ShuffleMixer: An Efficient ConvNet for Image Super-Resolution

Anonymous Author(s)
Affiliation
Address
email

Abstract

Lightweight and efficiency are critical drivers for the practical application of image super-resolution (SR) algorithms. We propose a simple and effective approach, ShuffleMixer, for lightweight image super-resolution that combines large convolution and channel split-shuffle operation. In contrast to previous SR models that stack multiple small kernel convolutions or complex operators to learn representations, we explore a large kernel ConvNet for mobile-friendly SR design. Specifically, we develop a large depthwise convolution and two projection layers based on channel splitting and shuffling as the basic component to mix features efficiently. Since the contexts of natural images are strongly locally correlated, using large depthwise convolutions only is insufficient to reconstruct fine details. To overcome this problem while keeping it lightweight and efficient, we introduce the Fused-MBConv into the proposed network to model the local connectivity of different features. Experimental results demonstrate that the proposed ShuffleMixer is about $6\times$ smaller than the state-of-the-art methods in terms of model parameters and FLOPs while achieving competitive performance.

1 Introduction

Single image super-resolution (SISR) aims to recover a high-resolution image from a low-resolution one. This is a classic problem that has attracted lots of attention recently due to the rapid development of high-definition devices, such as Ultra-High Definition Television, Samsung Galaxy S22 Ultra, iPhone 13 Pro Max, and HUAWEI P50 Pro, and so on. Thus, it is of great interest to develop an efficient and effective method to estimate high-resolution images to be better displayed on these devices.

Recently, convolutional neural network (CNN) based SR models [8, 9, 1, 15, 24, 45] have achieved impressive reconstruction performance. However, these networks hierarchically extract local features, which highly rely on stacking deeper or more complex models to enlarge the receptive fields for performance improvements. As a result, the required computational budget makes these heavy SR models difficult to deploy on resource-constrained mobile devices in practical applications [44].

To alleviate heavy SR models, various methods have been proposed to reduce model complexity or speed up runtime, including efficient operation design [31, 27, 35, 9, 15, 1, 32, 43, 42, 22, 26], neural architecture search [6, 34], knowledge distillation [12, 13], and structural re-parameterization methodology [7, 22, 44]. These methods are mainly based on improved small spatial convolutions or advanced training strategies, and large kernel convolutions are rarely explored. Moreover, they mostly focus on one of the efficiency indicators and do not perform well in real resource-constrained tasks. Thus, the need to obtain a better trade-off between complexity, latency, and SR quality is imperative.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

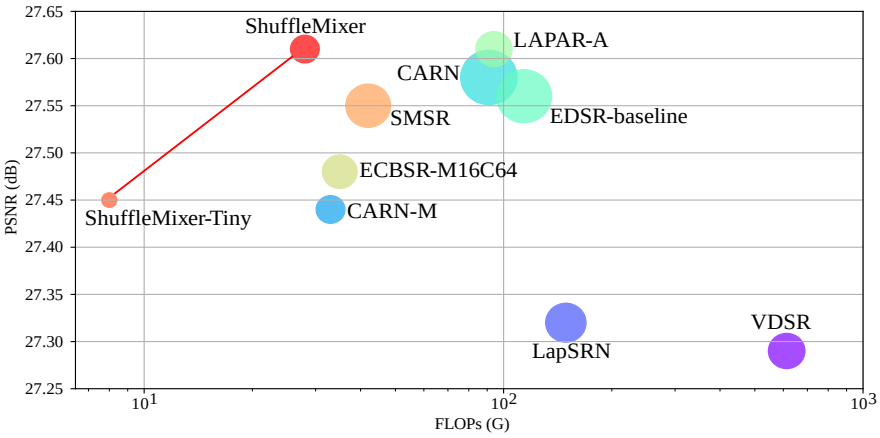


Figure 1: Model complexity and performance comparison between our proposed ShuffleMixer family and other lightweight methods on B100 [2] for $\times 4$ SR. Circle sizes indicate the numbers of parameters. ShuffleMixer achieves better trade-off.

A large receptive field involves more feature interactions, which helps reconstruct more refined results in tasks such as super-resolution that require dense per-pixel predictions. Recent visual transformer (ViT)-based approaches [10, 25, 29, 23, 40] employ a multi-head self-attention (MHSA) mechanism to learn long-range feature representations, which lead to their state-of-the-art performance in various vision tasks. However, MHSA is not friendly to enlarging the receptive field of an efficient SR design. Its complexity grows quadratically with the input resolution (the size is usually large and constant during SR training). Regular convolution with large kernels is also a simple but heavyweight approach to obtaining efficient receptive fields. To make large kernel convolutions practical, using depthwise convolutions with large kernel sizes [26, 38, 7] is an effective alternative. Since depthwise convolutions share connection weights between spatial locations and remain independent between channels, this property makes it challenging to capture sufficient interactions. In lightweight network design, therefore, it is essential to improve the learning capability of DW Convs.

In this paper, we develop a simple but effective network named ShuffleMixer that introduces large kernel convolutions for lightweight SR design. The core idea is to fuse non-local and local spatial locations within a feature mixing block with fewer parameters and FLOPs. Specifically, we employ depthwise convolutions with large kernel sizes to aggregate spatial information from a large region. For channel mixing, we introduce channel splitting and shuffling strategies to reduce model parameters and computational cost and improve network capability. We then build an effective shuffle mixer layer based on these two operators. To further improve the learning capability, we embed the Fused-MBConv into the mixer layer to boost local connectivity. Taken together, we find that the ShuffleMixer network with a simple module can obtain state-of-the-art performance. Figure 1 shows that our ShuffleMixer achieves a better trade-off with the least parameters and FLOPs among all existing lightweight SR methods.

The overall contributions of this paper can be summarized as follows: (1) We introduce large kernel convolutions for efficient SR design, thereby achieving a large effective receptive field. (2) We adopt a channel splitting and shuffling strategy to improve the efficiency of channel projection. (3) We utilize Fused-MBConv to enhance local connectivity with lower model complexity for better learning capability. Benefiting from the combination of these modules, extensive experiments show that the proposed ShuffleMixer is comparable to state-of-the-art efficient SR models on several benchmark datasets.

2 Related Work

CNN-based Efficient SR. CNN-based methods adopt various ways to reduce model complexity. FSRCNN [9] and ESPCN [32] employ post-upsampling layers to reduce the computational burden from predefined inputs significantly. Namhyuk *et al.* [1] uses group convolution and cascading connection upon a recursive network to save parameters. Hui *et al.* [15] proposes a lightweight

information multi-distillation network (IMDN) to aggregate features by applying feature splitting and concatenation operations, and the improved IMDN variants [43, 22] won the AIM2020 and NTIRE2022 Efficient SR challenge. Meanwhile, an increasingly popular approach is to search for a well-constrained architecture as a multi-objective evolution problem [6, 34]. Another branch is to compress and accelerate a heavy deep model through knowledge distillation [13, 12]. Note that fewer parameters and FLOPs do not sufficiently mean faster runtime on mobile devices because FLOPs ignore several important latency-related factors such as memory access cost (MAC) and degree of parallelism [27, 31]. In this paper, we analyze factors affecting the efficiency of SR models and develop a mobile-friendly SR network.

Transformer-based SR. Transformers were initially proposed for language tasks, which stacked the multi-head self-attention and feed-forward MLP layers to learn long-range relations among its inputs. Dosovitskiy *et al.* [10] first applied a vision transformer to image recognition. Since then, ViT-based models have become increasingly applicable to both high-level and low-level vision tasks. For image super-resolution, Chen *et al.* [4] develop a pre-trained image processing transformer (IPT) that directly applies the vanilla ViT to non-overlapped patch embeddings. Liang *et al.* [23] follow Swin Transformer [25] and propose a window-based self-attention model for image restoration tasks and achieve excellent results. Window-based self-attention is much more computationally efficient than global self-attention, but it is still a time-consuming and memory-intensive operation.

Models with Large Kernels. AlexNet [19] is a classic large-kernel convolutional neural network model that inspired many subsequent works. Global Convolutional Network [30] uses symmetric, separable large filters to improve semantic segmentation performance. Due to the high computational cost and a large number of parameters, large-size convolutional filters became not popular after VGG-Net [33]. However, large convolution kernels have recently gained attention with the development of efficient convolution techniques and new architectures such as transformers and MLPs. ConvMixer [38] replaces the mixer component of ViTs [25, 10] or MLPs [37] with large kernel depthwise convolutions. ConvNeXt [26] uses 7×7 depthwise kernels to redesign a standard ResNet and achieves comparable results to Transformers. RepLKNet [7] enlarges the convolution kernel to 31×31 to build a pure CNN model, which obtains better results than Swin Transformer [25] on ImageNet. Unlike these methods that focus on building big models for high-level vision tasks, we explore the possibility of large convolution kernels for lightweight model design in image super-resolution.

3 Proposed Method

We aim to develop an efficient large-kernel CNN model for the SISR task. To meet the efficiency goal, we introduce key designs to the feature mixing block employed to encode information efficiently. This section first presents the overall pipeline of our proposed ShuffleMixer network in detail. Then, we formulate the feature mixing block, which acts as a basic module for building the ShuffleMixer network. Finally, we provide detail on the training loss function.

3.1 ShuffleMixer Architecture

The overall ShuffleMixer architecture. Given a low-resolution image $I_{LR} \in \mathbb{R}^{3\times H\times W}$, ShuffleMixer first employs a convolution to extract overlapping patch embeddings $Z_0 \in \mathbb{R}^{C\times H\times W}$, where C, $H \times W$ denote the number of feature channels and the spatial resolution, respectively. Then, the resulting embeddings Z_0 are fed to a sequence of feature mixing blocks (FMBs) to produce deeper features $Z_1 \in \mathbb{R}^{C\times H\times W}$. Each FMB contains two shuffle mixer layers and a fused-MBConv [35]. The mixer layer performs feature interactions from a wide range, and the Fused-MBConv is applied to enhance feature local connectivity. After many applications of the FMB module, we utilize an upscaling operation to enlarge the feature size and pass it to a convolution layer to generate residual image $I_R \in \mathbb{R}^{3\times H\times W}$. The upsampler module only consists of a conv1x1 layer and a sub-pixel shuffle convolution for saving parameters as much as possible. Finally, we use bilinear interpolation to upscale I_{LR} and add it to the residual image to obtain the final high-resolution image I_{SR}: $I_{SR} = I_{LR} + I_R$.

The Feature Mixing Block itself consists of two shuffle mixer layers and a Fused-MBConv. As illustrated at the top of Fiugre 2, the mixer layer applies mixing operation on Z_0 on spatial and channel dimensions. To mix features at a large spatial range, we introduce depthwise convolution with a large kernel size. Depthwise convolution is a parameter-efficient operation widely used in

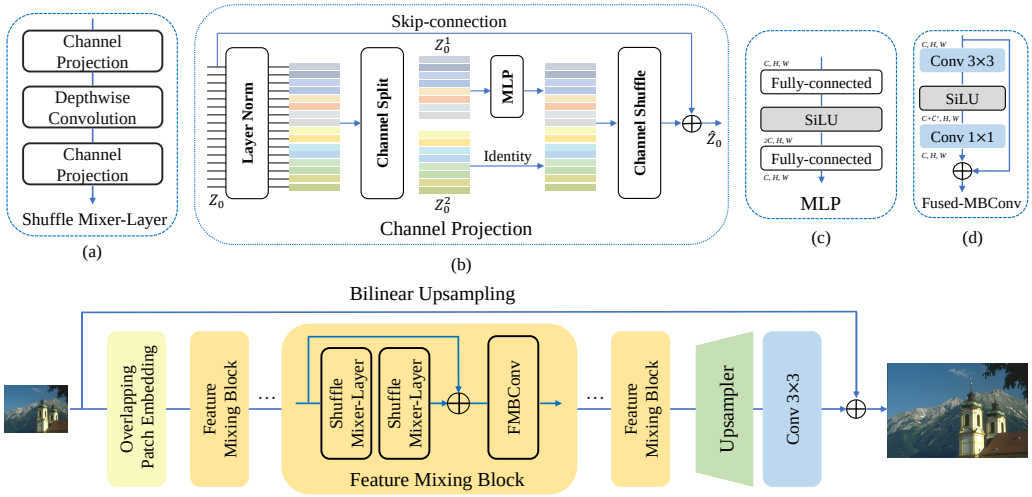


Figure 2: An overview of the proposed ShuffleMixer. Our method consists of overlap patch embedding, feature mixing, and upsampling. The key component is the feature mixing block (FMB) which contains two shuffle mixer layers and (d) Fused-MBConv. Each (a) shuffle mixer layer is further composed of two (b) channel projection layers and a large-kernel depthwise convolution. The channel projection includes channel splitting/shuffling, (c) MLP layers, and skip connections.

lightweight model design [31, 35]. Large kernel DW filters have large effective receptive fields, allowing them to involve more elements for modeling feature dependencies. As we investigated in Table 3, large-kernel depthwise convolutions consistently improve SR performance while maintaining computational efficiency.

With the help of depthwise convolution, the computational cost of the shuffle mixer layer is mainly caused by channel projections. To improve the efficiency of channel projection, we further introduce a channel splitting and shuffling (CSS) strategy [27]. More specifically, the input feature Z_0 is first to split into Z_0^1 and Z_0^2. Then, a channel MLP performs channel mixing on the split features Z_0^1. Finally, a channel shuffling operation is employed to enable the exchange of information on the concatenate feature. Therefore, the parameter complexity of the channel projection layer drops from $\Omega(4C^2)$ to $\Omega(C^2)$. This procedure can be formulated as follow:

$$
\begin{aligned}
[Z_0^1, Z_0^2] &= \text{Split}(\text{LayerNorm}(Z_0)) \\
\hat{Z}_0^1 &= W_1(\sigma(W_0(Z_0^1))))) \\
\hat{Z}_0 &= \text{Shuffle}([\hat{Z}_0^1, Z_0^2])) + Z_0
\end{aligned}
\tag{1}
$$

where σ is SiLU [11] nonlinearity function, W_0 and W_1 are the fully-connected layers; Split and Shuffle represent the splitting and shuffling of features in the channel dimension. As we only process half of the channels, channel shuffling does not guarantee that all features are processed. Inspired by the MobileNetv2 block, we thus repeat the channel projection layer and arrange them before and after DW Conv to learn visual representations. From our study, as listed in Table 2, the enhanced mixer layer achieves quite similar results to the ConvMixer block [38] while using fewer parameters and FLOPs.

Since the content of natural images is locally correlated, the stacked FMB modules do not fully exploit local features, and it requires more capacity to model feature representations for better SR performance. Therefore, we embed a few convolutional blocks into the large-kernel model to enhance local connectivity. Concretely, we evenly add the Fused-MBConv after every two shuffle mixer layers. Fused-MBConv originally contains an expansion conv3x3, a SE layer, and a reduction conv1x1. Using such a Fused-MBConv significantly increases parameters and FLOPs, motivating us to make some changes to match the computation requirements. We remove the SE layer first, as the SiLU can be treated as a gating mechanism to some extent. Note that the inference time slows down

significantly as the hidden dimension expands. Instead of expanding the hidden channel with a large constant factor (by default, 6) of conv3x3, we then limit the number of output channels and expand it to $C + C^{'}$ ($C^{'}$ is experimentally set to16), as shown in Figure 2(d). We also study several operations for this mixing process, and more details can be seen in Sec 4.3.

3.2 Learning Strategy

To constrain the network training, a straightforward way is to ensure that the content of the network output is close to that of the ground truth image:

$$\mathcal{L}_p = \|I_{SR} - I_{GT}\|_1, \tag{2}$$

where I_{SR} and I_{GT} denote the output result and the corresponding ground truth HR image. We note that only using the pixel-wise loss function does not effectively help high-frequency details estimation [5]. We accordingly employ a frequency constraint to regularize network training. The proposed loss function for the network training is defined as:

$$\mathcal{L} = \mathcal{L}_p + \lambda \|\mathcal{F}(I_{SR}) - \mathcal{F}(I_{gt})\|_1 \tag{3}$$

where $\mathcal{F}$ denotes the Fast Fourier transform, and λ is a weight parameter that is set to be 0.1 empirically.

4 Experimental Results

4.1 Datasets and implementation

Datasets. We train our models on the DF2K dataset, a merged dataset with DIV2K [36] and Flickr2K [24], which contains 3450 (800 + 2650) high-quality images. We obtain LR images by downscaling reference HR images using Bicubic interpolation. During the testing stage, we evaluate our models with the peak signal to noise ratio (PSNR) and the structural similarity index (SSIM) on five standard benchmark datasets - Set5 [3], Set14 [41], B100 [2], Urban100 [14] and Manga109 [28]. All PSNR and SSIM results are calculated on the Y channel (YCbCr color space).

Implementation details. We train our model in RGB channels and augment the input patches with random horizontal flips and 90 degree rotations. In each training mini-batch, we randomly crop 64 patches of size 64×64 from LR images as the input. The proposed model is trained by minimizing L1 loss and the frequency loss [5] with Adam [18] optimizer for 300,000 total iterations. The learning rate is set to a constant 5×10^{-4}. All experiments are conducted with the PyTorch framework on an Nvidia Tesla V100 GPU.

We provide two models according to the number of feature channels (C) and the kernel size (K) of DW convolution, and the number of FMB modules is 5. The configuration of ShuffleMixer and ShuffleMixer-Tiny are set to C64-K7 and C32-K3, respectively. The training code and models will be available to the public.

4.2 Comparisons with State-of-the-Art Methods

To evaluate the performance of our approach, we compare the proposed ShuffleMixer with state-of-the-art lightweight frameworks, including SRCNN [8], FSRCNN [9], VDSR [17], DRCN [16], LapSRN [20], CARN [1], EDSR-baseline [24], FALSR-A [6], IMDN [15], LAPAR [21], ECBSR [44], and SMSR [39].

Table 1 shows quantitative comparisons for $\times 2$, $\times 3$, and $\times 4$ SR. In addition to PSNR/SSIM metrics, we also list the number of parameters and FLOPs. The number of FLOPs is tested under the setting of super-resolving an image to 1280×720 pixels. In Figure 1, we compare FLOPS and the number of parameters on the B100 $\times 4$ dataset. Here, our ShuffleMixer model obtains competitive results with even fewer parameters and FLOPs. Especially, ShuffleMixer has a similar number of parameters to CARN-M, but our model outperforms it by a large margin on all benchmark datasets. Even with only 113K parameters, ShuffleMixer-Tiny achieves better performance than many existing methods. With regard to the $\times 2$ and $\times 3$ SR setting, the proposed ShuffleMixer family is capable of achieving similar performance.

Table 1: Comparisons on multiple benchmark datasets for efficient SR networks. All results are calculated on the **Y**-channel. The FLOPs is calculated corresponding to an HR image of size 1280×720. Best and second-best performance are in red and blue color, respectively. Blanked entries link to results not reported in previous works.

Scale	Method	Params	FLOPs	Set5	Set14	B100	Urban100	Manga109
×2	SRCNN [8]	57K	53G	36.66/0.9542	32.42/0.9063	31.36/0.8879	29.50/0.8946	35.74/0.9661
	FSRCNN [9]	12K	6G	37.00/0.9558	32.63/0.9088	31.53/0.8920	29.88/0.9020	36.67/0.9694
	ESPCN [32]	21K	5G	36.83/0.9564	32.40/0.9096	31.29/0.8917	29.48/0.8975	-
	VDSR [17]	665K	613G	37.53/0.9587	33.03/0.9124	31.90/0.8960	30.76/0.9140	37.22/0.9729
	DRCN [16]	1,774K	17,974G	37.63/0.9588	33.04/0.9118	31.85/0.8942	30.75/0.9133	37.63/0.9723
	LapSRN [20]	813K	30G	37.52/0.9590	33.08/0.9130	31.80/0.8950	30.41/0.9100	37.27/0.9740
	CARN-M [1]	412K	91G	37.53/0.9583	33.26/0.9141	31.92/0.8960	31.23/0.9193	-
	CARN [1]	1,592K	223G	37.76/0.9590	33.52/0.9166	32.09/0.8978	31.92/0.9256	-
	EDSR-baseline [24]	1,370K	316G	37.99/0.9604	33.57/0.9175	32.16/0.8994	31.98/0.9272	38.54/0.9769
	FALSR-A [6]	1021K	235G	37.82/0.9595	33.55/0.9168	32.12/0.8987	31.93/0.9256	-
	IMDN [15]	694K	161G	38.00/0.9605	33.63/0.9177	32.19/0.8996	32.17/0.9283	38.88/0.9774
	LAPAR-C [21]	87K	35G	37.65/0.9593	33.20/0.9141	31.95/0.8969	31.10/0.9178	37.75/0.9752
	LAPAR-A [21]	548K	171G	38.01/0.9605	33.62/0.9183	32.19/0.8999	32.10/0.9283	38.67/0.9772
	ECBSR-M16C64 [44]	596K	137G	37.90/0.9615	33.34/0.9178	32.10/0.9018	31.71/0.9250	-
	SMSR [39]	985K	132G	38.00/0.9601	33.64/0.9179	32.17/0.8990	32.19/0.9284	38.76/0.9771
	ShuffleMixer-Tiny(Ours)	108K	25G	37.85/0.9600	33.33/0.9153	31.99/0.8972	31.22/0.9183	38.25/0.9761
	ShuffleMixer(Ours)	394K	91G	38.01/0.9606	33.63/0.9180	32.17/0.8995	31.89/0.9257	38.83/0.9774
×3	SRCNN [8]	57K	53G	32.75/0.9090	29.28/0.8209	28.41/0.7863	26.24/0.7989	30.59/0.9107
	FSRCNN [9]	12K	5G	33.16/0.9140	29.43/0.8242	28.53/0.7910	26.43/0.8080	30.98/0.9212
	VDSR [17]	665K	613G	33.66/0.9213	29.77/0.8314	28.82/0.7976	27.14/0.8279	32.01/0.9310
	DRCN [16]	1,774K	17,974G	33.82/0.9226	29.76/0.8311	28.80/0.7963	27.15/0.8276	32.31/0.9328
	CARN-M [1]	412K	46G	33.99/0.9236	30.08/0.8367	28.91/0.8000	27.55/0.8385	-
	CARN [1]	1,592K	119G	34.29/0.9255	30.29/0.8407	29.06/0.8034	28.06/0.8493	-
	EDSR-baseline [24]	1,555K	160G	34.37/0.9270	30.28/0.8417	29.09/0.8052	28.15/0.8527	33.45/0.9439
	IMDN [15]	703K	72G	34.36/0.9270	30.32/0.8417	29.09/0.8046	28.17/0.8519	33.61/0.9445
	LAPAR-C [21]	99K	28G	33.91/0.9235	30.02/0.8358	28.90/0.7998	27.42/0.8355	32.54/0.9373
	LAPAR-A [21]	594K	114G	34.36/0.9267	30.34/0.8421	29.11/0.8054	28.15/0.8523	33.51/0.9441
	SMSR [39]	993K	68G	34.40/0.9270	30.33/0.8412	29.10/0.8050	28.25/0.8536	33.68/0.9445
	ShuffleMixer-Tiny(Ours)	114K	12G	34.07/0.9250	30.14/0.8382	28.94/0.8009	27.54/0.8373	33.03/0.9400
	ShuffleMixer(Ours)	415K	43G	34.40/0.9272	30.37/0.8423	29.12/0.8051	28.08/0.8498	33.69/0.9448
×4	SRCNN [8]	57K	53G	30.48/0.8628	27.49/0.7503	26.90/0.7101	24.52/0.7221	27.66/0.8505
	FSRCNN [9]	12K	5G	30.71/0.8657	27.59/0.7535	26.98/0.7150	24.62/0.7280	27.90/0.8517
	ESPCN [32]	25K	1G	30.52/0.8697	27.42/0.7606	26.87/0.7216	24.39/0.7241	-
	VDSR [17]	665K	613G	31.35/0.8838	28.01/0.7674	27.29/0.7251	25.18/0.7524	28.83/0.8809
	DRCN [16]	1,774K	17,974G	31.53/0.8854	28.02/0.7670	27.23/0.7233	25 .14/0.7510	28.98/0.8816
	LapSRN [20]	813K	149G	31.54/0.8850	28.19/0.7720	27.32/0.7280	25.21/0.7560	29.09/0.8845
	CARN-M [1]	412K	33G	31.92/0.8903	28.42/0.7762	27.44/0.7304	25.62/0.7694	-
	CARN [1]	1,592K	91G	32.13/0.8937	28.60/0.7806	27.58/0.7349	26.07/0.7837	-
	EDSR-baseline [24]	1,518K	114G	32.09/0.8938	28.58/0.7813	27.57/0.7357	26.04/0.7849	30.35/0.9067
	IMDN [15]	715K	41G	32.21/0.8948	28.58/0.7811	27.56/0.7353	26.04/0.7838	30.45/0.9075
	LAPAR-C [21]	115K	25G	31.72/0.8884	28.31/0.7740	27.40/0.7292	25.49/0.7651	29.50/0.8951
	LAPAR-A [21]	659K	94G	32.15/0.8944	28.61/0.7818	27.61/0.7366	26.14/0.7871	30.42/0.9074
	ECBSR-M16C64 [44]	603K	35G	31.92/0.8946	28.34/0.7817	27.48/0.7393	25.81/0.7773	-
	SMSR [39]	1006K	42G	32.12/0.8932	28.55/0.7808	27.55/0.7351	26.11/0.7868	30.54/0.9085
	ShuffleMixer-Tiny(Ours)	113K	8G	31.88/0.8912	28.46/0.7779	27.45/0.7313	25.66/0.7690	29.96/0.9006
	ShuffleMixer(Ours)	411K	28G	32.21/0.8953	28.66/0.7827	27.61/0.7366	26.08/0.7835	30.65/0.9093

Although IMDN [15], LAPAR-A [21] and SMSR [39] obtain comparable PSNR/SSIM performance, ShuffleMixer requires only a relatively small amount of model complexity. Meanwhile, we compare the GPU run time with fast and lightweight models on $\times 4$ SR: CARN [1], CARN-M [1] and LAPAR-A [21], and the proposed method has fast inference speed. Our ShuffleMixer-Tiny and ShuffleMixer speeds 0.016s and 0.021s to reconstruct an HR image of size 1280×720, respectively. As a comparison, the runtimes are 0.017s, 0.019s, and 0.031s for CARN-M, CARN, and LAPAR-A. Note that Pytorch has poor support for large-kernel depthwise convolution; employing optimized depthwise convolutions can further accelerate the inference time of our method, as suggested in [7]. All these results demonstrate the effectiveness of our method.

As shown in Figure 3, we present visual comparisons on Set14 and Urban100 datasets for a $\times 4$ scale. The qualitative comparison results demonstrate that our proposed methods can produce more visually pleasing results. The structures and details are better recovered.

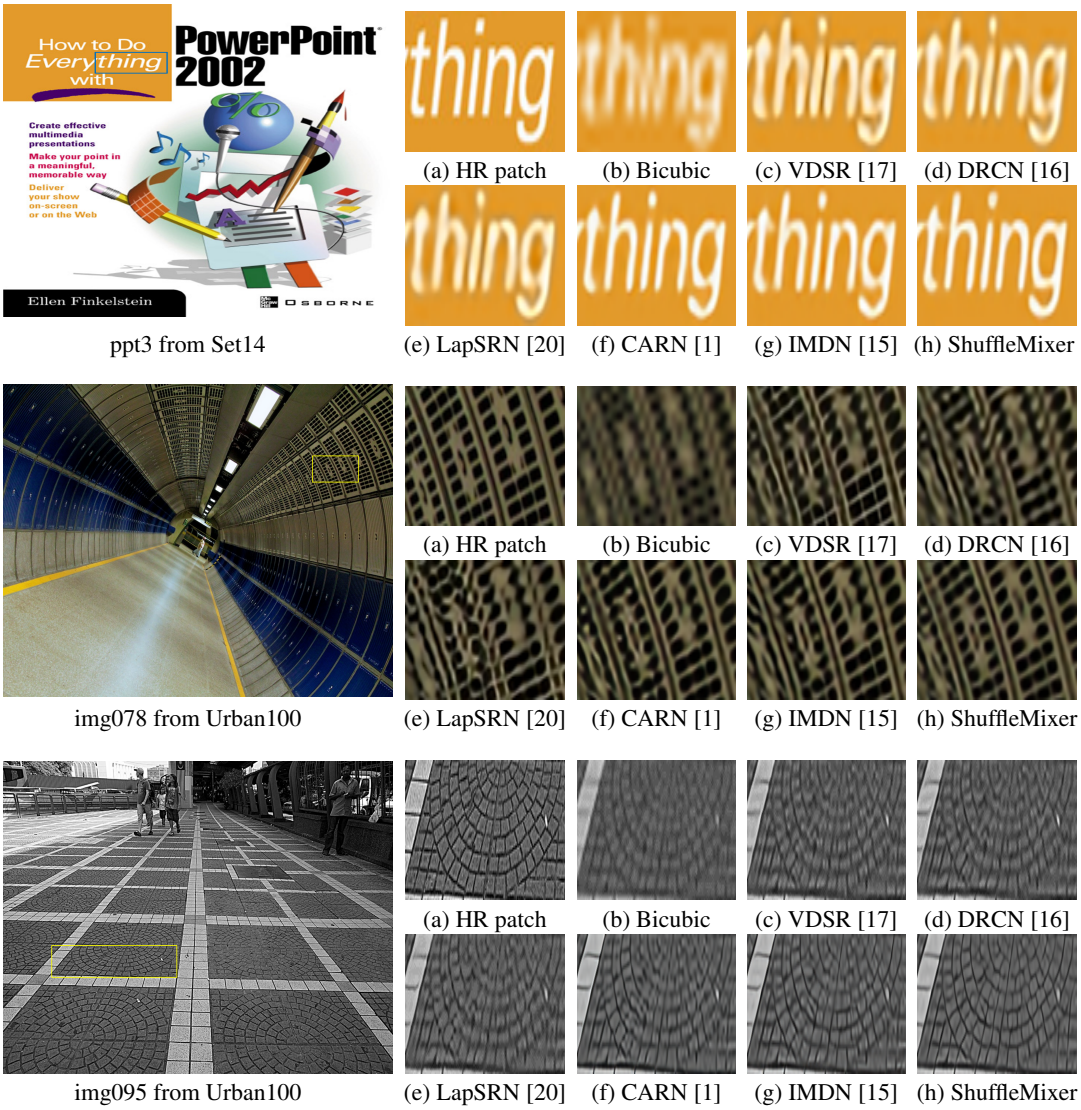


Figure 3: Visual comparisons of ShuffleMixer with other SR methods on Set14 and Urban100 datasets for $\times 4$ SR. Our method recovers the image with clearer structures.

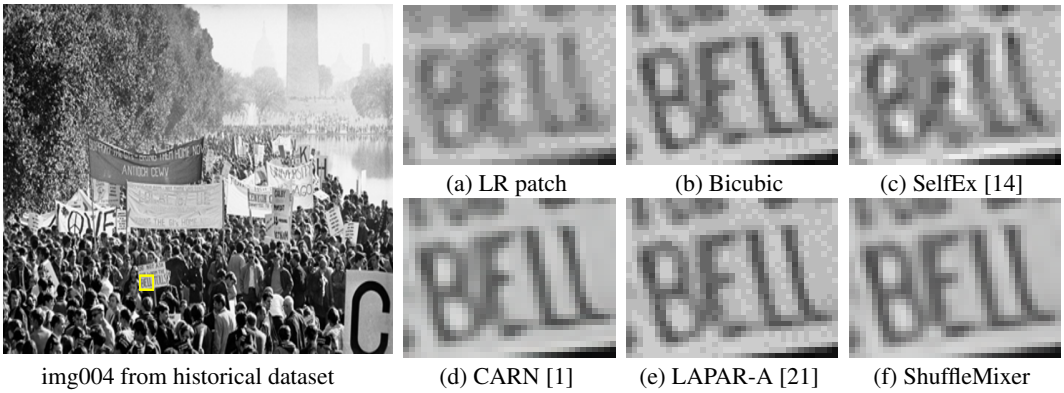


Figure 4: Visual comparisons on historical dataset [20] on $\times 3$ scale. Compared with the results in (b)–(e), the super-resolved image (f) generated by our approach is much clearer with fewer artifacts.

Table 2: Ablation studies of the shuffler mixer layer and feature mixing Bblock on $\times 4$ SR on DIV2K validation set[36]. The FLOPs is tested by fvcore on an LR image of size 256×256. PSNR results are evaluated on **RGB** channels.

		(a) Shuffle Mixer Layer		(b) Feature Mixing Block				
	Baseline	CSS	CDC	Conv	S-Conv	C-Conv	S-ResBlock	S-FMBConv
Params(K)	55.9	24.7	35.5	81.7	81.7	128	128	113
FLOPs(G)	5.2	3.2	3.8	6.9	6.9	9.9	9.9	8.9
PSNR(dB)	28.54	28.39	28.55	28.68	28.72	28.76	28.80	28.78

Table 3: Experimental results on different kernel settings of depthwise convolution. We test the PSNR result on $\times 4$ DIV2K validation set, and compute the FLOPs with the LR input of 256×256.

Kernel Size	LR Size	PSNR	Params	FLOPs
3×3	256×256	28.78	113K	8.9G
5×5	256×256	28.80	118K	9.2G
7×7	256×256	28.85	141K	10.7G
9×9	256×256	28.86	151K	11.4G
11×11	256×256	28.86	164K	12.2G
13×13	256×256	28.85	179K	13.2G

We further evaluate our approach on real low-quality images. One example from the historical dataset [20] is shown in Figure 4. The results by [36, 21] show visible artifacts. Our method and CARN [1] generate smooth detail, but our results have a clearer structure.

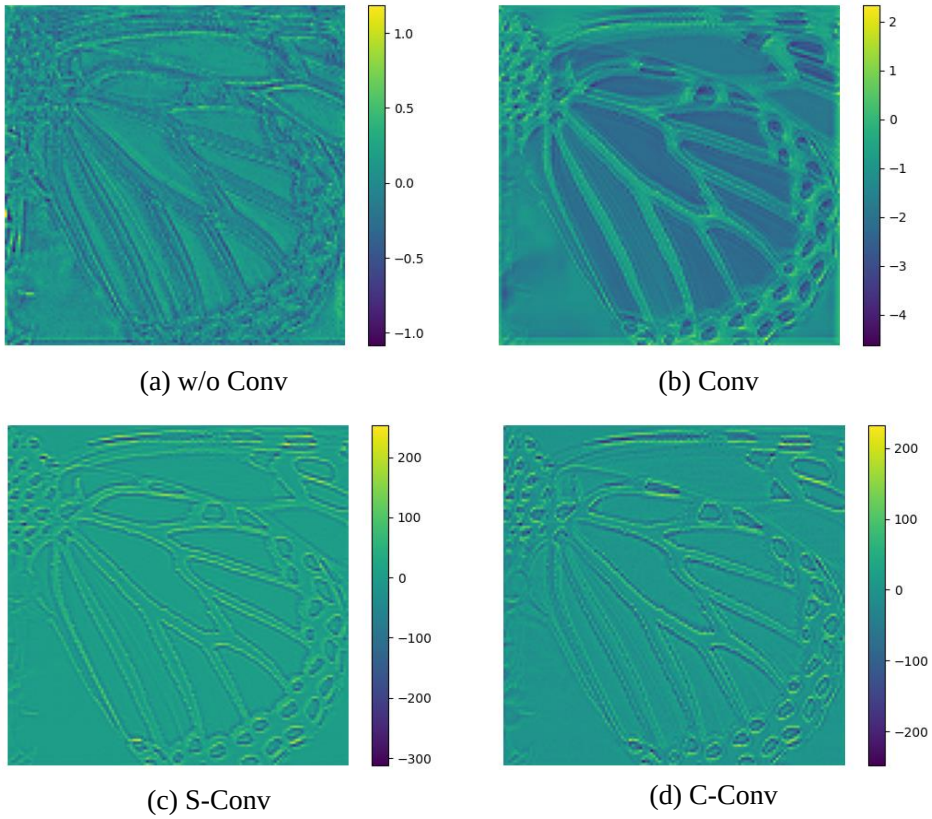
4.3 Analysis and Discussions

The core idea of ShuffleMixer lies in the shuffle mixer layer, feature mixing block, and large kernel convolution. In this subsection, we evaluate them respectively on the proposed tiny model and train them on $\times 4$ DIV2K dataset [36].

Effectiveness of the shuffle mxier layer. To verify the efficiency of the shuffle mixer, we use 10 ConvMixer [38] blocks to build a baseline model. Unlike the original ConvMixer block, we remove BatchNorm and apply LayerNorm before the channel MLP layer to make it more adaptable to the SR task. The kernel of depthwise convolution is set 3, and the number of channels is 32. When applying the channel splitting and shuffling (CSS) strategy, the number of parameters is reduced from 55.9K to 24.7K, and the performance is also 0.15dB lower than the baseline. This result reflects that the split operation limits the representation capability of the channel projection layer. To compensate for the lack of PSNR, we repeat the CSS-based projection layer to enable more cross-group feature mixing (denoted by CDC). Table 2(a) shows a quantitative comparison where we find that CDC achieves very similar performance to the baseline model while reducing parameters from 55.9K to 35.5K and FLOPs from 5.2G to 3.8G.

Effectiveness of the feature mixing block. To validate the effectiveness of the proposed feature mixing block, we first embed a conv3x3 layer after each two shuffle mixer layers, and it has a gain of 0.13dB over the baseline. To further analyze the effect of feature fusion manners, we study the S-Conv (take an element-wise summation between input and output features followed by a conv3x3 layer) and C-Conv (concatenate input and output features on channel dimension followed by a conv3x3 layer). As shown in Table 2(b), they all improve over the baseline; C-Conv achieves better PSNR performance while having more computational cost. Figure 5 exhibits the average feature map before the upsampler module along the channel axis, which illustrates that enhancing local connectivity between features is helpful for grabbing finer details. Based on the S-Conv, we additionally replace the conv3x3 layer with basic residual blocks (S-ResBlock) and Fused-MBConv (S-FMBConv). Table 2(b) shows that S-FMBConv obtains a balanced trade-off between model complexity and SR performance.

Effectiveness of large depthwise convolution. To demonstrate the effect of a large kernel, we try different kernel sizes ranging from 3 to 13 and test their performance separately. From the results shown in Table 3, we can see that the performance generally improves with increasing kernel size.



(a) w/o Conv (b) Conv

(c) S-Conv (d) C-Conv

Figure 5: Visualization of the (average) feature map before upsampler module. We calculate mean values of features on the channel dimension.

Take size 7×7, for example, and it leads to 0.08dB performance improvement by only increasing 8K parameters and 1.8G FLOPs.

5 Conclusion

In this paper, we have proposed a lightweight deep model to solve the image super-resolution problem. The proposed deep model, i.e., ShuffleMixer, contains a shuffler mixer layer with a larger effective receptive field to extract non-local feature representations efficiently. We have introduced the Fused-MBConv to model the local connectivity of features generated by the shuffler mixer layer, which is critical for improving SR performance. We both qualitatively and quantitatively evaluate the proposed ShuffleMixer on commonly used benchmarks. Experimental results demonstrate that the proposed ShuffleMixer is much more efficient while achieving competitive performance than the state-of-the-art methods.

Broader Impact

This paper is an exploratory work on lightweight and efficient image super-resolution using a large-kernel ConvNet. This approach can be deployed in some resource-constrained environments to improve image quality, such as processing photos taken by smartphones and reducing bandwidth during video calls or meetings. However, super-resolution technology has also brought some negative effects, such as criminals using this technology to enhance people's facial or body features, thereby allowing identity information to leak. It is worth noting that the positive social impact of image super-resolution far outweighs the potential problems. We call on people to use this technology and its derivative applications without harming the personal interests of the public.

References

[1] 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] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *PAMI*, 33(5):898–916, 2011.

[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 135.1–135.10, 2012.

[4] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. In *CVPR*, pages 12299–12310, 2021.

[5] Sung-Jin Cho, Seo-Won Ji, Jun-Pyo Hong, Seung-Won Jung, and Sung-Jea Ko. Rethinking coarse-to-fine approach in single image deblurring. In *ICCV*, pages 4641–4650, 2021.

[6] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search. In *ICPR*, pages 59–64, 2021.

[7] Xiaohan Ding, Xiangyu Zhang, Yizhuang Zhou, Jungong Han, Guiguang Ding, and Jian Sun. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. *arXiv preprint arXiv:2203.06717*, 2022.

[8] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *PAMI*, 38(2):295–307, 2016.

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

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[11] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

[12] Qinquan Gao, Yan Zhao, Gen Li, and Tong Tong. Image super-resolution using knowledge distillation. In *ACCV*, pages 527–541, 2019.

[13] Zibin He, Tao Dai, Jian Lu, Yong Jiang, and Shu-Tao Xia. Fakd: Feature-affinity based knowledge distillation for efficient image super-resolution. In *ICIP*, pages 518–522, 2020.

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

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

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

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

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

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[20] 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 624–632, 2017.

[21] Wenbo Li, Kun Zhou, Lu Qi, Nianjuan Jiang, Jiangbo Lu, and Jiaya Jia. Lapar: Linearly-assembled pixel-adaptive regression network for single image super-resolution and beyond. In *NeurIPS*, pages 20343–20355, 2020.

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

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

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

[25] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pages 10012–10022, 2021.

[26] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.

[27] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, pages 116–131, 2018.

[28] Yusuke Matsui, Kota Ito, Yuji Aramaki, Toshihiko Yamasaki, and Kiyoharu Aizawa. Sketch-based manga retrieval using manga109 dataset. *arXiv preprint arXiv:1510.04389*, 2015.

[29] Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *ICLR*, 2022.

[30] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters – improve semantic segmentation by global convolutional network. In *CVPR*, pages 4353–4361, 2017.

[31] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018.

[32] 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.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[34] Dehua Song, Chang Xu, Xu Jia, Yiyi Chen, Chunjing Xu, and Yunhe Wang. Efficient residual dense block search for image super-resolution. pages 12007–12014, 2020.

[35] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *ICML*, volume 139, pages 10096–10106, 2021.

[36] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, and Lei Zhang. Ntire 2017 challenge on single image super-resolution: Methods and results. In *CVPR Workshops*, 2017.

[37] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, pages 24261–24272, 2021.

[38] Asher Trockman and J Zico Kolter. Patches are all you need? In *ICLR*, 2022.

[39] Longguang Wang, Xiaoyu Dong, Yingqian Wang, Xinyi Ying, Zaiping Lin, Wei An, and Yulan Guo. Exploring sparsity in image super-resolution for efficient inference. In *CVPR*, pages 4917–4926, 2021.

[40] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Restormer: Efficient transformer for high-resolution image restoration. In *CVPR*, 2022.

[41] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *Curves and Surfaces*, pages 711–730, 2012.

[42] Kai Zhang, Shuhang Gu, Radu Timofte, et al. Aim 2019 challenge on constrained super-resolution: Methods and results. In *ICCV Workshops*, 2019.

[43] Kai Zhang, Martin Danelljan, Yawei Li, and et al. AIM 2020 challenge on efficient super-resolution: Methods and results. In *ECCV Workshops*, pages 5–40, 2020.

[44] Xindong Zhang, Hui Zeng, and Lei Zhang. Edge-oriented convolution block for real-time super resolution on mobile devices. In *ACM MM*, pages 4034–4043, 2021.

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

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [No]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]