Trilevel Neural Architecture Search for Efficient Single Image Super-Resolution

Yan Wu[1], Zhiwu Huang[2], Suryansh Kumar[1], Rhea Sanjay Sukthanker[1], Radu Timofte[1,3], Luc Van Gool[1,4]
[1] ETH Zürich, Switzerland [2] SMU, Singapore [3] JMU, Germany [4] KU Leuven, Belgium
wuyan@student.ethz.ch, zwhuang@smu.edu.sg rhea.sukthanker@inf.ethz.ch,
{sukumar, radu.timofte, vangool}@vision.ee.ethz.ch

Abstract

A deep neural network-based solution to the single image super-resolution (SISR) problem not only strives for better performance but also aims at a lighter and computationally efficient model. To this end, we propose a new neural architecture search (NAS) method that performs a trilevel search leading to efficient, optimized, and compressed architecture within favorable search cost. The key idea is to introduce hierarchical modeling on network-, cell-, and kernel-level structures (trilevel) for solving SISR. To make the search differentiable and efficient on the trilevel spaces, we exploit a new sparsestmax technique which is excellent at generating sparse distributions of individual neural architecture candidates so that they can be better disentangled for the final selection from the enlarged search space. We further introduce the sorting technique to the sparsestmax relaxation for better network-level compression. Evaluations on benchmark datasets show our method's clear superiority over the state-of-the-art NAS based SISR, thereby showing a good trade-off between model size, performance, and efficiency.

1. Introduction

Single image super-resolution (SISR) upscales a single low-resolution image to its high-resolution description. Efficient SISR methods typically opt for the following variations at different levels in the neural architecture design: *(i)* A **network-level** optimization of proper positions the upsampling layers and network depth. *(ii)* A **cell-level** optimization to improve the capacities of encoding or upscaling. *(iii)* A **kernel-level** optimization to make a trade-off between an operator's capacity and its width (i.e., the number of kernel channels) for a compressed model (see Fig.1 (Left)). However, manually performing these optimizations for a favorable network design requires a lot of time, effort, and domain expertise. Further, handcrafted architectures are often not optimal and may be computationally inefficient for real-world applications.

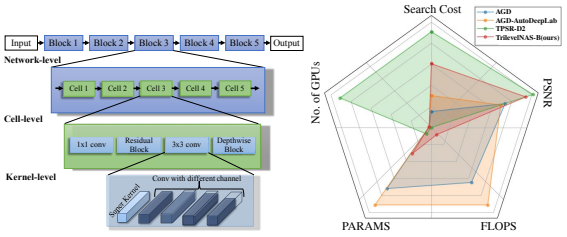

Figure 1. (Left) A typical trilevel neural architecture design for SISR. (Right) Radar plots (*smaller covering area show better method*) showing our method's comparison with other state-of-the-art NAS-based SR methods, i.e., AutoGAN-Distiller (AGD) [2], AutoDeepLab [7] and tiny perceptual SR (TPSR) [5], in terms of 5 metrics: inverse PSNR, FLOPS, parameter size, number of used GPUs, and search time. The smallest covering area shows that our TrilevelNAS for SISR clearly work better overall (i.e., keep the best balance among the 5 metrics).

To overcome the above shortcomings, this paper proposes a neural architecture search (NAS) method to automate all the three-levels in neural architectures' design process for efficient SISR. Few NAS-based methods for efficient SISR have emerged recently [2, 3, 5], but generally, their algorithms perform search only at one or two of the above three-levels. In particular, the tiny perceptual SR (TPSR) method [5] exploits a reinforcement learning (RL) based NAS method to search for optimal cell-level architecture. However, like most RL-based NAS algorithms, the success of TPSR requires enormous search cost i.e., 40 NVIDIA GeForce GTX1080 Ti server GPUs with 12 days of search time, hence practically challenging. Other work such as AutoGan-Distiller (AGD) [2] exploits a differentiable NAS method to search for optimal architectures on both cell- and kernel-levels. As this method overlooks the network-level compression, their learned neural architecture still has large model size with slow FLOPS rate.

Motivated by the above limitations with the existing NAS-based SISR, we introduce a novel differentiable trilevel NAS algorithm for the efficient search of SISR network. Our method aims to keep a good balance among search complexity (neural architecture search time and the number of employed GPUs), searched architecture performance (Peak Signal-to-Noise Ratio (PSNR)), and efficiency

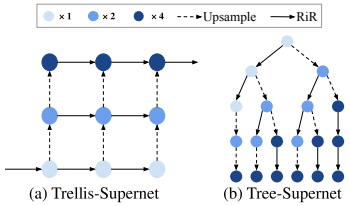

Figure 2. (a) Trellis-supernet modeling [7] on 2 Residual-in-Residual (RiR) blocks and 2 Upsampling blocks. Vertical and horizontal axes show upsampling scale and layer number, respectively. Each intermediate feature map (blue) has two paths to traverse: horizontal (RiR block) and vertical ($\times$2Upsampling layer). A feasible path traverse from the start of the node to the end node. (b) The proposed tree-supernet modeling with 2 RiR and 2 Upsampling layers. Like Trellis, each intermediate feature map has 2 paths. However, each path is independent and do not merge. Any path that starts from the root is a feasible path.

(Floating-Point Operations per Second (FLOPS) and model parameter size). We efficiently solve the trilevel NAS problem for SISR with the on-device computational resource[1], and the derived architecture gives comparable performance to the best available methods (see Fig. 1 (Right)).

To automate the trilevel neural architectures' design optimization, we introduce a discrete search space at different levels as well as a differentiable NAS algorithm as follows:

(a) Trilevel Search Space and Modeling. We define search space at three different levels (see Fig.1 (Left)). The network-level search space is composed of all the candidate network paths with various depths. The cell-level space contains all possible candidate operations. Lastly, the kernel-level space is a single convolutional kernel with a defined subset of the convolution kernel dimensions. Inspired by [2,8], we model cell- and kernel-level search spaces with supercell and superkernel, respectively. However, [2, 8] both overlook the modeling of network-level search space, which is crucial for network-level optimization and compression. One way is to adopt the trellis-like supernet modeling in Fig.2(a) from [7]. Yet, in trellis-supernet, any possible path's information flow is highly-entangled, and thus the derived path and cell architectures may not be optimal. Also, we get a fixed network depth from it. To overcome those issues, we introduce a tree-like supernet, where paths are enumerated independently (see Fig.2(b)) and hence can be pruned more easily. Intuitively, the proposed tree-like supernet can disentangle the network path dependency, providing a better trade-off between cell sharing across layers, memory consumption and search time.

(b) Efficient Differentiable Search Algorithm. For continuous relaxation of the search spaces, differentiable NAS methods often employ softmax [8]. But, softmax may not give a clear-cut dominant operation for selection to optimal architecture design. Hence, we exploit the sparsestmax strategy [11] for continuous relaxation of cell-level search space. The sparsestmax generates sparse distribution and preserves softmax's vital properties (e.g., differentiability and convexity). For network-level search space modeling, we proposed a sorted sparsestmax to produce the network-level path sparsity that is arranged in descending order. The proposed sorted sparsestmax enable us to prune the tail of the network path for notable model compression.

In summary, this paper makes three-fold contributions.

- We exploit a novel trilevel search space and modeling to allow for more comprehensive neural architecture optimization and compression.
- To make the neural architecture search on the trilevel space differentiable and efficient, we introduce a sorted sparsestmax based NAS algorithm.
- Compared to the state-of-the art methods, our method plays a good balance between SISR performance and efficiency on standard Set5 [1] and Set14 [14].

2. Proposed Trilevel Search Space

Network Level Search Space. Following AGD and SR-ResNet [4], we define our network-level search space. By fixing the network's stem (the first convolution layer) and header (the last convolution layer), we search for the path of the remaining 5 residual-in-residual (RiR) blocks and two upsampling blocks. We replace the dense blocks in RiR modules with 5 sequential layers containing searchable cell-level operators and kernel-level widths (see TrilevelNAS-A in Fig.3(a)). For efficient upsampling block design, we replace the two upsampling blocks [2] with one sub-pixel block, which contains a convolution layer (with an output channel number being $n \times n \times 3$), and a PixelShuffel layer to reach the target resolution with an upscaling factor of n (see TrilevelNAS-B in Fig.3(b)).

Network Level Search Space Modeling. We avoid using a trellis-like structure to model the network-level search space because the trellis modeling aims to traverse all the network blocks' sequential paths. As shown in Fig.2(a), each path starts from the first node and goes along a set of arrows to the target. Clearly, all the paths share most of the nodes and arrows leading to many redundant sharing among the network paths, the cells, and the kernels. Although the architecture sharing strategy saves training memory, it dramatically limits the search space. Further, the tight entanglement is likely to affect the learning on each path's contribution and the pruning of unnecessary paths. Consequently, we propose a tree structure for a flexible network-level path search modeling. As shown in Fig.2(b), each node is merely connected to its father (if applicable) and children; hence the dependencies are highly relaxed. Relaxing the correlations of distinct paths enables a flexible network-level search space, and the better disentanglement among the paths allows us to perform pruning on redundant paths. Moreover, the lower dependency on paths may lead

[1] By on-device, we mean a computer with 1 GPU (16GB RAM).

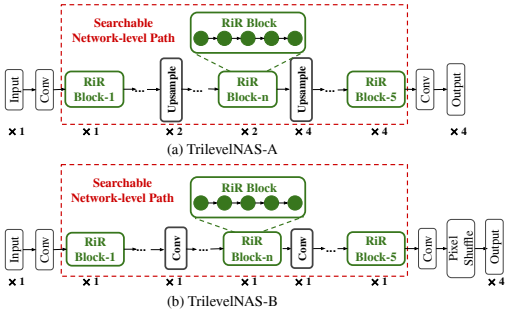

Figure 3. (a) Image SR supernet backbone *TrilevelNAS-A* with 5 RiR blocks and 2 upsampling layers. (b) Image SR supernet backbone *TrilevelNAS-B* with 5 RiR blocks and a pixel shuffle layer.

to a reliable association among cells and kernels due to their hierarchical connection.

We followed [2] to model the cell and kernel level search space, At cell-level, we search for 5 RiR blocks with each block containing 5 searchable cells, i.e., in total, 25 searchable cells. For cell-level search space modeling, we follow the DARTS. To model kernel-level search space, we follow AGD [2] superkernel framework.

3. Proposed Approach for Optimization

We propose a sorted sparsestmax based modeling and optimization for more efficient differentiable NAS on the suggested trilevel search space.

• **Supernet Modeling with Sorted Sparsestmax.** First, we define a set of contribution weights β for all the feature maps from the involved network paths based on our suggested model (Fig.2(b)). So, the output of the supernet is a weighted combination of all the intermediate feature maps. Given a tree model of the network-level search space with N paths being $P = \{P_1, \ldots, P_N\}$ where P_i path has M_i feature maps, the output of the whole supernet (i.e., the mixture of all the candidate paths) is defined as:

$$O_{tree} = \sum_{i=0}^{N} \sum_{j=0}^{M_i} F_N(\beta_{i,j}; \beta) f_{i,j}, \tag{1}$$

where, $f_{i,j}$ is the feature map at the j-th layer of P_i, and $F_N(\beta_{i,j}; \beta)$ indicates the normalized combination weight over the feature map $f_{i,j}$. For the final architecture, we aim at selecting **one single** node from the tree-supernet as the final output node so that the path from the root node to this selected output node is finally chosen for the network-level design. Hence, we employ the sparsestmax instead of softmax and sparsemax to ensure the roughly complete sparsity when normalizing the combination weights. For detailed evaluations on the superiority of sparsestmax over softmax and sparsemax [9, 10], we refer readers to [11].

While sparsestmax gives sparse distributions, it is not aligned well with our sequential setup. Given P_i path with M_i feature maps, sparsestmax produces unordered non-zero combination weights on the feature maps. In that case, we cannot prune the network path well unless the sparsity is ordered. To perform network level pruning we must have non-zero combination weights in descending order along the path so that all the zero weights appear at the tail of the path. Accordingly, we exploited sorted sparsestmax, which imposes an ordering constraint to the weights β_i within each path P_i. It helps the output feature maps from shallower layers to share more contributions to the supernet. We formulate *sorted sparsestmax* as

$$F_N(\beta, r) := \underset{q \in \Delta_r^{k-1}}{\arg\min} \|q - \beta\|_2^2 + \lambda \sum_i \sum_j (\beta_{i,j} - \beta_{i,j-1}), \tag{2}$$

where $\Delta_r^{k-1} := \{q \in \mathbb{R}^K | 1^T q = 1, \|q - u\|_2 \geq r, q \geq 0\}$ indicates a simplex with a circular constraint $1^T q = 1, \|q - u\|_2 \geq r$, $u = \frac{1}{K}1$ is the center of the simplex, 1 is a vector of ones, and r is radius of the circle [11]. Here λ is a trade-off constant. The additional ordering constraint allows for a rough descending order from root node to leaves, such that those bottom nodes can be trimmed for a better network-level pruning. In particular, the ordering constraint to β, which implicitly results in an ordered q due to the target of minimizing $\|q - \beta\|$. In other words, it serves as a *soft* constraint, and thus our algorithm can seek for an optimal trade-off of model performance and network depth.

• **Supercell and Superkernel Modeling.** Since the cell-level network has no sequential properties, we directly apply *sparsestmax* to relax the discrete search space into a continuous one, i.e., $C_i = \sum_{o \in O} f_N(\alpha_i^{(o)}; \alpha_i) o(C_{i-1})$, where α is a set of operations' contribution weights, f_N corresponds to *sparsestmax* [11], $o(C_{i-1})$ is one operations selected from the cell-level space O over C_{i-1}. For the continuous relaxation of the kernel-level search space, we adhere to apply the differentiable *Gumbel-softmax* sampling to keep the setup consistent with AGD [2]. As for optimizing expansion ratio parameters γ_i, we follow AGD to conduct the sampling-based strategy.

Proxy Task and Optimization. For our trilevel NAS task, instead of training the model from scratch, we leverage a pre-trained state-of-the-art SISR model i.e. ESRGAN [13] via knowledge distillation technique as used in AGD [2]. Therefore, the search phase's proxy task aims to search for a model G by minimizing the knowledge distillation distance d between the output from the model G and the pre-trained ESRGAN model G_0. Besides, we consider involving the model efficiency term H in our target for better compression. Overall, the training objective of the proposed TrilevelNAS is as follows:

$$\min_{G, \alpha, \beta, \gamma} \frac{1}{N} \sum_{i=0}^{N} d(G(x_i; \alpha, \beta, \gamma), G_0(x_i)) + \lambda_f H(G; \alpha, \beta, \gamma), \tag{3}$$

where α, β, γ are the parameters for the continuous relaxation of the trilevel architecture search space as defined previously. Following AGD [2], we compute $d(G, G_0)$ with a combination of content loss L_c (avoid color shift), perceptual loss L_p (preserve visual and semantic details), and em-

Method	Path	Params (M)	GFLOPS (256×256)	PSNR Set5	PSNR Set14	Search Cost #(GPUs)× Days	Type
ESRGAN [13]	-	16.70	1176.6	30.44	26.28	-	Manual
ESRGAN-prune [6]	-	1.6	113.1	28.07	25.21	-	Manual
SRGAN [4]	-	1.52	166.7	29.40	26.02	-	Manual
TPSR [5]	-	**0.06**	-	29.60	26.88	40×12	One-level NAS
AGD [2]†	[0,0,0,0,0,1,1]*	0.56	117.7	30.36	27.21	1×2	Bi-level NAS
AGD-AutoDeepLab	[0,0,0,1,0,0,1]	0.71	165.8	30.48	27.23	1×4	Tri-level NAS
TrilevelNAS-A	[0,0,0,1,0,1]	**0.34**	117.4	30.34	27.29	1×8	Tri-level NAS
TrilevelNAS-B	[0,0,0]	**0.24**	**15.4**	29.80	27.06	1×8	Tri-level NAS

Table 1. Quantitative results of visualization-oriented SR models with scaling factor 4. As for the listed Path results, we use '0' to indicate a *RiR* block and '1' is a *Upsampling* block/*Conv* layer in TrilevelNAS-A/TrilevelNAS-B respectively.† Reproduced AGD with official setup and implementation, and * means that the path is fixed rather than being searched. The statistics clearly show that our method can supply a lighter model with manageable computational resources, and its PSNR performance favorable compares to the best methods. Hence, the proposed method performs better with all the evaluation metrics combined.

ploy the FLOPS measure for model efficiency. The detailed algorithm is presented in Supplementary Material.

4. Experiments and Results

We performed search and training on DIV2K and Flickr2K datasets [12], followed by evaluation on the searched architectures for popular benchmarks including Set5 [1], Set14 [14]. We implemented the Network-Level search with two types of backbones: *(i)* **TrilevelNAS-A** and *(ii)* **TrilevelNAS-B** (see Fig.3). More experimental details and results are presented in Supplementary Material.

We adopt ESRGAN [13] model as the teacher to search for a visualization-oriented SR model. We compare our method to state-of-art SR models (ESRGAN [13], a pruned ESRGAN baseline [6] and SRGAN [4]) and NAS-based visualization-oriented SR model (AGD [2] and TPSR [5]). Note that we reproduce the AGD results with their official code and default experiment setup. Besides, for a fair comparison to our Tree-supernet-based Trilevel NAS, we apply the Trellis-supernet design of AutoDeepLab to the AGD backbone to implement another Trilevel NAS for image SR.

Table 1 shows the statistical comparison of our approach against competing methods. We additionally report the derived network path ('0' in path indicate the RiR blocks; '1' represent upsampling layers and convolution layers in TrilevelNAS-A and TrilevelNAS-B, respectively). Compared to the AutoDeepLab-based Trilevel NAS (AGD-AutoDeepLab), our TrellisNAS can further perform network-level pruning (i.e., prune RiR blocks or convolution layers), which allows for significantly reduced model complexity. With comparable performance, TrelevelNAS-A prunes one redundant RiR block and derives a much lighter model with a 0.34MB model size. Even with smaller model size, TrilevelNAS-A still has comparable FLOPS against AGD. We notice that most of the FLOPS consumption comes from the blocks operating on high-dimension feature maps. Remarkably, our new backbone TrilevelNAS-B with post-upsampling framework has the potential to search for a flexible network-level path with both small model size and light GFLOPs. Compared to the original AGD and TrilevelNAS-A, the FLOPs consumption of TrilevelNAS-B is reduced by 4×. Though TPSR derives a tiny architecture, its search cost (60× of our TrilevelNAS' search cost) is extremely expensive, and we also see a clear PSNR performance drop with this tiny model.

References

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

[2] Yonggan Fu, Wuyang Chen, Haotao Wang, Haoran Li, Yingyan Lin, and Zhangyang Wang. Autogan-distiller: Searching to compress generative adversarial networks. In *ICML*, 2020.

[3] Yong Guo, Yongsheng Luo, Zhenhao He, Jin Huang, and Jian Chen. Hierarchical neural architecture search for single image super-resolution. *IEEE SPL*, 2020.

[4] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Tejani, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 2017.

[5] Royson Lee, Łukasz Dudziak, Mohamed Abdelfattah, Stylianos I Venieris, Hyeji Kim, Hongkai Wen, and Nicholas D Lane. Journey towards tiny perceptual super-resolution. In *ECCV*, 2020.

[6] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[7] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*, 2019.

[8] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2018.

[9] Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *ICML*, 2016.

[10] Vlad Niculae and Mathieu Blondel. A regularized framework for sparse and structured neural attention. In *NeurIPS*, 2017.

[11] Wenqi Shao, Tianjian Meng, Jingyu Li, Ruimao Zhang, Yudian Li, Xiaogang Wang, and Ping Luo. Ssn: Learning sparse switchable normalization via sparsestmax. In *CVPR*, 2019.

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

[13] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *ECCV workshops*, 2018.

[14] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *ICCS*, 2010.