Achieving on-Mobile Real-Time Super-Resolution with Neural Architecture and Pruning Search

Zheng Zhan*[1], Yifan Gong*[1], Pu Zhao*[1], Geng Yuan[1], Wei Niu[2], Yushu Wu[1], Tianyun Zhang[3], Malith Jayaweera[1], David Kaeli[1], Bin Ren[2], Xue Lin[1], Yanzhi Wang[1]
[1]Northeastern University, [2]College of William & Mary, [3]Cleveland State University
{zhan.zhe, gong.yifa, zhao.pu, xue.lin, yanz.wang}@northeastern.edu,
t.zhang85@csuohio.edu, kaeli@ece.neu.edu, bren@cs.wm.edu

Abstract

Though recent years have witnessed remarkable progress in single image super-resolution (SISR) tasks with the prosperous development of deep neural networks (DNNs), the deep learning methods are confronted with the computation and memory consumption issues in practice, especially for resource-limited platforms such as mobile devices. To overcome the challenge and facilitate the real-time deployment of SISR tasks on mobile, we combine neural architecture search with pruning search and propose an automatic search framework that derives sparse super-resolution (SR) models with high image quality while satisfying the real-time inference requirement. To decrease the search cost, we leverage the weight sharing strategy by introducing a supernet and decouple the search problem into three stages, including supernet construction, compiler-aware architecture and pruning search, and compiler-aware pruning ratio search. With the proposed framework, we are the first to achieve real-time SR inference (with only tens of milliseconds per frame) for implementing 720p resolution with competitive image quality (in terms of PSNR and SSIM) on mobile platforms (Samsung Galaxy S20).

1. Introduction

In recent year, people have ever-increasing demands for image processing to achieve higher resolutions, leading to the rapid development of SR. In general, the SR principle is to convert low-resolution images to high-resolution images with clearer details and more information. It has been adopted in various applications such as crime scene analysis to identify unnoticeable evidence or medical image processing for more accurate diagnosis.

*Equal contribution.

With the fast growth of live streaming and video recording, video contents enjoy high popularity. However, videos often have lower resolution due to the limited communication bandwidth or higher resolution of the display. Besides, live streaming usually has a *real-time*[1] requirement that the latency of each frame should not exceed a threshold. Thus, it is desirable to achieve real-time SR for video locally.

Compared with the classic interpolation algorithms to improve image or video resolution, deep learning-based SR can deliver higher visual qualities by learning the mappings from the low-resolution to high-resolution images from external datasets. Despite its superior visual performance, deep learning-based SR is usually more expensive with large amounts of computations and huge power consumption (typically hundreds of watts on powerful GPUs) [19, 17, 53], leading to difficulties for the real-time implementations. Moreover, in practice, as SR is often deployed on edge devices such as mobile phones for live streaming or video capturing due to the wide spread of mobile phones, the limited memory and computing resources on edge devices make it even harder for achieving real-time SR.

Weight pruning [60, 22, 26] is often adopted to remove the redundancy in DNNs to reduce the resource requirement and accelerate the inference. There are various pruning schemes including unstructured pruning [23, 22, 20, 44], coarse-grained structured pruning [50, 71, 70, 47, 42], and fine-grained structured pruning [45, 18, 21]. Unstructured pruning removes arbitrary weights, leading to irregular pruned weight matrices and limited hardware parallelism. Structured pruning maintains a full matrix format of the remaining weights such that the pruned model is compatible with GPU acceleration for inference. Recently, fine-grained structured pruning including pattern-based pruning and block-based pruning are proposed to provide a finer pruning granularity for higher accuracy while exhibiting

[1]We believe targeting sub 100ms can be reasonably called real-time [49] and we require the real-time implementation to be faster than 50ms.

certain regularities which can be optimized with compilers to improve hardware parallelism. To achieve inference acceleration of SR models, we focus on conventional structured pruning and fine-grained structured pruning.

Prior works usually use fixed pruning scheme for the whole model. As different pruning schemes can achieve different SR and acceleration performance, a new optimization dimension is introduced to find the most-suitable pruning configuration for each layer instead of for the whole model. Besides, as the performance of pruning depends on the original unpruned model, it is also essential to search an unpruned starting model with high SR performance.

In this paper, to facilitate the real-time SR deployment on edge devices, we propose a framework incorporating architecture and pruning search to find the most suitable cell-wise SR block configurations and layer-wise pruning configurations. Our implementation can achieve real-time SR inference with competitive SR performance on mobile devices. We summarize our contribution as follows.

- We propose an architecture and pruning search framework to automatically find the best configuration of the SR block in each cell and pruning scheme for each layer, achieving real-time SR implementation on mobile devices with high image quality.
- We train a supernet to provide a well-trained unpruned model for all possible combinations of the SR block in each supernet cell before the architecture and pruning search. Thus there is no need to train a separate unpruned model for each combination with multiple epochs, saving tremendous training efforts.
- Different from previous works with fixed pruning scheme for all layers or fixed SR blocks for all cells, we automatically search the best-suited SR block for each cell and pruning scheme for each layer. To reduce the complexity, we decouple the pruning ratio search and employ Bayesian optimization (BO) to accelerate the SR block and pruning scheme search.
- With the proposed method, we are the first to achieve real-time SR inference (with only tens of milliseconds per frame) for implementing 720p resolution with competitive image quality (in terms of PSNR and SSIM) on mobile platforms (Samsung Galaxy S20). Our achievements facilitate various practical applications with real-time requirements such as live streaming or video communication.

2. Background and Related Works

2.1. Preliminaries on Deep Learning-based SR

SISR aims to generate a high resolution image from the low-resolution version. The usage of DNNs for SR task was first proposed in SRCNN [16] and later works try to improve the upscaling characteristic and image quality with larger networks [33, 41, 68, 67, 14]. However, SR models are resource-intensive due to maintaining or upscaling the spatial dimensions of the feature map for each layer. Therefore, the number of multiply-accumulate (MAC) operations is typically counted in gigabits, leading to high inference latency (seconds per image) on a powerful GPU.

Several attempts were made to design lightweight SR models for practical applications, including using upsampling operator at the end of a network [17, 53], adopting channel splitting [31], and using wider activation [64]. Specifically, work [64] proposed WDSR-A and WDSR-B blocks, which are two of the state-of-the-art SR building blocks with high image quality. Besides, inspired by the success of neural architecture search (NAS), latest SR works try to establish more efficient and lightweight SR models by leveraging NAS approaches [12, 54, 37, 13]. But the proposed models are still too large with tremendous resource demands. Furthermore, they do not consider practical mobile deployments with limited hardware resource. For mobile deployment, the winner of the PIRM challenge [57] and MobiSR [38] are the few works that make progress for SR inference on mobiles. But the latency is still far from real time, requiring nearly one second per frame.

2.2. DNN Model Pruning

Weight pruning reduces the redundancy in DNNs for less storage and computations. Existing pruning schemes can be divided into *unstructured pruning*, *coarse-grained structured pruning*, and *fine-grained structured pruning*.

Unstructured pruning allows weights at arbitrary locations to be removed [22, 20, 15], as shown in Figure 1 (a). Despite the high accuracy, its irregular weight matrices are not compatible with GPU acceleration. Coarse-grained structured pruning [60, 27, 26, 65, 28] keeps structured regularity of remaining weights such as channel pruning prunes entire channels as in Figure 1 (b). The key advantage is that a full matrix format is maintained, thus facilitating hardware acceleration. However, coarse-grained structured pruning often leads to non-negligible accuracy degradation [59].

Fine-grained structured pruning includes block-based pruning [18] and pattern-based pruning [51, 45, 21]. They incorporate the benefits from fine-grained pruning while maintaining structures that can be exploited for hardware accelerations with the help of compiler. Block-based pruning divides the weight matrix of a DNN layer into multiple equal-sized blocks and applies structured pruning independently to each block, as shown in Figure 1 (c). Pattern-based pruning is a combination of kernel pattern pruning and connectivity pruning, as illustrated in Figure 1 (d). Kernel pattern pruning removes weights by forcing the remaining weights in a kernel to form a specific kernel pattern.

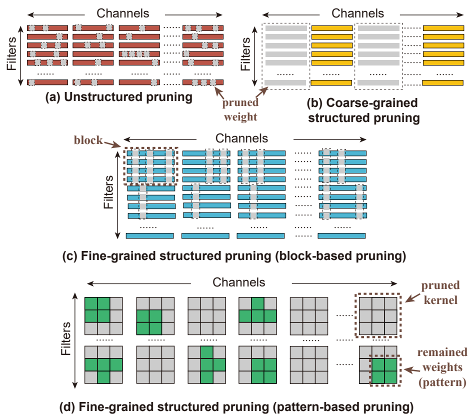

Figure 1. (a) Unstructured pruning; (b) coarse-grained structured pruning (channel); (c) fine-grained structured pruning (block-based); and (d) fine-grained structured sparsity (pattern-based).

Connectivity pruning removes entire redundant kernels and is the supplementary to kernel pattern pruning for a higher compression rate. With an appropriate pruning regularity degree, compiler-level code generation can be exploited to achieve a high hardware parallelism.

2.3. DNN Acceleration Frameworks on Mobile

On-mobile DNN inference has attracted many interests from both industry and academia [35, 36, 61, 32, 63, 25]. Representative DNN acceleration frameworks, including Tensorflow-Lite [1], Alibaba MNN [2], Pytorch-Mobile [3], and TVM [10], are designed to support inference acceleration on mobile. Several graph optimization techniques are used in these frameworks, including layer fusion, constant folding, and runtime optimizations on both mobile CPU and GPU. But the missing piece is that sparse (pruned) models for further speedup are not supported. Recently, some efforts are made to accelerate pattern-based pruned models on mobile with compiler-based optimizations [51, 45]. But they suffer difficulties when generalized to DNN layers other than 3×3 convolutional (CONV) layers.

2.4. Motivation

State-of-the-art SR methods leverage huge DNNs to pursue high image quality, causing extremely high computation cost. Thus, it is difficult to achieve real-time SR even on powerful GPUs, not to mention mobile devices with limited resource. But due to the widespread of mobile phones and the popular video communication and live streaming applications with high resolution requirements, it is desirable to implement on-mobile real-time SR with high image quality.

SR models usually constitute several cascaded SR blocks. Different blocks have different latency performance, while different combinations can form various SR models with different image quality. Meanwhile, with

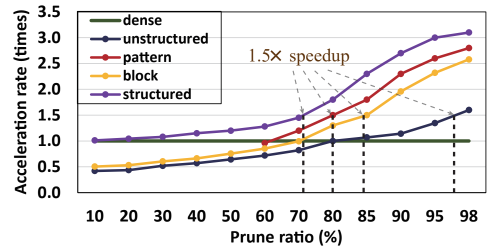

Figure 2. Inference acceleration rate vs. pruning ratio of different pruning schemes. Results are measured on a Samsung Galaxy S20 smartphone, and a typical 3×3 CONV layer in WDSR block with 24/48 input/output channels and 320×180 feature size is used.

weight pruning for acceleration, each layer may prefer a different pruning scheme, resulting in different accuracy and acceleration performance. For instance, Figure 2 illustrates the acceleration curves of different pruning schemes on a given 3×3 CONV layer. Hence, it is desirable to find the best-suited combination of SR blocks and per-layer pruning scheme and ratio to achieve high image quality while satisfying the real-time execution requirement.

Finding the satisfied network architecture and pruning configurations is too complex to be solved manually. Thus an automatic architecture and pruning search method [58] is desired. However, it is expensive to directly search in a large space, including block number (depth), block type, per-layer pruning scheme, and per-layer pruning ratio. Hence, we decouple the search into several stages and solve them separately.

3. Framework Overview

The objective is to combine architecture search with pruning search to find sparse SR models facilitating various practical applications such as live streaming or video communication. The sparse SR models should satisfy the real-time inference requirement (with only tens of milliseconds per frame) for high upscaling resolution such as 720p (1280×720) on mobile devices, with competitive image quality with the state-of-the-art methods.

The searching problem involves the determination of the number of stacked cells, the type of selected block in each cell, and pruning scheme and pruning ratio for each layer of the SR network. Direct search in such a high-dimensional search space is computationally expensive. To reduce the search cost in terms of time and computation, we leverage the weight sharing strategy by introducing a supernet and decouple the search problem into three stages: 1) supernet construction, 2) compiler-aware architecture and pruning search, and 3) compiler-aware pruning ratio determination. Supernet construction includes supernet initialization that determines the number of stacked cells, and supernet training that provides a good starting point for the following two steps. Then, a combination of block determination and pruning scheme selection for each layer is performed. The goal is to find a desirable structure that maximizes the image quality while satisfying the target latency t with the aid of compiler optimizations. Specifically, when $t \leq 50$ms, the target latency meets the real-time requirement. The following step is the automatic pruning ratio determination with the reweighted dynamic regularization method. We show the overall framework in Figure 3.

4. Supernet Construction

In architecture and pruning search, the accuracy of a model (architecture) after pruning largely depends on the accuracy of unpruned starting model. To obtain the well-trained starting models for various architectures with satisfying SR performance, the straightforward method is to perform training for each new architecture, which usually costs huge training efforts. Instead of training separate models respectively, we train a supernet such that, for any new model, we can activate the corresponding path in the supernet to derive the well-trained unpruned model immediately without further efforts to train each new model from scratch. Thus, the supernet can significantly reduce the training time for the unpruned models, thereby accelerating the search.

The architecture of the supernet is illustrated in the Figure 3 (a). We encode the architecture search space $\mathcal{A}$ with a supernet, denoted as $\mathcal{S}(\mathcal{A}, W)$, where W represents the weight collection. The supernet is constituted of N stacked cells and each cell contains K SR block choices. In our work, we adopt WDSR-A and WDSR-B, which are two highly efficient SR blocks with high image quality, as block choices. Note that our framework is not restricted by the WDSR blocks and can be generalized to different kinds of SR residual blocks. The output of each SR block k in cell n connects with all of the SR blocks in the next cell $n+1$. We define the choice of one SR block (WDSR-A or WDSR-B) for each supernet cell as one path segment, and all of the possible combinations of the N path segments form the architecture search space $\mathcal{A}$ with a size of K^N. Then one path is the collection of N path segments for all cells, denoting one SR candidate model. During supernet computation, only one path is activated while other unselected SR blocks do not participate into the computation.

To construct a supernet, there are two necessary steps: 1) determine the number of stacked cells of the supernet and initialize the supernet, and 2) fully train the supernet to provide a good starting point with high image quality and low overhead for the following SR candidate nets search.

4.1. Determine Cell Number with Latency Models

The number of stacked cells N of the supernet should be determined beforehand to guarantee the SR candidate models have the potential to satisfy the target latency t on mobile devices. Several widely used techniques in SR such as pixel shuffling (a.k.a., sub-pixel convolution) and global residual path are often hard to optimize and accelerate, resulting in a fixed latency overhead. Moreover, the skip (identity) connection structure in a block of a cell leads to a certain execution overhead that is difficult to be reduced and is accumulated with the number of stacked blocks.

To determine the number of stacked cells, we build a latency model enabling fast and precise estimation of the overall model inference latency on the target device (e.g., Samsung S20 smartphone). The latency model contains the look-up-tables of inference latency for different types of layers used in SR models (e.g., 1×1 CONV, 3×3 CONV, 5×5 CONV, skip connection, and sub-pixel convolution). For each layer type, several different settings are considered, including the number of filters and input and output feature map size. Our latency model is compiler-aware, built by measuring real-world inference latency on the target device with compiler optimizations incorporated. More details about our compiler optimization techniques are shown in Appendix A. The latency model building time can be ignored since no training process is involved, and the building process can be conducted in parallel with the supernet training. We only build once for a specific device. Moreover, we also include the sparse inference latency for different types of layers under different pruning schemes and pruning ratios in our latency model, which will be used in the pruning search stage (more discussion in Section 5.1).

Therefore, the overall inference latency on the target device can be estimated by accumulating the per-layer latency inquired from our latency model. With a target latency t for the SR candidate models, the suitable number of stacked cells can be determined. Furthermore, decoupling the supernet depth determination from the search space of the candidate SR models can greatly reduce the search complexity.

4.2. Supernet Training

After the supernet is initialized, the next step is to train its weights $\mathcal{W}$ to minimize the loss function $\mathcal{L}(\mathcal{A}, \mathcal{W})$. The well-trained supernet provides a good starting point for the following network architecture and pruning search as candidate net architecture a directly inherits weights from the path $\mathcal{W}(a)$ in the supernet. Note that the weights $\mathcal{W}$ of the supernet should be optimized in a way that all the candidate architectures $a \in \mathcal{A}$ with weights $\mathcal{W}(a)$ are optimized simultaneously. However, jointly optimizing the architecture parameters a and model parameters $\mathcal{W}(a)$ often introduces extra complexities. Furthermore, it may lead to the situation that some nodes in the graph are well trained while others are poorly trained, incurring unfair comparison for paths of different levels of maturity in the supernet.

To mitigate this problem, we adopt a single-path sampling & training strategy to accelerate the convergence of supernet training. Specifically, for each training batch, we

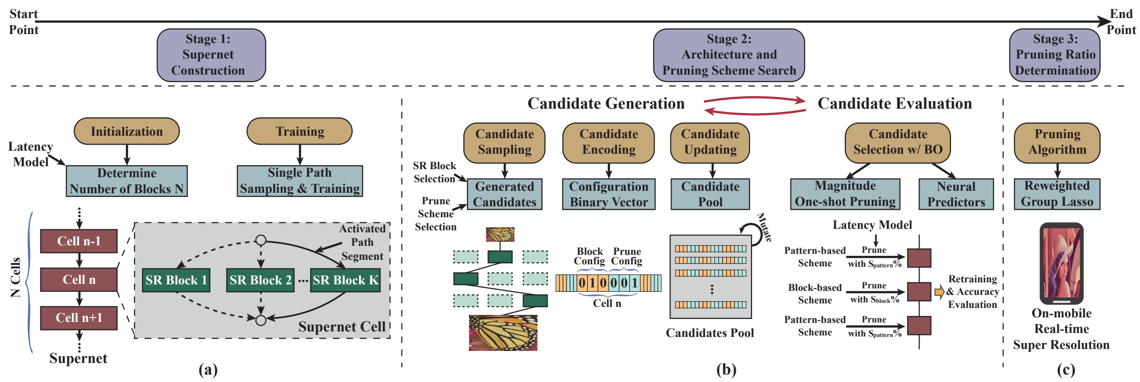

Figure 3. Framework overview. The framework is composed of three stages to reduce the search cost: (a) stage 1: supernet construction, (b) stage 2: architecture and pruning search, and (c) stage 3: pruning ratio determination.

only activate and train one random path while other unselected SR blocks are skipped. In this way, the architecture selection and model weights updating are decoupled. This strategy is hyper-parameter free, and each path is a SR model providing a well-trained unpruned starting point for the following architecture and pruning search.

5. Architecture and Pruning Search

We define each *architecture and pruning candidate* as a configuration to select one SR block for each supernet cell together with choosing the pruning scheme for each layer. The architecture and pruning search aims to find the best cell-wise SR block selection and layer-wise pruning scheme configuration, i.e., the candidate with the highest image quality satisfying the target latency t. The search consists of two main steps: 1) candidate generation and 2) candidate evaluation. In each iteration, candidate generation samples architecture and pruning candidates, which are further evaluated in the candidate evaluation process. To improve search efficiency, we adopt evolutionary-based candidate updating in candidate generation and BO in candidate evaluation to obtain the best candidate.

5.1. Candidate Generation

5.1.1 Candidate Sampling

The candidate generation samples *architecture and pruning candidates* from the search space. Each candidate g is a directed acyclic graph denoting the cell-wise SR block selection and layer-wise pruning scheme selection. For SR block selection in each supernet cell, we can choose from WDSR-A block or WDSR-B block. For the pruning scheme, we can choose channel pruning [60], pattern-based pruning [45], or block-based pruning [18] for each layer. Different from previous works with fixed pruning scheme for all layers, we can choose different pruning schemes for different layers, which is also supported by our compiler code generation. Note that the difference between the candidate g and the candidate network architecture a is that g includes the per-layer pruning scheme selection.

We encode each candidate with a binary vector by assigning a binary feature for each possible cell-wise block choice and layer-wise pruning scheme selection, denoting whether the block or pruning scheme is adopted or not.

Decoupling pruning ratio search. To prune the model, we also need to configure the layer-wise pruning ratio corresponding to each pruning scheme. As it is expensive to search the continuous pruning ratio values for each layer, at this step, we simply set the layer-wise pruning ratio to a minimal value satisfying the target latency t. Therefore, we can focus on pruning scheme search first. To determine the minimal pruning ratio, we can estimate the latency of the unpruned model t' and the target latency t, and obtain the minimal speedup required for the whole model, which is t'/t. To satisfy the overall speedup, we simply require each layer to achieve this minimal speedup t'/t[2]. Then, according to the latency model (detailed in Section 4.1) and the layer-wise speedup, we can obtain the layer-wise minimal pruning ratio corresponding to each pruning scheme.

5.1.2 Candidate Updating

In each iteration, we need to generate a pool of new candidates. To make the candidates updating more efficient, the evolutionary-based candidate updating method is adopted. We keep a record of all evaluated candidates with their evaluation performance. To generate new candidates, we mutate the candidates with the best evaluation performance in the records by randomly changing one SR block of one random cell or one pruning scheme of one random layer. Specifically, we first select H candidates with highest evaluation performance, and mutate each of them iteratively until C new proposals are derived.

[2] We find that each layer in the SR model has similar computation amount. Thus it is reasonable to adopt the same speedup for each layer.

Algorithm 1 Evaluation with BO

Input: Observation data $\mathcal{D}$, BO batch size B, BO acquisition function $\phi(\cdot)$
Output: The best candidate $g*$
for steps **do**
  Generate a pool of candidates $\mathcal{G}_c$;
  Train an ensemble of neural predictors with $\mathcal{D}$;
  Select $\{\hat{g}^i\}_{i=1}^B = \arg\max_{g \in \mathcal{G}_c} \phi(g)$;
  Evaluate the candidate and obtain reward $\{r^i\}_{i=1}^B$ of $\{\hat{g}^i\}_{i=1}^B$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup (\{\hat{g}^i\}_{i=1}^B, \{r^i\}_{i=1}^B)$;
end for

5.2. Candidate Evaluation

As it incurs a high time cost to prune and retrain the model following each candidate, BO [11] is adopted to expedite the candidate evaluation. With the generated C candidates, we first use BO to select B ($B < C$) candidates with potentially better performance. Then the selected candidates are evaluated to obtain the accurate SR performance while the unselected candidates are not evaluated. The number of actually evaluated candidates is reduced in this way.

BO includes two main components, i.e., training an ensemble of neural predictors and selecting candidates based on acquisition function values enabled by the predictor ensemble. To make use of BO, the ensemble of neural predictors provides an average SR prediction with its corresponding uncertainty estimate for each unseen candidate. Then BO is able to choose the candidate which maximizes the acquisition function. We show the full algorithm in Algorithm 1 and specify BO in the following.

5.2.1 Bayesian Optimization with Neural Predictors

Neural predictor. The neural predictor is a neural network repeatedly trained on the current set of evaluated candidates with their evaluation performance to predict the reward of unseen candidates. It is a neural network with 8 sequential fully-connected layers of width 40 trained by the Adam optimizer with a learning rate of 0.01. For the loss function to train neural predictors, mean absolute percentage error (MAPE) is adopted as it can give a higher weight to candidates with higher evaluation performance:

$$\mathcal{L}(m_{\text{pred}}, m_{\text{true}}) = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{m_{\text{pred}}^{(i)} - m_{\text{UB}}}{m_{\text{true}}^{(i)} - m_{\text{UB}}} - 1\right|, \quad (1)$$

where $m_{\text{pred}}^{(i)}$ and $m_{\text{true}}^{(i)}$ are the predicted and true values, respectively, of the reward for the i-th candidate in a batch, and m_{UB} is a global upper bound on the maximum true reward. Note that the training of the predictors does not cost too much efforts due to their simple architectures.

Ensemble of neural predictors. To incorporate BO, it also needs an uncertainty estimate for the prediction. So we adopt an ensemble of neural predictors to provide the uncertainty estimate. More specifically, we train P neural predictors using different random weight initializations and training data orders. Then for any candidate g, we can obtain the mean and standard deviation of these P predictions. Formally, we train an ensemble of P predictors, $\{f_p\}_{p=1}^P$, where $f_p(g)$ provides a predicted reward for a candidate g. The mean prediction and its deviation are given by

$$\hat{f}(g) = \frac{1}{P}\sum_{p=1}^{P} f_p(g), \text{ and } \hat{\sigma}(g) = \sqrt{\frac{\sum_{p=1}^{P}(f_p(g) - \hat{f}(g))^2}{P-1}}. \quad (2)$$

Selection with acquisition function. After training an ensemble of neural predictors, we can obtain the acquisition function value for candidates in the pool and select a small portion of candidates with the largest acquisition function values. We choose the upper confidence bound (UCB) [55] as the acquisition function shown below:

$$\phi_{\text{UCB}}(g) = \hat{f}(g) + \beta\hat{\sigma}(g), \quad (3)$$

where the tradeoff parameter β is set to 0.5.

5.2.2 Evaluation with magnitude-based pruning

After selecting the candidates from the pool, we need to measure the performance of the selected candidate g to update the neural predictors. For faster evaluation, magnitude-based pruning framework [23] (with two steps including pruning and retraining) is adopted to perform the actual pruning for candidate g to obtain its evaluation performance. Note that multiple candidates can be evaluated in parallel. Once the evaluation finishes, their actual performances are recorded as a reference such that the candidate generation can sample better candidates.

6. Pruning Ratio Determination

After finding the best SR block configuration for each cell and the pruning scheme for each layer, we adopt a pruning ratio determination process to derive the suitable layer-wise pruning ratio. Unlike prior works (i.e., group Lasso regularization [60, 27, 43] or Alternating Direction Methods of Multipliers (ADMM) [66, 52, 39]) that suffers from significant accuracy loss or complicated pruning ratio tuning, we adopt the reweighted group Lasso [9, 46] method to determine the layer-wise prune ratio automatically.

The basic idea is to assign a penalty to each weight or pruning pattern, and dynamically reweight the penalties. More specifically, during the training (pruning) process, the reweighted method reduces the penalties on weights with larger magnitudes, thus enlarging the more critical weights, and increases the penalties on weights with smaller magnitudes, thus decreasing negligible weights. After convergence, the desired pruning ratio for each layer is determined automatically. The reweighted method can be adopted for different pruning schemes and layer types. We show the detailed reweighted pruning algorithm in Appendix B.

7. Experiments

7.1. Methodology

Datasets: All SR models were trained on the training set of the DIV2K [4] dataset with 800 training images. For the evaluation, four benchmark datasets Set5 [6], Set14 [62], B100 [48] and Urban100 [29] are employed as test sets, and the PSNR and SSIM indices are calculated on the luminance channel (a.k.a. Y channel) of YCbCr color space.

Evaluation Platforms and Running Configurations: The training codes are implemented with the PyTorch API. 8 Nvidia TITAN RTX GPUs are used to conduct the architecture and pruning search. We train an ensemble of 20 predictors and 8 models are evaluated in parallel in each step. Since we start from a well-trained supernet, we retrain 2 epochs for each one-shot pruned candidate model for fast evaluation. The search process takes 6 GPU days. The latency is measured on the GPU of an off-the-shelf Samsung Galaxy S20 smartphone, which has the Qualcomm Snapdragon 865 mobile platform with a Qualcomm Kryo 585 Octa-core CPU and a Qualcomm Adreno 650 GPU. Each test takes 50 runs on different inputs with 8 threads on CPU, and all pipelines on GPU. As different runs do not vary greatly, only the average time is reported for readability.

7.2. Comparison with State-of-the-Art

The comparison of our SR model obtained through the proposed framework with state-of-the-art methods are shown in Table 1. Some extremely large models [68, 67, 14] could take several seconds for them to upscale only one image on a large GPU. Therefore, those results are not included in Table 1. PSNR and SSIM are adopted as metrics to evaluate the image quality by convention. The evaluations are conducted on tasks with different scales including $\times2$, $\times3$, and $\times4$. For a fair comparison, we start from different low resolution inputs but the outputs have the same high resolution (720p–1280×720).

To make a comprehensive study, we set the target latency t to different values for each scale. Particularly, as real-time execution typically requires at least 20 frames per second (FPS), we adopt $t = 50$ms for $\times2$ and $\times3$ upscaling task and $t = 40$ms for $\times4$ upscaling task to obtain models that satisfy real-time inference requirement. As shown in Table 1, with a target latency $t = 450$ms, our model outperforms CARN-M and FALSR-C with higher PSNR/SSIM using much fewer MACs for a $\times2$ upscaling. With $t = 150$ms, our model has better PSNR/SSIM than FSRCNN, MOREMNAS-C, and TPSR-NOGAN with similar or even fewer MACs. Furthermore, both of our models for the two different target latency cases achieve higher PSNR/SSIM with fewer MACs compared with SRCNN and LapSRN. Compared with ESRN-V, EDSR, and WDSR, our model greatly saves the MACs while still maintaining high PSNR/SSIM. Specially, we even obtain a extremely lightweight model that meets the real-time requirement by setting $t = 50$ms and the model still maintains satisfying PSNR/SSIM. As for the $\times4$ scaling task, our model obtained with a target latency $t = 120$ms prevails SRCNN, FSRCNN and FEQE-P over MACs, PSNR, and SSIM on the four datasets. With a target latency $t = 170$ms, our model outperforms DI-based and CARN-M in PSNR/SSIM with similar or even much fewer MACs. Moreover, with $t = 40$ms, our model attains real-time inference while keeping competitive PSNR/SSIM.

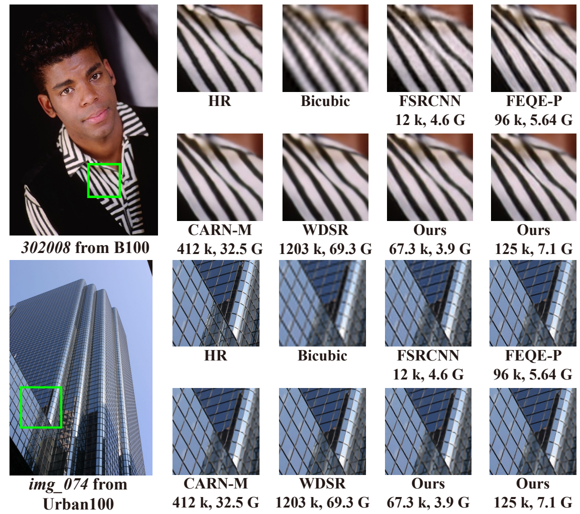

Figure 4. Visual comparison with other SR models on $\times4$ scale. Model parameters and MACs are listed under model name. More results can be found in Appendix D.

7.3. Searched Results for Real-Time SR on Mobile

We further examine the real-time performance of our SR model assisted with the compiler-based optimizations. As shown in Figure 5, with the same SR model derived with the proposed method, our method with compiler optimizations achieves the highest FPS for various scales compared with implementations by other acceleration frameworks including MNN [2] and PyTorch Mobile [3]. The models are obtained by setting $t = 50$ms for $\times2$ and $\times3$, and $t = 40$ms for $\times4$. We can observe from Figure 5 that our proposed method can satisfy the real-time requirement with a FPS higher than 20 for $\times2$ and $\times3$, and higher than 25 for $\times4$.

MobiSR and FEQE-P also conduct SR inference on mobile devices. They achieve 2792ms and 912ms inference

Scale	Model	Params (K)	Multi-Adds (G)	Set5 (PSNR/SSIM)	Set14 (PSNR/SSIM)	B100 (PSNR/SSIM)	Urban100 (PSNR/SSIM)
× 2	SRCNN [16]	57	52.7	36.66/0.9542	32.42/0.9063	31.36/0.8879	29.50/0.8946
	FSRCNN [17]	12	6.0	37.00/0.9558	32.63/0.9088	31.53/0.8920	29.88/0.9020
	MOREMNAS-C [13]	25	5.5	37.06/0.9561	32.75/0.9094	31.50/0.8904	29.92/0.9023
	TPSR-NOGAN [37]	60	14.0	37.38/0.9583	33.00/0.9123	31.75/0.8942	30.61/0.9119
	LAPSRN [34]	813	29.9	37.52/0.9590	33.08/0.9130	31.80/0.8950	30.41/0.9100
	CARN-M [31]	412	91.2	37.53/0.9583	33.26/0.9141	31.92/0.8960	31.23/0.9193
	FALSR-C [12]	408	93.7	37.66/0.9586	33.26/0.9140	31.96/0.8965	31.24/0.9187
	ESRN-V [54]	324	73.4	37.85/0.9600	33.42/0.9161	32.10/0.8987	31.79/0.9248
	EDSR [41]	1518	458.0	37.99/0.9604	33.57/0.9175	32.16/0.8994	31.98/0.9272
	WDSR [64]	1203	274.1	38.10/0.9608	33.72/0.9182	32.25/0.9004	32.37/0.9302
	Ours ($t = 450$ms)	106	24.3	37.81/0.9599	33.37/0.9153	32.07/0.8980	31.58/0.9225
	Ours ($t = 150$ms)	52	11.7	37.52/0.9582	33.24/0.9140	31.88/0.8953	31.18/0.9180
	Ours ($t = 50$ms,**real-time**)	14	3.1	37.32/0.9549	33.17/0.9071	31.67/0.8885	30.35/0.8986
× 4	SRCNN [16]	57	52.7	30.48/0.8628	27.49/0.7503	26.90/0.7101	24.52/0.7221
	FSRCNN [17]	12	4.6	30.71/0.8657	27.59/0.7535	26.98/0.7150	24.62/0.7280
	TPSR-NOGAN [37]	61	3.6	31.10/0.8779	27.95/0.7663	27.15/0.7214	24.97/0.7456
	FEQE-P [57]	96	5.6	31.53/0.8824	28.21/0.7714	27.32/0.7273	25.32/0.7583
	DI-BASED [28]	92	7.0	31.84/0.889	28.38/0.775	27.40/0.730	25.51/0.765
	CARN-M [31]	412	32.5	31.92/0.8903	28.42/0.7762	27.44/0.7304	25.62/0.7694
	ESRN-V [54]	324	20.7	31.99/0.8919	28.49/0.7779	27.50/0.7331	25.87/0.7782
	EDSR [41]	1518	114.5	32.09/0.8938	28.58/0.7813	27.57/0.7357	26.04/0.7849
	DHP-20 [40]	790	34.1	31.94/ —	28.42/ —	27.47/ —	25.69/ —
	IMDN [30]	715	—	32.21/0.8948	28.58/0.7811	27.56/0.7353	26.04/0.7838
	WDSR [64]	1203	69.3	32.27/0.8963	28.67/0.7838	27.64/0.7383	26.26/0.7911
	Ours ($t = 170$ms)	125	7.1	31.93/0.8906	28.42/0.7763	27.44/0.7307	25.66/0.7715
	Ours ($t = 120$ms)	67	3.9	31.77/0.8886	28.34/0.7730	27.33/0.7280	25.41/0.7615
	Ours ($t = 40$ms, **real-time**)	12	0.7	30.74/0.8671	27.68/0.7562	26.98/0.7156	24.65/0.7299

† Results on ×3 scaling task are shown in Appendix C.

Table 1. Comparison of searched results with state-of-the-art efficient SR models.

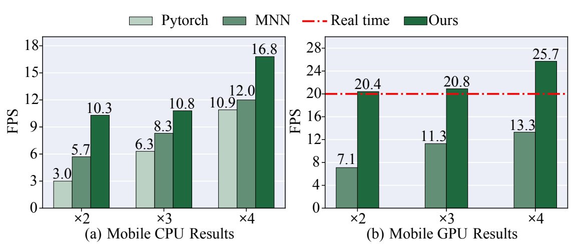

Figure 5. On-mobile inference comparisons with state-of-the-art mobile acceleration frameworks.

latency on a mobile GPU, respectively, which are far from the real-time requirement. We highlight that we are the first to achieve real-time SR inference (higher than 20 FPS for ×2 and ×3, and 25 for ×4) for implementing 720p resolution upsaling with competitive image quality (in terms of PSNR) on mobile platforms (Samsung Galaxy S20).

7.4. Ablation study

We investigate the influence of architecture search and pruning search separately. For ×2 upscaling, architecture search only achieves a 37.84 PSNR on Set5, slightly higher than ours. But as the computations are not reduced by pruning, it suffers from low inference speed (1.82 FPS). Starting from WDSR blocks, pruning search only with $t = 150$ms achieves 6.8 FPS with a lower PSNR (37.40 on Set5). Thus, we can see that pruning search significantly improves the speed performance while architecture search helps mitigate the SR performance loss due to pruning.

To promote the reproducibility and evaluate speedup using the same framework, we also implement our derived models and other baseline models including CARN-M [31] and FSRCNN [17] with the open-source MNN framework. We compare their PSNR and FPS performance and observe that we can achieve higher FPS and PSNR than the baselines. More details are attached in Appendix E.

8. Conclusion

We combine architecture search with pruning search and propose an automatic search framework that derives sparse SR models satisfying real-time execution requirement on mobile devices with competitive image quality.

Acknowledgment

The work is partly supported by Army Research Office/Army Research Laboratory via grant W911NF-20-1-0167 (YIP) to Northeastern University, the NSF CCF-2047516 (CAREER), and Jeffress Memorial Trust Awards in Interdisciplinary Research to William & Mary.

References

[1] https://www.tensorflow.org/mobile/tflite/. 3, 12

[2] https://github.com/alibaba/MNN. 3, 7, 12

[3] https://pytorch.org/mobile/home. 3, 7, 12

[4] Eirikur Agustsson and Radu Timofte. 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. 7

[5] Arash Ashari, Shirish Tatikonda, Matthias Boehm, Berthold Reinwald, Keith Campbell, John Keenleyside, and P Sadayappan. On optimizing machine learning workloads via kernel fusion. *ACM SIGPLAN Notices*, 50(8):173–182, 2015. 12

[6] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. 2012. 7

[7] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. 12

[8] Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Alexandre V Evfimievski, and Prithviraj Sen. On optimizing operator fusion plans for large-scale machine learning in systemml. *arXiv preprint arXiv:1801.00829*, 2018. 12

[9] Emmanuel J Candes, Michael B Wakin, and Stephen P Boyd. Enhancing sparsity by reweighted 1 minimization. *Journal of Fourier analysis and applications*, 14(5-6):877–905, 2008. 6, 13

[10] Tianqi Chen, Thierry Moreau, et al. Tvm: An automated end-to-end optimizing compiler for deep learning. In *USENIX*, pages 578–594, 2018. 3, 12

[11] Yutian Chen, Aja Huang, et al. Bayesian optimization in alphago. *arXiv:1812.06855*, 2018. 6

[12] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search. *arXiv preprint arXiv:1901.07261*, 2019. 2, 8

[13] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Multi-objective reinforced evolution in mobile neural architecture search. In *European Conference on Computer Vision (ECCV) Workshops*, pages 99–113. Springer, 2020. 2, 8

[14] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. Second-order attention network for single image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11065–11074, 2019. 2, 7

[15] Xiaoliang Dai, Hongxu Yin, and Niraj Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 2019. 2

[16] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *European conference on computer vision*, pages 184–199, 2014. 2, 8, 15

[17] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*, pages 391–407. Springer, 2016. 1, 2, 8, 14, 15

[18] Peiyan Dong, Siyue Wang, et al. Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition. *arXiv:2002.11474*, 2020. 1, 2, 5

[19] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4860–4874, 2017. 1

[20] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR*, 2018. 1, 2

[21] Yifan Gong, Zheng Zhan, Zhengang Li, Wei Niu, Xiaolong Ma, Wenhao Wang, Bin Ren, Caiwen Ding, Xue Lin, Xiaolin Xu, et al. A privacy-preserving-oriented dnn pruning and mobile acceleration framework. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, pages 119–124, 2020. 1, 2

[22] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *NeurIPS*, pages 1379–1387, 2016. 1, 2

[23] Song Han, Jeff Pool, et al. Learning both weights and connections for efficient neural network. In *NeurIPS*, pages 1135–1143, 2015. 1, 6

[24] Song Han, Jeff Pool, et al. Learning both weights and connections for efficient neural network. In *NeuIPS*, pages 1135–1143, 2015. 15

[25] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 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)*, pages 123–136. ACM, 2016. 3

[26] Yang He, Ping Liu, et al. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, 2019. 1, 2

[27] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, pages 1389–1397, 2017. 2, 6, 13

[28] Zejiang Hou and Sun-Yuan Kung. Efficient image super resolution via channel discriminative deep neural network pruning. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3647–3651. IEEE, 2020. 2, 8

[29] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5197–5206, 2015. 7

[30] Zheng Hui, Xinbo Gao, Yunchu Yang, and Xiumei Wang. Lightweight image super-resolution with information multi-distillation network. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2024–2032, 2019. 8

[31] Zheng Hui, Xiumei Wang, and Xinbo Gao. Fast and accurate single image super-resolution via information distillation network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 723–731, 2018. 2, 8, 14, 15

[32] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 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)*, pages 82–95. ACM, 2017. 3

[33] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1646–1654, 2016. 2

[34] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 624–632, 2017. 8

[35] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 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*, page 23. IEEE Press, 2016. 3

[36] Nicholas D Lane, Petko Georgiev, and Lorena Qendro. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 283–294. ACM, 2015. 3

[37] Royson Lee, Łukasz Dudziak, Mohamed Abdelfattah, Stylianos I Venieris, Hyeji Kim, Hongkai Wen, and Nicholas D Lane. Journey towards tiny perceptual super-resolution. In *European Conference on Computer Vision (ECCV)*, pages 85–102. Springer, 2020. 2, 8

[38] Royson Lee, Stylianos I Venieris, Lukasz Dudziak, Sourav Bhattacharya, and Nicholas D Lane. Mobisr: Efficient on-device super-resolution through heterogeneous mobile processors. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019. 2

[39] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3977–3986, 2019. 6, 13

[40] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. Dhp: Differentiable meta pruning via hypernetworks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*, pages 608–624. Springer, 2020. 8

[41] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017. 2, 8, 15

[42] Ning Liu, Xiaolong Ma, et al. Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates. In *AAAI*, 2020. 1

[43] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2736–2744, 2017. 6, 13

[44] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018. 1

[45] Xiaolong Ma et al. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In *AAAI*, 2020. 1, 2, 3, 5, 12

[46] Xiaolong Ma, Zhengang Li, Yifan Gong, Tianyun Zhang, Wei Niu, Zheng Zhan, Pu Zhao, Jian Tang, Xue Lin, Bin Ren, et al. Blk-rew: A unified block-based dnn pruning framework using reweighted regularization method. *arXiv preprint arXiv:2001.08357*, 2020. 6

[47] Xiaolong Ma, Geng Yuan, Sheng Lin, Caiwen Ding, Fuxun Yu, Tao Liu, Wujie Wen, Xiang Chen, and Yanzhi Wang. Tiny but accurate: A pruned, quantized and optimized memristor crossbar framework for ultra efficient dnn implementation. In *ASP-DAC*, 2020. 1

[48] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. 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*, volume 2, pages 416–423. IEEE, 2001. 7

[49] Robert B. Miller. Response time in man-computer conversational transactions. AFIPS 1968 (Fall, part I). 1

[50] Chuhan Min, Aosen Wang, Yiran Chen, Wenyao Xu, and Xin Chen. 2pfpce: Two-phase filter pruning based on conditional entropy. *arXiv preprint arXiv:1809.02220*, 2018. 1

[51] Wei Niu et al. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. *arXiv:2001.00138*, 2020. 2, 3, 12, 13

[52] Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 925–938. ACM, 2019. 6, 13, 15

[53] Wenzhe Shi, Jose Caballero, Huszar, 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*, pages 1874–1883, 2016. 1, 2

[54] Dehua Song, Chang Xu, Xu Jia, Yiyi Chen, Chunjing Xu, and Yunhe Wang. Efficient residual dense block search for image super-resolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12007–12014, 2020. 2, 8, 15

[55] Niranjan Srinivas, Andreas Krause, et al. Gaussian process optimization in the bandit setting: No regret and experimental design. In *ICML*, 2010. 6

[56] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision

and Pattern Recognition (CVPR), pages 2820–2828, 2019. 15

[57] Thang Vu, Cao Van Nguyen, Trung X Pham, Tung M Luu, and Chang D Yoo. Fast and efficient image quality enhancement via desubpixel convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018. 2, 8

[58] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2078–2087, 2020. 3, 15

[59] Yanzhi Wang, Shaokai Ye, Zhezhi He, Xiaolong Ma, Linfeng Zhang, Sheng Lin, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, et al. Non-structured dnn weight pruning considered harmful. *arXiv preprint arXiv:1907.02124*, 2019. 2

[60] Wei Wen, Chunpeng Wu, et al. Learning structured sparsity in deep neural networks. In *NeurIPS*, pages 2074–2082, 2016. 1, 2, 5, 6, 13

[61] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 129–144. ACM, 2018. 3

[62] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE transactions on image processing*, 19(11):2861–2873, 2010. 7

[63] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*, pages 351–360, 2017. 3

[64] Jiahui Yu, Yuchen Fan, Jianchao Yang, Ning Xu, Zhaowen Wang, Xinchao Wang, and Thomas Huang. Wide activation for efficient and accurate image super-resolution. *arXiv preprint arXiv:1808.08718*, 2018. 2, 8, 15

[65] Ruichi Yu, Ang Li, et al. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*, pages 9194–9203, 2018. 2

[66] Tianyun Zhang, Shaokai Ye, et al. Systematic weight pruning of dnns using alternating direction method of multipliers. *ECCV*, 2018. 6, 13, 15

[67] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 286–301, 2018. 2, 7

[68] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2472–2481, 2018. 2, 7

[69] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018. 15

[70] Xiaotian Zhu, Wengang Zhou, and Houqiang Li. Improving deep neural network sparsity through decorrelation regularization. In *IJCAI*, 2018. 1

[71] Zhuangwei Zhuang, Mingkui Tan, et al. Discrimination-aware channel pruning for deep neural networks. In *NeurIPS*, pages 875–886, 2018. 1