Learning Sparse Masks for Efficient Image Super-Resolution

Longguang Wang[1], Xiaoyu Dong[2], Yingqian Wang[1], Xinyi Ying[1], Zaiping Lin[1], Wei An[1], Yulan Guo[1]
[1]National University of Defense Technology
[2]Aerospace Information Research Institute, Chinese Academy of Sciences
{wanglongguang15,yulan.guo}@nudt.edu.cn

Abstract

Current CNN-based super-resolution (SR) methods process all locations equally with computational resources being uniformly assigned in space. However, since high-frequency details mainly lie around edges and textures, less computational resources are required for those flat regions. Therefore, existing CNN-based methods involve much redundant computation in flat regions, which increases their computational cost and limits the applications on mobile devices. To address this limitation, we develop an SR network (SMSR) to learn sparse masks to prune redundant computation conditioned on the input image. Within our SMSR, spatial masks learn to identify "important" locations while channel masks learn to mark redundant channels in those "unimportant" regions. Consequently, redundant computation can be accurately located and skipped while maintaining comparable performance. It is demonstrated that our SMSR achieves state-of-the-art performance with $41\%/33\%/27\%$ *FLOPs being reduced for* $\times 2/3/4$ *SR.*

1. Introduction

The goal of single image super-resolution (SR) is to recover a high-resolution (HR) image from a single low-resolution (LR) observation. Due to the powerful feature representation and model fitting capabilities of deep neural network, CNN-based SR methods have achieved significant performance improvements against traditional ones. Recently, many efforts have been made for real-world applications, including few-shot SR [37, 38], blind SR [11, 45], and scale-arbitrary SR [14, 41]. With the popularity of intelligent edge devices like smartphones and wearable devices, efficient SR is also under great demand [16, 2].

Since the pioneering work of SRCNN [6], deeper networks have been extensively studied for image SR. In VDSR [18], SRCNN is first deepened to 20 layers. Then, a very deep and wide architecture with over 60 layers is introduced into EDSR [27]. Later, Zhang *et al.* further increased the network depth to over 100 and 400 in RDN [47] and RCAN [46], respectively. Although a deep network usually improves SR performance, it also leads to high computational cost and limits the applications on mobile devices. To address this problem, several efforts have been made to reduce model size through information distillation [16] and efficient feature reuse [2]. Nevertheless, these networks still involve redundant computation. Compared to an HR image, missing details in its LR image mainly exist in regions of edges and textures. Consequently, less computational resources are required in those flat regions. However, these CNN-based SR methods process all locations equally, resulting in much redundant computation within flat regions.

In this paper, we propose a sparse mask SR (SMSR) network to skip redundant computation for efficient image SR. Specifically, we learn spatial masks to dynamically identify "important" regions (*e.g.*, edge and texture regions) and use channel masks to mark redundant channels in those "unimportant" regions. These two kinds of masks work jointly to accurately locate redundant computation. During network training, we soften these binary masks using Gumbel softmax trick to make them differentiable. During inference, we use sparse convolution to skip redundant computation.

Our main contributions can be summarized as: 1) We develop an SMSR network to dynamically skip redundant computation for efficient image SR. 2) We propose to locate redundant computation by learning spatial and channel masks. These two kinds of masks work jointly for a fine-grained location of redundant computation. 3) Experimental results show that our SMSR achieves state-of-the-art performance with better inference efficiency. For example, our SMSR outperforms previous methods for $\times 2$ SR on the Set14 dataset with a 39% FLOPs reduction and a $\times 1.6$ speedup on mobile devices.

2. Related Work

In this section, we first review several major works for CNN-based single image SR. Then, we discuss CNN acceleration techniques related to our work, including adaptive inference and network pruning.

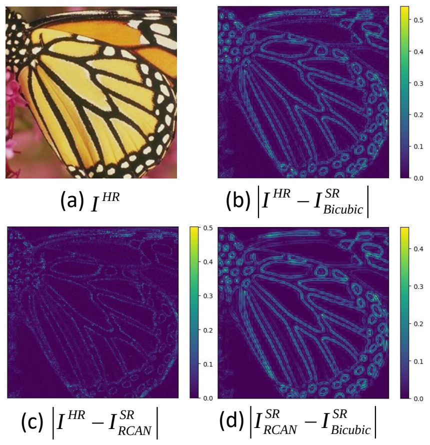
(a) I^{HR}
(b) $\left|I^{HR} - I^{SR}_{Bicubic}\right|$
(c) $\left|I^{HR} - I^{SR}_{RCAN}\right|$
(d) $\left|I^{SR}_{RCAN} - I^{SR}_{Bicubic}\right|$

Figure 1. Absolute difference between $I^{SR}_{Bicubic}$, I^{SR}_{RCAN} and I^{HR} in luminance channel.

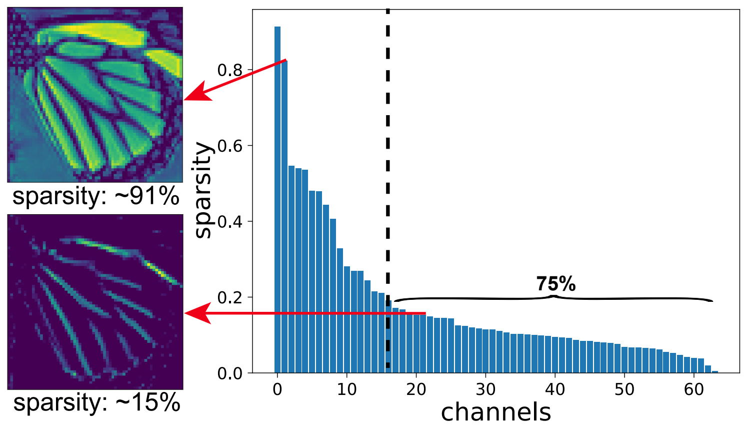

Figure 2. Visualization of feature maps after ReLU layer in the first backbone block of RCAN. Note that, sparisy is defined as the ratio of activated pixels in the corresponding channels.

Single Image SR. CNN-based methods have dominated the research of single image SR due to their strong representation and fitting capabilities. Dong *et al.* [6] first introduced CNNs to learn an LR-to-HR mapping for single image SR. Kim *et al.* [18] then proposed a deeper network with 20 layers (namely, VDSR). Recently, deeper networks are extensively studied for image SR. Lim *et al.* [27] proposed a very deep and wide network (namely, EDSR) by cascading modified residual blocks. Zhang *et al.* [47] further combined residual learning and dense connection to build RDN with over 100 layers. Although these networks achieve state-of-the-art performance, their high computational cost and memory footprint limit the applications on mobile devices. To address this problem, several lightweight networks are developed [21, 16, 2]. Specifically, lightweight distillation blocks are used for feature learning in IDN [16], while a cascading mechanism is introduced to encourage efficient feature reuse in CARN [2]. Different from these manually designed networks, Chu *et al.* [5] developed a compact architecture using neural architecture search. Although existing lightweight SR networks successfully reduce the model size, redundant computation is still involved and hinders them to achieve better computational efficiency.

Adaptive Inference. Adaptive inference techniques [42, 36, 34, 10, 24] have attracted increasing interest since they can adapt the network structure according to the input. One active branch of adaptive inference techniques is to dynamically select an inference path at the levels of layers. Specifically, Wu *et al.* [43] proposed a BlockDrop approach for ResNets to dynamically drop several residual blocks for efficiency. Mullapudi *et al.* [34] proposed an HydraNet with multiple branches and used a gating approach to dynamically choose a set of them at test time. Another popular branch is to dynamically identify "unimportant" regions and skip the computation within these regions. On top of ResNets, Figurnov *et al.* [8] proposed a spatially adaptive computation time (SACT) mechanism to stop computation for a spatial position when the features become "good enough". Liu *et al.* [29] introduced adaptive inference for SR by producing a map of local network depth to adapt the number of convolutional layers implemented at different locations. However, these methods only focus on spatial redundancy without considering redundancy in channel dimension.

Network Pruning. Network pruning [12, 30, 31] is widely used to remove a set of redundant parameters. As a popular branch of network pruning methods, structured pruning approaches are usually used to prune the network at the level of channels and even layers [23, 30, 31, 13]. Specifically, Li *et al.* [23] used L_1 norm to measure the importance of different filters and then pruned less important ones. Liu *et al.* [30] imposed a sparsity constraint on scaling factors of the batch normalization layers and identified channels with lower scaling factors as less informative ones. Different from these static structured pruning methods, Lin *et al.* [28] conducted runtime neural network pruning according to the input image. Recently, Gao *et al.* [9] introduced a feature boosting and suppression method to dynamically prune unimportant channels at inference time. Nevertheless, these pruning methods treat all spatial locations equally without taking their different importance into consideration.

3. Motivation

Existing CNN-based SR networks include much redundant computation in flat regions since these regions are equally processed as regions of edges and textures. To demonstrate this, we first illustrate the intrinsic sparsity of single image SR task and then investigate the feature sparsity in state-of-the-art SR networks.

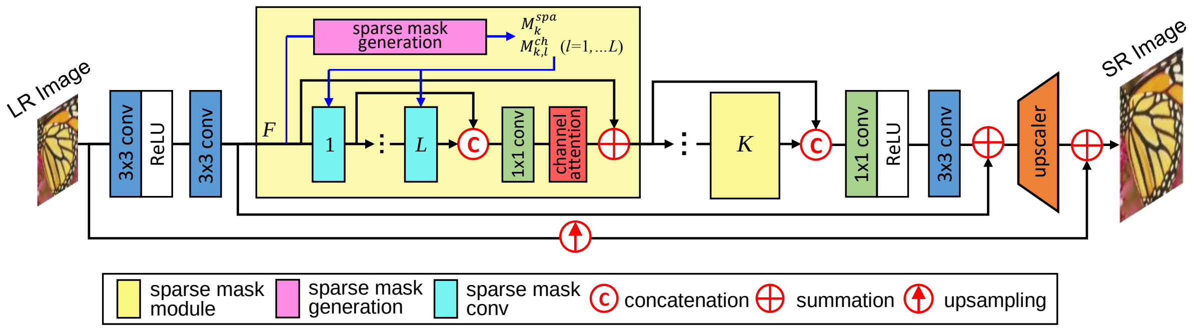

Figure 3. An overview of our SMSR network.

Given an HR image I^{HR} and its LR version I^{LR} ($\times4$ downsampled), we super-resolve I^{LR} using Bicubic and RCAN to obtain $I^{SR}_{Bicubic}$ and I^{SR}_{RCAN}, respectively. Figure 1 shows the absolute difference between $I^{SR}_{Bicubic}$, I^{SR}_{RCAN} and I^{HR} in luminance channel. It can be observed in Fig. 1(b) that $I^{SR}_{Bicubic}$ is "good enough" for flat regions, with noticeable missing details existing in a small proportion of regions only ($\sim$17% pixels with $|I^{HR}-I^{SR}_{Bicubic}|>0.1$). That is, SR task is intrinsically sparse in spatial domain. Compared to Bicubic, RCAN performs better in edge regions while achieving comparable performance in flat regions (Fig. 1(c)). Although RCAN focuses on recovering high-frequency details within edge regions (Fig. 1(d)), those flat regions are equally processed at the same time. Consequently, redundant computation is involved.

Figure 2 further illustrates the feature maps after ReLU layer in a backbone block of RCAN. It can be observed that the spatial sparsity varies significantly for different channels. Moreover, a considerable number of channels are quite sparse (sparsity $\leq$ 0.2), with only edge and texture regions being activated. That is, computation in those flat regions is redundant since these regions are not activated after ReLU. In summary, RCAN activates only a few channels for "unimportant" regions (*e.g.*, flat regions) and more channels for "important" regions (*e.g.*, edge regions).

Motivated by these observations, we learn sparse masks to locate and skip redundant computation for efficient inference. Specifically, our spatial masks dynamically identify "important" regions while the channel masks mark redundant channels in those "unimportant" regions. Compared to network pruning methods [9, 28, 13], we take region redundancy into consideration and only prune channels for "unimportant" regions. Different from adaptive inference networks [36, 25], we further investigate the redundancy in channel dimension to locate redundant computation at a finer-grained level.

4. Our SMSR Network

Our SMSR network uses sparse mask modules (SMM) to prune redundant computation for efficient image SR. Within each SMM, spatial and channel masks are first generated to locate redundant computation, as shown in Fig. 3. Then, the redundant computation is dynamically skipped using sparse mask convolutions. Since only important computation is performed, our SMSR can reduce computational cost while maintaining comparable performance.

4.1. Sparse Mask Generation

1) Training Phase

Spatial Mask. The goal of spatial mask is to identify "important" regions in feature maps. As shown in Fig. 4(a), $F\in R^{C\times H\times W}$ is first fed to an hourglass block to produce $F^{spa}\in R^{2\times H\times W}$. Then, Gumbel softmax trick [17] is used to obtain a softened spatial mask $M_k^{spa}\in R^{H\times W}$:

$$M_k^{spa}[x,y]=\frac{\exp\Big(\big(F^{spa}[1,x,y]+G_k^{spa}[1,x,y]\big)/\tau\Big)}{\sum_{i=1}^{2}\exp\Big(\big(F^{spa}[i,x,y]+G_k^{spa}[i,x,y]\big)/\tau\Big)}, \tag{1}$$

where x,y are vertical and horizontal indices, $G_k^{spa}\in R^{2\times H\times W}$ is a Gumbel noise tensor with all elements following Gumbel(0,1) distribution and τ is a temperature parameter. When $\tau\to\infty$, samples from Gumbel softmax distribution become uniform. That is, all elements in M_k^{spa} are 0.5. When $\tau\to0$, samples from Gumbel softmax distribution become one-hot. That is, M_k^{spa} becomes binary.

Channel Mask. In addition to spatial masks, channel masks are used to mark redundant channels in those "unimportant" regions. For the l^{th} convolutional layer of the k^{th} SMM, we feed auxiliary parameter $S_{k,l}\in R^{2\times C}$ to a Gumbel softmax layer to generate softened channel masks $M_{k,l}^{ch}\in R^{C}$:

$$M_{k,l}^{ch}[c]=\frac{\exp\Big(\big(S_{k,l}[1,c]+G_{k,l}^{ch}[1,c]\big)/\tau\Big)}{\sum_{i=1}^{2}\exp\Big(\big(S_{k,l}[i,c]+G_{k,l}^{ch}[i,c]\big)/\tau\Big)}, \tag{2}$$

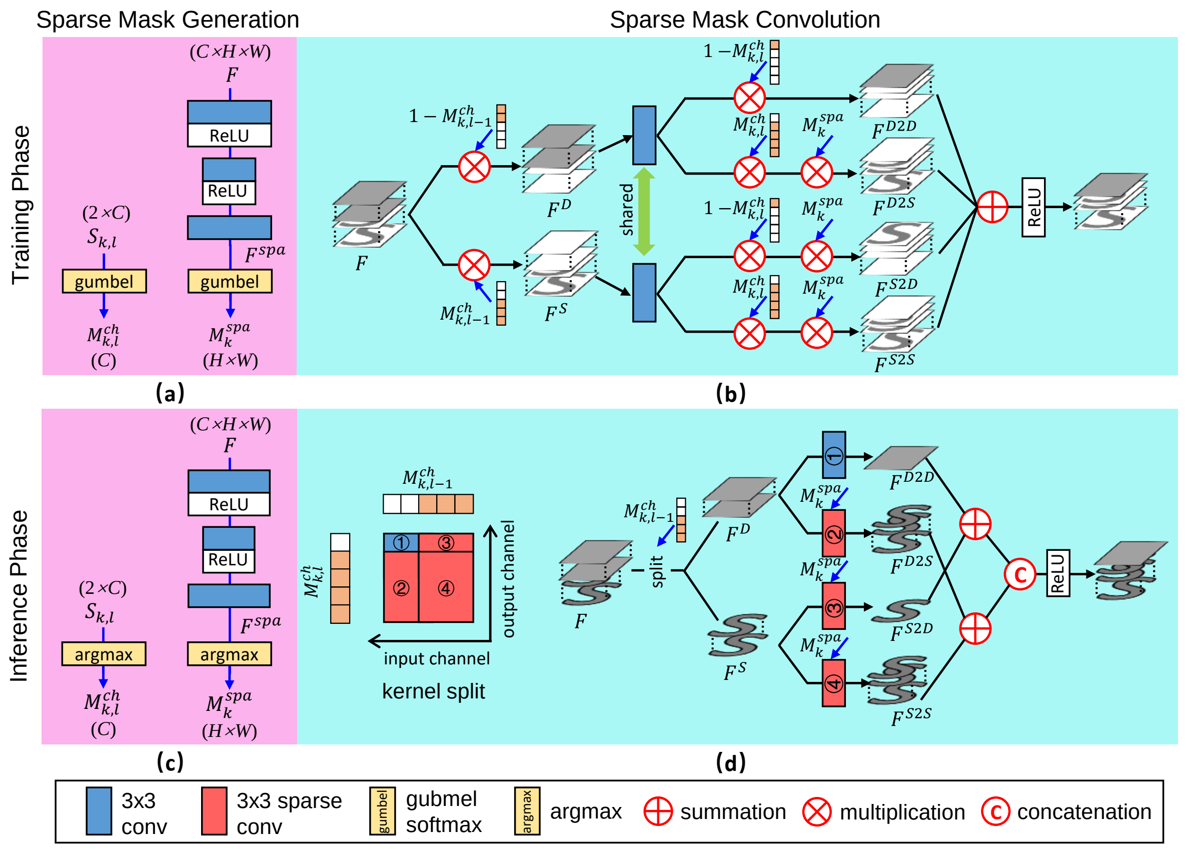

Figure 4. An illustration of sparse mask generation and sparse mask convolution.

where c is channel index and $G^{ch}_{k,l} \in R^{2\times C}$ is a Gumbel noise tensor.

Sparsity Regularization. Based on spatial and channel masks, we define a sparsity term $\eta_{k,l}$:

$$\eta_{k,l} = \frac{1}{C\times H\times W}\sum_{c,x,y}\begin{pmatrix} M^{ch}_{k,l}[c]\times M^{spa}_{k}[x,y]+ \\ (1-M^{ch}_{k,l}[c])\times I[x,y] \end{pmatrix}, \quad (3)$$

where $I \in R^{H\times W}$ is a tensor with all ones. Further, we introduce a sparsity regularization loss based on $\eta_{k,l}$ to encourage lower sparsity:

$$L_{reg} = \frac{1}{K\times L}\sum_{k,l}\eta_{k,l}. \quad (4)$$

Training Strategy. During the training phase, the temperature parameter τ in Gumbel softmax layers is initialized as 1 and gradually decreased to 0.4:

$$\tau = \max(0.4,\ \ 1-\frac{t}{T_{temp}}), \quad (5)$$

where t is the number of epochs and T_{temp} is empirically set to 500 in our experiments.

2) Inference Phase

During training, Gumbel softmax distributions are forced to approach one-hot distributions as τ decreases. Therefore, we replace the Gumbel softmax layers with argmax layers after training to obtain binary spatial and channel masks, as shown in Fig. 4(c).

4.2. Sparse Mask Convolution

1) Training Phase

To enable backpropagation of gradients at all locations, we do not explicitly perform sparse convolution during training. Instead, we multiply the results of a vanilla "dense" convolution with predicted spatial and channel masks, as shown in Fig. 4(b). Specifically, input feature F is first multiplied with $(1-M^{ch}_{k,l-1})$ and $M^{ch}_{k,l-1}$ to obtain F^D and F^S, respectively. That is, channels with "dense" feature maps and "sparse" feature maps in F are separately activated. Next, F^D and F^S are passed to two convolutions with shared weights. The resulting features are then multiplied with different combinations of $(1-M^{ch}_{k,l})$, $M^{ch}_{k,l}$ and M^{spa}_{k} to activate different parts of the features. Finally, all these features are summed up to generate final features. Thanks to Gumbel softmax trick used in mask generation, gradients at all locations can be preserved to optimize the kernel weights of convolutional layers.

2) Inference Phase

During the inference phase, sparse convolution is performed based on the predicted spatial and channel masks, as shown in Fig. 4(d). Take the l^{th} layer in the k^{th} SMM as an example, its kernel is first splitted into four sub-kernels according to $M^{ch}_{k,l-1}$ and $M^{ch}_{k,l}$ to obtain four convolutions. Meanwhile, input feature F is splitted into F^D and F^S based on $M^{ch}_{k,l-1}$. Then, F^D is fed to convolutions ① and ② to produce F^{D2D} and F^{D2S}, while F^S is fed to convolutions ③ and ④ to produce F^{S2D} and F^{S2S}. Note that, F^{D2D} is produced by a vanilla "dense" convolution while F^{D2S}, F^{S2D} and F^{S2S} are generated by sparse convolutions with only "important" regions (marked by M^{spa}_k) being computed. Finally, features obtained from these four branches are summed and concatenated to produce output features. Using sparse mask convolution, computation for redundant channels within those "unimportant" regions can be skipped for efficient inference.

5. Experiments

5.1. Implementation Details

We used 800 training images and 100 validation images from the DIV2K dataset [1] as training and validation set. For evaluation, we used five benchmark datasets including Set5 [3], Set14 [44], B100 [32], Urban100 [15], and Manga109 [33]. Peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) were used as evaluation metrics to measure SR performance. Following the evaluation protocol in [47, 46], we cropped borders and calculated the metrics in the luminance channel.

During training, 16 LR patches of size 96×96 and their corresponding HR patches were randomly cropped. Data augmentation was then performed through random rotation and flipping. We set $C = 64, L = 4, K = 5$ for our SMSR. We used the Adam method [20] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for optimization. The initial learning rate was set to 2×10^{-4} and reduced to half after every 200 epochs. The training was stopped after 1000 epochs. The overall loss for training is defined as $L = L_{SR} + \lambda L_{reg}$, where L_{SR} is the L_1 loss between SR results and HR images. To maintain training stability, we used a warmup strategy for L_{reg}:

$$\lambda = \lambda_0 \times \min(\frac{t}{T_{warm}}, 1), \tag{6}$$

where t is the number of epochs, T_{warm} is empirically set to 50 and λ_0 is set to 0.1.

5.2. Comparison with State-of-the-arts

We compare our SMSR with ten state-of-the-art methods, including SRCNN [6], FSRCNN [7], VDSR [18], DRCN [19], LapSRN [21], MemNet [39], SRFBN-S [26], IDN [16], CARN [2], and FALSR-A [5]. Quantitative results are presented in Table 1 and visual results are shown in Fig. 6.

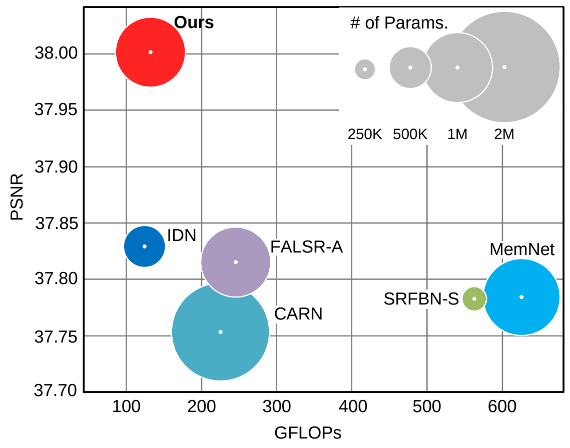

Figure 5. Trade-off between PSNR performance, number of parameters and FLOPs. Results are achieved on Set5 for ×2 SR.

Quantitative Results. As shown in Table 1, our SMSR outperforms the state-of-the-art methods on most datasets. For example, our SMSR achieves much better performance than CARN for ×2 SR, with the number of parameters and FLOPs being reduced by 38% and 41%, respectively. With a comparable model size, our SMSR performs favorably against FALSR-A and achieves better inference efficiency in terms of FLOPs (131.6G vs. 234.7G). With comparable computational complexity in terms of FLOPs (131.6G vs. 127.7G), our SMSR achieves much higher PSNR values than IDN. Using sparse masks to skip redundant computation, our SMSR reduces $41\%/33\%/27\%$ FLOPs for $\times 2/3/4$ SR while maintaining the state-of-the-art performance. We further show the trade-off between performance, number of parameters and FLOPs in Fig. 5. We can see that our SMSR achieves the best PSNR performance with low computational cost.

Qualitative Results. Figure 6 compares the qualitative results achieved on Urban100, Set14 and Manga109. Compared to other methods, our SMSR produces better visual results with fewer artifacts, such as the lattices in img_004 and the stripes on the building in img_033.

5.3. Model Analysis

We first conduct experiments to demonstrate the effectiveness of sparse masks. Then, we further investigate the effect of sparsity and visualize sparse masks for discussion.

Effectiveness of Sparse Masks. To demonstrate the effectiveness of our sparse masks, we first introduced variant 1 by removing both spatial and channel masks. Then, we produce variants 2 and 3 by adding channel masks and spatial

Table 1. Comparative results achieved for $\times 2/3/4$ SR. FLOPs is computed based on HR images with a resolution of 720p (1280×720). For SMSR, average sparsities on all datasets ($0.51/0.61/0.67$ for $\times 2/3/4$ SR) are used to calculate FLOPs, with full FLOPs being shown in brackets. Best and second best results are **highlighted** and underlined.

Model	Scale	#Params	FLOPs	Set5	Set14	B100	Urban100	Manga109
Bicubic	×2	-	-	33.66/0.9299	30.24/0.8688	29.56/0.8431	26.88/0.8403	30.80/0.9339
SRCNN [6]	×2	57K	52.7G	36.66/0.9542	32.45/0.9067	31.36/0.8879	29.50/0.8946	35.60/0.9663
FSRCNN [7]	×2	12K	6.0G	37.00/0.9558	32.63/0.9088	31.53/0.8920	29.88/0.9020	36.67/0.9710
VDSR [18]	×2	665K	612.6G	37.53/0.9590	33.05/0.9130	31.90/0.8960	30.77/0.9140	37.22/0.9750
DRCN [19]	×2	1774K	9788.7G	37.63/0.9588	33.04/0.9118	31.85/0.8942	30.75/0.9133	37.55/0.9732
LapSRN [21]	×2	813K	29.9G	37.52/0.9591	33.08/0.9130	31.08/0.8950	30.41/0.9101	37.27/0.9740
MemNet [39]	×2	677K	623.9G	37.78/0.9597	33.28/0.9142	32.08/0.8978	31.31/0.9195	37.72/0.9740
SRFBN-S [26]	×2	282K	574.4G	37.78/0.9597	33.35/0.9156	32.00/0.8970	31.41/0.9207	38.06/0.9757
IDN [16]	×2	553K	127.7G	37.83/0.9600	33.30/0.9148	32.08/0.8985	31.27/0.9196	38.01/0.9749
CARN [2]	×2	1592K	222.8G	37.76/0.9590	33.52/0.9166	32.09/0.8978	31.92/0.9256	38.36/0.9765
FALSR-A [5]	×2	1021K	234.7G	37.82/0.9595	33.55/0.9168	32.12/0.8987	31.93/0.9256	-/-
SMSR	×2	985K	(224.1G)131.6G	**38.00/0.9601**	**33.64/0.9179**	**32.17/0.8990**	**32.19/0.9284**	**38.76/0.9771**
Bicubic	×3	-	-	30.39/0.8682	27.55/0.7742	27.21/0.7385	24.46/0.7349	26.95/0.8556
SRCNN [6]	×3	57K	52.7G	32.75/0.9090	29.30/0.8215	28.41/0.7863	26.24/0.7989	30.48/0.9117
FSRCNN [7]	×3	12K	5.0G	33.16/0.9140	29.43/0.8242	28.53/0.7910	26.43/0.8080	31.10/0.9210
VDSR [18]	×3	665K	612.6G	33.67/0.9210	29.78/0.8320	28.83/0.7990	27.14/0.8290	32.01/0.9340
DRCN [19]	×3	1774K	9788.7G	33.82/0.9226	29.76/0.8311	28.80/0.7963	27.14/0.8279	32.24/0.9343
MemNet [39]	×3	677K	623.9G	34.09/0.9248	30.01/0.8350	28.96/0.8001	27.56/0.8376	32.51/0.9369
SRFBN-S [26]	×3	375K	686.4G	34.20/0.9255	30.10/0.8372	28.96/0.8010	27.66/0.8415	33.02/0.9404
IDN [16]	×3	553K	57.0G	34.11/0.9253	29.99/0.8354	28.95/0.8013	27.42/0.8359	32.71/0.9381
CARN [2]	×3	1592K	118.8G	34.29/0.9255	30.29/0.8407	29.06/0.8034	28.06/0.8493	33.50/0.9440
SMSR	×3	993K	(100.5G)67.8G	**34.40/0.9270**	**30.33/0.8412**	**29.10/0.8050**	**28.25/0.8536**	**33.68/0.9445**
Bicubic	×4	-	-	28.42/0.8104	26.00/0.7027	25.96/0.6675	23.14/0.6577	24.89/0.7866
SRCNN [6]	×4	57K	52.7G	30.48/0.8628	27.50/0.7513	26.90/0.7101	24.52/0.7221	27.58/0.8555
FSRCNN [7]	×4	12K	4.6G	30.71/0.8657	27.59/0.7535	26.98/0.7150	24.62/0.7280	27.90/0.8610
VDSR [18]	×4	665K	612.6G	31.35/0.8830	28.02/0.7680	27.29/0.7260	25.18/0.7540	28.83/0.8870
DRCN [19]	×4	1774K	9788.7G	31.53/0.8854	28.02/0.7670	27.23/0.7233	25.18/0.7524	28.93/0.8854
LapSRN [21]	×4	813K	149.4G	31.54/0.8850	28.19/0.7720	27.32/0.7270	25.21/0.7560	29.09/0.8900
MemNet [39]	×4	677K	623.9G	31.74/0.8893	28.26/0.7723	27.40/0.7281	25.50/0.7630	29.42/0.8942
SRFBN-S [26]	×4	483K	852.9G	31.98/0.8923	28.45/0.7779	27.44/0.7313	25.71/0.7719	29.91/0.9008
IDN [16]	×4	553K	32.3G	31.82/0.8903	28.25/0.7730	27.41/0.7297	25.41/0.7632	29.41/0.8942
CARN [2]	×4	1592K	90.9G	**32.13/0.8937**	**28.60**/0.7806	**27.58**/0.7349	26.07/0.7837	30.47/0.9084
SMSR	×4	1006K	(57.2G)41.6G	32.12/0.8932	28.55/**0.7808**	27.55/**0.7351**	**26.11/0.7868**	**30.54/0.9085**

Table 2. Comparative results achieved on Set14 by our SMSR with different settings for $\times 2$ SR.

Model	Spatial Mask	Channel Mask	Conv	#Params.	FLOPs	PSNR	SSIM
1	✗	✗	Vanilla	985K	1.00×	33.65	0.9180
2	✗	✓	Vanilla	587K	0.60×	33.53	0.9169
3	✓	✗	Sparse	985K	0.65×	33.60	0.9176
4 (Ours)	✓	✓	Sparse	985K	0.61×	33.64	0.9179

masks, respectively. Comparative results are shown in Table 2. Without spatial and channel masks, variant 1 processes all locations and all channels equally. Therefore, it has a high computational cost. Using channel masks, redundant channels are pruned at all spatial locations in variant 2. Although variant 2 has fewer parameters and FLOPs, it suffers from a notable performance drop (33.53 vs. 33.65) since beneficial information within "important" regions of these pruned channels are discarded. With only spatial masks, variant 3 suffers from a training conflict between efficiency and performance since redundant computation in channel dimension cannot be well handled. Consequently, its FLOPs is reduced with a performance drop (33.60 vs. 33.65). Using both spatial and channel masks, our SMSR can effectively locate and skip redundant computation at a finer-grained level to reduce FLOPs by 39% while main-

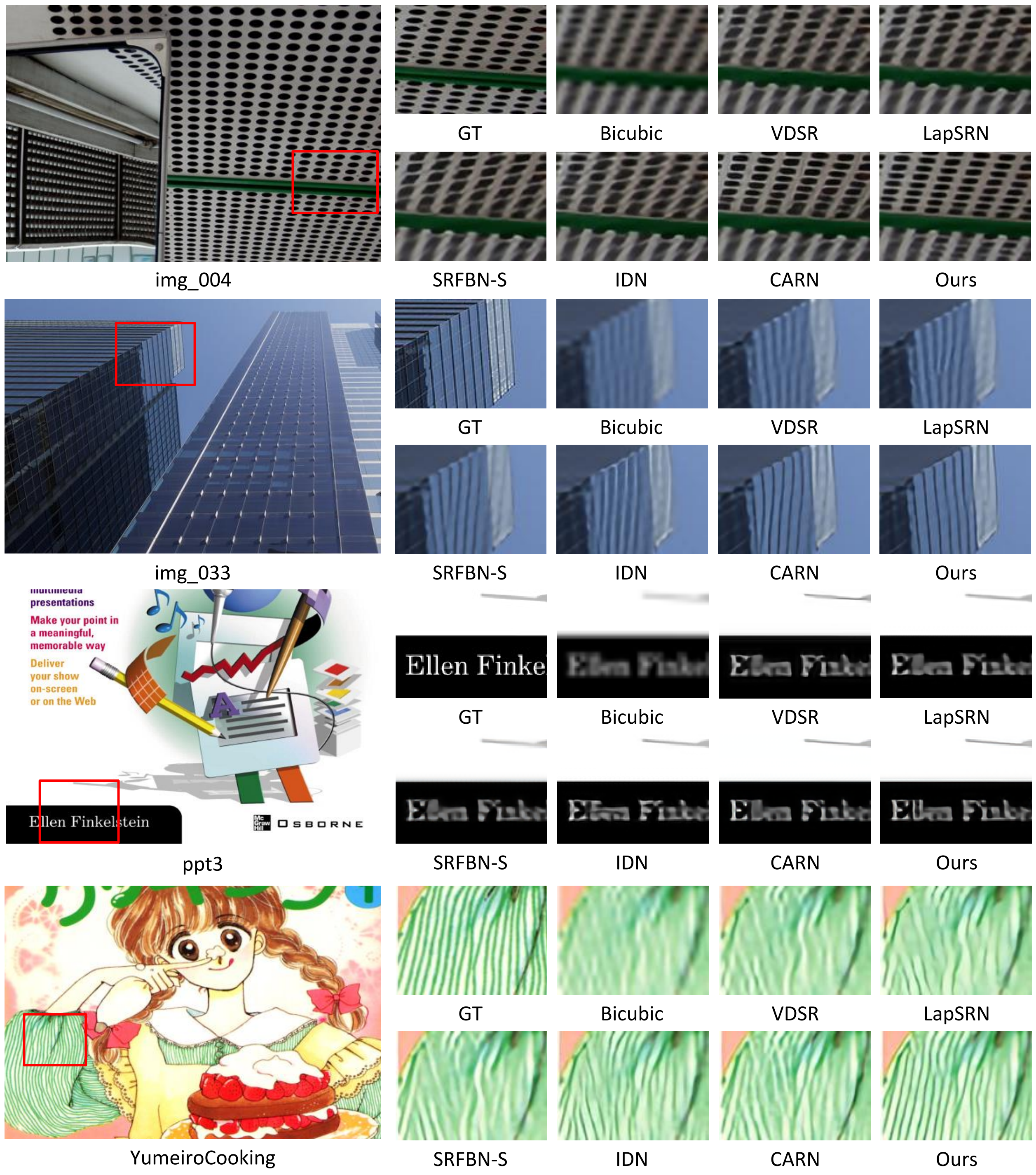

Figure 6. Visual comparison on the Urban100, Set14 and Manga109 datasets for ×4 SR.

taining comparable performance (33.64 vs. 33.65).

Effect of Sparsity. To investigate the effect of sparsity, we retrained our SMSR using higher λ_0 to encourage lower sparsity. Nvidia RTX2080Ti, Intel I9-9900K and Kirin 990/810 are used as platforms of GPU, CPU and mobile processor for evaluation. For fair comparison of memory consumption and inference time, all convolutional layers in the backbone of different networks are implemented using im2col [4] based convolutions to avoid the effects of different implementation methods like Winograd [22] and FFT

Table 3. Comparative results achieved on Set14 by our SMSR with different sparsities for ×2 SR.

Model	Conv	λ_0	Sparsity	#Params.	FLOPs	Memory	Time				PSNR	SSIM
							GPU	CPU	Kirin 990	Kirin 810		
baseline	Vanilla	0	1	985K	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	33.65	0.9180
1	Sparse	0.1	0.54	985K	0.61×	0.89×	1.22×	0.79×	0.64×	0.57×	33.64	0.9179
2	Sparse	0.2	0.36	985K	0.46×	0.87×	1.11×	0.73×	0.55×	0.50×	33.61	0.9174
3	Sparse	0.3	0.27	985K	0.38×	0.85×	1.04×	0.68×	0.54×	0.45×	33.52	0.9169
IDN [16]	-	-	-	553K	0.57×	0.91×	1.04×	0.73×	0.71×	0.60×	33.30	0.9148
CARN [2]	-	-	-	1592K	0.99×	1.01×	1.00×	0.89×	0.96×	1.15×	33.52	0.9166
FALSR-A [5]	-	-	-	1021K	1.04×	2.02×	1.11×	1.05×	1.02×	0.92×	33.55	0.9168

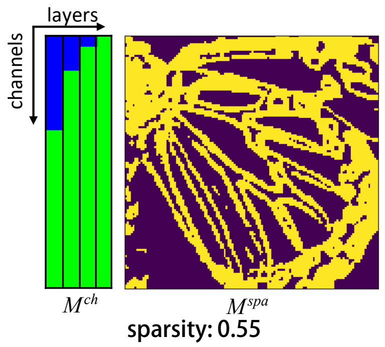

Figure 7. Visualization of sparse masks. Blue regions in M^{ch} represents channels with "dense" feature maps, while green regions refers to channels with "sparse" feature maps. In M^{spa}, "important" locations are marked in yellow.

[40]. Comparative results are presented in Table 3.

As λ_0 increases, our SMSR produces lower sparsities with more FLOPs and memory consumption being reduced. Further, our network also achieves significant speedup on CPU and mobile processors. Since sparse convolution relies on specialized software to realize acceleration on general GPUs [35], the advantage of our SMSR cannot be fully exploited without specific optimization. Compared to other state-of-the-art methods, our SMSR (variant 1) obtains much better performance with lower memory consumption and shorter inference time on mobile processors. This clearly demonstrates the great potential of our SMSR for applications on mobile devices.

Visualization of Sparse Masks. We visualize the sparse masks generated within the first SMM for ×2 SR in Fig. 7. More results are provided in the supplemental material. It can be seen that M^{spa} marks locations around edges and textures as "important" ones, which is consistent with our observations in Sec. 3. Moreover, we can see that channels marked by M^{ch} (*i.e.*, green regions) are more dominant for deeper layers. This indicates that a subset of channels in shallow layers are informative enough for "unimportant" regions and our network progressively focuses more on "important" regions as the depth increases. Overall, our spatial and channel masks work jointly for a fine-grained location of redundant computation.

6. Conclusion

In this paper, we propose a sparse mask SR network for efficient image SR. Our spatial and channel masks work jointly to locate redundant computation at a fine-grained level such that our network can effectively reduce computation cost while maintaining comparable performance. Extensive experiments demonstrate that our network achieves the state-of-the-art performance with significant FLOPs reduction and a speedup on mobile devices.

References

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

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

[3] 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] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *IWFHR*, 2006.

[5] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, Jixiang Li, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search. *arXiv*, abs/1901.07261, 2019.

[6] 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.

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

[8] Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P. Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, pages 1790–1799, 2017.

[9] Xitong Gao, Yiren Zhao, Lukasz Dudziak, Robert D. Mullins, and Cheng-Zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *ICLR*, 2019.

[10] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *CVPR*, pages 9224–9232, 2018.

[11] Jinjin Gu, Hannan Lu, Wangmeng Zuo, Chao Dong, , 1The School of ScienceEngineering, The Chinese University of Hong Kong, and Shenzhen. Blind super-resolution with iterative kernel correction. In *CVPR*, 2019.

[12] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, pages 1135–1143, 2015.

[13] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, pages 4340–4349, 2019.

[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.

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

[16] Zheng Hui, Xiumei Wang, and Xinbo Gao. Fast and accurate single image super-resolution via information distillation network. In *CVPR*, 2018.

[17] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

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

[19] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Deeply-recursive convolutional network for image super-resolution. In *CVPR*, pages 1637–1645, 2016.

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

[21] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. In *CVPR*, pages 5835–5843, 2017.

[22] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *CVPR*, pages 4013–4021, 2016.

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[24] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. In *ICCV*, pages 1891–1900, 2019.

[25] Xiaoxiao Li, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *CVPR*, pages 6459–6468, 2017.

[26] Zhen Li, Jinglei Yang, Zheng Liu, Xiaomin Yang, Gwanggil Jeon, and Wei Wu. Feedback network for image super-resolution. In *CVPR*, pages 3867–3876, 2018.

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

[28] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *NeurIPS*, pages 2181–2191, 2017.

[29] Ming Liu, Zhilu Zhang, Liya Hou, Wangmeng Zuo, and Lei Zhang. Deep adaptive inference networks for single image super-resolution. *arXiv*, 2020.

[30] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, pages 2755–2763, 2017.

[31] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5068–5076, 2017.

[32] 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.

[33] 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.

[34] Ravi Teja Mullapudi, William R. Mark, Noam Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *CVPR*, pages 8080–8089, 2018.

[35] Jongsoo Park, Sheng R. Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. Faster cnns with direct sparse convolutions and guided pruning. In *ICLR*, 2017.

[36] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. Sbnet: Sparse blocks network for fast inference. In *CVPR*, pages 8711–8720, 2018.

[37] Assaf Shocher, Nadav Cohen, and Michal Irani. ”Zero-shot” super-resolution using deep internal learning. In *CVPR*, 2018.

[38] Jae Woong Soh, Sunwoo Cho, and Nam Ik Cho. Meta-transfer learning for zero-shot super-resolution. In *CVPR*, 2020.

[39] Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. Memnet: A persistent memory network for image restoration. In *ICCV*, pages 4549–4557, 2017.

[40] Nicolas Vasilache, Jeff Johnson, Michaël Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A GPU performance evaluation. In *ICLR*, 2015.

[41] Longguang Wang, Yingqian Wang, Zaiping Lin, Jungang Yang, Wei An, and Yulan Guo. Learning for scale-arbitrary super-resolution from scale-specific networks. *arXiv*, 2020.

[42] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, volume 11217, pages 420–436, 2018.

[43] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogério Schmidt Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, pages 8817–8826, 2018.

[44] 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.

[45] Kai Zhang, Luc Van Gool, and Radu Timofte. Deep unfolding network for image super-resolution. In *CVPR*, 2020.

[46] 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.

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