55$ SR (40.64/40.94 vs. 40.39/40.71).

Compared to Meta-RDN and Meta-RCAN, our ArbRDN and ArbRCAN achieves comparable or better performance for most scale factors. For example, our ArbRCAN produces notable performance improvements for $\times 3.4/3.65$ SR on Manga109 (33.12/32.29 vs. 33.00/32.22). Moreover, our ArbRDN and ArbRCAN achieve much better efficiency than Meta-RDN and Meta-RCAN, respectively. Compared to RCAN, Meta-RCAN has a comparable model size with larger memory consumption and longer running time. In

[1]To perform SR with non-integer scale factors (*e.g.*, $\times 1.6$) using baseline networks (*e.g.*, RCAN), the LR image is first super-resolved for $\times 2$ SR and then bicubicly downscaled to the expected resolution. More analyses are included in the supplemental material.

Table 3. PSNR results achieved on 5 benchmark datasets for symmetric scale factors. Note that, the memory consumption is calculated on an LR image with a size of 100×100. The running time is averaged over B100 on $\times 2/3/4$ SR. "+ft" means that the networks are fine-tuned on our training set.

	Params.	Memory*	Time*	Set5			Set14			B100			Urban100			Manga109		
				$\times 2$	$\times 1.6$	$\times 1.55$	$\times 2$	$\times 1.5$	$\times 1.65$	$\times 2$	$\times 1.4$	$\times 1.85$	$\times 2$	$\times 1.9$	$\times 1.95$	$\times 2$	$\times 1.7$	$\times 1.95$
Bicubic	-	-	-	33.66	36.10	36.24	30.24	32.87	31.83	29.56	32.95	30.11	26.88	27.25	27.05	30.80	32.91	31.12
EDSR-$\times 2$ [16](+Bicubic)	39.7M	0.9G	0.05s	38.19	40.39	40.71	33.95	37.10	35.95	32.36	36.79	33.02	32.95	33.06	32.69	39.18	40.88	39.13
ArbEDSR (Ours)	39.2M	1.0G	0.23s	**38.19**	**40.64**	**40.94**	**34.05**	**37.51**	**36.22**	**32.37**	**36.92**	**33.23**	**33.02**	**33.61**	**33.30**	**39.22**	**41.20**	**39.24**
RDN-$\times 2$ [10](+Bicubic)	21.6M	0.4G	0.08s	**38.24**	40.51	40.53	34.01	37.24	36.10	32.34	36.83	33.15	32.89	33.05	32.79	39.18	41.06	39.31
Meta-RDN [14]	21.4M	1.1G	0.38s	38.23	40.66	40.94	34.03	37.52	36.24	32.35	36.93	33.21	**33.03**	33.60	33.26	39.31	41.33	39.60
Meta-RDN [14]+ft	21.4M	1.1G	0.38s	38.21	40.65	40.94	34.05	37.53	36.26	32.34	36.91	33.21	33.01	**33.61**	**33.27**	**39.32**	**41.35**	**39.61**
ArbRDN (Ours)	22.6M	0.6G	0.18s	38.23	**40.67**	**40.95**	**34.07**	**37.53**	**36.27**	**32.37**	**36.93**	**33.21**	33.00	33.51	33.19	39.28	41.32	39.54
RCAN-$\times 2$ [12](+Bicubic)	15.2M	0.3G	0.27s	**38.27**	40.53	40.77	**34.12**	37.23	36.08	**32.40**	36.86	33.16	**33.18**	33.17	32.84	**39.42**	41.15	39.39
Meta-RCAN [14]	15.5M	0.9G	0.40s	38.22	40.66	40.93	34.00	37.51	36.17	32.36	**36.95**	33.22	33.12	33.62	33.30	39.32	41.30	39.59
Meta-RCAN [14]+ft	15.5M	0.9G	0.40s	38.21	40.63	40.93	34.03	37.50	36.20	32.35	36.95	33.22	33.10	**33.63**	**33.32**	39.34	41.31	**39.61**
ArbRCAN (Ours)	16.6M	0.5G	0.29s	38.26	**40.69**	**40.97**	34.09	**37.53**	**36.28**	32.39	36.93	**33.23**	33.14	33.55	33.25	39.37	**41.32**	39.56
				$\times 3$	$\times 2.4$	$\times 2.75$	$\times 3$	$\times 2.8$	$\times 2.95$	$\times 3$	$\times 2.2$	$\times 2.15$	$\times 3$	$\times 2.3$	$\times 2.35$	$\times 3$	$\times 2.7$	$\times 2.55$
Bicubic	-	-	-	30.39	32.41	31.06	27.55	27.84	27.46	27.21	28.88	29.12	24.46	25.91	25.72	26.95	27.77	28.27
EDSR-$\times 3$ [16](+Bicubic)	42.5M	1.0G	0.05s	34.68	36.45	**35.35**	30.53	30.90	30.49	29.27	31.38	**31.78**	28.82	31.13	30.91	34.19	35.18	35.75
ArbEDSR (Ours)	39.2M	1.3G	0.13s	**34.73**	**36.54**	35.34	**30.61**	**31.04**	**30.56**	**29.30**	**31.46**	31.70	**28.90**	**31.36**	**31.11**	**34.28**	**35.40**	**36.06**
RDN-$\times 3$ [10](+Bicubic)	21.7M	0.4G	0.08s	34.71	36.46	35.27	30.57	30.88	30.53	29.26	31.30	31.65	28.80	31.25	31.07	34.13	35.41	36.00
Meta-RDN [14]	21.4M	1.9G	0.32s	**34.73**	36.55	35.33	30.58	30.97	30.57	29.30	31.41	31.69	**28.93**	31.33	31.13	34.40	35.58	36.21
Meta-RDN [14]+ft	21.4M	1.9G	0.32s	34.70	36.55	35.35	30.58	30.97	30.57	29.28	31.42	31.67	28.88	31.33	31.12	34.42	35.59	**36.22**
ArbRDN (Ours)	22.6M	0.8G	0.13s	30.71	**36.55**	**35.35**	**30.59**	**30.98**	**30.58**	**29.30**	**31.45**	**31.69**	28.86	**31.33**	**31.14**	**34.43**	**35.60**	36.20
RCAN-$\times 3$ [12](+Bicubic)	15.3M	0.3G	0.27s	34.76	36.51	35.31	30.62	30.90	30.53	29.31	31.31	31.68	**29.01**	31.34	31.15	34.42	35.50	36.06
Meta-RCAN [14]	15.5M	1.7G	0.41s	34.76	36.58	35.36	30.58	31.00	30.56	29.29	31.44	31.70	28.96	31.43	31.20	34.40	35.55	36.21
Meta-RCAN [14]+ft	15.5M	1.7G	0.41s	34.72	36.59	35.38	30.58	30.99	30.56	29.28	31.46	31.70	28.93	31.44	31.22	34.44	35.60	36.24
ArbRCAN (Ours)	16.6M	0.8G	0.29s	**34.76**	**36.59**	**35.39**	**30.64**	**31.01**	**30.59**	**29.32**	**31.48**	**31.72**	28.98	**31.48**	**31.26**	**34.55**	**35.64**	**36.27**
				$\times 4$	$\times 3.1$	$\times 3.25$	$\times 4$	$\times 3.2$	$\times 3.95$	$\times 4$	$\times 3.2$	$\times 3.55$	$\times 4$	$\times 3.7$	$\times 3.85$	$\times 4$	$\times 3.4$	$\times 3.65$
Bicubic	-	-	-	28.42	29.89	29.21	26.00	26.98	25.68	25.96	26.91	26.32	23.14	23.38	23.14	24.89	25.97	25.41
EDSR-$\times 4$ [16](+Bicubic)	42.1M	1.2G	0.05s	32.47	34.25	33.35	28.81	29.95	28.63	27.73	28.84	28.25	**26.65**	27.06	26.69	31.04	32.51	31.79
ArbEDSR (Ours)	39.2M	1.7G	0.10s	**32.51**	**34.48**	**33.92**	**28.83**	**30.07**	**28.72**	**27.74**	**28.91**	**28.30**	26.62	**27.12**	**26.73**	**31.26**	**32.90**	**32.14**
RDN-$\times 4$ [10](+Bicubic)	21.7M	0.3G	0.07s	32.47	34.36	33.91	28.81	30.01	28.69	27.72	28.85	28.25	26.61	27.17	26.83	31.00	32.70	31.99
Meta-RDN [14]	21.4M	2.6G	0.29s	**32.49**	34.42	**33.93**	28.86	30.06	**28.75**	27.75	28.90	**28.31**	**26.70**	**27.24**	**26.91**	31.34	33.02	32.24
Meta-RDN [14]+ft	21.4M	2.6G	0.29s	32.46	34.41	33.91	**28.86**	30.06	28.74	**27.75**	28.90	28.30	26.68	27.20	26.87	31.35	**33.02**	32.24
ArbRDN (Ours)	22.6M	1.2G	0.13s	32.42	**34.43**	33.92	28.82	**30.08**	28.71	27.73	**28.90**	28.30	26.61	27.15	26.85	**31.35**	32.99	**32.24**
RCAN-$\times 4$ [12](+Bicubic)	15.2M	0.3G	0.23s	**32.63**	34.37	33.92	28.85	30.00	28.72	27.75	28.86	28.27	**26.75**	27.20	26.89	31.20	32.76	32.04
Meta-RCAN [14]	15.5M	3.1G	0.39s	32.56	34.46	33.98	28.85	30.08	28.73	27.75	28.86	28.30	26.71	**27.25**	**26.93**	31.33	33.00	32.22
Meta-RCAN [14]+ft	15.5M	3.1G	0.39s	32.55	34.44	33.99	28.85	30.08	28.73	27.75	28.88	28.30	26.71	27.24	26.93	31.35	33.02	32.23
ArbRCAN (Ours)	16.6M	1.1G	0.29s	32.55	**34.50**	**34.03**	**28.87**	**30.08**	**28.74**	**27.76**	**28.93**	**28.33**	26.68	27.22	26.90	**31.36**	**33.12**	**32.29**

* Officially released codes for Meta-RDN and Meta-RCAN are used to test the memory consumption and running time. Since generating an input matrix for the weight prediction network (Line 224 of *trainer.py* in the Github repository of Meta-SR) and post-processing (Line 235 of *trainer.py*) are also necessary for a single inference of Meta-SR, these operations are included for a fair comparison of running time.

contrast, our ArbRCAN takes shorter running time (0.29s vs. 0.39s) and much less memory consumption (1.1G vs. 3.1G). This clearly demonstrates the high efficiency of our plug-in module.

Qualitative Results. Figure 8 compares the visual results achieved on two images of the Manga109 and Urban100 datasets. From the zoom-in regions, we can see that our ArbRCAN produces results with better perceptual quality and fewer artifacts. For the second test image, Meta-RDN and Meta-RCAN cannot faithfully recover the stripes and suffer distorted artifacts. In contrast, our ArbRCAN produces clearer and finer details.

4.5. Results for SR with Asymmetric Scale Factors

In this section, we test our ArbEDSR, ArbRDN and ArbRCAN on the SR task with asymmetric scale factors[2]. Comparative results are presented in Table 4, while visual comparison is provided in Fig. 9.

Quantitative Results. It can be observed from Table 4 that baseline networks (*e.g.*, RCAN) have limited performance on asymmetric scale factors since their filters are fixed. Meta-RCAN uses meta-learning to generate filters for different scale factors to produce better results, with PSNR values being improved from 37.48/33.31/33.82 to 37.74/33.61/34.23 on Manga109. Moreover, fine-tuning Meta-RCAN on our training set further introduces marginal improvements (37.80/33.67/34.28 vs. 37.74/33.61/34.23). However, the performance is still inferior to our ArbRCAN even after fine-tuning. Using scale-aware convolutions to dynamically customize filters conditioned on the scale information, our ArbRCAN can adapt to the input scale factor to achieve better performance (37.93/33.81/34.41 vs. 37.80/33.67/34.28).

[2]To perform SR with asymmetric scale factors (*e.g.*, $\frac{\times 2.5}{\times 3.5}$) using baseline networks (*e.g.*, RCAN) and Meta-SR (*e.g.*, Meta-RCAN), the LR image is first super-resolved for $\times 4$ and $\times 3.5$ SR, respectively. Then, the resultant images are resized to the expected resolution using bicubic interpolation. More analyses are included in the supplemental material.

Table 4. PSNR results achieved for asymmetric scale factors. Note that, the memory consumption is calculated on an LR image with a size of 100×100 for $\frac{\times 2}{\times 4}$ SR. The running time is averaged over B100 on $\frac{\times 2}{\times 4}$ SR.

	Params.	Memory	Time	Set5			Set14			B100			Urban100			Manga109		
				$\frac{\times 1.5}{\times 4}$	$\frac{\times 1.5}{\times 3.5}$	$\frac{\times 1.6}{\times 3.05}$	$\frac{\times 4}{\times 2}$	$\frac{\times 3.5}{\times 2}$	$\frac{\times 3.5}{\times 1.75}$	$\frac{\times 4}{\times 1.4}$	$\frac{\times 1.5}{\times 3}$	$\frac{\times 3.5}{\times 1.45}$	$\frac{\times 1.6}{\times 3}$	$\frac{\times 1.6}{\times 3.8}$	$\frac{\times 3.55}{\times 1.55}$	$\frac{\times 2.5}{\times 2}$	$\frac{\times 2.8}{\times 3.5}$	$\frac{\times 3.35}{\times 2.7}$
Bicubic	-	-	-	30.01	30.83	31.40	27.25	27.88	27.27	27.45	28.86	27.94	25.93	24.92	25.19	29.61	26.47	26.86
EDSR [16]+Bicubic	42.1M	0.7G	0.04s	33.95	34.89	35.59	30.29	30.91	31.36	29.33	31.24	29.96	30.61	28.77	29.23	37.08	32.99	33.46
ArbEDSR (Ours)	39.2M	0.9G	0.14s	**34.32**	**35.33**	**36.02**	**30.51**	**31.15**	**31.46**	**29.52**	**31.38**	**30.20**	**31.06**	**29.32**	**29.98**	**37.70**	**33.54**	**34.16**
RDN [10]+Bicubic	21.7M	0.4G	0.08s	34.12	35.04	35.63	30.32	31.02	31.16	29.34	31.29	29.98	30.68	28.75	29.30	37.43	33.27	33.77
Meta-RDN [14]+Bicubic	21.4M	3.1G	0.49s	34.19	35.17	35.79	30.39	31.06	31.36	29.43	31.28	30.09	30.77	29.04	29.63	37.74	33.61	34.22
Meta-RDN [14]+Bicubic+ft	21.4M	3.1G	0.49s	34.22	35.19	35.80	30.42	31.06	31.35	29.47	31.30	30.12	30.85	29.11	29.70	37.80	33.64	34.26
ArbRDN (Ours)	22.6M	0.7G	0.13s	**34.31**	**35.26**	**35.98**	**30.47**	**31.12**	**31.42**	**29.52**	**31.36**	**31.19**	**31.02**	**29.23**	**29.91**	**37.88**	**33.74**	**34.36**
RCAN [12]+Bicubic	15.2M	0.4G	0.27s	34.14	35.05	35.67	30.35	31.02	31.21	29.35	31.30	29.98	30.72	28.81	29.34	37.48	33.31	33.82
Meta-RCAN [14]+Bicubic	15.5M	2.8G	0.61s	34.20	35.17	35.81	30.40	31.05	31.33	29.43	31.26	30.09	30.73	29.03	29.67	37.74	33.61	34.23
Meta-RCAN [14]+Bicubic+ft	15.5M	2.8G	0.61s	34.26	35.24	35.86	30.46	31.10	31.40	29.47	31.30	30.14	30.86	29.14	29.75	37.80	33.67	34.28
ArbRCAN (Ours)	16.6M	0.7G	0.29s	**34.37**	**35.40**	**36.05**	**30.55**	**31.27**	**31.54**	**29.54**	**31.40**	**30.22**	**31.13**	**29.36**	**30.04**	**37.93**	**33.81**	**34.41**

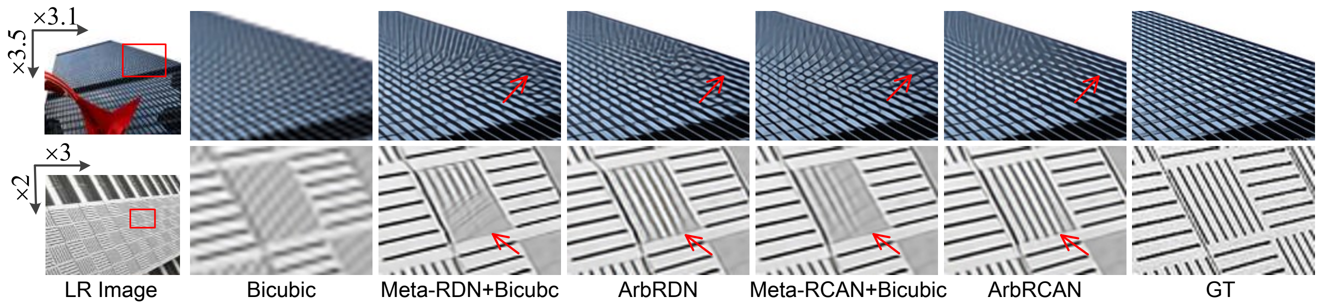

Figure 9. Visual comparison for SR with asymmetric scale factors on Urban100.

In addition to higher PSNR results, our ArbRCAN also has much smaller memory cost (0.7G vs. 2.8G) and shorter running time (0.29s vs. 0.61s) as compared to Meta-RCAN. Since Meta-RCAN needs to super-resolve an LR image to a size larger than the expected one before bicubic downscaling to perform asymmetric SR, redundant computational and memory cost is involved. In contrast, our ArbRCAN can directly super-resolve the LR image to the expected size with better efficiency.

Qualitative Results. Figure 9 illustrates the visual results achieved on two images of the Urban100 dataset. It can be observed from the zoom-in regions that our ArbRCAN produces better results for different asymmetric scale factors. Specifically, we can see from the second row that, our ArbRDN and ArbRCAN faithfully recover the stripes while other methods suffer blurring artifacts. This further demonstrates the superior performance of our networks on asymmetric scale factors.

4.6. Results for SR in the Wild

In many real-world applications, continuous magnification of an image to an arbitrary size is favored by customers such that they can stretch an image to any resolution they like. However, all existing SR network including RDN, RCAN and Meta-RCAN rely on post-processing (*i.e.*, bicubic interpolation) to achieve scale-arbitrary SR, which is difficult to obtain optimal performance. In contrast, our network provides an end-to-end framework to produce better results. In this section, ArbRCAN is used to super-resolve a real-world image (an HR image in B200 [29]) to different resolutions for visual comparison, as shown in Fig. 10. When the input image is continuously magnified, the text becomes easier for recognition and our ArbRCAN consistently produces better perceptual quality than other methods.

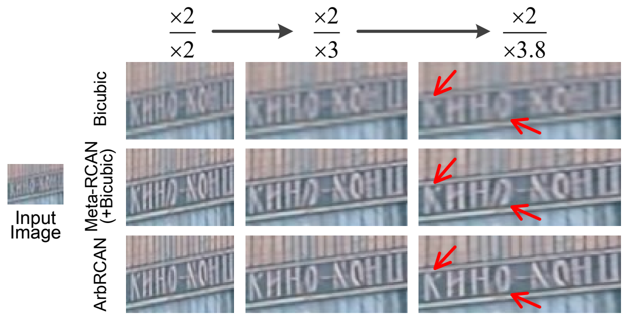

Figure 10. Visual comparison on a real-world image.

5. Conclusions

In this paper, we proposed a plug-in module to enable existing image SR networks for scale-arbitrary SR with a single model. Experimental results show that baseline networks equipped with our module can produce promising results on SR tasks with non-integer and asymmetric scale factors, while maintaining state-of-the-art performance on integer scale factors. Moreover, our module can be easily adapted to scale-specific networks with small additional computational and memory cost.

References

[1] Jian Sun, Zongben Xu, and Heung-Yeung Shum. Image super-resolution using gradient profile prior. In *CVPR*, 2008. 1, 2

[2] Kaibing Zhang, Xinbo Gao, Dacheng Tao, and Xuelong Li. Single image super-resolution with non-local means and steering kernel regression. *IEEE Trans. Image Process.*, 21(11):4544–4556, nov 2012. 1, 2

[3] Chih-Yuan Yang and Ming-Hsuan Yang. Fast direct super-resolution by simple functions. In *ICCV*, pages 561–568, 2013. 1, 2

[4] Radu Timofte, Vincent De Smet, and Luc J. Van Gool. Anchored neighborhood regression for fast example-based super-resolution. In *ICCV*, pages 1920–1927, 2013. 1, 2

[5] Shuhang Gu, Wangmeng Zuo, Qi Xie, Deyu Meng, Xiangchu Feng, and Lei Zhang. Convolutional sparse coding for image super-resolution. In *ICCV*, pages 1823–1831, 2015. 1, 2

[6] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita. Deep back-projection networks for super-resolution. In *CVPR*, pages 1664–1673, 2018. 1, 2

[7] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *ECCV*, pages 184–199, 2014. 1, 2

[8] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *CVPR*, pages 1646–1654, 2016. 1, 2

[9] Jose Caballero, Christian Ledig, Andrew P. Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *CVPR*, pages 2848–2857, 2017. 1, 2

[10] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *CVPR*, pages 2472–2481, 2018. 1, 2, 4, 5, 6, 7, 8

[11] Yajun Qiu, Ruxin Wang, Dapeng Tao, and Jun Cheng. Embedded block residual network: A recursive restoration model for single-image super-resolution. In *ICCV*, 2019. 1

[12] 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 1646–1654, 2018. 1, 2, 4, 6, 7, 8

[13] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. Second-order attention network for single image super-resolution. In *CVPR*, 2019. 1, 2

[14] Xuecai Hu, Haoyuan Mu, Xiangyu Zhang, Zilei Wang, Jian Sun, and Tieniu Tan. Meta-SR: A magnification-arbitrary network for super-resolution. In *CVPR*, 2019. 1, 2, 4, 5, 6, 7, 8

[15] Jianchao Yang, Zhaowen Wang, Zhe Lin, S. Cohen, and T. Huang. Coupled dictionary training for image super-resolution. *IEEE Trans. Image Process.*, 21(8):3467–3478, aug 2012. 2

[16] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *CVPR*, 2017. 2, 4, 6, 7, 8

[17] Ying Fu, Jian Chen, Tao Zhang, and Yonggang Lin. Residual scale attention network for arbitrary scale image super-resolution. *Neurocomputing*, 427:201–211, 2021. 2

[18] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. *arXiv*, 2019. 2

[19] Brandon Yang, Gabriel Bender, Quoc V. Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. In *NeurIPS*, pages 1305–1316, 2019. 2

[20] Yikang Zhang, Jian Zhang, Qiang Wang, and Zhao Zhong. Dynet: Dynamic convolution for accelerating convolutional neural networks. *arXiv*, 2020. 2

[21] Zhi Tian, Chunhua Shen, and Hao Chen. Conditional convolutions for instance segmentation. In *ECCV*, 2020. 2

[22] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In *ECCV*, pages 391–407, 2016. 2

[23] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. In *ECCV*, pages 252–268, 2018. 2

[24] Minghao Yin, Zhuliang Yao, Yue Cao, Xiu Li, Zheng Zhang, Stephen Lin, and Han Hu. Disentangled non-local neural networks. In *ECCV*, 2020. 3

[25] Wenzhe Shi, Jose Caballero, Ferenc Huszar, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*, pages 1874–1883, 2016. 4

[26] Eirikur Agustsson and Radu Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *CVPRW*, pages 1122–1131, 2017. 4

[27] 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. 4

[28] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International Conference on Curves and Surfaces*, volume 6920, pages 711–730, 2010. 4

[29] David Martin, Charless Fowlkes, Doron Tal, Jitendra Malik, et al. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, 2001. 4, 8

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

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

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, and *et al.* PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 4

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5