Efficient Hardware-aware Neural Architecture Search for Image Super-resolution on Mobile Devices

Xindong Zhang[1,2,⋆], Hui Zeng[2,⋆], and Lei Zhang[1,2,⋆⋆]

[1] Dept. of Computing, The Hong Kong Polytechnic University
[2] OPPO Research, China
{csxdzhang, cslzhang}@comp.polyu.edu.hk, cshzeng@gmail.com

Abstract. With the ubiquitous use of mobile devices in our daily life, how to design a lightweight network for high-performance image super-resolution (SR) has become increasingly important. However, it is difficult and laborious to manually design and deploy different SR models on different mobile devices, while the existing network architecture search (NAS) techniques are expensive and unfriendly to find the desired SR networks for various hardware platforms. To mitigate these issues, we propose an efficient hardware-aware neural architecture search (EHANAS) method for SR on mobile devices. First, EHANAS supports searching in a large network architecture space, including the macro topology (e.g., number of blocks) and microstructure (e.g., kernel type, channel dimension, and activation type) of the network. By introducing a spatial and channel masking strategy and a re-parameterization technique, we are able to finish the whole searching procedure using one single GPU card within one day. Second, the hardware latency is taken as a direct constraint on the searching process, enabling hardware-adaptive optimization of the searched SR model. Experiments on two typical mobile devices demonstrate the effectiveness of the proposed EHANAS method, where the searched SR models obtain better performance than previously manually designed and automatically searched models. The source codes of EHANAS can be found at https://github.com/xindongzhang/EHANAS.

Keywords: NAS · Super-resolution · Mobile devices.

1 Introduction

Image super-resolution (SR) aims at recovering high-resolution (HR) images from their degraded low-resolution (LR) counterparts. Benefiting from the rapid development of deep learning [22], convolutional neural networks (CNN) based SR models [7,19,27,6,50] have exhibited superior performance over traditional SR methods and shown promising practical values in improving image quality. However, many existing SR networks employ very complicated and cumbersome backbones in order to achieve high reconstruction accuracy, which is infeasible for many real-world applications running on

⋆ Equal contribution
⋆⋆ Corresponding author. This work is supported by the Hong Kong RGC RIF grant (R5001-18) and the PolyU-OPPO Joint Innovation Lab.

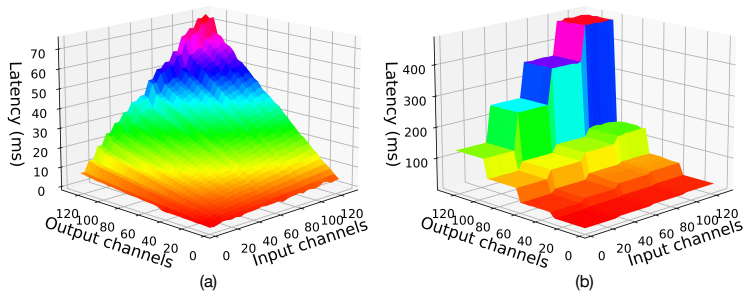

Fig. 1. Latency of applying a 3×3 convolution with different input and output channels to an image of 540×960 resolution using stride 1. The hardware platforms are (a) GPU of Dimensity 1000+ and (b) DSP of Snapdragon 865.

resource-limited edge or mobile devices. Therefore, how to design and train lightweight and efficient SR (LESR) models has been becoming increasingly important and attracting much attention.

One popular trend of LESR is to design convolution blocks with low-FLOPs (e.g., group-wise convolution, depth-wise convolution, and element-wise operator) and parameter-free operators (e.g., splitting, shuffling, concatenation and attention) [36,8,17,16,29,1,40]. However, the FLOPs and number of parameters of an SR model cannot faithfully reflect its real hardware latency [34,46,49]. It still requires a large amount of human labor and resources to adapt and deploy such models on hardware devices. Network pruning is another direction to obtain LESR models by removing less important parameters or blocks from larger SR models [42,25]. However, the pruned model may be sub-optimal on both SR accuracy and inference latency.

The huge space of possible network architectures intertwined with the various hardware platforms makes it very challenging and cumbersome to manually design and deploy SR models on different hardware devices. On one hand, the network architecture varies from macro topology (e.g., number of blocks) to microstructure (e.g., kernel type, channel dimension, and activation type). On the other hand, different mobile or edge devices (e.g., CPU, GPU, DSP, and NPU) can have very different hardware resources such as computational capacity, memory access speed, and supporting operators. As shown in Figure1, even the same operator (e.g., a 3×3 convolution) can have very different hardware latency when evaluated on the same device using different input and output channels.

To alleviate the laborious work on network design and deployment, researchers have made several attempts to leverage the network architecture search (NAS) technique to autonomously search for LESR models [5,4,14,37,23,45]. However, most of these works conduct NAS based on computationally expensive reinforcement learning [14,23] or evolutionary algorithms [5,4,37,45], which take hundreds or even thousands of GPU hours to search a model. To speed up the searching process, one may compromise on the searching space (e.g., only searching a part of the microstructure) or searching step (e.g., using a large step), which may lead to sub-optimal models. In addition, most of the existing methods use the indirect proxy (e.g., FLOPs and number of parameters) to guide the searching process, without considering the hardware setting of target devices [5,4,14,37,23].

To solve the above-mentioned problems, we propose an efficient and hardware-aware NAS (EHANAS) method for SR on mobile devices. Our EHANAS is based on the differential NAS (DNAS) methods [43,41], which have recently proven their effectiveness and efficiency on image classification tasks compared to the NAS methods based on reinforcement learning or evolutionary algorithms [2,52,51,35,28]. However, with multiple convolution branches and intermediate feature maps, the DNAS methods still cost considerable computation and memory resources for searching each block. In this paper, we propose to search a set of spatial and channel masks, which can be used to re-parameterize one single convolution to achieve block search. In this way, our block search takes almost the same computation and memory cost as training one normal convolution block. Combined with a differential strategy for searching the number of blocks, our EHANAS model can be trained using one single GPU card and the whole process can be finished within one day. In addition, the hardware latency is taken as a direct constraint into the objective function using a pre-computed latency look-up table (LUT) from the target hardware device, making the searched model well optimized for the target device.

Our contributions are summarized as follows:

1) We present a highly efficient neural architecture search method for training and deploying SR models on mobile devices, using less than one GPU day.
2) We take the hardware latency as a direct constraint in the optimization process, which enables hardware-adaptive optimization of the searched model.
3) Experiments on two typical mobile devices validate the effectiveness of the proposed EHANAS method. The searched SR models achieve higher SR accuracy than previous ones with comparable or less hardware latency.

2 Related work

LESR network design. Most of the manually designed LESR models mainly focus on reducing the number of FLOPs or network parameters. The pioneer LESR models, such as FSRCNN [8] and ESPCN [36], have validated the efficiency of a plain topology with several convolution blocks. Since then, a lot of more powerful modules have been proposed to improve SR performance while maintaining low FLOPs and small model sizes. For example, Ahn et al. [1] introduced a cascading residual network (CARN) with group convolution that has low FLOPs. Hui et al. [17] designed an information distillation network (IDN) to compress the number of filters per layer. IDN was then extended to the information multi-distillation network (IMDN) [16] which won the AIM 2019 constrained image SR challenge [47]. Liu et al. [29] further improved IMDN to residual feature distillation block (RFDB) and won the AIM 2020 [46] SR challenge. In NTIRE 2022 [26], more architectures and metrics are proposed and discussed for designing LESR. However, neither FLOPs nor parameters can faithfully reflect the real latency, especially on mobile devices [49]. Recently, Zhang et al. [49] carefully designed an edge-oriented convolution block for efficient SR on mobile devices. However, such a manual design consumes a huge amount of human labor and resources. In this work, we propose to automatically search a LESR model for the given target device and latency constraint in a very efficient way.

Network pruning. Network pruning focuses on removing redundant or less important network parameters or blocks, which is a popular approach to compress large models to smaller ones [9,13,30,33]. For example, Li et al. [25] proposed a differentiable meta pruning method and applied it to SR models with a slight performance drop. Zhan et al. [45] utilized a hardware-aware filter pruning algorithm to design LESR models with latency constraints. Though network pruning is effective at reducing computational cost and accelerating inference speed, its upper bound is determined by the original model and it is very likely to generate a sub-optimal solution for the given hardware devices.

Neural architecture search. Neural architecture search (NAS) has emerged as a promising direction for automating the design of LESR models [4,5,14,24,37,38]. However, most of them use computationally intensive reinforcement learning or evolution methods to search models and employ indirect proxies (i.e., parameters and FLOPs) as optimization constraints. As a result, the searched models are usually unfriendly for mobile devices. Single-path NAS [38] provides a micro search space for coarse-grained hardware-aware search. However, it does not support searching the depth of the network which is vital for real-time models on mobile devices and likely to yield sub-optimality. Very recently, Zhan et al. [45] proposed a hardware-aware neural architecture and pruning search method, which takes hardware latency as a direct constraint. However, the proposed NAS method is based on a slow evolution algorithm and it contains three separate stages, including network search, pruning, and tuning. Different from it, we design a totally differentiable NAS pipeline where the searching and tuning are finished in one single stage in less than one GPU day.

3 Methodology

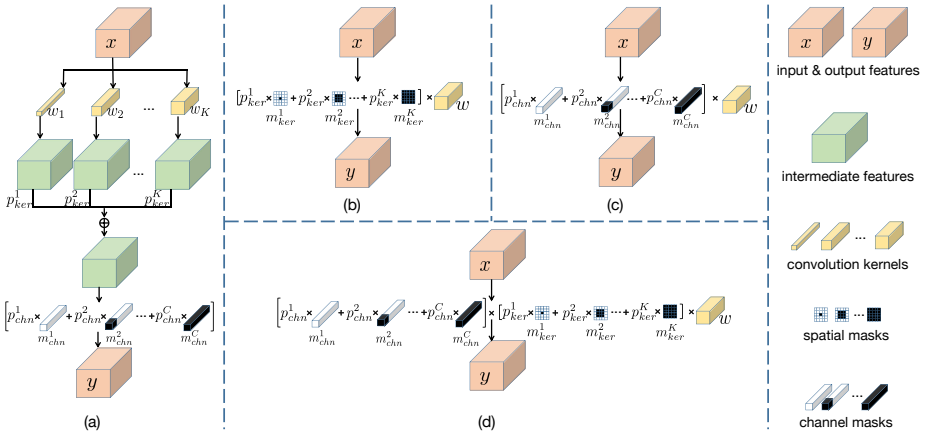

Fig. 2. Illustration of kernel type and channel dimension search. (a) The two-step search process of the DNAS method [41], contains multiple convolution branches and intermediate feature maps. (b) Kernel masking and (c) channel masking of the proposed method. (d) Re-parameterization of both kernel and channel masking for efficient search, where only one single convolution is required.

In this section, we discuss in detail the search for micro and macro network topology, hardware latency estimation, and latency-constrained optimization.

3.1 Block search

The convolution block is the core module for feature extraction in SR models. A typical convolution block consists of three variables: kernel type, channel dimension, and activation type [3]. The kernel type and channel dimension search in the existing DNAS method [41] is shown in Figure 2(a). As can be seen, in the kernel selection step, the input feature is passed into multiple convolution kernels, resulting in multiple intermediate features. The weighted summed feature is then fed into the channel selection step. Although the whole process is differentiable, the computation and memory cost increases linearly with the number of candidate kernels. To address this issue, we propose a kernel and channel masking strategy to conduct block searches at almost constant computation and memory costs.

Kernel mask. Our kernel masking strategy is shown in Figure 2(b). Given the input feature x, instead of employing multiple independent convolution kernels as candidates, we use a set of spatial masks $\{m_{ker}^k\}_{k=1}^K$ to dynamically adjust the spatial size of a shared convolution kernel w, where K is the number of kernel masks. In this way, the kernel search can be achieved by using one single convolution as:

$$y = (\sum_{k=1}^{K} p_{\text{ker}}^k \cdot m_{\text{ker}}^k \cdot w) * x + b \tag{1}$$

where y denotes the output feature, p_{ker}^k is the probability of choosing the k-th kernel mask and p_{ker}^k follows a Gumbel-Softmax distribution [18,31,10]:

$$p_{ker}^k = \frac{exp[(\theta_k + \epsilon_k)/\tau]}{\sum_{k=1}^{K} exp[(\theta_k + \epsilon_k)/\tau]} \tag{2}$$

where θ, ϵ, and τ represent the sampling parameter, Gumbel noise, and temperature, respectively.

With the consideration of mobile-friendly models, we set the spatial size of w to 5×5 and employ five kernel masks, including 1×1, 1×3, 3×1, 3×3 and 5×5.

Channel mask. Similarly, we can employ a set of channel masks to select the channel dimension as shown in Figure 2(c). Denote by m_{chn}^c the c-th channel mask and by p_{chn}^c the corresponding probability (also following Gumbel-Softmax distribution), the channel selection process can be formulated as:

$$y = (\sum_{c=1}^{C} p_{\text{chn}}^c \cdot m_{\text{chn}}^c) \cdot (w * x + b) \tag{3}$$

where C is the total channel dimension of features.

[3] We do not consider those more complicated operators such as splitting, skip connection, and attention for block search since they are not friendly for resource-limited mobile devices [49].

Re-parameterization for efficient search. The kernel and channel selection method shown in Figure 2(a) has two steps, which doubles the training latency. Benefiting from our kernel and channel masking strategy, Eq. 1 and Eq. 3 can be naturally merged into one single convolution:

$$y = w_{rep} * x + b_{rep} \tag{4}$$

where w_{rep} and b_{rep} are the weight and bias after re-parameterization:

$$\begin{cases} w_{rep} = (\sum_{c=1}^{C} p_{chn}^{c} \cdot m_{chn}^{c}) \cdot (\sum_{k=1}^{K} p_{ker}^{k} \cdot m_{ker}^{k}) \cdot w \\ b_{rep} = (\sum_{c=1}^{C} p_{chn}^{c} \cdot m_{chn}^{c}) \cdot b \end{cases} \tag{5}$$

The merged convolution is shown in Figure 2(d).

Activation search. The activation function is important for providing non-linear transformation capability to the SR model. However, most powerful activation functions are not supported by mobile devices. We thus incorporate two mobile-friendly activation functions, i.e., ReLU and identity mapping, as candidates for activation search. Denote by p_{act}^{0} and p_{act}^{1} the probabilities of selecting ReLU and identity mapping, respectively. The activation search can be formulated as follows:

$$\mathcal{A}(x) = p_{act}^{0} \cdot ReLU(x) + p_{act}^{1} \cdot x \tag{6}$$

where $\mathcal{A}$ denotes the weighted activation operation. Since both ReLU and identity mapping are linear functions, $\mathcal{A}$ could be further re-parameterized as follows:

$$\mathcal{A}(x) = \begin{cases} x, x \geq 0 \\ p_{act}^{1} \cdot x, x < 0 \end{cases} \tag{7}$$

Compared to Eq. 6, Eq. 7 consumes less memory access and computation cost, which can further accelerate the searching speed.

The whole block search process, denoted by $\mathcal{C}$, can be formulated as follows:

$$\mathcal{C}(x) = \mathcal{A}(W_{rep} * x + b_{rep}) \tag{8}$$

3.2 Network search

Based on the above block, we could search the overall network topology. Following prior arts on designing mobile-friendly SR models [49], we employ a plain network topology consisting of N blocks $\{C_n\}_{n=1}^{N}$, one skip connection, and one PixelShuffle operation $\mathcal{E}$, as shown in Figure 3(a). The network search can be simplified to the search of the number of blocks, which is usually achieved by using discrete optimization in reinforce-learning or evolution-based NAS methods [45,37,4,5,14]. However,

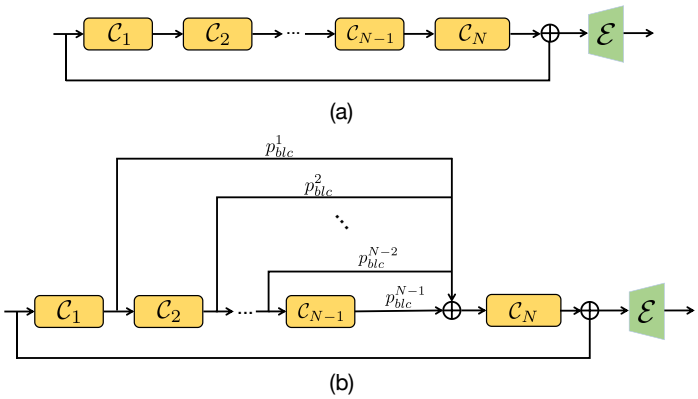

Fig. 3. Illustration of the network search. (a) A mobile-friendly plain topology for mobile SR consisting of N blocks $\{\mathcal{C}_n\}_{n=1}^{N}$, one skip connection, and one PixelShuffle operation $\mathcal{E}$. (b) The differential search of the number of blocks.

this is infeasible for DNAS. As shown in Figure 3(b), we design a differential method to achieve this goal.

Specifically, we assume the network has at least two blocks where the first block C_1 has a fixed number of input channels and the last block C_N has a fixed number of output channels, depending on the SR settings. For each block $C_n, n \leq N-1$, we add a skip connection with probability p^n_{blc} to the input of the last block C_N. Denote by y_n the output of the n-th block and by $\circ$ the symbol of composition functions, we have:

$$y_n = \begin{cases} \mathcal{C}_1(x), n = 1 \\ \mathcal{C}_n \circ \ldots \circ \mathcal{C}_1(x), 1 < n < N \end{cases} \tag{9}$$

Then the output of the network, denoted by $\mathcal{N}$, is a weighted summation of all paths as follows:

$$\mathcal{N}(x) = \mathcal{E} \circ [\mathcal{C}_N \circ (\sum_{n=1}^{N-1} p^n_{blc} \cdot y_n) + x] \tag{10}$$

In this way, the search for block numbers is differentiable. As the probability of block paths $\{p^n_{blc}\}_{n=1}^{N-1}$ converges to a one-hot categorical vector, the depth of the searched network can be determined.

3.3 Hardware latency estimation

To search for a model that fits a target device, we take the hardware latency as a direct constraint in the optimization process. Following prior arts [41] [4], we first calculate

[4] The DL-based latency prediction model [48,12] can be also easily integrated into our framework, while we use pre-calculated LUT in this work for the purpose of straight latency comparison [49] and simplicity.

the hardware latency of each operator at different parameter sizes and record it using a Lookup Table (LUT), as shown in Figure 1. Under our formulation, the latency estimation is naturally differentiable. Specifically, using $LUT_k[c_i, c_o]$ to record the latency of the k-th kernel with input channel c_i and output channel c_o, the latency of the re-parameterized convolution (Eq. 4) can be calculated as:

$$T_{conv} = \sum_{c_o=1}^{C} p_{chn}^{c_o} \cdot (\sum_{c_i=1}^{C} p_{chn}^{c_i} \cdot (\sum_{k=1}^{K} p_{ker}^{k} \cdot LUT_k[c_i, c_o])) \tag{11}$$

Note that the input channel of the current layer is the output channel of the former layer. We do not annotate the layer index in the above formula for simplicity.

Similarly, using $LUT_R[c_o]$ and $LUT_I[c_o]$ to record the latency of ReLU and identity mapping with output channel c_o, the latency of the re-parameterized activation (Eq. 7) can be calculated as:

$$T_{act} = \sum_{c_o=1}^{C} p_{chn}^{c_o} \cdot (p_{act}^{0} \cdot LUT_R[c_o] + p_{act}^{1} \cdot LUT_I[c_o]) \tag{12}$$

Then the latency of the n-th convolution block can be easily obtained as follows:

$$T(\mathcal{C}_n) = T_{conv}^{n} + T_{act}^{n} \tag{13}$$

where T_{conv}^{n} and T_{act}^{n} are the latency of the n-th convolution and activation. The total latency of the network is the weighted summation of all blocks plus several additional operators:

$$T(\mathcal{N}) = \sum_{n=1}^{N-1} [p_{blc}^{n} \cdot T(\mathcal{C}_n)] + T_{add} \tag{14}$$

where T_{add} contains the latency of the last convolution, one skip connection and one PixelShuffle operation.

3.4 Latency constrained optimization

Previous works on LESR search usually contain several stages such as architecture search, network pruning, and tuning [45,4,5,14,37,24]. Benefiting from our totally differential pipeline, we can perform model architecture searching and tuning in just one single stage, which can greatly reduce the time for hardware adaptation. Denote by I_{LR} and I_{HR} a pair of LR and HR images, the latency constrained loss function for SR is defined as follows:

$$\mathcal{L}(\mathcal{N}) = ||\mathcal{N}(I_{LR}) - I_{HR}||_1 + \beta||T(\mathcal{N}) - T_0||_1 \tag{15}$$

where $\mathcal{N}$ is the network parameters to be optimized, T_0 is the required latency on a target device, and β is a weighting parameter to balance the reconstruction loss and latency loss.

4 Experiments

In this section, we employ the proposed EHANAS method to search for LESR models on two typical hardware devices, including the DSP of snapdragon 865 and the GPU of Dimensity 1000+, under certain latency constraints and compare their performance with existing state-of-the-art LESR models on five SR benchmark datasets. We also conduct a comprehensive ablation study to analyze the influence of searching space on the performance of searched models.

4.1 Datasets and implementation details

We employ the DIV2K dataset [39], which consists of 800 training images, to search and train our EHANAS models. The DIV2K validation set and four benchmark SR datasets, i.e., Set5 [3], Set14 [44], BSD100 [32] and Urban [15], are used for model evaluation and comparison. PSNR and SSIM indexes calculated on the Y channel of YCbCr color space are used for performance evaluation.

We use bicubic interpolation to obtain LR images and randomly crop 32 patches of size 64×64 from the LR images as input for each training batch. Random rotations of $90°$, $180°$, and $270°$ are selected as the strategies for data augmentation. Since our EHANAS is a one-shot architecture, we use two ADAM optimizers for architecture search and weight tuning concurrently. Two typical hardware devices, i.e., the DSP of snapdragon 865 and the GPU of Dimensity 1000+, are employed to search hardware-aware LESR models. The latency look-up tables of different operators are pre-computed on both devices. We train the model for 500 epochs using a fixed learning rate at 5×10^{-4}. The parameter β for balancing the fidelity score and latency constraint in Eq. 15 is initialized as 1×10^{-4} for 10 epochs of warm-up, then increased to 3×10^{-2} linearly.

4.2 Quantitative comparison

We first quantitatively compare the searched models by EHANAS against existing representative LESR models, including SRCNN [7], FSRCNN [8], ESPCN [36], ECBSR [49], VDSR [19], LapSRN [21], CARN-M [1], MoreMNAS-{B, C} [5], FALSR-{B, C} [4], TPSR-NoGAN [24], EDSR [27], IMDN [16], RLFN [20], FMEN [11] and EFDN [26]. In Table 1, we summarize the performance comparisons on the DSP of snapdragon 865 and GPU of dimensity 1000+. For the $\times 2$ up-scaling setting, which is more computationally intensive than the $\times 4$ setting, we search the EHANAS model on DSP of snapdragon 865 using three latency constraints, 30ms for real-time speed, 80ms for nearly real-time speed, and 600ms for better accuracy to verify the scalability of the proposed method. For $\times 4$ up-scaling, we search a super real-time, a real-time, and a larger model using 15ms, 30ms, and 150ms as latency constraints, respectively. As for the model settings on GPU of dimensity 1000+, we use {50ms, 200ms, 1200ms} and {20ms, 80ms, 400ms} for $\times 2$ and $\times 4$ upscaling tasks, respectively. In addition to the PSNR/SSIM indexes and latency, we also report some classical proxies such as the number of network parameters, FLOPs, activation, and convolution layers for reference.

Table 1. Performance comparison of different SR models on five benchmarks. PSNR/SSIM scores on Y channel are reported on each dataset. #Params, #FLOPs, #Acts, #Conv, and #Lat represent the total number of network parameters, floating-point operations, activation, convolution layers, and inference latency, respectively. The #FLOPs and #Acts are measured under the setting of generating an SR image of 1280×720 resolution on both scales. The #Lat is measured by generating an SR image of 1920×1080 resolution using the DSP of Snapdragon 865 and GPU of Dimensity 1000+. The best results of each group are highlighted in **bold**.

Scale	Model	#Params (K)	#FLOPs (G)	#Acts (M)	#Conv	#Lat (s)		Set5 [3]	Set14 [44]	B100 [32]	Urban100 [15]	DIV2K [39]
						DSP	GPU					
× 2	Bicubic	—	—	—	—	—	—	33.68/0.9307	30.24/0.8693	29.56/0.8439	26.88/0.8408	32.45/0.9043
	SRCNN [7]	24.00	52.70	89.39	3	1.591	0.890	36.66/0.9542	32.42/0.9063	31.36/0.8879	29.50/0.8946	34.61/0.9334
	ESPCN [36]	21.18	4.55	23.04	3	0.072	0.080	36.83/0.9544	32.40/0.9060	31.29/0.8870	29.48/0.8948	34.63/0.9335
	FSRCNN [8]	12.46	6.00	40.53	8	0.114	0.076	36.98/0.9556	32.62/0.9087	31.50/0.8904	29.85/0.9009	34.74/0.9340
	MOREMNAS-C [5]	25.00	5.50	269.11	49	—	—	37.06/0.9561	32.75/0.9094	31.50/0.8904	29.92/0.9023	34.87/0.9356
	ECBSR-M4C16 [49]	10.20	2.34	19.35	6	0.033	0.046	37.33/0.9580	32.81/0.9100	31.66/0.8931	30.31/0.9091	35.15/0.9380
	TPSR-NoGAN [24]	60.00	14.00	50.69	14	—	—	37.38/0.9583	33.00/0.9123	31.75/0.8942	30.61/0.9119	—
	EHANAS-GPU-50ms	11.90	2.74	35.02	8	—	0.049	37.42/0.9585	32.95/0.9122	31.71/0.8940	30.36/0.9092	35.23/0.9383
	EHANAS-DSP-30ms	35.40	8.13	44.97	7	0.029	—	**37.63/0.9593**	**33.15/0.9137**	**31.89/0.8960**	**30.98/0.9168**	**35.51/0.9408**
	IMDN-RTC [16]	19.70	4.57	65.20	28	1.101	1.076	37.51/0.9590	32.93/0.9122	31.79/0.8950	30.67/0.9120	35.34/0.9398
	LapSRN [21]	813.00	29.90	223.03	14	4.395	1.624	37.52/0.9590	33.08/0.9130	31.80/0.8950	30.41/0.9100	35.31/0.9400
	VDSR [19]	665.00	612.60	1121.59	20	8.946	3.379	37.53/0.9587	33.05/0.9127	31.90/0.8960	30.77/0.9141	35.43/0.9410
	CARN-M [1]	412.00	91.20	649.73	42	0.884	1.195	37.53/0.9583	33.26/0.9141	31.92/0.8960	31.23/0.9193	35.62/0.9420
	MOREMNAS-B [5]	1118.00	256.90	987.96	79	—	—	37.58/0.9584	33.22/0.9135	31.91/0.8959	31.14/0.9175	35.46/0.9402
	FALSR-B [4]	326.00	74.70	372.33	49	—	—	37.61/0.9585	33.29/0.9143	31.97/0.8967	31.28/0.9191	35.58/0.9408
	FALSR-C [4]	408.00	93.70	379.70	34	—	—	37.66/0.9586	33.26/0.9140	31.96/0.8965	31.24/0.9187	35.57/0.9407
	EDSR-R5C32 [27]	130.80	30.31	111.51	13	0.150	0.653	37.61/0.9590	33.06/0.9127	31.87/0.8959	30.90/0.9162	35.45/0.9407
	ECBSR-M10C32 [49]	94.70	21.81	82.02	12	0.062	0.203	37.76/0.9595	33.26/0.9136	32.04/0.8970	31.25/0.9190	35.68/0.9421
	EHANAS-GPU-200ms	127.90	29.47	92.16	10	—	0.208	37.80/**0.9597**	33.31/0.9147	**32.05**/0.8972	31.32/0.9191	35.72/0.9425
	EHANAS-DSP-80ms	99.40	22.89	89.40	10	0.077	—	**37.83**/0.9596	**33.33/0.9152**	32.04/**0.8974**	**31.36/0.9198**	**35.73/0.9426**
	ECBSR-M16C64 [49]	596.00	137.31	251.60	18	0.513	0.786	37.90/0.9600	33.34/0.9153	32.10/0.8982	31.71/0.9250	35.79/0.9430
	EDSR-R16C64 [27]	1334.90	307.89	546.51	37	1.447	2.372	37.99/0.9604	33.57/0.9175	32.16/0.8994	31.98/0.9272	35.85/0.9436
	IMDN [16]	660.30	152.04	406.43	34	10.610	12.792	37.99/0.9603	33.39/0.9156	32.14/0.8993	32.03/0.9279	35.87/0.9436
	EHANAS-GPU-1200ms	1336.3	307.89	376.93	18	—	1.175	38.00/0.9605	33.53/0.9174	32.15/0.8995	32.00/0.9275	38.85/0.9435
	EHANAS-DSP-600ms	1188.90	273.91	487.53	34	0.583	—	**38.05/0.9611**	**33.60/0.9180**	**32.18/0.8998**	**32.05/0.9280**	**35.89/0.9437**
× 4	Bicubic	—	—	—	—	—	—	28.43/0.8113	26.00/0.7025	25.96/0.6682	23.14/0.6577	28.10/0.7745
	SRCNN [7]	57.00	52.7	89.39	3	1.583	0.896	30.48/0.8628	27.49/0.7503	26.90/0.7101	24.52/0.7221	29.25/0.8090
	ESPCN [36]	24.90	1.44	6.45	3	0.026	0.032	30.52/0.8647	27.42/0.7516	26.87/0.7100	24.39/0.7211	29.32/0.8100
	FSRCNN [8]	12.00	5.00	10.81	8	0.032	0.028	30.70/0.8657	27.59/0.7535	26.96/0.7128	24.60/0.7258	29.36/0.8110
	ECBSR-M4C16 [49]	11.90	0.69	5.53	6	0.011	0.028	31.04/0.8785	27.78/0.7645	27.09/0.7200	24.79/0.7422	29.62/0.8187
	TPSR-NoGAN [24]	61.00	3.60	13.13	15	—	—	31.10/0.8779	27.95/0.7663	27.15/0.7214	24.97/0.7456	29.77/0.8200
	EHANAS-GPU-20ms	21.00	1.21	9.22	10	—	0.022	31.25/0.8812	28.03/0.7680	27.20/0.7248	25.01/0.7480	29.80/0.8221
	EHANAS-DSP-15ms	66.60	3.87	17.63	10	0.014	—	**31.57/0.8855**	**28.23/0.7725**	**27.31/0.7255**	**25.34/0.7595**	**29.94/0.8260**
	IMDN-RTC [16]	21.00	1.22	16.99	28	0.318	0.287	31.22/0.8810	27.92/0.7660	27.18/0.7217	24.98/0.7477	29.76/0.8200
	VDSR [19]	665.00	612.60	1121.59	20	9.036	3.365	31.35/0.8838	28.02/0.7678	27.29/0.7252	25.18/0.7525	29.82/0.8240
	LapSRN [21]	813.00	149.40	264.04	27	5.378	5.801	31.54/0.8850	28.19/0.7720	27.32/0.7280	25.21/0.7560	29.88/0.8250
	EDSR-R5C32 [27]	241.80	14.15	50.69	13	0.101	0.567	31.46/0.8845	28.07/0.7682	27.27/0.7250	25.21/0.7561	29.87/0.8251
	ECBSR-M10C32 [49]	98.10	5.65	21.20	12	0.017	0.063	31.66/0.8880	28.15/0.7725	27.34/0.7283	25.41/0.7650	29.98/0.8275
	EHANAS-GPU-80ms	194.5	11.20	23.04	7	—	0.079	31.73/0.8885	28.33/**0.7759**	27.39/0.7300	25.43/0.7651	30.05/0.8281
	EHANAS-DSP-30ms	177.70	10.45	47.89	26	0.027	—	**31.75/0.8887**	**28.34**/0.7758	**27.42/0.7302**	**25.45/0.7653**	**30.05/0.8283**
	CARN-M [1]	412.00	46.10	222.11	43	0.170	0.362	31.92/0.8903	28.42/0.7762	27.44/0.7304	25.62/0.7694	30.10/0.8311
	ECBSR-M16C64 [49]	602.90	34.73	63.59	18	0.071	0.209	31.92/0.8929	28.34/0.7756	27.48/0.7323	25.81/0.7780	30.15/0.8315
	EDSR-R16C64 [27]	1778.00	102.85	181.56	37	0.527	1.639	32.09/0.8938	28.58/0.7813	27.57/0.7357	26.04/0.7849	30.21/0.8336
	IMDN [16]	667.40	38.41	102.30	34	2.782	2.672	32.03/0.8929	28.42/0.7783	27.48/0.7320	25.96/0.7804	30.22/0.8336
	EHANAS-GPU-400ms	1513.2	87.16	105.98	20	—	0.395	32.02/0.8930	28.50/0.7802	27.53/0.7334	26.00/0.7844	30.22/0.8336
	EHANAS-DSP-150ms	1269.9	73.15	129.95	36	0.148	—	**32.11/0.8941**	**28.60/0.7814**	**27.58/0.7357**	**26.05/0.7850**	**30.25/0.8337**
	RLFN [20]	317.2	17.31	70.35	39	0.893	0.192	32.07/0.8927	28.59/0.7808	27.56/0.7349	26.09/0.7842	30.43/0.8367
	FMEN [11]	341.1	19.58	63.36	34	0.874	0.303	32.01/0.8925	28.56/0.7808	27.55/0.7346	26.00/0.7818	30.44/0.8367
	EFDN [26]	276.2	14.70	97.65	65	1.005	1.070	32.05/0.8920	28.57/0.7801	27.54/0.7342	25.99/0.7805	30.44/0.8365

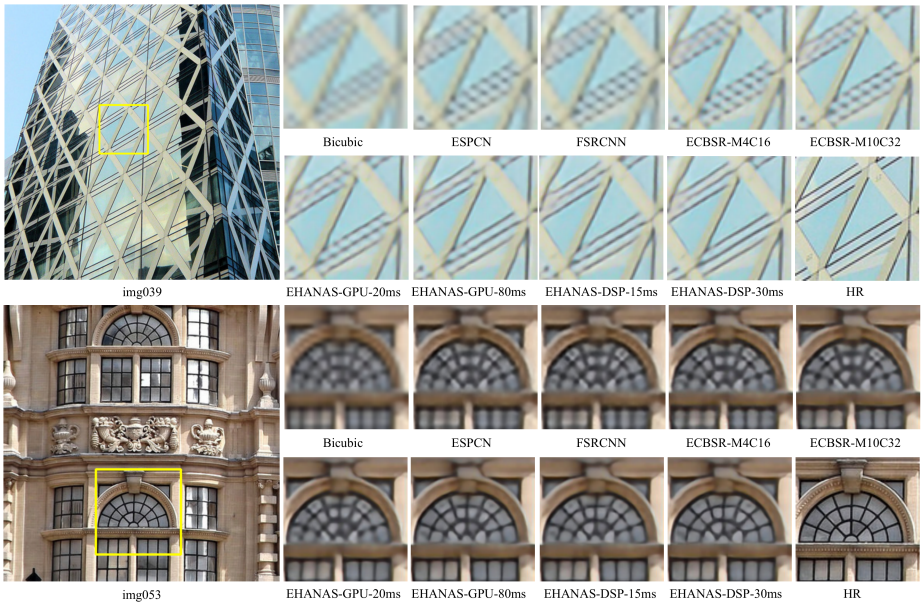

Fig. 4. Qualitative comparison of real-time and nearly real-time SR models on Urban100 for $\times$ 4 upscaling tasks. The searched EHANAS models can restore richer and sharper details than other competing models.

One can see that the searched models by EHANAS obtain clearly better performance than previous LESR models under most settings. Specifically, under the real-time setting of $\times 2$ up-scaling, our EHANAS-DSP-30ms model improves the PSNR by more than 0.3 dB on both the Set5 and DIV2K validation set compared to the second-best candidate ECBSR-M4C16. As for the nearly real-time setting of $\times 2$ up-scaling, EHANAS-DSP-80ms outperforms the second-best model ECBSR-M10C32 on all five datasets at comparable latency. One can see that our EHANAS-DSP-80ms model can obtain close PSNR/SSIM indexes to IMDN and EDSR-baseline, but it is more than 18 and 137 times faster than them on the DSP hardware. Furthermore, our larger model, EHANAS-DSP-600ms, achieves the best results among the five benchmarks, while it is around 2 and 18 times faster than EDSR-baseline and IMDN on DSP hardware, validating the scalability of our EHANAS method.

Similar observations can be made under the setting of $\times 4$ up-scaling, where the EHANAS-DSP-15ms, EHANAS-DSP-30ms and EHANAS-DSP-150ms models obtain higher PSNR and SSIM indexes on all datasets with less or comparable resources compared to their competitors. These results validate the effectiveness of using the proposed EHANAS method to search LESR models. Noted that the computational capacity of GPU of dimensity 1000+ is much lower than DSP of snapdragon 865, while our EHANAS method can also successfully search the desired models, achieving better PSNR/SSIM indexes than previous methods with lower computational cost and hardware resources.

We also compare our method with the top-ranking LESR designs in NTIRE 2022 [26], including RLFN [20], FMEN [11] and EFDN [26]. Our EHANAS-DSP-150ms

is slightly better except for the DIV2K, it may be that all of the three methods are specifically trained and tuned for this benchmark. Our EHANAS series have huge advantages over them on DSP hardware, it is because the three methods incorporate multi-branch design and DSP-unfriendly operations (e.g., PReLU, LReLU, and ESA) which may introduce a huge amount of memory access cost. Since the GPU hardware is programmable and computationally bounded, the above-mentioned drawbacks could be eased. Both RLFN and FMEN achieve promising efficiency. The three designs can be further enhanced on mobile scenarios by EHANAS, we will leave it as future research.

4.3 Qualitative comparison

In Figure 4, we qualitatively compare the SR results of several real-time and nearly real-time models under the $\times 4$ up-scaling setting by using two example images from the Urban100 dataset. Specifically, we compare our searched EHANAS-DSP-30ms, EHANAS-DSP-80ms, EHANAS-GPU-50ms, and EHANAS-GPU-200ms models with FSRCNN [8], ESPCN [36], ECBSR-M4C16 [49] and ECBSRM10C32 [49]. The bicubic upsampling and ground truth HR patches are also included as references. As can be seen from the figure, previous real-time SR models like ESPCN, FSRCNN, and ECBSR-M4C16 tend to recover blurry and smooth texture around long edges in "img039", while our EHANAS-GPU-50ms and EHANAS-DSP-30ms can reproduce more details and well preserve the edge structure to some extent. On image "img053", our EHANAS-DSP-30ms, EHANAS-DSP-80ms, EHANAS-GPU-50ms, and EHANAS-GPU-200ms models successfully generate sharper and clearer edge details than other competing methods.

In Figure 5, we provide the qualitative comparison of SR results among several representative and more complicated models, including CARN-M [1], EDSR-R16C64 [27] and IMDN [16]. On image "Barbara" from Set14, EHANAS-GPU-400ms and EHANAS-DSP-150ms well preserve the long edge structure, while other methods produce blurry or artificial details around edges. On image "img052", most of the compared methods yield either blurry or inaccurate edges and textures, while our EHANAS-DSP-150ms can restore more accurate and sharper details.

Table 2. Ablation studies on the search space. The baseline results are from ECBSR-M4C16 [49]. By default, the kernel type, channel dimension, block number and activation are set to 3×3, 32, 8, ReLU, respectively, when they are not searched.

Method	Kernel	Channel	#Block	Act.	#Lat(s)	Set5 [3]	Set14 [44]	B100 [32]	Urban100 [15]	DIV2K [39]
Baseline					0.033	37.33/0.9580	32.81/0.9100	31.66/0.8931	30.31/0.9091	35.15/0.9380
EHANAS	✓	✓			0.034	37.46/0.9586	32.98/0.9121	31.76/0.8944	30.56/0.9117	35.31/0.9393
EHANAS		✓	✓		0.035	37.51/0.9588	33.03/0.9130	31.80/0.8950	30.69/0.9131	35.37/0.9398
EHANAS	✓		✓		0.035	37.53/0.9588	33.03/0.9126	31.78/0.8949	30.65/0.9126	35.33/0.9395
EHANAS	✓	✓	✓		0.031	37.60/0.9592	33.12/0.9136	31.85/0.8958	30.91/0.9154	35.46/0.9401
EHANAS	✓	✓	✓	✓	**0.029**	**37.63/0.9593**	**33.15/0.9137**	**31.89/0.8960**	**30.98/0.9168**	**35.51/0.9408**

4.4 Search cost comparison

In this section, we compare the computation and memory cost by using the state-of-the-art DNAS method [41] and our proposed EHANAS to search SR models of different

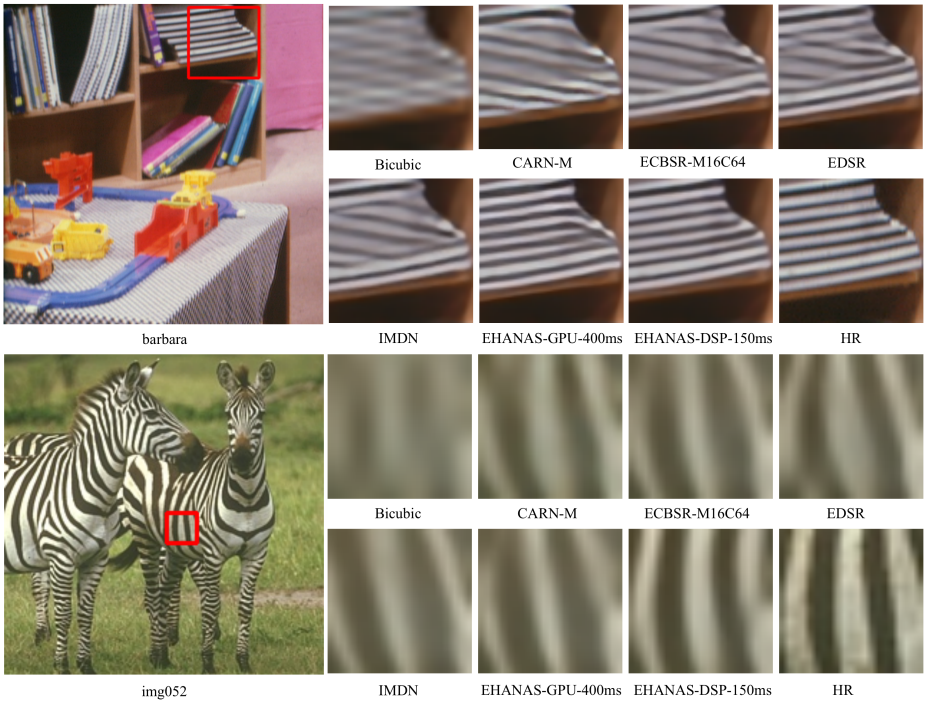

Fig. 5. Qualitative comparison of larger models for $\times$ 4 upscaling tasks on "Barbara" and "Img052" from B100 and Set14, respectively. The searched EHANAS models can restore clearer edges and structures than other competing models.

complexity. Specifically, we use the five kernel candidates as described in Sec. 3.1, set the maximum channel dimension to 64, and vary the number of blocks from 5 to 25 with step-length 5. The memory cost (measured in GB) and computing time of 100 training iterations (measured in seconds) of different settings are plotted in Figure 6.

One can see that both the memory and computation cost of DNAS is about 3 times as much as our proposed EHANAS. This is because the DNAS method uses multiple independent convolution branches, while our EHANAS employs only one shared convolution. As the cost increases linearly with the number of network blocks, EHANAS can save a lot of computation and time, especially when the SR models are to be deployed on multiple hardware devices with different latency constraints.

4.5 Ablation study

We conduct a series of ablation studies to investigate the influence of network search space on the searched model. Specifically, we conduct ablation experiments by using different combinations of the four variables, i.e., kernel type, channel dimension, activation type, and the number of blocks. Since one single variable can be directly determined for a certain latency, we search for at least two variables in each experiment. All models are searched under the $\times 2$ up-scaling task with a 30ms latency constraint. For network searching, the maximum channel dimension and number of blocks are set to 64

and 8, respectively. When not searched, the kernel type, channel dimension, block number, and activation are set to 3×3, 32, 8, and ReLU, respectively, following prior arts. The results of different model variants are reported in Table 2. The results of ECBSR-M4C16 [49], which is the state-of-the-art among manually design LESR models at the same latency level, are also reported for comparison.

As can be seen, all the searched model variants significantly outperform their baselines, which validates the advantage of the proposed EHANAS method over manual design. When performing searching with two variables, the combination of kernel type and block number (kernel+block for short) obtains the best performance, followed by channel+block. Searching with three variables, i.e., kernel+channel+block, obtains better performance than all variants of searching with two variables. Searching with all four variables further improves the performance. These results indicate that a larger search space can generally lead to a better model.

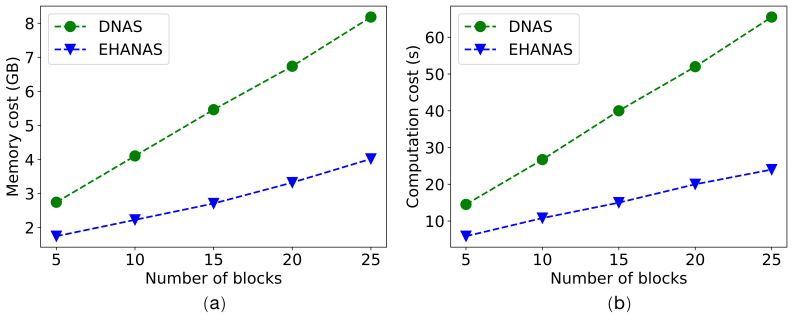

Fig. 6. Comparison of searching costs between the existing DNAS method [41] and our proposed EHANAS. (a) Memory cost (measured in GB) comparison of the entire model during the model searching process. (b) Computational cost (measured in second) comparison of training 100 iterations during the model searching process.

5 Conclusion and discussions

In this paper, we proposed an efficient hardware-aware neural architecture search (EHANAS) method to automatically search light-weight and efficient SR models on mobile devices. EHANAS could finish the searching of both macro and micro-network topologies within one GPU day. Experiments on two typical hardware devices, GPU of Dimensity 1000+ and DSP of Snapdragon 865, validated the effectiveness of EHANAS in searching SR models under desired latency constraints. The searched SR models could work in real-time or nearly real-time while exhibiting much better accuracy than previously manually designed and automatically searched models.

While EHANAS is very efficient by benefiting from the proposed masking and re-parameterization strategy, the shared weights may slightly condense the searching space. In addition, an individual model has to be searched for each device under each latency constraint. In the future, we will study how to search for shared SR models that can be easily adapted to different hardware devices.

References

1. Ahn, N., Kang, B., Sohn, K.A.: Fast, accurate, and lightweight super-resolution with cascading residual network. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 252–268 (2018) 2, 3, 9, 10, 12
2. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167 (2016) 3
3. Bevilacqua, M., Roumy, A., Guillemot, C., Alberi-Morel, M.L.: Low-complexity single-image super-resolution based on nonnegative neighbor embedding (2012) 9, 10, 12
4. Chu, X., Zhang, B., Ma, H., Xu, R., Li, Q.: Fast, accurate and lightweight super-resolution with neural architecture search. In: 2020 25th International Conference on Pattern Recognition (ICPR). pp. 59–64. IEEE (2021) 2, 4, 6, 8, 9, 10
5. Chu, X., Zhang, B., Xu, R.: Multi-objective reinforced evolution in mobile neural architecture search. In: European Conference on Computer Vision. pp. 99–113. Springer (2020) 2, 4, 6, 8, 9, 10
6. Dai, T., Cai, J., Zhang, Y., Xia, S.T., Zhang, L.: Second-order attention network for single image super-resolution. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11065–11074 (2019) 1
7. Dong, C., Loy, C.C., He, K., Tang, X.: Image super-resolution using deep convolutional networks. IEEE transactions on pattern analysis and machine intelligence **38**(2), 295–307 (2015) 1, 9, 10
8. Dong, C., Loy, C.C., Tang, X.: Accelerating the super-resolution convolutional neural network. In: European conference on computer vision. pp. 391–407. Springer (2016) 2, 3, 9, 10, 12
9. Dong, P., Wang, S., Niu, W., Zhang, C., Lin, S., Li, Z., Gong, Y., Ren, B., Lin, X., Tao, D.: Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition. In: 2020 57th ACM/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2020) 4
10. Dong, X., Yang, Y.: Searching for a robust neural architecture in four gpu hours. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1761–1770 (2019) 5
11. Du, Z., Liu, D., Liu, J., Tang, J., Wu, G., Fu, L.: Fast and memory-efficient network towards efficient image super-resolution. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 853–862 (2022) 9, 10, 11
12. Gao, Y., Gu, X., Zhang, H., Lin, H., Yang, M.: Runtime performance prediction for deep learning models with graph neural network. Tech. rep., Technical Report MSR-TR-2021-3. Microsoft (2021) 7
13. Gong, Y., Zhan, Z., Li, Z., Niu, W., Ma, X., Wang, W., Ren, B., Ding, C., Lin, X., Xu, X., et al.: A privacy-preserving-oriented dnn pruning and mobile acceleration framework. In: Proceedings of the 2020 on Great Lakes Symposium on VLSI. pp. 119–124 (2020) 4
14. Guo, Y., Luo, Y., He, Z., Huang, J., Chen, J.: Hierarchical neural architecture search for single image super-resolution. IEEE Signal Processing Letters **27**, 1255–1259 (2020) 2, 4, 6, 8
15. Huang, J.B., Singh, A., Ahuja, N.: Single image super-resolution from transformed self-exemplars. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5197–5206 (2015) 9, 10, 12
16. Hui, Z., Gao, X., Yang, Y., Wang, X.: Lightweight image super-resolution with information multi-distillation network. In: Proceedings of the 27th ACM International Conference on Multimedia. pp. 2024–2032 (2019) 2, 3, 9, 10, 12
17. Hui, Z., Wang, X., Gao, X.: Fast and accurate single image super-resolution via information distillation network. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 723–731 (2018) 2, 3

18. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144 (2016) 5
19. Kim, J., Lee, J.K., Lee, K.M.: Accurate image super-resolution using very deep convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1646–1654 (2016) 1, 9, 10
20. Kong, F., Li, M., Liu, S., Liu, D., He, J., Bai, Y., Chen, F., Fu, L.: Residual local feature network for efficient super-resolution. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 766–776 (2022) 9, 10, 11
21. Lai, W.S., Huang, J.B., Ahuja, N., Yang, M.H.: Deep laplacian pyramid networks for fast and accurate super-resolution. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 624–632 (2017) 9, 10
22. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015) 1
23. Lee, R., Dudziak, L., Abdelfattah, M., Venieris, S.I., Kim, H., Wen, H., Lane, N.: Journey towards tiny perceptual super-resolution. ECCV (2020) 2
24. Lee, R., Dudziak, Ł., Abdelfattah, M., Venieris, S.I., Kim, H., Wen, H., Lane, N.D.: Journey towards tiny perceptual super-resolution. In: European Conference on Computer Vision. pp. 85–102. Springer (2020) 4, 8, 9, 10
25. Li, Y., Gu, S., Zhang, K., Van Gool, L., Timofte, R.: Dhp: Differentiable meta pruning via hypernetworks. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16. pp. 608–624. Springer (2020) 2, 4
26. Li, Y., Zhang, K., Timofte, R., Van Gool, L., Kong, F., Li, M., Liu, S., Du, Z., Liu, D., Zhou, C., et al.: Ntire 2022 challenge on efficient super-resolution: Methods and results. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1062–1102 (2022) 3, 9, 10, 11
27. Lim, B., Son, S., Kim, H., Nah, S., Mu Lee, K.: Enhanced deep residual networks for single image super-resolution. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp. 136–144 (2017) 1, 9, 10, 12
28. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436 (2017) 3
29. Liu, J., Tang, J., Wu, G.: Residual feature distillation network for lightweight image super-resolution. In: European Conference on Computer Vision. pp. 41–55. Springer (2020) 2, 3
30. Ma, X., Guo, F.M., Niu, W., Lin, X., Tang, J., Ma, K., Ren, B., Wang, Y.: Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 5117–5124 (2020) 4
31. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. arXiv preprint arXiv:1611.00712 (2016) 5
32. Martin, D., Fowlkes, C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001. vol. 2, pp. 416–423. IEEE (2001) 9, 10, 12
33. Niu, W., Ma, X., Lin, S., Wang, S., Qian, X., Lin, X., Wang, Y., Ren, B.: Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 907–922 (2020) 4
34. Radosavovic, I., Kosaraju, R.P., Girshick, R., He, K., Dollár, P.: Designing network design spaces. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10428–10436 (2020) 2

35. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the aaai conference on artificial intelligence. vol. 33, pp. 4780–4789 (2019) 3
36. Shi, W., Caballero, J., Huszár, 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: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1874–1883 (2016) 2, 3, 9, 10, 12
37. Song, D., Xu, C., Jia, X., Chen, Y., Xu, C., Wang, Y.: Efficient residual dense block search for image super-resolution. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 12007–12014 (2020) 2, 4, 6, 8
38. Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., Marculescu, D.: Single-path nas: Device-aware efficient convnet design. arXiv preprint arXiv:1905.04159 (2019) 4
39. Timofte, R., Agustsson, E., Van Gool, L., Yang, M.H., Zhang, L.: Ntire 2017 challenge on single image super-resolution: Methods and results. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp. 114–125 (2017) 9, 10, 12
40. Vu, T., Van Nguyen, C., Pham, T.X., Luu, T.M., Yoo, C.D.: Fast and efficient image quality enhancement via desubpixel convolutional neural networks. In: Proceedings of the European Conference on Computer Vision (ECCV) Workshops. pp. 0–0 (2018) 2
41. Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K., et al.: Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12965–12974 (2020) 3, 4, 5, 7, 12, 14
42. Wang, L., Dong, X., Wang, Y., Ying, X., Lin, Z., An, W., Guo, Y.: Exploring sparsity in image super-resolution for efficient inference. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4917–4926 (2021) 2
43. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10734–10742 (2019) 3
44. Zeyde, R., Elad, M., Protter, M.: On single image scale-up using sparse-representations. In: International conference on curves and surfaces. pp. 711–730. Springer (2010) 9, 10, 12
45. Zhan, Z., Gong, Y., Zhao, P., Yuan, G., Niu, W., Wu, Y., Zhang, T., Jayaweera, M., Kaeli, D., Ren, B., et al.: Achieving on-mobile real-time super-resolution with neural architecture and pruning search. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4821–4831 (2021) 2, 4, 6, 8
46. Zhang, K., Danelljan, M., Li, Y., Timofte, R., Liu, J., Tang, J., Wu, G., Zhu, Y., He, X., Xu, W., et al.: Aim 2020 challenge on efficient super-resolution: Methods and results. In: European Conference on Computer Vision. pp. 5–40. Springer (2020) 2, 3
47. Zhang, K., Gu, S., Timofte, R., Hui, Z., Wang, X., Gao, X., Xiong, D., Liu, S., Gang, R., Nan, N., et al.: Aim 2019 challenge on constrained super-resolution: Methods and results. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW). pp. 3565–3574. IEEE (2019) 3
48. Zhang, L.L., Han, S., Wei, J., Zheng, N., Cao, T., Yang, Y., Liu, Y.: nn-meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services. pp. 81–93 (2021) 7
49. Zhang, X., Zeng, H., Zhang, L.: Edge-oriented convolution block for real-time super resolution on mobile devices. In: Proceedings of the 29th ACM International Conference on Multimedia. pp. 4034–4043 (2021) 2, 3, 5, 6, 7, 9, 10, 12, 14

50. Zhang, Y., Li, K., Li, K., Wang, L., Zhong, B., Fu, Y.: Image super-resolution using very deep residual channel attention networks. In: Proceedings of the European conference on computer vision (ECCV). pp. 286–301 (2018) 1
51. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016) 3
52. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018) 3