

# Learning for Scale-Arbitrary Super-Resolution from Scale-Specific Networks

Longguang Wang, Yingqian Wang, Zaiping Lin, Jungang Yang, Wei An, and 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 a single specific integer scale (*e.g.*,  $\times 2$ ,  $\times 3$ ,  $\times 4$ ), and cannot be used for non-integer and asymmetric SR. In this paper, we propose to learn a scale-arbitrary image SR network from scale-specific networks. Specifically, we propose 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, we introduce a scale-aware knowledge transfer paradigm to transfer knowledge from scale-specific networks to the scale-arbitrary network. Our plug-in module can be easily adapted to existing networks to achieve scale-arbitrary SR. These networks plugged with our module can achieve 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.

**Keywords:** Image Super-Resolution, Scale-Arbitrary Super-Resolution, Knowledge Transfer, Meta-Learning

## 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. Numerous CNN-based methods [7,8,9,10,11] have been developed and the SR performance has been significantly improved.

Although recent CNN-based single image SR networks [10,11,12,13] have achieved promising performance, they are developed for image SR with a single specific integer scale (*e.g.*,  $\times 2$ ,  $\times 3$ ,  $\times 4$ ). In real-world applications, non-integer SR (*e.g.*, from  $100 \times 100$  to  $220 \times 220$ ) and asymmetric SR (*e.g.*, from  $100 \times 100$  to  $220 \times 420$ ) are also necessary such that customers can zoom in an image arbitrarily for better view of details. However, SR networks for specific integer scales cannot be used for scale-arbitrary SR in real-world scenarios.

To address this limitation, Hu *et al.* [14] proposed a Meta-SR network to predict weights of filters for different scale factors using meta-learning. Meta-SR produces promising results on non-integer scale factors. However, it still has few limitations. First, the scale information is only used for upsampling in the network. That is, features in the backbone are identical for SR tasks with different scale factors, which hinders the further improvement of performance. Second, Meta-SR is trained from scratch (which is time-consuming) and has a large memory overhead. Third, Meta-SR mainly focuses on SR with non-integer scale factors but cannot handle SR with asymmetric scale factors.

Since image SR tasks with multiple scales are inter-related [15], knowledge learned by powerful scale-specific networks [10,11,12] can be transferred to train a scale-arbitrary network. In this paper, we propose to learn a scale-arbitrary single image SR network from scale-specific networks. Specifically, we propose 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 achieve scale-aware feature extraction and the scale-aware upsampling layer is used for scale-arbitrary upsampling. Moreover, we use a scale-aware knowledge transfer paradigm to transfer knowledge from scale-specific networks for the training of the scale-arbitrary network. Our plug-in module can be easily adapted to existing networks for scale-arbitrary SR with small additional computational and memory costs. Baseline networks equipped with our module can produce promising results for non-integer and asymmetric SR, while maintaining comparable performance to their scale-specific counterparts for SR with integer scale factors. 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 propose a plug-in module for SR networks to achieve scale-arbitrary SR, including multiple scale-aware feature adaption blocks and a scale-aware upsampling layer. 2) We introduce a scale-aware knowledge transfer paradigm to transfer knowledge from scale-specific networks to a scale-arbitrary network. 3) Experimental results show that baseline networks equipped with our module produce promising results for scale-arbitrary SR while maintaining the state-of-the-art performance for SR with integer scale factors. A video demo is available at <https://youtu.be/AFm97PHWeGI>. (Please play the video in 1080P for better view.)

## 2 Related Work

In this section, we first briefly review several major works for CNN-based single image SR. Then, we discuss learning techniques related to our work, including multi-task learning, meta-learning and transfer learning.

### 2.1 Single Image Super-Resolution

Due to the powerful feature representation and model fitting capabilities of deep neural network, CNN-based single image SR methods [7,8,9,10,11] outperform

traditional methods [1,2,3,4,5,6] 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 proposed to achieve better performance. Tai *et al.* [16] proposed a deep recursive residual network (DRRN) to reduce the number of model parameters using recursive blocks with shared parameters.

Recently, Lim *et al.* [15] 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. Zhang *et al.* [11] proposed a residual dense network (RDN) by stacking several residual dense blocks. By combining residual connection and dense connection, RDN has a contiguous memory and achieves favorable performance against EDSR. Haris *et al.* [10] proposed a deep back-projection network (DBPN) with iterative up-sampling and down-sampling layers to provide error feedback information. Zhang *et al.* [12] and Dai *et al.* [13] further improved the image SR performance by introducing channel attention and second-order attention, respectively.

Although existing single image SR methods have achieved promising results, they are trained for SR with a single specific integer scale factor. To overcome this limitation, Lim *et al.* [15] proposed a multi-scale deep super-resolution system (MDSR) to integrate modules trained for multiple scale factors (*i.e.*,  $\times 2$ ,  $\times 3$ ,  $\times 4$ ). 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. However, Meta-SR cannot handle SR with asymmetric scale factors and has a large memory overhead.

## 2.2 Multi-Task Learning, Meta-Learning and Transfer Learning

Multi-task learning aims at developing a single model for multiple different tasks [17,18,19]. A multi-task learning network usually includes a common backbone and multiple output branches (paths) for different tasks. Multi-task learning is based on the intuition that multiple tasks are inter-related and can contribute to each other. However, for scale-arbitrary single image SR, a single network has to be used for SR with an arbitrary scale factor. Therefore, multi-task learning is unsuitable for our problem since an infinite number of scale factors have to be handled.

Meta-learning, also known as learning to learn, aims to learn meta-knowledge to make the process of learning from new data more effective and efficient [20]. Meta-learning is commonly employed in reinforcement learning [21,22,23] and optimization [24,25,26,27]. As one of the meta-learning strategies, weight prediction is applied in numerous tasks, including image recognition [28] and object detection [29]. In these networks [28,29], the weights of networks are learned from meta-learners rather than training data. In Meta-SR [14], meta-learning is used for SR to predict the weights of filters for different scale factors. Meta-learning is also used in our plug-in module to learn meta-knowledge.

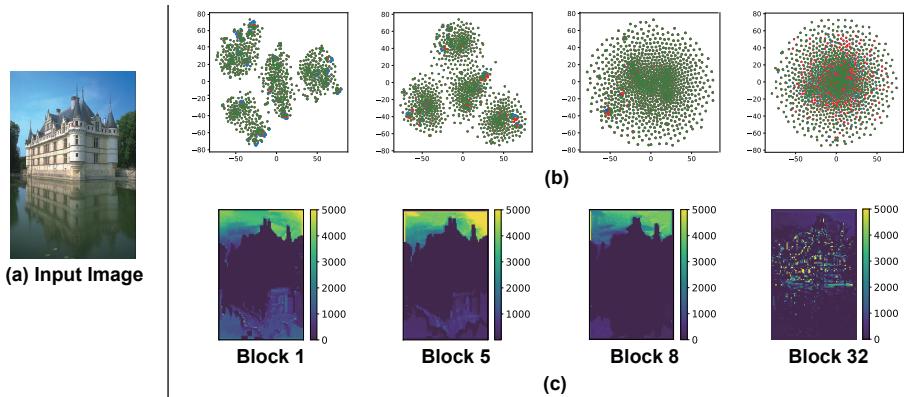


Fig. 1: Visualization of features from different blocks in EDSR. Blue, red and green dots represent features from the  $\times 2$ ,  $\times 3$  and  $\times 4$  SR networks, respectively.

Transfer learning aims at transferring knowledge learned from a source domain to a target domain [30]. The target domain can be a new task [31,30] or a new environment [32,33]. Since multi-scale SR are inter-related tasks [15], if we consider non-integer SR and asymmetric SR as target domains, SR with integer scale factors can be considered as source domains. Therefore, we are motivated to transfer knowledge from scale-specific SR networks to a scale-arbitrary network.

### 3 Methodology

#### 3.1 Motivation

Since SR tasks with different scale factors are inter-related [15], it is non-trivial to learn a scale-arbitrary SR network from scale-specific SR networks (*e.g.*,  $\times 2$ ,  $\times 3$ ,  $\times 4$ ). Therefore, we first investigate the relationship between  $\times 2/\times 3/\times 4$  SR tasks to provide insights for scale-arbitrary SR.

Specifically, we conduct experiments on the B100 dataset [34] to compare the feature distribution on specific layers in pre-trained  $\times 2/\times 3/\times 4$  SR networks. In our experiments, EDSR [15] is selected as the baseline network. Experimental results with RCAN [12] are provided in the supplementary material. First, we downsample an image of the B100 dataset to  $\frac{1}{4}$  size (denoted as  $I \in R^{H \times W}$ ). Then, we feed  $I$  to the EDSR network developed for  $\times 2/\times 3/\times 4$  SR. The features of the last layer in the  $i^{\text{th}}$  residual block (denoted as  $F_i^E \in R^{256 \times H \times W}, i = 1, 2, \dots, 32$ ) are then used for visualization.

In the first experiment, we use all of the 100 images in the B100 dataset and randomly select 10 positions in each image for feature sampling, resulting in 1000 feature samples for each block (*e.g.*,  $f_i^{E \times 2} \in R^{256 \times 1000}$ ). Then, we concatenate  $f_i^{E \times 2}$ ,  $f_i^{E \times 3}$  and  $f_i^{E \times 4}$  to produce  $f_i^E \in R^{256 \times 3000}$ . Finally,  $f_i^E$  is visualized using

the t-SNE method [35], as shown in Fig. 1(b). It is clear that dots of different colors are overlapped at most positions in blocks 1, 5 and 8. That is, the features at these positions in blocks 1, 5 and 8 of the  $\times 2$ ,  $\times 3$  and  $\times 4$  SR networks are prone to be scale-independent. In contrast, the distributions of dots with different colors in the last block are quite different. That means the features in the last block are prone to be scale-dependent.

In the second experiment, we use one image (as shown in Fig. 1(a)) as the input and investigate the scale-dependency of features in different regions. Specifically,  $F_i^{E \times 2}$ ,  $F_i^{E \times 3}$  and  $F_i^{E \times 4}$  are concatenated and then reshaped to  $R^{256 \times 3HW}$  for t-SNE transformation. Assume a feature sample  $F_i^{E \times 2}(p)$  at position  $p$  is projected to  $X_i^{E \times 2}(p)$  by the t-SNE transformation, a discrepancy measure between  $X_i^{E \times 2}(p)$ ,  $X_i^{E \times 3}(p)$  and  $X_i^{E \times 4}(p)$  is defined as:

$$D(p) = \|X_i^{E \times 2}(p) - \bar{X}(p)\|^2 + \|X_i^{E \times 3}(p) - \bar{X}(p)\|^2 + \|X_i^{E \times 4}(p) - \bar{X}(p)\|^2, \quad (1)$$

where

$$\bar{X}(p) = \frac{1}{3}(X_i^{E \times 2}(p) + X_i^{E \times 3}(p) + X_i^{E \times 4}(p)). \quad (2)$$

The discrepancy map is visualized in Fig. 1(c). Note that, lower discrepancy indicates higher feature similarity, *i.e.*, lower scale-dependency.

From Fig. 1(c) we can see that the discrepancy is low for regions with rich edges and texture (*e.g.*, the building) in blocks 1, 5 and 8. However, in texture-less regions (*e.g.*, the sky and the lake), the discrepancy is large. That means features in texture-rich regions are prone to be scale-independent while features in texture-less regions are prone to be scale-dependent. In our experiment, this observation holds for all blocks except the last one. For the last block, the discrepancy for texture-rich regions is significantly increased. For more results, please refer to the supplemental material.

In summary, scale-dependency is different for different blocks and regions. Motivated by this observation, we distinguish scale-dependent features from scale-independent ones, and then perform scale-aware feature adaption adaptively. Specifically, scale-independent features can be directly used for SR with arbitrary scale factors, while scale-dependent features should be adapted according to the scale factors.

### 3.2 Our Plug-in Module

The architecture of our plug-in module is shown in Fig. 2. Given a baseline network (*e.g.*, EDSR and RCAN) developed for SR with a specific integer scale factor, 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. 2(b). Following the backbone module, a scale-aware upsampling layer is used for scale-arbitrary upsampling.

**Scale-Aware Convolution** The scale-aware convolutional layer is illustrated in Fig. 2(c). First, the horizontal scale factor  $r_h$  and vertical scale factor  $r_v$  are fed

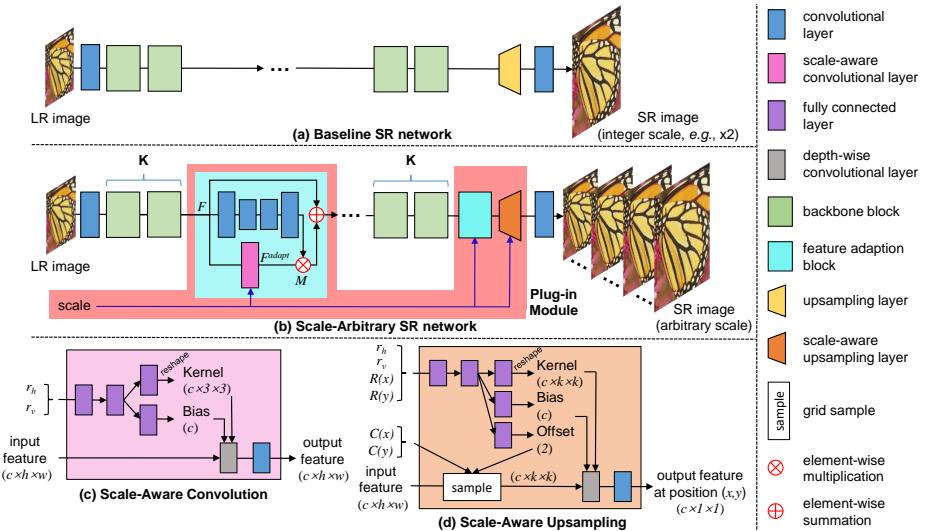


Fig. 2: An overview of our plug-in module.

to two fully connected layers, resulting a feature vector. Then, this feature vector is separately fed to a kernel head and a bias head to predict kernels and bias. Next, the predicted kernels and bias are used to perform depth-wise convolution on the input feature map. Finally, the resulting feature map is passed to a  $1 \times 1$  convolution to fuse the information across different channels, resulting in an output feature.

**Scale-Aware Feature Adaption** Given a feature map  $F$ , it is first fed to an hourglass module with four convolutions to generate a scale-dependency mask  $M$  with values ranging from 0 to 1. Then,  $F$  is fed to a scale-aware convolution for feature adaption, resulting in an adapted feature map  $F^{adapt}$ . Next,  $F$  and  $F^{adapt}$  are fused as:

$$F^{fuse} = F + F^{adapt} \times M, \quad (3)$$

where the scale-dependency mask  $M$  is used as a guidance.

Intuitively, in regions with low scale-dependency values, features are prone to be scale-independent and  $F$  can be directly used as  $F^{fuse}$ . In contrast, in regions with high scale-dependency values, features are prone to be scale-dependent. Therefore,  $F^{adapt}$  should be added into  $F^{fuse}$  for feature adaption.

**Scale-Aware Upsampling** Pixel shuffling layer [36] 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-variant convolution

step (*i.e.*,  $r^2$  convolutions for 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 up-sampling layer, as shown in Fig. 2(d). First, each pixel  $(x, y)$  in the HR space is projected to the LR space to compute its coordinates ( $C(x)$  and  $C(y)$ ) and relative distances ( $R(x)$  and  $R(y)$ ):

$$\begin{cases} C(x) = \frac{x + 0.5}{r_h} - 0.5 \\ C(y) = \frac{y + 0.5}{r_v} - 0.5 \end{cases}, \quad (4)$$

$$\begin{cases} R(x) = \frac{x + 0.5}{r_h} - \text{floor}\left(\frac{x + 0.5}{r_h}\right) - 0.5 \\ R(y) = \frac{y + 0.5}{r_v} - \text{floor}\left(\frac{y + 0.5}{r_v}\right) - 0.5 \end{cases}. \quad (5)$$

Then,  $R(x)$ ,  $R(y)$ ,  $r_h$  and  $r_v$  are concatenated and fed to two fully connected layers for feature extraction. The resulting features are fed to kernel/bias/offset heads to predict kernels, bias and offsets, respectively. Next, we sample a  $k \times k$  neighborhood centered at  $(C(x) + \delta_x, C(y) + \delta_y)$ , where  $\delta_x$  and  $\delta_y$  are predicted offsets along the  $x$  and  $y$  axes, respectively. Note that,  $k$  is set to 1 in our networks to control the model size. Finally, predicted kernels and bias are used to perform depth-wise convolution on the sampled features, followed by a  $1 \times 1$  convolution.

### 3.3 Scale-Aware Knowledge Transfer

For  $\times r$  ( $r = 2, 3, 4$ ) SR, the baseline SR network ( $\times r$ ) can be used as a teacher network for knowledge transfer. Intuitively, we use features in the backbone module as supervision. To exploit rich information in the hidden layers of the teacher's backbone module, the student network learns the teacher's outputs in every  $K$  blocks, as shown in Fig. 3. Note that, our scale-aware knowledge transfer paradigm is only used in the training phase. The knowledge transfer loss is defined as an L1 loss:

$$L_{transfer} = \frac{1}{N} \sum_{i=1}^N \left\| F_i^{GT} - F_i^{fuse} \right\|_1, \quad (6)$$

where  $N$  is the number of blocks used for knowledge transfer.

## 4 Experiments

### 4.1 Datasets and Metrics

We used the high-quality DIV2K dataset [37] for network training. This dataset contains 800 training images, 100 validation images and 100 test images. We then

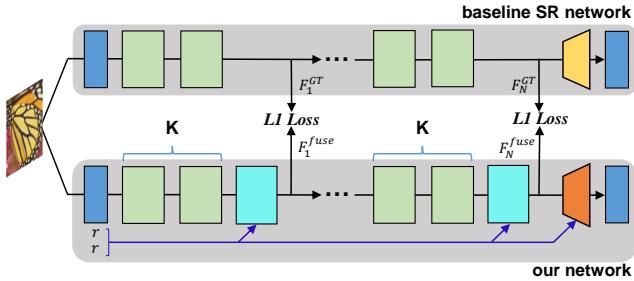


Fig. 3: An illustration of the scale-aware knowledge transfer paradigm.

used five benchmark datasets to test our module, including Set5 [38], Set14 [39], B100 [34], Urban100 [40], and Manga109 [41]. Peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) were used as evaluation metrics to measure SR performance. 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], we generated LR training images with scale factors varying from 1 to 4 with a stride of 0.1. During training, a scale factor was randomly selected for each batch and then 16 LR patches with the size of  $50 \times 50$  were randomly cropped. Meanwhile, their corresponding HR patches were also cropped. Note that, only symmetric scale factors were considered in the training phase. Data augmentation was performed through random rotation and random flipping.

The SR loss  $L_{SR}$  is defined as an L1 loss between SR results and HR images. The overall loss for training is defined as:

$$L = L_{SR} + \lambda L_{transfer}, \quad (7)$$

where  $\lambda$  is empirically set to 1. Note that,  $L_{transfer}$  is only used for batches with integer scale factors ( $r = 2, 3, 4$ ). In our experiments, EDSR [15] and RCAN [12] were used as baseline networks. We set  $K = 2$  for EDSR and  $K = 1$  for RCAN to control the model size. A pre-trained  $\times 4$  SR model was used to initialize the backbone blocks. Since the available pre-trained RDN models are implemented in Torch while our networks are implemented in PyTorch [42], RDN is not included in our baseline networks. We used the Adam [43] method with  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$  for optimization. The initial learning rate was set to  $1 \times 10^{-4}$  and reduced to half after every 15 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 70 epochs.

Table 1: Comparative results achieved on Set5. Note that, ‘SA conv’ represents the scale-aware convolution.

Model	Scale-Aware Adaption	Scale-Aware Upsampling	Scale-Aware Transfer	Average PSNR ( $\times 2, \times 3, \times 4$ )	Average PSNR ( $\times 1.1 \sim \times 4.0$ )	Average PSNR ( $\times 6, \times 7, \times 8$ )	Average PSNR ( $\times 5.1 \sim \times 8.0$ )
	SA Conv Mask						
Baseline	$\times$	$\times$	$\times$	$\times$	35.11	-	-
1	$\times$	$\times$	bicubic	$\times$	34.25	36.24	26.52
2	$\times$	$\times$	$\checkmark$	$\times$	35.10	37.00	26.74
3	$\checkmark$	$\times$	$\checkmark$	$\times$	35.14	37.06	26.73
4	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	35.14	37.06	26.79
5	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	<b>35.14</b>	<b>37.07</b>	<b>26.98</b>
							<b>27.64</b>

### 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 5 variants. All variants were re-trained for 70 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 performance is low (34.25/36.24/26.52/27.00) when bicubic upsampling is used. In contrast, the performance is significantly improved (35.10/37.00/26.74/27.44) when scale-aware upsampling is used. That is because, our scale-aware upsampling layer can learn an optimal filter for each scale factor while bicubic upsampling uses a fixed filter for all scale factors. By using scale-aware upsampling, variant 2 is able to achieve scale-arbitrary SR while maintaining comparable performance to the baseline network on  $\times 2/\times 3/\times 4$  SR.

**Scale-Aware Feature Adaption** Scale-aware feature adaption is used to adapt features to a specific scale factor. Note that, our scale-aware feature adaption block consists of two key components: scale-aware convolution and mask generation. To demonstrate their effectiveness, we first added scale-aware convolutions to variant 2. Then, we further added mask generation to variant 3. It can be observed from Table 1 that the performance benefits from both scale-aware convolution and mask generation, with PSNR values being improved from 35.10/37.00/26.74/27.44 to 35.14/37.06/26.79/27.55. That is because, features in the backbone can be adapted according to the scale information using our scale-aware feature adaption blocks. Note that, variant 4 (with mask generation) achieves comparable performance to variant 3 on SR with small scale factors, and outperforms variant 3 on SR with large scale factors (*e.g.*,  $\times 6/\times 7/\times 8$ ). That is because, the network with scale-dependency masks can detect scale-dependent

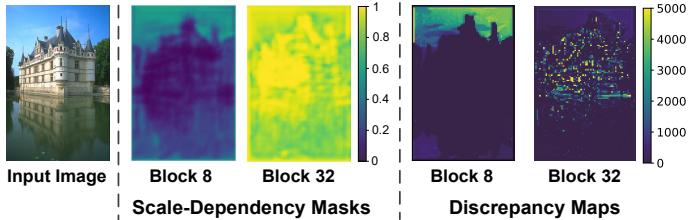


Fig. 4: Visualization of scale-dependency masks and their corresponding discrepancy maps.

features for feature adaption. Therefore, better generalization ability on large scale factors can be achieved.

We further illustrate two scale-dependency masks learned by variant 4 in Fig. 4. It can be observed that the masks are consistent with the discrepancy maps (also shown in Fig. 1). This clearly demonstrates that scale-aware feature adaption blocks can detect scale-dependent features and perform feature adaption adaptively.

**Scale-Aware Knowledge Transfer** The scale-aware knowledge transfer paradigm is used to transfer knowledge from scale-specific networks to our scale-arbitrary network. To demonstrate its effectiveness, we introduced variant 5 by including the scale-aware knowledge transfer paradigm for network training. It can be observed from Table 1 that the performance on SR with large and unseen scale factors (*i.e.*,  $\times 6/\times 7/\times 8$  and  $\times 5.1-\times 8.0$ ) is improved from 26.79/27.55 to 26.98/27.64. This clearly demonstrates that the knowledge transfer paradigm facilitates our network to achieve better generalization capability on SR with large scale factors while maintaining comparable performance on SR with small scale factors.

#### 4.4 Results for SR with Integer Scale Factors

In this section, we applied our plug-in module to two baseline networks EDSR and RCAN to produce two scale-arbitrary networks ArbEDSR and ArbRCAN. These two networks are compared to several existing networks, including SRCNN [7], VDSR [8], LapSRN [44], MemNet [45], SRMD [46], CARN [47], MSRN [48], DBPN [10], EDSR [15], RDN [11] and RCAN [12] for  $\times 2/\times 3/\times 4$  SR. Comparative results on 5 benchmark datasets are shown in Table 2.

**Quantitative Results** We can see from Table 2 that our ArbEDSR and ArbRCAN achieve comparable performance to their corresponding baseline networks (*i.e.*, EDSR and RCAN), with average PSNR improvements of 0.04 and 0.01 being achieved over all datasets and scales. Compared to EDSR, our ArbEDSR achieves comparable performance on B100 (27.74/0.7418 vs. 27.73/0.7417) for  $\times 4$  SR while achieving a notable performance improvement on Manga109 (31.29/

Table 2: PSNR/SSIM results achieved for  $\times 2/\times 3/\times 4$  SR. Note that, since the model of Meta-EDSR is unavailable, its results are directly copied from [14].

Model	Scale	Set5	Set14	B100	Urban100	Manga109
Bicubic	$\times 2$	33.66/0.9299	30.24/0.8688	29.56/0.8431	26.88/0.8403	30.80/0.9339
SRCNN [7]	$\times 2$	36.66/0.9542	32.45/0.9067	31.36/0.8879	29.50/0.8946	35.60/0.9663
VDSR [8]	$\times 2$	37.53/0.9590	33.05/0.9130	31.90/0.8960	30.77/0.9140	37.22/0.9750
LapSRN [44]	$\times 2$	37.52/0.9591	33.08/0.9130	31.08/0.8950	30.41/0.9101	37.27/0.9740
MemNet [45]	$\times 2$	37.78/0.9697	33.28/0.9142	32.08/0.8978	31.31/0.9195	37.72/0.9740
SRMD [46]	$\times 2$	37.79/0.9600	33.32/0.9159	32.05/0.8985	31.33/0.9204	38.07/0.9761
CARN [47]	$\times 2$	37.76/0.9590	33.52/0.9166	32.09/0.8978	31.51/0.9312	-/-
MSRN [48]	$\times 2$	38.08/0.9605	33.74/0.9170	32.23/0.9013	32.22/0.9326	38.82/0.9868
DBPN [10]	$\times 2$	38.09/0.9600	33.85/0.9190	32.27/0.9000	32.55/0.9324	38.89/0.9775
EDSR [15]	$\times 2$	38.19/0.9608	33.95/0.9200	32.36/0.9017	32.95/0.9357	39.18/0.9781
RDN [11]	$\times 2$	38.24/0.9614	34.01/0.9212	32.34/0.9017	32.89/0.9353	39.18/0.9780
RCAN [12]	$\times 2$	<b>38.27/0.9613</b>	<b>34.12/0.9214</b>	<b>32.40/0.9023</b>	<b>33.18/0.9370</b>	<b>39.42/0.9786</b>
Meta-EDSR [14]	$\times 2$	-/-	-/-	32.26/-	-/-	-/-
Meta-RDN [14]	$\times 2$	38.23/0.9610	34.03/0.9204	32.35/0.9009	33.03/0.9360	39.31/0.9781
ArbEDSR	$\times 2$	38.20/0.9611	34.00/0.9204	32.36/0.9016	32.94/0.9359	39.15/0.9780
ArbRCAN	$\times 2$	<b>38.24/0.9612</b>	<b>34.06/0.9214</b>	<b>32.37/0.9022</b>	<b>33.10/0.9363</b>	<b>39.40/0.9784</b>
Bicubic	$\times 3$	30.39/0.8682	27.55/0.7742	27.21/0.7385	24.46/0.7349	26.95/0.8556
SRCNN [7]	$\times 3$	32.75/0.9090	29.30/0.8215	28.41/0.7863	26.24/0.7989	30.48/0.9117
VDSR [8]	$\times 3$	33.67/0.9210	29.78/0.8320	28.83/0.7990	27.14/0.8290	32.01/0.9340
LapSRN [44]	$\times 3$	33.82/0.9227	29.87/0.8320	28.82/0.7980	27.07/0.8280	32.21/0.9350
MemNet [45]	$\times 3$	34.09/0.9248	30.01/0.8350	28.96/0.8001	27.56/0.8376	32.51/0.9369
SRMD [46]	$\times 3$	34.12/0.9254	30.04/0.8382	28.97/0.8025	27.57/0.8398	33.00/0.9403
CARN [47]	$\times 3$	34.29/0.9255	30.29/0.8407	29.06/0.8034	27.38/0.8404	-/-
MSRN [48]	$\times 3$	34.38/0.9262	30.34/0.8395	29.08/0.8041	28.08/0.8554	33.44/0.9427
EDSR [15]	$\times 3$	34.68/0.9294	30.53/0.8461	29.27/0.8098	28.82/0.8658	34.19/0.9484
RDN [11]	$\times 3$	34.71/0.9296	30.57/0.8468	29.26/0.8093	28.80/0.8653	34.13/0.9484
RCAN [12]	$\times 3$	<b>34.76/0.9300</b>	<b>30.62/0.8473</b>	<b>29.31/0.8108</b>	<b>29.01/0.8684</b>	<b>34.42/0.9497</b>
Meta-EDSR [14]	$\times 3$	-/-	-/-	29.22/-	-/-	-/-
Meta-RDN [14]	$\times 3$	34.73/0.9296	30.58/0.8465	29.30/0.8095	28.93/0.8673	34.40/0.9490
ArbEDSR	$\times 3$	34.72/0.9296	30.58/0.8469	29.30/0.8102	28.84/0.8664	34.34/0.9488
ArbRCAN	$\times 3$	<b>34.78/0.9301</b>	<b>30.62/0.8472</b>	<b>29.31/0.8109</b>	<b>28.98/0.8681</b>	<b>34.56/0.9497</b>
Bicubic	$\times 4$	28.42/0.8104	26.00/0.7027	25.96/0.6675	23.14/0.6577	24.89/0.7866
SRCNN [7]	$\times 4$	30.48/0.8628	27.50/0.7513	26.90/0.7101	24.52/0.7221	27.58/0.8555
VDSR [8]	$\times 4$	31.35/0.8830	28.02/0.7680	27.29/0.7260	25.18/0.7540	28.83/0.8870
LapSRN [44]	$\times 4$	31.54/0.8850	28.19/0.7720	27.32/0.7270	25.21/0.7560	29.09/0.8900
MemNet [45]	$\times 4$	31.74/0.8893	28.26/0.7723	27.40/0.7281	25.50/0.7630	29.42/0.8942
SRMD [46]	$\times 4$	31.96/0.8925	28.35/0.7787	27.49/0.7337	25.68/0.7731	30.09/0.9024
CARN [47]	$\times 4$	32.13/0.8937	28.60/0.7806	27.58/0.7349	26.07/0.7837	-/-
MSRN [48]	$\times 4$	32.07/0.8903	28.60/0.7751	27.52/0.7273	26.04/0.7896	30.17/0.9034
DBPN [10]	$\times 4$	32.47/0.8980	28.82/0.7860	27.72/0.7400	26.38/0.7946	30.91/0.9137
EDSR [15]	$\times 4$	32.47/0.8983	28.81/0.7877	27.73/0.7417	26.65/0.8033	31.04/0.9156
RDN [11]	$\times 4$	32.47/0.8990	28.81/0.7871	27.72/0.7419	26.61/0.8028	31.00/0.9151
RCAN [12]	$\times 4$	<b>32.63/0.8997</b>	<b>28.85/0.7882</b>	<b>27.75/0.7427</b>	<b>26.75/0.8062</b>	<b>31.20/0.9168</b>
Meta-EDSR [14]	$\times 4$	-/-	-/-	27.67/-	-/-	-/-
Meta-RDN [14]	$\times 4$	32.49/0.8984	28.86/0.7881	27.75/0.7419	26.70/0.8050	31.34/0.9175
ArbEDSR	$\times 4$	32.49/0.8980	28.85/0.7877	27.74/0.7418	26.61/0.8027	31.29/0.9164
ArbRCAN	$\times 4$	<b>32.54/0.8981</b>	<b>28.87/0.7878</b>	<b>27.76/0.7421</b>	<b>26.73/0.8046</b>	<b>31.39/0.9167</b>

0.9164 vs. 31.04/0.9156). This clearly demonstrates that the proposed plug-in module can enable scale-arbitrary SR without performance degradation on SR with integer scale factors.

Table 3: Comparison of model size, memory consumption and running time for  $\times 4$  SR. Note that, the memory consumption is calculated on LR input images with a size of  $100 \times 100$ . The running time is averaged over the B100 dataset.

	RDN	Meta-RDN	EDSR	Meta-EDSR	ArbEDSR	RCAN	Meta-RCAN	ArbRCAN
Params.	21.7M	21.4M	42.1M	39.2M	41.9M	15.2M	15.5M	15.6M
Memory	0.3G	2.6G	0.8G	10.2G	1.1G	0.3G	3.1G	0.4G
Time	0.07s	0.29s	0.03s	0.29s	0.11s	0.23s	0.39s	0.26s

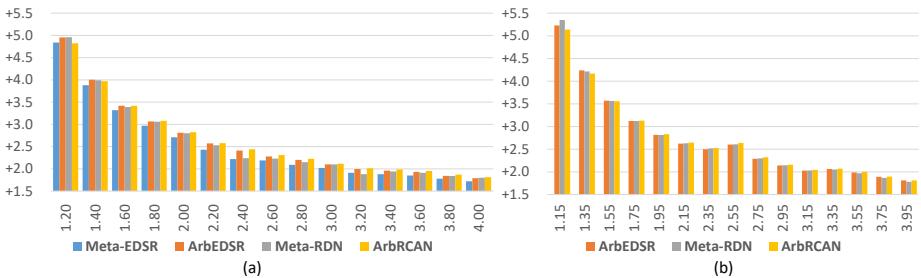


Fig. 5: PSNR performance improvements over Bicubic achieved by Meta-EDSR, ArbEDSR, Meta-RDN and ArbRCAN for different scale factors on B100. Note that, scale factors in (b) are not included in training.

**Efficiency** We further compare our ArbEDSR and ArbRCAN to RDN, Meta-RDN, EDSR, Meta-EDSR, RCAN and Meta-RCAN in terms of model size, memory consumption and running time. Comparative results are listed in Table 3. Both ArbRCAN and Meta-RCAN have comparable model size to their baseline network RCAN. However, our ArbRCAN takes shorter running time and much less memory consumption than Meta-RCAN. Note that, our ArbRCAN is trained for only 70 epochs while Meta-RCAN is trained for 1000 epochs. This clearly demonstrates the high efficiency of our plug-in module.

#### 4.5 Results for SR with Non-Integer Scale Factors

In this section, we compare ArbEDSR and ArbRCAN to Bicubic, Meta-EDSR, and Meta-RDN on SR with non-integer scale factors. The PSNR performance improvements over Bicubic achieved by different methods on B100 are shown in Fig. 5, and the average results are listed in Table 4. Besides, we also provide visual comparison in Fig. 6.

**Quantitative Results** As shown in Fig. 5(a), our ArbEDSR achieves better performance than Meta-EDSR on all scale factors. Moreover, our ArbRCAN outperforms Meta-RDN on all scale factors except 1.2 and 1.4. From Fig. 5(b), we can see that our ArbRCAN achieves the best performance on most scale factors. We can further see from Table 4 that our ArbEDSR outperforms Meta-

Table 4: Comparative results achieved on the B100 dataset. Note that, PSNR/SSIM values are averaged over corresponding scale factors.

Scale	Metric	Bicubic	Meta-EDSR [14]	Meta-RDN [14]	ArbEDSR	ArbRCAN
1.1-4.0 (stride 0.1)	PSNR	28.94	31.54	31.61	31.64	<b>31.64</b>
	SSIM	0.7938	-	0.8563	0.8571	<b>0.8575</b>
1.05-3.95 (stride 0.1)	PSNR	29.11	-	31.86	31.88	<b>31.89</b>
	SSIM	0.8029	-	0.8633	0.8630	<b>0.8634</b>

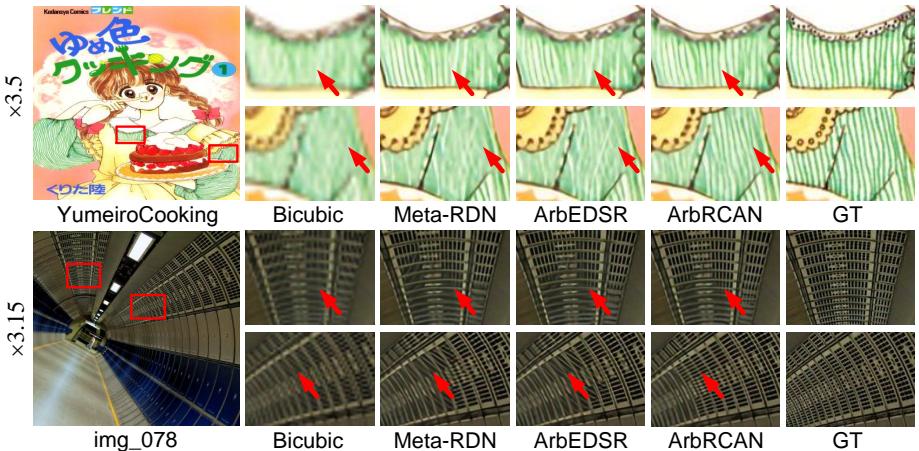


Fig. 6: Visual comparison for non-integer SR (*i.e.*,  $\times 3.5$  SR and  $\times 3.15$  SR).

EDSR by a notable margin, with an averaged PSNR value being improved from 31.54 to 31.64 on the B100 dataset. Moreover, our ArbRCAN achieves the best performance on both seen and unseen non-integer scale factors.

**Qualitative Results** Figure 6 illustrates the qualitative results on two images of the Manga109 and Urban100 datasets. From these zoom-in regions, we can see that our ArbRCAN produces better visual results than other methods with fewer artifacts. For example, Meta-RDN and ArbEDSR cannot recover the stripes reliably and suffer from obvious distorted artifacts on “img\_078” of the Urban100 dataset. In contrast, our ArbRCAN produces finer details.

#### 4.6 Results for SR with Asymmetric Scale Factors

In this section, we test our ArbEDSR and ArbRCAN on the SR task with asymmetric scale factors. Note that, samples with asymmetric scale factors were not included in the training data. We generated LR test images with scale factors varying from 1.5 to 4 (with a stride of 0.5) along the horizontal and vertical axes. Comparative results are illustrated in Fig. 7, while visual comparison is provided in Fig. 8.

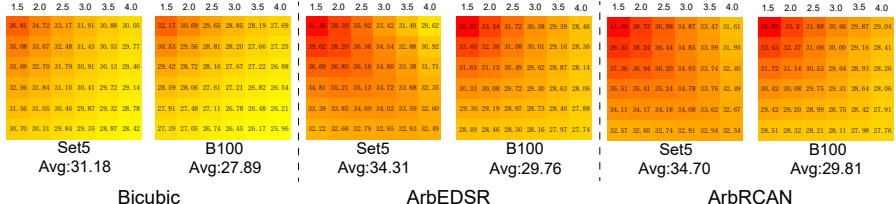


Fig. 7: PSNR performance achieved by Bicubic, ArbEDSR and ArbRCAN for SR with asymmetric scale factors. Horizontal and vertical axes show scale factors along these two dimensions.

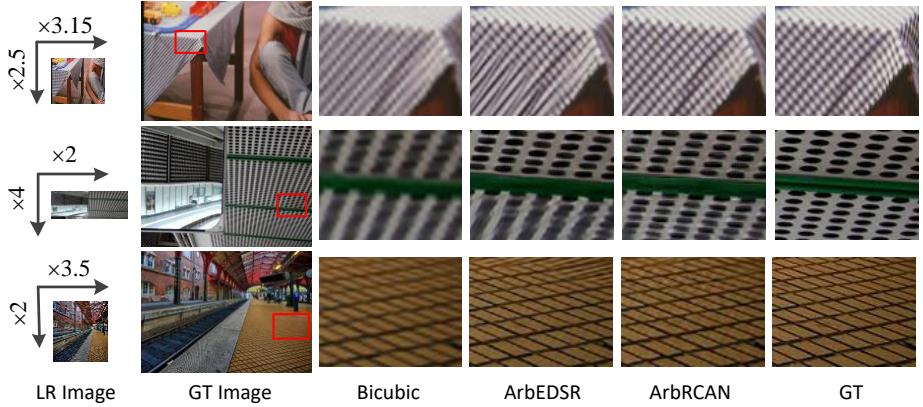


Fig. 8: Visual comparison for asymmetric SR on Set14 and Urban100.

**Quantitative Results** It can be observed from Fig. 7 that our ArbEDSR and ArbRCAN methods outperform Bicubic by notable margins for different scale factors on Set5 and B100. Moreover, our ArbRCAN outperforms ArbEDSR, with average PSNR values being improved from 34.31/29.76 to 34.70/29.81. Although only samples with symmetric scale factors are used for training, our networks show promising generalization capability on SR tasks with unseen asymmetric scale factors.

**Qualitative Results** Figure 8 qualitatively illustrates the results achieved on three images of the Set14 and Urban100 datasets. It can be observed from these zoom-in regions that our ArbRCAN produces better visual results than other methods for different asymmetric scale factors. Specifically, we can see from the first row that, our ArbRCAN recovers the grids reliably while Bicubic suffers from obvious blurring artifacts. This further demonstrates the superior performance of our networks on unseen asymmetric scale factors.

## 5 Conclusions

In this paper, we propose a versatile plug-in module to enable existing single image SR networks for scale-arbitrary SR. We also introduce a scale-aware knowledge transfer paradigm to transfer knowledge from scale-specific networks to a scale-arbitrary network. 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 SR tasks with integer scale factors. Moreover, our module can be easily adapted to scale-specific networks with small additional computational and memory costs.

## References

1. Sun, J., Xu, Z., Shum, H.: Image super-resolution using gradient profile prior. In: CVPR. (2008)
2. Zhang, K., Gao, X., Tao, D., Li, X.: Single image super-resolution with non-local means and steering kernel regression. IEEE Trans. Image Process. **21**(11) (nov 2012) 4544–4556
3. Yang, J., Wang, Z., Lin, Z., Cohen, S., Huang, T.: Coupled dictionary training for image super-resolution. IEEE Trans. Image Process. **21**(8) (aug 2012) 3467–3478
4. Yang, C., Yang, M.: Fast direct super-resolution by simple functions. In: ICCV. (2013) 561–568
5. Timofte, R., Smet, V.D., Gool, L.J.V.: Anchored neighborhood regression for fast example-based super-resolution. In: ICCV. (2013) 1920–1927
6. Gu, S., Zuo, W., Xie, Q., Meng, D., Feng, X., Zhang, L.: Convolutional sparse coding for image super-resolution. In: ICCV. (2015) 1823–1831
7. Dong, C., Loy, C.C., He, K., Tang, X.: Learning a deep convolutional network for image super-resolution. In: ECCV. (2014) 184–199
8. Kim, J., Lee, J.K., Lee, K.M.: Accurate image super-resolution using very deep convolutional networks. In: CVPR. (2016) 1646–1654
9. Caballero, J., Ledig, C., Aitken, A.P., Acosta, A., Totz, J., Wang, Z., Shi, W.: Real-time video super-resolution with spatio-temporal networks and motion compensation. In: CVPR. (2017) 2848–2857
10. Haris, M., Shakhnarovich, G., Ukita, N.: Deep back-projection networks for super-resolution. In: CVPR. (2018) 1664–1673
11. Zhang, Y., Tian, Y., Kong, Y., Zhong, B., Fu, Y.: Residual dense network for image super-resolution. In: CVPR. (2018) 2472–2481
12. Zhang, Y., Li, K., Li, K., Wang, L., Zhong, B., Fu, Y.: Image super-resolution using very deep residual channel attention networks. In: ECCV. (2018) 1646–1654
13. Dai, T., Cai, J., Zhang, Y., Xia, S.T., Zhang, L.: Second-order attention network for single image super-resolution. In: CVPR. (2019)
14. Hu, X., Mu, H., Zhang, X., Wang, Z., Sun, J., Tan, T.: Meta-SR: A magnification-arbitrary network for super-resolution. In: CVPR. (2019)
15. Lim, B., Son, S., Kim, H., Nah, S., Lee, K.M.: Enhanced deep residual networks for single image super-resolution. In: CVPR. (2017)
16. Tai, Y., Yang, J., Liu, X.: Image super-resolution via deep recursive residual network. In: CVPR. (2017) 2790–2798

17. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: ICCV. (2017) 2961–2969
18. Kendall, A., Gal, Y., Cipolla, R.: Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In: CVPR. (2018) 7482–7491
19. Ranjan, R., Patel, V.M., Chellappa, R.: Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. IEEE Trans. Pattern Anal. Mach. Intell. **41**(1) (2019) 121–135
20. Xiao, H., Kang, B., Liu, Y., Zhang, M., Feng, J.: Online meta adaptation for fast video object segmentation. IEEE Trans. Pattern Anal. Mach. Intell. (2019)
21. Schweighofer, N., Doya, K.: Meta-learning in reinforcement learning. Neural Networks **16**(1) (2003) 5–9
22. Finn, C., Levine, S., Abbeel, P.: Guided cost learning: Deep inverse optimal control via policy optimization. In: ICML. Volume 48. (2016) 49–58
23. Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., Levine, S.: Meta-reinforcement learning of structured exploration strategies. In: NeurIPS. (2018) 5307–5316
24. Hochreiter, S., Younger, A.S., Conwell, P.R.: Learning to learn using gradient descent. In: ICANN. Volume 2130. (2001) 87–94
25. Wang, Y., Hebert, M.: Learning to learn: Model regression networks for easy small sample learning. In: ECCV. Volume 9910. (2016) 616–634
26. Ravi, S., Larochelle, H.: Optimization as a model for few-shot learning. In: ICLR. (2017)
27. Lemke, C., Budka, M., Gabrys, B.: Metalearning: a survey of trends and technologies. Artif. Intell. Rev. **44**(1) (2015) 117–130
28. Cai, Q., Pan, Y., Yao, T., Yan, C., Mei, T.: Memory matching networks for one-shot image recognition. In: CVPR. (2018) 4080–4088
29. Yang, T., Zhang, X., Li, Z., Zhang, W., Sun, J.: Metaanchor: Learning to detect objects with customized anchors. In: NeurIPS. (2018) 318–328
30. Zamir, A.R., Sax, A., Shen, W.B., Guibas, L.J., Malik, J., Savarese, S.: Taskonomy: Disentangling task transfer learning. In: CVPR. (2018) 3712–3722
31. Li, Z., Hoiem, D.: Learning without forgetting. In: ECCV. Volume 9908. (2016) 614–629
32. Kulis, B., Saenko, K., Darrell, T.: What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In: CVPR. (2011) 1785–1792
33. Hoffman, J., Darrell, T., Saenko, K.: Continuous manifold based adaptation for evolving visual domains. In: CVPR. (2014) 867–874
34. Martin, D., Fowlkes, C., Tal, D., Malik, J., et al.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: ICCV. (2001)
35. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(Nov) (2008) 2579–2605
36. Shi, W., Caballero, J., Huszar, F., Totz, J., Aitken, A.P., Bishop, R., Rueckert, D., Wang, Z.: Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In: CVPR. (2016) 1874–1883
37. Agustsson, E., Timofte, R.: NTIRE 2017 challenge on single image super-resolution: Dataset and study. In: CVPR. (2017) 1122–1131
38. Bevilacqua, M., Roumy, A., Guillemot, C., Alberi-Morel, M.: Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In: BMVC. (2012) 1–10
39. Zeyde, R., Elad, M., Protter, M.: On single image scale-up using sparse-representations. In: International Conference on Curves and Surfaces. Volume 6920. (2010) 711–730

40. Huang, J., Singh, A., Ahuja, N.: Single image super-resolution from transformed self-exemplars. In: CVPR. (2015) 5197–5206
41. Matsui, Y., Ito, K., Aramaki, Y., Fujimoto, A., Ogawa, T., Yamasaki, T., Aizawa, K.: Sketch-based manga retrieval using manga109 dataset. Multimedia Tools Appl. **76**(20) (2017) 21811–21838
42. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., *et al.*: PyTorch: An imperative style, high-performance deep learning library. In: NeurIPS. (2019)
43. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR. (2015)
44. Lai, W., Huang, J., Ahuja, N., Yang, M.: Deep laplacian pyramid networks for fast and accurate super-resolution. In: CVPR. (2017) 5835–5843
45. Tai, Y., Yang, J., Liu, X., Xu, C.: Memnet: A persistent memory network for image restoration. In: ICCV. (2017) 4549–4557
46. Zhang, K., Zuo, W., Zhang, L.: Learning a single convolutional super-resolution network for multiple degradations. In: CVPR. (2017)
47. Ahn, N., Kang, B., Sohn, K.: Fast, accurate, and lightweight super-resolution with cascading residual network. In: ECCV. (2018) 252–268
48. Li, J., Fang, F., Mei, K., Zhang, G.: Multi-scale residual network for image super-resolution. In: ECCV. (2018) 517–532