Compiler-Aware Neural Architecture Search for On-Mobile Real-time Super-Resolution

Yushu Wu[*1], Yifan Gong[*1], Pu Zhao[1], Yanyu Li[1], Zheng Zhan[1], Wei Niu[2], Hao Tang[3], Minghai Qin[1], Bin Ren[2], and Yanzhi Wang[1]

[1] Northeastern University, Boston MA 02115, USA
{wu.yushu,gong.yifa}@northeastern.edu
[2] College of William and Mary, Williamsburg VA 23185, USA
[3] CVL, ETH Zürich, Zürich 8092, Switzerland

Abstract. Deep learning-based super-resolution (SR) has gained tremendous popularity in recent years because of its high image quality performance and wide application scenarios. However, prior methods typically suffer from large amounts of computations and huge power consumption, causing difficulties for real-time inference, especially on resource-limited platforms such as mobile devices. To mitigate this, we propose a compiler-aware SR neural architecture search (NAS) framework that conducts depth search and per-layer width search with adaptive SR blocks. The inference speed is directly taken into the optimization along with the SR loss to derive SR models with high image quality while satisfying the real-time inference requirement. Instead of measuring the speed on mobile devices at each iteration during the search process, a speed model incorporated with compiler optimizations is leveraged to predict the inference latency of the SR block with various width configurations for faster convergence. With the proposed framework, we achieve real-time SR inference for implementing 720p resolution with competitive SR performance (in terms of PSNR and SSIM) on GPU/DSP of mobile platforms (Samsung Galaxy S21). Codes are available at link.

Keywords: Super Resolution; Real-Time; On-Mobile; NAS

1 Introduction

As a classic vision task, single-image-super-resolution (SISR) restores the original high-resolution (HR) image based on a down-sampled low-resolution (LR) one. It can be applied in various applications, such as low-resolution media data enhancement or video/image upscaling for high resolution display panels. Various classic [38,24,66,67] and deep learning (DL)-based [20,21,62,81,52] SR methods have been proposed in the past. Compared with classic interpolation algorithms to improve image/video resolution, DL-based methods take advantage of learning mappings from LR to HR images from external datasets. Thus most recent

---

[*] Both authors contributed equally.

SR works emerge in the DL area. However, one major limitation of existing DL-based SR methods is their high computation and storage overhead to achieve superior image quality, leading to difficulties to implement real-time SR inference even on powerful GPUs, not to mention resource limited edge devices. Due to the ever-increasing popularity of mobile devices and interactive on-mobile applications (such as live streaming), it is essential to derive lightweight SR models with both high image quality and low on-mobile inference latency.

There exist several works targeting at efficient SR models, including using upsampling operator at the end of a network [21,62], adopting channel splitting [34], using wider activation [81], and combining lightweight residual blocks with variants of group convolution [52]. Neural architecture search (NAS) is applied to derive the optimal architecture in many vision tasks. Latest works [15,63,44,16] try to derive fast, lightweight, and accurate SR networks via NAS. However, their models are still too large to be implemented on mobile devices. Furthermore, these methods usually take the parameter numbers and computation counts (such as multiply-accumulate (MAC) operations) into the optimization for model efficiency, without considering the actual on-mobile implementation performance such as the inference latency. The actual mobile deployment of SR mobiles has rarely been investigated. The most relevant works are the winner of the PIRM challenge [68], MobiSR [45], and work [85]. But they either require nearly one second per frame for inference, far beyond real-time, or take a long search time.

Targeting at achieving real-time inference of accurate SR model for 720p resolution on various resource-limited hardware such as mobile GPU and DSP, this paper proposes a compiler-aware NAS framework. An adaptive SR block is introduced to conduct the depth search and per-layer width search. Each convolution (CONV) layer is paired with a mask layer in the adaptive SR block for the width search, while the depth search is reached by choosing a path between the skip connection and the masked SR block. The mask can be trained along with the network parameters via gradient descent optimizers, significantly saving training overhead. Instead of using MACs as the optimization target, the latency performance is directly incorporated into the objective function with the usage of a speed model. Our implementation can support real-time SR inference with competitive SR performance on various resource-limited platforms, including mobile GPU and DSP. The contributions are summarized below:

- We propose a framework to search for the appropriate depth and per-layer width with adaptive SR blocks.
- We introduce a general compiler-aware speed model to predict the inference speed on the target device with corresponding compiler optimizations.
- The proposed framework can directly optimize the inference latency, providing the foundations for achieving real-time SR inference on mobile.
- Our proposed framework can achieve real-time SR inference (with only tens of milliseconds per frame) for the implementation of 720p resolution with competitive SR performance (in terms of PSNR and SSIM) on mobile (Samsung Galaxy S21). Our achievements can facilitate various practical SR applications with real-time requirements such as live streaming or video communication.

2 Related Work

SR Works. In recent years, most SR works have shifted their approaches from classic methods to DL-based methods with significant SR performance improvements. From the pioneering SRCNN [20] to later works with shortcut operator, dense connection, and attention mechanism [41,48,88,87,17], the up-scaling characteristic have dramatically boosted at the cost of high storage and computation overhead. Most of the works mentioned above even take seconds to process only one image on a powerful GPU, let alone mobile devices or video applications.

Efficient SR. Prior SR works are hard to be implemented on resource-limited platforms due to high computation and storage cost. To obtain more compact SR models, FSRCNN [21] postpones the position of the upsampling operator. IDN [35] and IMDN [34] utilize the channel splitting strategy. CARN-M [7] explores a lightweight SR model by combining efficient residual blocks with group convolutions. SMSR [70] learns sparse masks to prune redundant computation for efficient inference. ASSLN [89] and SRPN [90] leverage structure-regularized pruning and impose regularization on the pruned structure to guarantee the alignment of the locations of pruned filters across different layers. SR-LUT [40] uses look-up tables to retrieve the precomputed HR output values for LR input pixels, with a more significant SR performance degradation. However, these SR models do not consider the actual mobile deployment, and the sizes of the models are still too large. The actual SR deployment is rarely investigated. The winner of the PIRM challenge [68], MobiSR [45], and work [85] explore the on-device SR, but the models take seconds for a single image, far from real time, or require long search time. Work [37] considers real-time SR deployed on the powerful mobile TPU, which is not widely adopted such as mobile CPU/GPU.

NAS for SR. NAS has been shown to outperform heuristic networks in various applications. Recent SR works start to leverage NAS to find efficient, lightweight, and accurate SR models. Works [15,16,85] leverage reinforced evolution algorithms to achieve SR as a multi-objective problem. Work [6] uses a hierarchical search strategy to find the connection with local and global features. LatticeNet [56] learns the combination of residual blocks with the attention mechanism. Work [74,32,19] search lightweight architectures at different levels with differentiable architecture search (DARTS) [51]. DARTS based methods introduce architecture hyper-parameters which are usually continuous rather than binary, incurring additional bias during selection and optimization. Furthermore, the above-mentioned methods typically take the number of parameters or MACs into the objective function, rather than on-mobile latency as discussed in Sec. 3. Thus they can hardly satisfy the real-time requirement.

Hardware Acceleration. A significant emphasis on optimizing the DNN execution has emerged in recent years [43,75,36,79,29,60,22,39,25]. There are several representative DNN acceleration frameworks including Tensorflow-Lite [1], Alibaba MNN [2], Pytorch-Mobile [3], and TVM [13]. These frameworks include several graph optimization techniques such as layer fusion, and constant folding.

3   Motivation and Challenges

With the rapid development of mobile devices and real-time applications such as live streaming, it is essential and desirable to implement real-time SR on resources-limited mobile devices. However, it is challenging. To maintain or up-scale the spatial dimensions of feature maps based on large input/output size, SR models typically consume tens of or hundreds of GMACs (larger than several GMACs in image classification [54,69]), incurring difficulties for real-time inference. For example, prior works on mobile SR deployment [45] and [68] achieve 2792ms and 912ms on-mobile inference latency, respectively, far from real-time.

We can adopt NAS or pruning methods to find a lightweight SR model with fast speed on mobile devices. But there are several challenges: (C1) tremendous searching overhead with NAS, (C2) misleading magnitude during pruning, (C3) speed incorporation issues, and (C4) heuristic depth determination.

Tremendous Searching Overhead with NAS. In NAS, the exponentially growing search space leads to tremendous search overhead. Specifically, the RL-based [93,91,94] or evolution-based NAS methods [61,64,78] typically need to sample large amounts of candidate models from the search space and train each candidate architecture with multiple epochs, incurring long search time and high computation cost. Besides, differentiable NAS methods [11,8,51] build super-nets to train multiple architectures simultaneously, causing significant memory cost and limited discrete search space up-bounded by the available memory. To mitigate these, there are certain compromised strategies, such as proxy tasks (to search on CIFAR and target on ImageNet) [61,92,78] and performance estimation (to predict/estimate the architecture performance with some metrics) [4,65,49].

Misleading Magnitude during Pruning. Pruning can also be adopted to reduce the model size, which determines the per-layer pruning ratio and pruning positions. With the assumption that weights with smaller magnitudes are less important for final accuracy, magnitude-based pruning [30,58,31,86,72,26,47,57,84] is widely employed to prune weights smaller than a threshold. However, the assumption is not necessarily true, and weight magnitudes can be misleading. Magnitude-based pruning is not able to achieve importance shifting during pruning. As detailed in Appendix **??**, in iterative magnitude pruning, small weights pruned first are not able to become large enough to contribute to the accuracy. Thus layers pruned more at initial will be pruned more and more, causing a non-recoverable pruning policy. It becomes pure exploitation without exploration.

Speed Incorporation Issues. To achieve real-time inference on mobile, it is essential to obtain the on-mobile speed performance when searching architectures. However, it is non-trivial to achieve this since testing speed requires an additional process to interact with the mobile device for a few minutes, which can hardly be incorporated into a typical model training. To mitigate this, certain methods [54,69,53] adopt weight number or computation counts as an estimation of the speed performance. Other methods [73,18,77] first collect on-mobile speed data and then build lookup tables with the speed data to estimate the speed.

Heuristic Depth Determination. Reducing model depth can avoid all computations in the removed layers, thus significantly accelerating the inference.

Since previous NAS works do not incorporate a practical speed constraint or measurement during optimization, their search on model depth is usually heuristic. Designers determine the model depth according to a simple rule that the model should satisfy an inference budget, without a specific optimization method [49,50,92,78,8,51]. More efforts are devoted to searching other optimization dimensions such as kernel size or width rather than model depth.

4 Our Method

We first introduce the framework, then discuss the components of the framework in detail. We also specify how it can deal with the challenges in Sec. 3.

4.1 Framework with Adaptive SR Block

In the framework, we perform a compiler-aware architecture depth and per-layer width search to achieve real-time SR on mobile devices. The search space contains the width for each CONV layer and the number of stacked SR blocks in the model, which is too large to be explored with a heuristic method. Therefore, we propose an adaptive SR block to implement the depth and per-layer width search, and the model is composed of multiple adaptive SR blocks. Fig. 1 shows the architecture of the adaptive SR block. It consists of a masked SR block, a speed model, and an aggregation layer. The adaptive SR block has two inputs (and outputs) corresponding to the SR features and the accumulated speed, respectively. It achieves per-layer width search with mask layers in the masked SR blocks and depth search with aggregation layer to choose a path between the skip connection and the masked SR block. Besides, to obtain the on-mobile speed performance, we adopt a speed model to predict the speed of the masked SR block. The speed model is trained on our own dataset with speed performance of various block width configurations measured through compiler optimizations for significant inference acceleration to achieve accurate speed prediction.

4.2 Per-Layer Width Search with Mask Layer for C1 and C2

Width search is performed for each CONV layer in a typical WDSR block [81]. WDSR is chosen as our basic building blocks since it has demonstrated high efficiency in SR tasks [83,82,14]. Note that our framework is not limited to the WDSR block and can be easily extended to various residual SR blocks [7,48,35] in the literature. To satisfy the real-time requirement, we perform a per-layer width search to automatically select an appropriate number of channels for each CONV layer in the WDSR block. Specifically, we insert a differentiable mask layer (a depth-wise 1×1 CONV layer) after each CONV layer to serve as the layer-wise trainable mask, as shown below,

$$\boldsymbol{a}_l^n = \boldsymbol{m}_l^n \odot (\boldsymbol{w}_l^n \odot \boldsymbol{a}_{l-1}^n), \tag{1}$$

where $\odot$ denotes the convolution operation. $\boldsymbol{w}_l^n \in R^{o \times i \times k \times k}$ is the weight parameters in the lth CONV layer of the nth block, with o output channels, i input
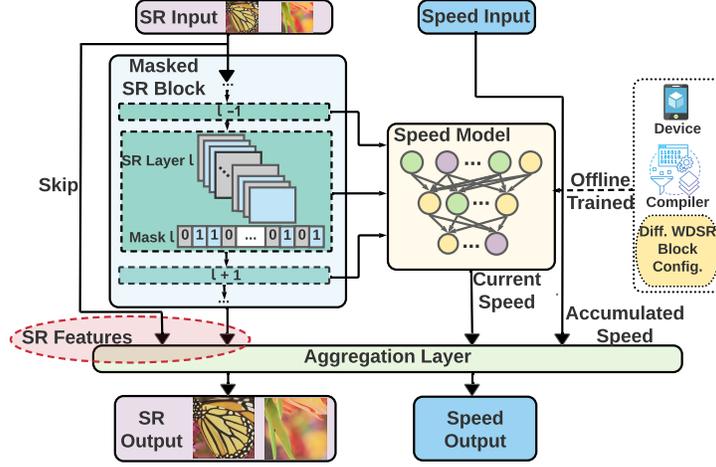
Fig. 1: Architecture of the adaptive SR block search.

channels, and kernels of size $k \times k$. $\boldsymbol{a}_l^n \in R^{B \times o \times s \times s'}$ represents the output features of lth layer (with the trainable mask), with o channels and $s \times s'$ feature size. B denotes the batch size. $\boldsymbol{m}_l^n \in R^{o \times 1 \times 1 \times 1}$ is the corresponding weights of the depth-wise CONV layer (i.e., the mask layer).

We use each element of $\boldsymbol{m}_l^n$ as the pruning indicator for the corresponding output channel of $\boldsymbol{w}_l^n \odot \boldsymbol{a}_{l-1}^n$. Larger elements of $\boldsymbol{m}_l^n$ mean that the corresponding channels should be preserved while smaller elements indicate pruning the channels. Formally, we use a threshold to convert $\boldsymbol{m}_l^n$ into a binary mask,

$$\boldsymbol{b}_l^n = \begin{cases} 1, \boldsymbol{m}_l^n > thres. \\ 0, \boldsymbol{m}_l^n \leq thres. \end{cases} \quad \text{(element-wise)}, \tag{2}$$

where $\boldsymbol{b}_l^n \in \{0,1\}^{o \times 1 \times 1 \times 1}$ is the binarized $\boldsymbol{m}_l^n$. We initialize $\boldsymbol{m}_l^n$ with random values between 0 and 1, and the adjustable $thres$ is set to 0.5 in our case. The WDSR block with the proposed mask layers is named as masked SR block.

Thus we are able to obtain a binary mask for each CONV layer. The next problem is how to make the mask trainable, as the binarization operation is non-differentiable, leading to difficulties for back-propagation. To solve this, we integrate Straight Through Estimator (STE) [9] as shown below,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_l^n} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_l^n}, \tag{3}$$

where we directly pass the gradients through the binarization. The STE method is originally proposed to avoid the non-differentiable problems in quantization tasks [55,80]. Without STE, some methods adopt complicated strategies to deal with the non-differentiable binary masks such as [28,27].

With the binarization and the STE method, we are able to build a trainable mask to indicate whether the corresponding channel is pruned or not. Our mask

generation and training are more straightforward and simpler. For example, proxyless-NAS [12] transforms the real-valued weights to binary gates with a probability distribution, and adopts complex mask updating procedure (such as task factorizing). SMSR [70] adopts Gumbel softmax to perform complex sparse mask CONV. Unlike proxylessNAS or SMSR, we generate binary masks simply via a threshold and train the masks directly via STE.

4.3 Speed Prediction with Speed Model for `C3`

To achieve real-time SR inference on mobile devices, we take the inference speed into the optimization to satisfy a given real-time threshold. It is hard to measure the practical speed or latency of various structures on mobile devices during optimization. Traditionally, the inference speed may be estimated roughly with the number of computations [54,69,53] or a latency lookup table [73,18,77], which can hardly provide an accurate speed. To solve this problem, we adopt a DNN-based speed model to predict the inference speed of the block. The input of the speed model is the width of each CONV layer in the block, and it outputs the block speed. As shown in Fig. 1, the width of each CONV layer can be obtained through the mask layer. Thus the speed model can work perfectly with the width search, dealing with `C3` to provide speed performance of various architectures.

To train such a speed model, we first need to build a speed dataset with block latency of various layer width configurations in the block. Next, we can train a speed model based on the dataset to predict the speed. We find that the trained speed model is accurate in predicting the speed of different layer widths in the block (with 5% error at most). We show the details about the dataset, speed model, and the prediction accuracy in Sec. 5 and Appendix B.

We highlight that our speed model not only takes the masks as inputs to predict the speed, but also back-propagates the gradients from the speed loss (Eq. (10)) to update the masks as detailed in Sec. 4.5, rather than just predicting performance forwardly such as [71]. That is why we build the speed model based on DNNs instead of loop-up tables. The trainable masks and the speed model are combined comprehensively to solve the problem more efficiently.

4.4 Depth Search with Aggregation Layer for `C4`

Although reducing the per-layer width can accelerate the inference, removing the whole block can avoid the computations of the whole block, thus providing higher speedup. Hence, besides width search, we further incorporate depth search to automatically determine the number of adaptive SR blocks in the model. Note that although per-layer width search may also converge to zero width, which eliminates the entire block, we find that in most cases, there are usually a few channels left in each block to promote the SR performance, leading to difficulties in removing the whole block. Thus it is necessary to incorporate depth search.

To perform depth search, we have two paths in each adaptive SR block. As shown in Fig. 1, one path is the skip connection, and the other path is the masked SR block. In the aggregation layer, there is a parameter like a switch to

control which path the SR input goes through. If the SR input chooses the skip path, the masked SR block is skipped, and the latency of this block is just 0, leading to significant inference acceleration. The aggregation layer plays a key role in the path selection. It contains two trainable parameters α_s and α_b. In the forward pass, it selects the skip path or the masked WDSR block path based on the relative relationship of α_s and α_b, as shown below,

$$\boldsymbol{\beta}_s = 0 \text{ and } \beta_b = 1, \text{if } \alpha_s \leq \alpha_b, \tag{4}$$

$$\boldsymbol{\beta}_s = 1 \text{ and } \beta_b = 0, \text{if } \alpha_s > \alpha_b, \tag{5}$$

where the binarized variables β_s and β_b denote the path selection (β_s=1 means choosing the skip path and β_b=1 means choosing the masked SR block path). Since the comparison operation is non-differentiable, leading to difficulties for back-propagation, similarly we adopt STE [9] to make it differentiable as below,

$$\frac{\partial \mathcal{L}}{\partial \alpha_s} = \frac{\partial \mathcal{L}}{\partial \beta_s}, \quad \frac{\partial \mathcal{L}}{\partial \alpha_b} = \frac{\partial \mathcal{L}}{\partial \beta_b}. \tag{6}$$

In the aggregation layer, the forward computation can be represented below,

$$\boldsymbol{a}^n = \beta_s \cdot \boldsymbol{a}^{n-1} + \beta_b \cdot \boldsymbol{a}_L^n, \tag{7}$$

$$v_n = v_{n-1} + \beta_b \cdot v_c, \tag{8}$$

where $\boldsymbol{a}^n$ is the SR output features of the nth adaptive SR block. $\boldsymbol{a}_L^n$ is the SR output features of masked SR block in the nth adaptive SR block, and L is the maximum number of CONV layers in each block and we have $l \leq L$. v_n is the accumulated speed or latency until the nth adaptive SR block and v_c is the speed of the current block which is predicted by the speed model. By training α_s and α_b, the model can learn to switch between the skip path and the SR path to determine the model depth, thus dealing with C4.

4.5   Training Loss

Multiple adaptive SR blocks can form the SR model, which provides two outputs including the typical SR outputs and the speed outputs. The training loss is a combination of a typical SR loss and a speed loss as below,

$$\mathcal{L}_{SPD} = \max\{0, v_N - v_T\}, \tag{9}$$

$$\mathcal{L} = \mathcal{L}_{SR} + \gamma \mathcal{L}_{SPD}, \tag{10}$$

where v_T is the real-time threshold, v_N is the accumulated speed of N blocks, and γ is a parameter to control their relative importance. The objective is to achieve high SR performance while the speed can satisfy a real-time threshold. To summarize, with the trainable masks, the speed model, and the aggregation layer in the adaptive SR block, our search algorithm achieves the following advantages:

– The mask can be trained along with the network parameters via gradient descent optimizers, thus dealing with C1 to save search overhead compared with previous one-shot pruning [31,23] or NAS methods [93,91] to train multiple epochs for each candidate architecture with huge searching efforts.
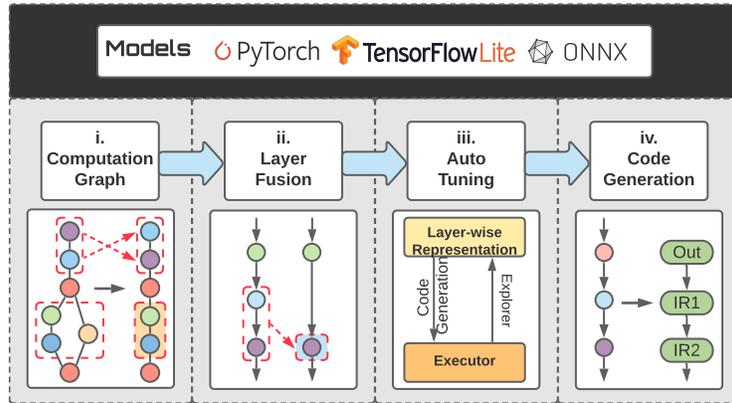
Fig. 2: The overview of compiler optimizations.

- Compared with magnitude-based threshold pruning, we decouple the trainable masks from original model parameters, thus enabling exploitation and overcoming the drawbacks of magnitude-based pruning, dealing with C2.
- We use the speed model for predicting the speed to solve C3, which is differentiable regarding the trainable mask. Thus the mask is trained to find a model with both high SR performance and fast inference speed.
- We also incorporate depth search though aggregation layers to deal with C4.

5   Compiler Awareness with Speed Model

To satisfy the speed requirement with a given latency threshold on a specific mobile device, it is required to obtain the actual inference latency on the device. It is non-trivial to achieve this as the model speed varies with different model width and depth. It is unrealistic to measure the actual on-mobile speed during the search, as the search space is quite large, and testing the mobile speed of each candidate can take a few minutes, which is not compatible with DNN training.

To solve this problem, we adopt a speed model to predict the inference latency of the masked SR block with various width configurations. With the speed model, we can obtain the speed prediction as outputs by providing the width of each CONV layer in the SR block as inputs. It is fully compatible with the trainable mask, enabling differentiable model speed with respect to the layer width.

To obtain the speed model, we first build a latency dataset with latency data measured on the hardware platforms incorporated with compiler optimizations. Then the DNN speed model is trained based on the latency dataset.

Compiler Optimization. To build a latency dataset, we need to measure the speed of various block configurations on mobile devices. Compiler optimizations are adopted to accelerate the inference speed during speed testing. It is essential to incorporate compiler optimizations as they can significantly accelerate the inference speed. The overview of the compiler optimizations is shown in Fig. 2.

To fully exploit the parallelism for a higher speedup, the key features of SR have to be considered. As the objective of SR is to obtain a HR image from its LR counterpart, each layer has to maintain or upscale the spatial dimensions of the feature, leading to larger feature map size and more channels compared with classification tasks. Therefore, the data movements between the memory and cache are extremely intensive. To reduce the data movements for faster inference, we adopt two important optimization techniques: 1) operator fusion and 2) decreasing the amount of data to be copied between CPU and GPU.

Operator fusion is a key optimization technique adopted in many state-of-the-art DNN execution framework [1,2,3]. However, these frameworks usually adopt fusion approaches based on certain patterns that are too restrictive to cover the diversity of operators and layer connections. To address this problem, we classify the existing operations in the SR model into several groups based on the mapping between the input and output, and develop rules for different combinations of the groups in a more aggressive fusion manner. For instance, CONV operation and depth-to-space operation can be fused together. With layer fusion, both the memory consumption of the intermediate results and the number of operators can be reduced. An auto-tuning process is followed to determine the best-suited configurations of parameters for different mobile CPUs/GPUs and Domain Specific Language (DSL) based code generation. After that, a high-level DSL is leveraged to specify the operator in the computational graph of a DNN model. We show more details about compiler optimization in Appendix C.

Latency Dataset. To train the speed model, we first measure and collect the inference speed of the WDSR block under various CONV layer width configurations. After that, a dataset of the WDSR block on-mobile speed with different configurations can be built. We vary the number of filters in each CONV layer as the different width configurations. The inference time is measured on the target device (Samsung Galaxy S21) by stacking 20 WDSR blocks with the same configuration, and the average latency is used as the inference time to mitigate the overhead of loading data on mobile GPU. As the maximum number of CONV layers in each masked WDSR block is L, each data point in the dataset can be represented as a tuple with $L+2$ elements: $\{\mathcal{F}_{CONV^1}, \cdots, \mathcal{F}_{CONV^{L+1}}, \mathcal{T}_{inference}\}$, where $\mathcal{F}_{CONV^i}$, for $i \in \{1, \cdots, L\}$, indicates the number of input channels for the ith CONV layer, $\mathcal{F}_{CONV^{L+1}}$ is the number of output channels for the last CONV layer, and $\mathcal{T}_{inference}$ is the inference speed for this configuration measured in milliseconds. The entire dataset is composed of 2048 data points.

Speed Model. With the latency dataset, the speed model can be trained on the collected data points. The inference speed estimation is a regression problem, thus, a network with 6 fully-connected layers combined with ReLU activation is used as the speed model. During the speed model training, 90% of the data is used for training and the rest is for validation. After training, the speed model can predict the inference time of various block configurations with high accuracy. From our results, the speed model only incurs 5% of deviation for the speed prediction. The speed model has two advantages: (1) It is compatible with the width search framework as the trainable mask can be directly fed into the speed

Scale	Method	Params (K)	MACs (G)	Latency (ms)	PSNR Set5	Set14	B100	Urban100	SSIM Set5	Set14	B100	Urban100
×2	FSRCNN [21]	12	6.0	128.47	37.00	32.63	31.53	29.88	0.9558	0.9088	0.8920	0.9020
	MOREMNAS-C [16]	25	5.5	—	37.06	32.75	31.50	29.92	0.9561	0.9094	0.8904	0.9023
	TPSR-NOGAN [44]	60	14.0	—	37.38	33.00	31.75	30.61	0.9583	0.9123	0.8942	0.9119
	LapSRN [42]	813	29.9	—	37.52	33.08	31.80	30.41	0.9590	0.9130	0.8950	0.9100
	CARN-M [7]	412	91.2	1049.92	37.53	33.26	31.92	31.23	0.9583	0.9141	0.8960	0.9193
	FALSR-C [15]	408	93.7	—	37.66	33.26	31.96	31.24	0.9586	0.9140	0.8965	0.9187
	ESRN-V [63]	324	73.4	—	37.85	33.42	32.10	31.79	0.9600	0.9161	0.8987	0.9248
	EDSR [48]	1518	458.0	2031.65	37.99	33.57	32.16	31.98	0.9604	0.9175	0.8994	0.9272
	WDSR [81]	1203	274.1	1973.31	38.10	33.72	32.25	32.37	0.9608	0.9182	0.9004	0.9302
	SMSR [70]	985	131.6	—	38.00	33.64	32.17	32.19	0.9601	0.9179	0.8990	0.9284
	SRPN-L [90]	609	139.9	—	38.10	33.70	32.25	32.26	0.9608	0.9189	0.9005	0.9294
	Ours (v_T=100ms)	47	11.0	**98.90**	37.64	33.16	31.91	31.08	0.9591	0.9136	0.8961	0.9170
	Ours (v_T=70ms)	28	6.6	**66.09**	37.49	33.05	31.81	30.76	0.9584	0.9123	0.8946	0.9135
	Ours (v_T=40ms, real-time)	11	2.5	**34.92**	37.19	32.80	31.60	30.15	0.9572	0.9099	0.8919	0.9054
×4	FSRCNN [21]	12	4.6	98.13	30.71	27.59	26.98	24.62	0.8657	0.7535	0.7150	0.7280
	TPSR-NOGAN [44]	61	3.6	55.82	31.10	27.95	27.15	24.97	0.8779	0.7663	0.7214	0.7456
	FEQE-P [68]	96	5.6	82.81	31.53	28.21	27.32	25.32	0.8824	0.7714	0.7273	0.7583
	CARN-M [7]	412	32.5	374.15	31.92	28.42	27.44	25.62	0.8903	0.7762	0.7304	0.7694
	ESRN-V [63]	324	20.7	—	31.99	28.49	27.50	25.87	0.8919	0.7779	0.7331	0.7782
	IDN [35]	600	32.3	—	31.99	28.52	27.52	25.92	0.8928	0.7794	0.7339	0.7801
	EDSR [48]	1518	114.5	495.90	32.09	28.58	27.57	26.04	0.8938	0.7813	0.7357	0.7849
	DHP-20 [46]	790	34.1	—	31.94	28.42	27.47	25.69	—	—	—	—
	IMDN [34]	715	40.9	—	32.21	28.58	27.56	26.04	0.8948	0.7811	0.7353	0.7838
	WDSR [81]	1203	69.3	533.02	32.27	28.67	27.64	26.26	0.8963	0.7838	0.7383	0.7911
	SR-LUT-S [40]	77	—	—	29.77	26.99	26.57	23.94	0.8429	0.7372	0.6990	0.6971
	SMSR [70]	1006	41.6	—	32.12	28.55	27.55	26.11	0.8932	0.7808	0.7351	0.7868
	SRPN-L [90]	623	35.8	—	32.24	28.69	27.63	26.16	0.8958	0.7836	0.7373	0.7875
	Ours (v_T=100ms)	188	10.8	**93.50**	32.02	28.50	27.51	25.83	0.8922	0.7778	0.7328	0.7769
	Ours (v_T=70ms)	116	6.7	**64.95**	31.88	28.43	27.46	25.69	0.8905	0.7760	0.7312	0.7715
	Ours (v_T=40ms, real-time)	66	3.7	**36.46**	31.73	28.28	27.34	25.44	0.8878	0.7725	0.7281	0.7620

∗ Some latency results are not reported as the models are not open-source or contain operators that cannot run on mobile GPU.
† The latency results are measured on the GPU of Samsung Galaxy S21.

Table 1: Comparison with SOTA efficient SR models for implementing 720p.

model. (2) It makes the model speed differentiable with respect to the masks, and back-propagates gradients to update the masks, thus the model can update the model speed by adjusting the layer width though back-propagation.

6 Experiments

6.1 Experimental Settings

SR Datasets. All SR models are trained on the training set of DIV2K [5] with 800 training images. For evaluation, four benchmark datasets Set5 [10], Set14 [76], B100 [59], and Urban100 [33] are used for test. The PSNR and SSIM are calculated on the luminance channel (a.k.a. Y channel) in the YCbCr color space.
Evaluation Platforms and Running Configurations. The training codes are implemented with PyTorch. 8 GPUs are used to conduct the search, which usually finishes in 10 hours. The latency is measured on the GPU of an off-the-shelf Samsung Galaxy S21 smartphone, which has the Qualcomm Snapdragon 888 mobile platform with a Qualcomm Kryo 680 Octa-core CPU and a Qualcomm Adreno 660 GPU. Each test takes 50 runs on different inputs with 8 threads on CPU, and all pipelines on GPU. The average time is reported.
Training Details. 48×48 RGB image patches are randomly sampled from LR images for each input minibatch. We use the architecture of WDSR with 16
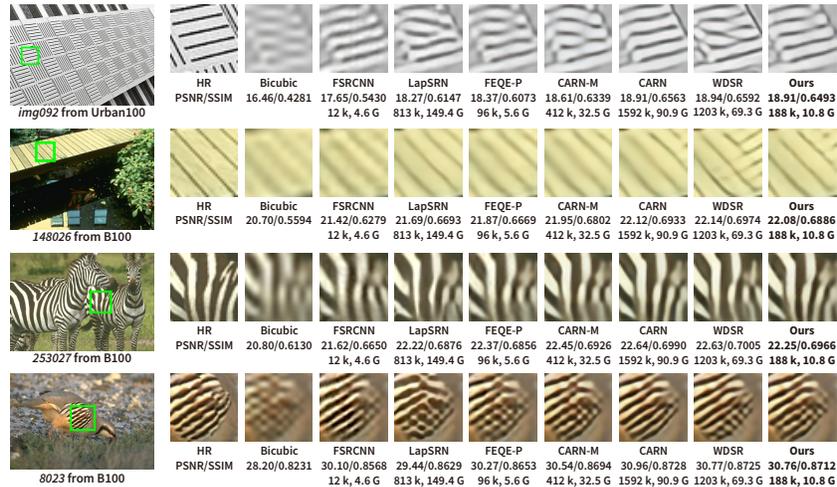
Fig. 3: Visual Comparisons with other methods on Urban100/B100 for ×4 SR.

blocks as the backbone of our NAS process. Considering the huge input size of SR (normally nHD–640×360 inputs or higher resolution for ×2 task), a compact version of the WDSR block is chosen to fit the mobile GPU, where the largest filer number for each CONV layer is 32, 146, and 28, respectively. The backbone is initialized with the parameters of the pretrained WDSR model. Traditional MAE loss is used to measure the differences between the SR image and the ground-truth as the SR loss. The parameter γ in the training loss denoted as Eq. (10) is set to 0.01. The first 20 epochs are used for the NAS process, and the following 30 epochs for fine-tuning the searched model. ADAM optimizers with β_1=0.9, β_2=0.999, and ϵ=1 × 10^{-8} are used for both model optimization and fine-tuning process. The learning rate is initialized as 1×10^{-4} and reduced by half at 10, 16 epochs and at 20, 25 epochs in the NAS and fine-tuning process, respectively. The details of the searched architecture are in Appendix D.

Baseline Methods. We compare with some traditional human-designed SR models such as FSRCNN and EDSR. Besides, some baselines optimizing the speed or hardware with NAS approaches are also covered. For example, TPSR-NOGAN, FALSR-C, ESRN-V optimize the SR efficiency to facilitate the deployment on end devices. Moreover, we compare with some methods exploring the sparsity in SR models such as DHP, SMSR, and SRPN-L for efficient inference.

6.2 Experimental Results

Comparison with Baselines on SR Performance. The comparisons of the models obtained by the proposed framework with state-of-the-art efficient SR works are shown in Table 1. Two commonly used metrics (PSNR and SSIM) are adopted to evaluate image quality. The evaluations are conducted on ×2 and ×4 scales. For a fair comparison, we start from different low-resolution inputs
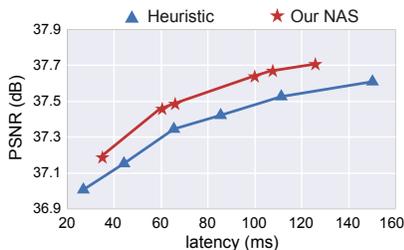
Fig. 4: Comparison of ×2 SR results between searched models and heuristic models on Set5 with latency measured on the GPU of Samsung Galaxy S21.

Search Method		Latency (ms)	Set 5		Urban100	
Width Search	Depth Search		PSNR	SSIM	PSNR	SSIM
✗	✗	150.92	37.62	0.9589	31.03	0.9164
✗	✓	111.58	37.65	0.9591	31.10	0.9172
✓	✗	108.38	37.65	0.9591	31.02	0.9161
✓	✓	**98.90**	37.64	0.9591	31.08	0.9170

Table 2: Comparison of different search schema for ×2 scales. The performance is evaluated on Set5 and Urban100 datasets

but the high-resolution outputs are 720p (1280×720). To make a comprehensive study, the latency threshold v_T is set to different values. Specifically, as real-time execution typically requires at least 25 frames/sec (FPS), the latency threshold v_T is set as 40ms to obtain SR models for real-time inference.

For ×2 scale, the model obtained with latency threshold v_T=100ms outperforms TPSR-NOGAN, LAPSRN, and CARN-M in terms of PSNR and SSIM with fewer parameters and MACs. Compared with FALSR-C, ESRN-V, EDSR, WDSR, SMSR, and SRON-L, our model greatly reduces the model size and computations with a competitive image quality performance. By setting v_T as 70ms, our model has similar parameters and MACs as MOREMNAS-C, but achieves higher PSNR and SSIM performance. Similar results can be obtained on the ×4 scale. Furthermore, for both scales, by setting v_T as 40ms, we obtain extremely lightweight models and the models still maintain satisfying PSNR and SSIM performance on all four datasets. Although SR-LUT uses look-up tables for efficient SR inference, it suffers from more significant SR performance degradation.

The visual comparisons with other SR methods for ×4 up-scaling task are shown in Fig. 3. Our model can recover the details comparable or even better than other methods by using fewer parameters and computations.

Comparison with Baselines on Speed Performance. In general, our method can achieve real-time SR inference (higher than 25 FPS) for implementing 720p resolution up-scaling with competitive image quality in terms of PSNR and SSIM on mobile platforms (Samsung Galaxy S21). Compared with [70] which also explore the sparsity of SR models, our method can achieve more significant model size and computation reduction (our 11GMACs v.s. 131.6GFLOPs [70] for ×2 scale), leading to faster speed (our 11.3ms v.s. 52ms [70] on Nvidia A100 GPU).

Comparison with Heuristic Models. We compare our searched models with heuristic models, which are obtained by evenly reducing the depth and width from the WDSR model. Since we do not search per-layer width in heuristic models, the width is the same among all blocks in one heuristic model. For a fair comparison, the same compiler optimization framework is adopted for both searched models and heuristic models. As shown in Fig. 4, we can see that the

NAS approach can achieve faster inference than the heuristic models under the same PSNR, demonstrating the effectiveness of the search approach.

Compiler Optimization Performance. To demonstrate the effectiveness of our compiler optimizations, we implement CARN-M [7], FSRCNN [21], and our searched model with the open-source MNN framework. By comparing their PSNR and FPS performance, we find that our model can achieve higher FPS and PSNR than the baseline models, with detailed results in Appendix E. We also compare with the compilation of [36] detailed in Appendix F.

Performance on Various Devices. Our main results are trained and tested on the mobile GPU. We highlight that our method can be easily applied to all kinds of devices with their corresponding speed models. To demonstrate this, we perform compiler optimizations for the DSP on the mobile device and train the corresponding speed model. With the new speed model, we use our method to search an SR model for the DSP, which can achieve 37.34 PSNR on Set5 with 32.51 ms inference speed for $\times 2$ up-scaling task, detailed in Appendix G.

6.3   Ablation Study

For the ablation study, we investigate the influence of depth search and per-layer width search separately for $\times 2$ scale task. Multiple runs are taken for each search method with different latency threshold v_T so that the searched models have similar PSNR and SSIM on Set5 to provide a clear comparison. From the results in Table 2, we can see that both depth search only and width search only can greatly reduce the latency with better image quality than non-search case. Specifically, as a missing piece in many prior SR NAS works, depth search provides better PSNR and SSIM performance than width search on Urban100 with a slightly higher latency, which shows the importance of this search dimension. By combining depth search and width search, we could reach faster inference with similar PSNR and SSIM than conducting either search alone.

7   Conclusion

We propose a compiler-aware NAS framework to achieve real-time SR on mobile devices. An adaptive WDSR block is introduced to conduct depth search and per-layer width search. The latency is directly taken into the optimization objective with the leverage of a speed model incorporated with compiler optimizations. With the framework, we achieve real-time SR inference for the implementation of 720p with competitive SR performance on mobile.

References

1. https://www.tensorflow.org/mobile/tflite/ 3, 10
2. https://github.com/alibaba/MNN 3, 10
3. https://pytorch.org/mobile/home 3, 10
4. Abdelfattah, M.S., Mehrotra, A., Dudziak, Ł., Lane, N.D.: Zero-cost proxies for lightweight {nas}. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=0cmMMy8J5q 4
5. Agustsson, E., Timofte, R.: Ntire 2017 challenge on single image super-resolution: Dataset and study. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (July 2017) 11
6. Ahn, J.Y., Cho, N.I.: Neural architecture search for image super-resolution using densely constructed search space: Deconas. In: 2020 25th International Conference on Pattern Recognition (ICPR). pp. 4829–4836. IEEE (2021) 3
7. 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) 3, 5, 11, 14
8. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: International Conference on Machine Learning. pp. 550–559 (2018) 4, 5
9. Bengio, Y., Léonard, N., Courville, A.C.: Estimating or propagating gradients through stochastic neurons for conditional computation. CoRR **abs/1308.3432** (2013), http://arxiv.org/abs/1308.3432 6, 8
10. Bevilacqua, M., Roumy, A., Guillemot, C., Alberi-Morel, M.L.: Low-complexity single-image super-resolution based on nonnegative neighbor embedding (2012) 11
11. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Smash: one-shot model architecture search through hypernetworks. arXiv preprint arXiv:1708.05344 (2017) 4
12. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. ICLR (2019) 7
13. Chen, T., Moreau, T., et al.: Tvm: An automated end-to-end optimizing compiler for deep learning. In: USENIX. pp. 578–594 (2018) 3
14. Cheng, G., Matsune, A., Li, Q., Zhu, L., Zang, H., Zhan, S.: Encoder-decoder residual network for real super-resolution. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (June 2019) 5
15. Chu, X., Zhang, B., Ma, H., Xu, R., Li, Q.: Fast, accurate and lightweight super-resolution with neural architecture search. arXiv preprint arXiv:1901.07261 (2019) 2, 3, 11
16. Chu, X., Zhang, B., Xu, R.: Multi-objective reinforced evolution in mobile neural architecture search. In: European Conference on Computer Vision (ECCV) Workshops. pp. 99–113. Springer (2020) 2, 3, 11
17. 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) 3
18. Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., et al.: Chamnet: Towards efficient network design through platform-aware model adaptation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11398–11407 (2019) 4, 7
19. Ding, M., Lian, X., Yang, L., Wang, P., Jin, X., Lu, Z., Luo, P.: Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers. In:

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2021) 3

20. Dong, C., Loy, C.C., He, K., Tang, X.: Learning a deep convolutional network for image super-resolution. In: European conference on computer vision. pp. 184–199 (2014) 1, 3

21. 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) 1, 2, 3, 11, 14

22. Dong, P., Wang, S., et al.: Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition. arXiv:2002.11474 (2020) 3

23. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. ICLR (2018) 8

24. Freeman, W.T., Jones, T.R., Pasztor, E.C.: Example-based super-resolution. IEEE Computer graphics and Applications **22**(2), 56–65 (2002) 1

25. Gong, Y., Yuan, G., Zhan, Z., Niu, W., Li, Z., Zhao, P., Cai, Y., Liu, S., Ren, B., Lin, X., et al.: Automatic mapping of the best-suited dnn pruning schemes for real-time mobile acceleration. ACM Transactions on Design Automation of Electronic Systems (TODAES) **27**(5), 1–26 (2022) 3

26. 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

27. Guan, Y., Liu, N., Zhao, P., Che, Z., Bian, K., Wang, Y., Tang, J.: Dais: Automatic channel pruning via differentiable annealing indicator search (2020) 6

28. Guo, S., Wang, Y., Li, Q., Yan, J.: Dmcp: Differentiable markov channel pruning for neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1539–1547 (2020) 6

29. Han, S., Shen, H., Philipose, M., Agarwal, S., Wolman, A., Krishnamurthy, A.: Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys). pp. 123–136. ACM (2016) 3

30. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. International Conference on Learning Representations (ICLR) (2016) 4

31. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV). pp. 1389–1397 (2017) 4, 8

32. Huang, H., Shen, L., He, C., Dong, W., Huang, H., Shi, G.: Lightweight image super-resolution with hierarchical and differentiable neural architecture search. arXiv preprint arXiv:2105.03939 (2021) 3

33. 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) 11

34. 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, 11

35. 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) 3, 5, 11

36. Huynh, L.N., Lee, Y., Balan, R.K.: Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In: Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys). pp. 82–95. ACM (2017) 3, 14

37. Ignatov, A., Timofte, R., Denna, M., Younes, A.: Real-time quantized image super-resolution on mobile npus, mobile ai 2021 challenge: Report. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2525–2534 (2021) 3

38. Irani, M., Peleg, S.: Improving resolution by image registration. CVGIP: Graphical models and image processing **53**(3), 231–239 (1991) 1

39. Jian, T., Gong, Y., Zhan, Z., Shi, R., Soltani, N., Wang, Z., Dy, J.G., Chowdhury, K.R., Wang, Y., Ioannidis, S.: Radio frequency fingerprinting on the edge. IEEE Transactions on Mobile Computing (2021) 3

40. Jo, Y., Kim, S.J.: Practical single-image super-resolution using look-up table. In: CVPR (2021) 3, 11

41. Kim, J., Kwon Lee, J., Mu Lee, K.: 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) 3

42. 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) 11

43. Lane, N.D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L., Qendro, L., Kawsar, F.: Deepx: A software accelerator for low-power deep learning inference on mobile devices. In: Proceedings of the 15th International Conference on Information Processing in Sensor Networks. p. 23. IEEE Press (2016) 3

44. 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 (ECCV). pp. 85–102. Springer (2020) 2, 11

45. Lee, R., Venieris, S.I., Dudziak, L., Bhattacharya, S., Lane, N.D.: Mobisr: Efficient on-device super-resolution through heterogeneous mobile processors. In: The 25th Annual International Conference on Mobile Computing and Networking. pp. 1–16 (2019) 2, 3, 4

46. 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) 11

47. Li, Z., Gong, Y., Ma, X., Liu, S., Sun, M., Zhan, Z., Kong, Z., Yuan, G., Wang, Y.: Ss-auto: A single-shot, automatic structured weight pruning framework of dnns with ultra-high efficiency. arXiv preprint arXiv:2001.08839 (2020) 4

48. 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) 3, 5, 11

49. Lin, M., Wang, P., Sun, Z., Chen, H., Sun, X., Qian, Q., Li, H., Jin, R.: Zen-nas: A zero-shot nas for high-performance deep image recognition. In: 2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021 (2021) 4, 5

50. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 19–34 (2018) 5

51. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018) 3, 4, 5

52. Liu, H., Lu, Z., Shi, W., Tu, J.: A fast and accurate super-resolution network using progressive residual learning. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1818–1822. IEEE (2020) 1, 2
53. Liu, S., Zheng, C., Lu, K., Gao, S., Wang, N., Wang, B., Zhang, D., Zhang, X., Xu, T.: Evsrnet: Efficient video super-resolution with neural architecture search. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 2480–2485 (2021). https://doi.org/10.1109/CVPRW53098.2021.00281 4, 7
54. Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.T., Sun, J.: Metapruning: Meta learning for automatic neural network channel pruning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 3296–3305 (2019) 4, 7
55. Liu, Z.G., Mattina, M.: Learning low-precision neural networks without straight-through estimator (ste). arXiv preprint arXiv:1903.01061 (2019) 6
56. Luo, X., Xie, Y., Zhang, Y., Qu, Y., Li, C., Fu, Y.: Latticenet: Towards lightweight image super-resolution with lattice block. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16. pp. 272–289. Springer (2020) 3
57. Ma, X., Li, Z., Gong, Y., Zhang, T., Niu, W., Zhan, Z., Zhao, P., Tang, J., Lin, X., Ren, B., et al.: Blk-rew: A unified block-based dnn pruning framework using reweighted regularization method. arXiv preprint arXiv:2001.08357 (2020) 4
58. Mao, H., Han, S., et al.: Exploring the regularity of sparse structure in convolutional neural networks. arXiv:1705.08922 (2017) 4
59. 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) 11
60. 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. arXiv preprint arXiv:2001.00138 (2020) 3
61. 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) 4
62. Shi, W., Caballero, J., Huszar, F., et al.: 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) 1, 2
63. 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, 11
64. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946 (2019) 4
65. Tanaka, H., Kunin, D., Yamins, D.L., Ganguli, S.: Pruning neural networks without any data by iteratively conserving synaptic flow. arXiv preprint arXiv:2006.05467 (2020) 4
66. Timofte, R., De Smet, V., Van Gool, L.: Anchored neighborhood regression for fast example-based super-resolution. In: Proceedings of the IEEE international conference on computer vision. pp. 1920–1927 (2013) 1
67. Timofte, R., De Smet, V., Van Gool, L.: A+: Adjusted anchored neighborhood regression for fast super-resolution. In: Asian conference on computer vision. pp. 111–126. Springer (2014) 1

68. 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, 3, 4, 11

69. 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) 4, 7

70. Wang, L., et al.: Exploring sparsity in image super-resolution for efficient inference. In: CVPR (2021) 3, 7, 11, 13

71. Wen, W., Liu, H., Chen, Y., Li, H., Bender, G., Kindermans, P.J.: Neural predictor for neural architecture search. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.M. (eds.) Computer Vision – ECCV 2020. pp. 660–676. Springer International Publishing, Cham (2020) 7

72. Wen, W., Wu, C., et al.: Learning structured sparsity in deep neural networks. In: NeurIPS. pp. 2074–2082 (2016) 4

73. 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 Conference on Computer Vision and Pattern Recognition (CVPR). pp. 10734–10742 (2019) 4, 7

74. Wu, Y., Huang, Z., Kumar, S., Sukthanker, R.S., Timofte, R., Van Gool, L.: Trilevel neural architecture search for efficient single image super-resolution. arXiv preprint arXiv:2101.06658 (2021) 3

75. Xu, M., Zhu, M., Liu, Y., Lin, F.X., Liu, X.: Deepcache: Principled cache for mobile deep vision. In: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking. pp. 129–144. ACM (2018) 3

76. Yang, J., Wright, J., Huang, T.S., Ma, Y.: Image super-resolution via sparse representation. IEEE transactions on image processing **19**(11), 2861–2873 (2010) 11

77. Yang, T.J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H.: Netadapt: Platform-aware neural network adaptation for mobile applications. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 285–300 (2018) 4, 7

78. Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., Xu, C.: Cars: Continuous evolution for efficient neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1829–1838 (2020) 4, 5

79. Yao, S., Hu, S., Zhao, Y., Zhang, A., Abdelzaher, T.: Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In: Proceedings of the 26th International Conference on World Wide Web. pp. 351–360 (2017) 3

80. Yin, P., Lyu, J., Zhang, S., Osher, S., Qi, Y., Xin, J.: Understanding straight-through estimator in training activation quantized neural nets. arXiv preprint arXiv:1903.05662 (2019) 6

81. Yu, J., Fan, Y., Yang, J., Xu, N., Wang, Z., Wang, X., Huang, T.: Wide activation for efficient and accurate image super-resolution. arXiv preprint arXiv:1808.08718 (2018) 1, 2, 5, 11

82. Yu, J., Huang, T.: Autoslim: Towards one-shot architecture search for channel numbers. arXiv preprint arXiv:1903.11728 (2019) 5

83. Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., Huang, T.S.: Free-form image inpainting with gated convolution. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019) 5

84. Yuan, G., Ma, X., Niu, W., Li, Z., Kong, Z., Liu, N., Gong, Y., Zhan, Z., He, C., Jin, Q., et al.: Mest: Accurate and fast memory-economic sparse training framework on the edge. Advances in Neural Information Processing Systems **34**, 20838–20850 (2021) 4

85. Zhan, Z., Gong, Y., Zhao, P., Yuan, G., Niu, W., Wu, Y., Zhang, T., Jayaweera, M., Kaeli, D.R., Ren, B., Lin, X., Wang, Y.: Achieving on-mobile real-time super-resolution with neural architecture and pruning search. In: ICCV (2021) 2, 3

86. Zhang, T., Ma, X., Zhan, Z., Zhou, S., Ding, C., Fardad, M., Wang, Y.: A unified dnn weight pruning framework using reweighted optimization methods. In: 2021 58th ACM/IEEE Design Automation Conference (DAC). pp. 493–498. IEEE (2021) 4

87. 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) 3

88. Zhang, Y., Tian, Y., Kong, Y., Zhong, B., Fu, Y.: Residual dense network for image super-resolution. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2472–2481 (2018) 3

89. Zhang, Y., Wang, H., Qin, C., Fu, Y.: Aligned structured sparsity learning for efficient image super-resolution. Advances in Neural Information Processing Systems **34** (2021) 3

90. Zhang, Y., Wang, H., Qin, C., Fu, Y.: Learning efficient image super-resolution networks via structure-regularized pruning. In: International Conference on Learning Representations (2021) 3, 11

91. Zhong, Z., Yan, J., Wu, W., Shao, J., Liu, C.L.: Practical block-wise neural network architecture generation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2423–2432 (2018) 4, 8

92. Zhou, D., Zhou, X., Zhang, W., Loy, C.C., Yi, S., Zhang, X., Ouyang, W.: Econas: Finding proxies for economical neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11396–11404 (2020) 4, 5

93. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: ICLR (2017) 4, 8

94. 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 (CVPR). pp. 8697–8710 (2018) 4