Hierarchical Neural Architecture Search for Single Image Super-Resolution

Yong Guo, Yongsheng Luo, Zhenhao He, Jin Huang, and Jian Chen

***Abstract*—Deep neural networks have exhibited promising performance in image super-resolution (SR). Most SR models follow a hierarchical architecture that contains both the cell-level design of computational blocks and the network-level design of the positions of upsampling blocks. However, designing SR models heavily relies on human expertise and is very labor-intensive. More critically, these SR models often contain a huge number of parameters and may not meet the requirements of computation resources in real-world applications. To address the above issues, we propose a Hierarchical Neural Architecture Search (HNAS) method to automatically design promising architectures with different requirements of computation cost. To this end, we design a hierarchical SR search space and propose a hierarchical controller for architecture search. Such a hierarchical controller is able to simultaneously find promising cell-level blocks and network-level positions of upsampling layers. Moreover, to design compact architectures with promising performance, we build a joint reward by considering both the performance and computation cost to guide the search process. Extensive experiments on five benchmark datasets demonstrate the superiority of our method over existing methods.**

***Index Terms*—Neural Architecture Search, Super-Resolution.**

I. Introduction

IMAGE super-resolution (SR) is an important computer vision task that aims at designing effective models to reconstruct the high-resolution (HR) images from the low-resolution (LR) images [1]–[5]. Most SR models consist of two components, namely several upsampling layers that increase spatial resolution and a set of computational blocks (e.g., residual block) that increase the model capacity. These two kinds of blocks/layers often follow a two-level architecture, where the network-level architecture determines the positions of the upsampling layers (e.g., SRCNN [6] and LapSRN [7]) and the cell-level architecture controls the computation of each block/layer (e.g., RCAB [8]). In practice, designing deep models is often very labor-intensive and the hand-crafted architectures are often not optimal in practice.

Regarding this issue, many efforts have been made to automate the model designing process via Neural Architecture Search (NAS) [9]. Specifically, NAS methods seek to find the optimal cell architecture [9]–[13] or a whole network architecture [14]–[17]. However, existing NAS methods may suffer from two limitations if we apply them to search for an optimal SR architecture.

First, it is hard to directly search for the optimal two-level SR architecture. For SR models, both the cell-level blocks and network-level positions of upsampling layers play very important roles. However, existing NAS methods only focus on one of the architecture levels. Thus, how to simultaneously find the optimal cell-level block and network-level positions of upsampling layers is still unknown.

Second, most methods only focus on improving SR performance but ignore the computational complexity. As a result, SR models are often very large and become hard to be applied to real-world applications [18], [19] when the computation resources are limited. Thus, it is important to design promising architectures with low computation cost.

To address the above issues, we propose a novel Hierarchical Neural Architecture Search (HNAS) method to automatically design SR architectures. Unlike existing methods, HNAS simultaneously searches for the optimal cell-level blocks and the network-level positions of upsampling layers. Moreover, by considering the computation cost to build the joint reward, our method is able to produce promising architectures with low computation cost.

Our contributions are summarized as follows:

- We propose a novel Hierarchical Neural Architecture Search (HNAS) method to automatically design cell-level blocks and determine network-level positions of upsampling layers.
- We propose a joint reward that considers both the SR performance and the computation cost of SR architectures. By training HNAS with such a reward, we can obtain a series of architectures with different performance and computation cost.

Manuscript received March 10, 2020; revised May 6, 2020; accepted June 12, 2020. Date of publication June 18, 2020; date of current version July 30, 2020. This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2019B1515130001, in part by the Guangdong Special Branch Plans Young Talent with Scientific and Technological Innovation under Grant 2016TQ03X445, in part by the Guangzhou Science and Technology Planning Project under Grant 201904010197, and in part by the Natural Science Foundation of Guangdong Province under Grant 2016A030313437. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Saurabh Prasad *(Corresponding author: Jian Chen.)*

Yong Guo, Yongsheng Luo, and Zhenhao He are with the School of Software Engineering, South China University of Technology, Guangzhou 510641, China (e-mail: guoyongcs@gmail.com; seluoyongsheng@mail.scut.edu.cn; sezhenhao.he@mail.scut.edu.cn).

Jian Chen is with the School of Software Engineering, South China University of Technology, Guangzhou 510641, China, and also with the Guangdong Key Laboratory of BigData Analysis and Processing, Guangzhou 510006, China (e-mail: ellachen@scut.edu.cn).

Jin Huang is with the School of Computer Science, South China Normal University, Guangzhou 510631, China (e-mail: huangjin@m.scnu.edu.cn).

This letter has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/LSP.2020.3003517

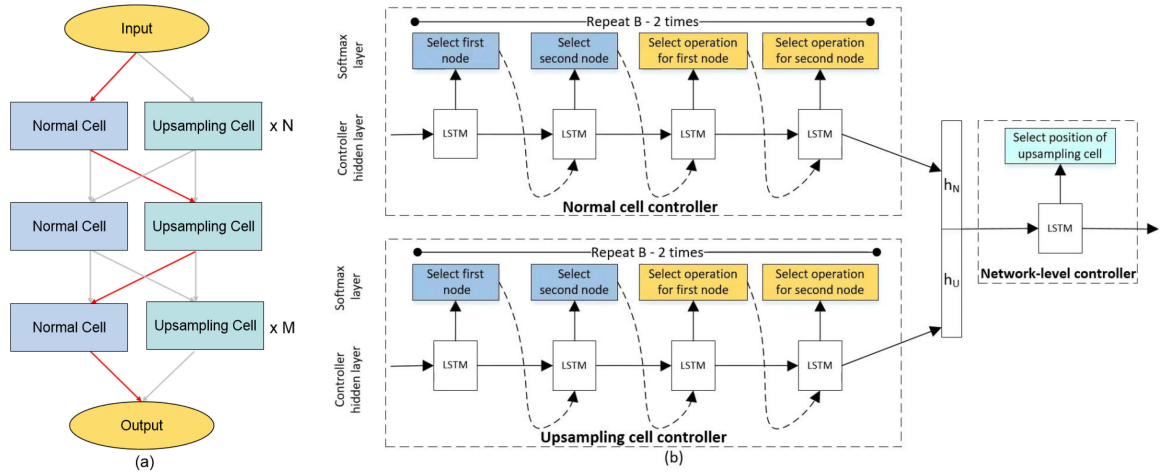

Fig. 1. The overview of the proposed HNAS method. (a) The architecture of the two-branch super network. The red line represents a searched model with the upsampling layer at a specific layer. N and M denote the number of layers before and after the upsampling blocks. (b) The proposed hierarchical controller.

- Extensive experiments on several benchmark datasets demonstrate the superiority of the proposed method.

II. Proposed Method

In this paper, we propose a Hierarchical Neural Architecture Search (HNAS) method to automatically design promising two-level SR architectures, with good performance and low computation cost. To this end, we first define our hierarchical search space that consists of a cell-level search space and a network-level search space. Then, we propose a hierarchical controller as an agent to search for good architectures. To search for promising SR architectures with low computation cost, we develop a joint reward by considering both the performance and computation cost. We show the overall architecture and the controller model of HNAS in Figure 1.

A. Hierarchical SR Search Space

In general, SR models often consist of two components, namely several upsampling layers that increase spatial resolution and a series of computational blocks that increase the model capacity. These two components form a two-level architecture, where the cell-level identifies the computation of each block and the network-level determines the positions of the upsampling layers. Based on the hierarchical architecture, we propose a hierarchical SR search space that contains a cell-level search space and a network-level search space.

Cell-level search space. In the cell-level search space, as shown in Fig. 2, we represent a cell as a directed acyclic graph (DAG) [9], [11], [20], [21], where the nodes denote the feature maps in deep networks and the edges denote some computational operations, convolution. In this paper, we define two kinds of cells: (1) the normal cell that controls the model capacity and keeps the spatial resolution of feature maps unchanged, and (2) the upsampling cell that increases the spatial resolution. To design these cells, we collect the two sets of operations that have been widely used in SR models. We show the candidate operations for both cells in TABLE I.

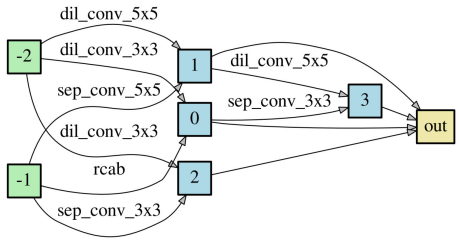

Fig. 2. An example of DAG that represents a cell architecture.

TABLE I
Candidate Operations for Normal Cell and Upsampling Cell

Normal Cell/Block	Upsampling Cell/Block
• identity (skip connection)	• area interpolation
• 3×3 dilated convolution	• bilinear interpolation
• 5×5 dilated convolution	• nearest-neighbor interpolation
• 3×3 separable convolution	• sub-pixel layer
• 5×5 separable convolution	• deconvolution layer
• up and down-projection block [15]	
• residual channel attention block [8]	

For the normal cell, we consider seven candidate operations, including identity mapping, 3×3 and 5×5 dilated convolution, 3×3 and 5×5 separable convolution, up and down-projection block (UDPB) [22], and residual channel attention block (RCAB) [8]. For the upsampling cell, we consider 5 widely used operations to increase spatial resolution. Specifically, there are 3 interpolation-based upsampling operations, including area interpolation, bilinear interpolation [6], nearest-neighbor interpolation [23]. Moreover, we also consider 2 trainable convolutional layers, namely the deconvolution layer (also known as transposed convolution) [24] and the sub-pixel convolution [25].

Based on the candidate operations, the goal of HNAS is to select the optimal operation for each edge of DAG and learn the optimal connectivity among nodes (See more details in Section II-B).

Network-level search space. Note that the position of upsampling block/layer plays an important role in both the performance and computation cost of SR models. Specifically, if we put the upsampling block in a very shallow layer, the feature map

would increase too early and hence significantly increase the computational cost of the whole model. By contrast, when we put the upsampling block in a deep layer, there would be little or no layers to process the upsampled features and hence the computation to obtain high-resolution images may be insufficient, leading to suboptimal SR performance. Regarding this issue, we seek to find the optimal position of the upsampling block for different SR models.

To this end, we design a two-branch super network whose architecture is fixed through the whole search process. As shown in Fig. 1, there are two kinds of cells (i.e, normal cell and upsampling cell) at each layer. Given a specific position of the upsampling cell, we set the selected layer to the upsampling cell and set the others to the normal cells. In this way, the model with a specific position of the upsampling cell becomes a sub-network of the proposed super network. Let N and M denote the number of layers before and after the upsampling blocks. Thus, there are $L = M + N + 1$ blocks in total. We will show how to determine the position of the upsampling layer in Section II-B.

B. Hierarchical Controller for HNAS

Based on the hierarchical search space, we seek to search for the optimal cell-level and network-level architectures. Following [9], [14], we use a long short-term memory (LSTM) [26] as the controller to produce candidate architectures (represented by a sequence of tokens [14]). Regarding the two-level hierarchy of SR models, we propose a hierarchical controller to produce promising architectures. Specifically, we consider two kinds of controllers, including a cell-level controller that searches for the optimal architectures for both normal block and upsampling block, and a network-level controller that determines the positions of upsampling layers.

Cell-level controller. We utilize a cell-level controller to find the optimal computational DAG with B nodes (See example in Fig. 2). In a DAG, the input nodes -2 and node -1 denote the outputs of the second nearest and the nearest cell in front of the current block, respectively. The remaining $B-2$ nodes are intermediate nodes, each of which also takes two previous nodes in this cell as inputs. For each intermediate node, the controller makes two kinds of decisions: 1) which previous node should be taken as input and 2) which operation should be applied to each edge. All of these decisions can be represented as a sequence of tokens and thus can be predicted using the LSTM controller [14]. After repeating $B-2$ times, all of the $B-2$ nodes are concatenated together to obtain the final output of the cell, the output node.

Network-level controller. Once we have the normal block and upsampling block, we seek to further determine where we should put the upsampling block to build the SR model. Given a model with L layers, we predict the position, an integer ranging from 1 to L, where we put the upsampling block. Since such a position relies on the design of both normal and upsampling blocks, we build the network-level controller that takes the embeddings (i.e, hidden states) of two kinds of blocks as inputs to determine the position. Specifically, let h_N and h_U denote the last hidden states of the controllers for normal block and upsampling block, respectively. We concatenate these embeddings as the initial state of the network level controller (See Fig. 1(b)). Since the network-level controller considers the information of the architecture design of both the normal and upsampling blocks, it becomes possible to determine the position of the upsampling block.

Algorithm 1: Training Method for HNAS.

Require: The number of iterations T, learning rate η, shared parameters w, controller parameters θ.

Initialize w and θ.
for $i = 1$ to T **do**
  // Update w by minimizing the training loss
  for each iteration on training data **do**
    Sample $\alpha \sim \pi(\alpha; \theta)$;
    $w \leftarrow w - \eta \nabla_w \mathcal{L}(\alpha, w)$.
  end for
  // Update θ by maximizing the reward
  for each iteration on validation data **do**
    Sample $\alpha \sim \pi(\alpha; \theta)$;
    $\theta \leftarrow \theta + \eta \mathcal{R}(\alpha) \nabla_\theta \log \pi(\alpha; \theta, \Omega_i)$;
  end for
end for

C. Training and Inference Methods

To train HNAS, we first propose the joint reward to guide the architecture search process. Then, we depict the detailed training and inference methods of HNAS.

Joint reward. Designing promising architectures with low computation cost is important for real-world SR applications. To this end, we build a joint reward by considering both performance and computation cost to guide the architecture search process. Given any architecture α, let $\mathrm{PSNR}(\alpha)$ be the PSNR performance of α, $\mathrm{Cost}(\alpha)$ be the computation cost of α in terms of FLOPs (i.e., the number of multiply-add operations). The joint reward can be computed by

$$\mathcal{R}(\alpha) = \lambda * \mathrm{PSNR}(\alpha) - (1 - \lambda) * \mathrm{Cost}(\alpha), \tag{1}$$

where λ controls the trade-off between the PSNR performance and the computation cost. Such a trade-off exists when there is a limited budget of computation resources and we can adjust λ in the proposed joint reward function to meet different requirements of real-world applications. In general, a larger λ makes the controller pay more attention to improving the PSNR performance but regardless of the computation cost. By contrast, a smaller λ makes the controller focus more on reducing the computation cost.

Training method for HNAS. With the joint reward, following [9], [14], we apply the policy gradient [27] to train the controller. We show the training method in Algorithm 1. To accelerate the training process, we adopt the parameter sharing technique [9], we construct a large computational graph, where each subgraph represents a neural network architecture, hence forcing all architectures to share the parameters.

Let θ and w be the parameters of the controller model and the shared parameters. The goal of HNAS is to learn an optimal policy $\pi(\cdot)$ and produce candidate architectures by conduct

TABLE II
COMPARISONS WITH THE STATE-OF-THE-ART METHODS BASED ON $\times 2$ SUPER-RESOLUTION TASK. RESULTS MARKED WITH "†" WERE OBTAINED BY TRAINING THE CORRESPONDING ARCHITECTURES USING OUR SETUP. "-" DENOTES THE RESULTS THAT ARE NOT REPORTED

Model	#FLOPs (G)	SET5 PSNR / SSIM	SET14 PSNR / SSIM	B100 PSNR / SSIM	Urban100 PSNR / SSIM	Manga109 PSNR / SSIM
Bicubic	-	33.65 / 0.930	30.24 / 0.869	29.56 / 0.844	26.88 / 0.841	30.84 / 0.935
SRCNN [6]	52.7	36.66 / 0.954	32.42 / 0.906	31.36 / 0.887	29.50 / 0.894	35.72 / 0.968
VDSR [22]	612.6	37.53 / 0.958	33.03 / 0.912	31.90 / 0.896	30.76 / 0.914	37.16 / 0.974
DRCN [23]	17,974.3	37.63 / 0.958	33.04 / 0.911	31.85 / 0.894	30.75 / 0.913	37.57 / 0.973
DRRN [24]	6,796.9	37.74 / 0.959	33.23 / 0.913	32.05 / 0.897	31.23 / 0.918	37.92 / 0.976
SelNet [25]	225.7	37.89 / 0.959	**33.61** / 0.916	32.08 / 0.898	-	-
CARN [26]	222.8	37.76 / 0.959	33.52 / 0.916	32.09 / 0.897	31.92 / 0.925	38.36 / 0.976
MoreMNAS-A [27]	238.6	37.63 / 0.958	33.23 / 0.913	31.95 / 0.896	31.24 / 0.918	-
FALSR [28]	74.7	37.61 / 0.958	33.29 / 0.914	31.97 / 0.896	31.28 / 0.919	37.46 / 0.974
Residual Block [29]†	47.5	36.72 / 0.955	32.20 / 0.905	31.30 / 0.888	29.53 / 0.897	33.36 / 0.962
RCAB [8]†	84.9	37.66 / 0.959	33.17 / 0.913	31.93 / 0.896	31.19 / 0.918	37.80 / 0.974
Random†	111.7	37.83 / 0.959	33.31 / 0.915	31.98 / 0.897	31.42 / 0.920	38.31 / 0.976
HNAS-A ($\lambda = 0.2$)	**30.6**	37.84 / 0.959	33.39 / 0.916	32.06 / 0.898	31.50 / 0.922	38.15 / 0.976
HNAS-B ($\lambda = 0.6$)	48.2	37.92 / 0.960	33.46 / 0.917	32.08 / 0.898	31.66 / 0.924	38.46 / 0.977
HNAS-C ($\lambda = 0.9$)	83.6	**38.11** / **0.964**	33.60 / **0.920**	**32.17** / **0.902**	**31.93** / **0.928**	**38.71** / **0.985**

sampling $\alpha \sim \pi(\alpha)$. To encourage exploration, we introduce an entropy regularization term into the objective. In this way, we can train as diverse architectures as possible in the super network and thus alleviate the unfairness issue [28] that some sub-networks (or candidate operations) may be over-optimized while the others are under-optimized.

Inferring Architectures. Based on the learned policy $\pi(\cdot)$, we conduct sampling to obtain promising architectures. Specifically, we first sample several candidate architectures and then select the architecture with the highest validation performance. Finally, we build SR models using the searched architectures (including both the cell-level blocks and network-level position of upsampling blocks) and train them from scratch.

III. EXPERIMENTS

In the experiments, we use the DIV2K dataset [37] to train all the models and conduct comparisons on five benchmark datasets, including Set5 [38], Set14 [39], BSD100 [40], Urban100 [41], and Manga109 [42]. We compare different models in terms of PSNR, SSIM, and FLOPs. Please see more training details in supplementary. We have made the code of HNAS available at https://github.com/guoyongcs/HNAS-SR.

A. Quantitative Results

In this experiment, we consider three settings (i,e., $\lambda = \{0.2, 0.6, 0.9\}$) and use HNAS-A/B/C to represent the searched architectures under these settings, respectively. We show the detailed architectures in supplementary. Table II shows the quantitative comparisons for $2\times$ SR. Note that all FLOPs are measured based on a $3 \times 480 \times 480$ input LR image. Compared with the hand-crafted models, our models tend to yield higher PSNR and SSIM and lower fewer FLOPs. Specifically, HNAS-A yields the lowest FLOPs but still outperforms a large number of baseline methods. Moreover, when we gradually increase λ, HNAS-B and HNAS-C take higher computation cost and yield better performance. These results demonstrate that HNAS can produce architectures with promising performance and low computation cost.

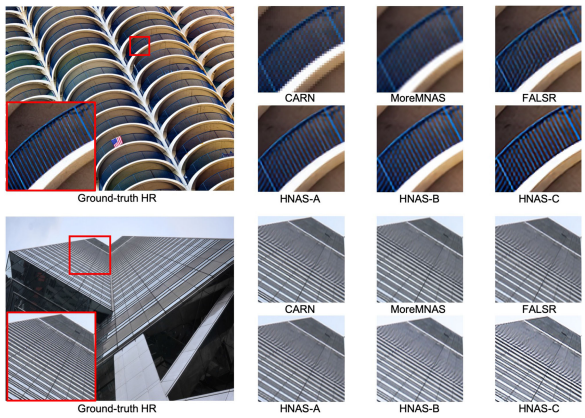

Fig. 3. Visual comparisons of different methods for $2\times$ SR.

B. Visual Results

To further show the effectiveness of the proposed method, we also conduct visual comparisons between HNAS and several state-of-the-arts. We show the results in Fig. 3. From Fig. 3, the considered baseline methods often produce very blurring images with salient artifacts. By contrast, the searched models by HNAS are able to produce sharper images than other methods. These results demonstrate the effectiveness of the proposed method.

IV. CONCLUSION

In this paper, we have proposed a novel Hierarchical Neural Architecture Search (HNAS) method to automatically search for the optimal architectures for image super-resolution (SR) models. Since most SR models follow the two-level architecture design, we define a hierarchical SR search space and develop a hierarchical controller to produce candidate architectures. Moreover, we build a joint reward by considering both SR performance and computation cost to guide the search process of HNAS. With such a joint reward, HNAS is able to design promising architectures with low computation cost. Extensive results on five benchmark datasets demonstrate the effectiveness of the proposed method.

REFERENCES

[1] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, "Learning low-level vision," *Int. J. Comput. Vision*, vol. 40, no. 1, pp. 25–47, 2000.

[2] Y. Guo *et al.*, "Closed-loop matters: Dual regression networks for single image super-resolution," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2020, pp. 5407–5416.

[3] W. Yang, W. Wang, X. Zhang, S. Sun, and Q. Liao, "Lightweight feature fusion network for single image super-resolution," *IEEE Signal Process. Lett.*, vol. 26, no. 4, pp. 538–542, Apr. 2019.

[4] Y. Yang, D. Zhang, S. Huang, and J. Wu, "Multilevel and multiscale network for single-image super-resolution," *IEEE Signal Process. Lett.*, vol. 26, no. 12, pp. 1877–1881, Dec. 2019.

[5] L. Huang, J. Zhang, Y. Zuo, and Q. Wu, "Pyramid-structured depth map super-resolution based on deep dense-residual network," *IEEE Signal Process. Lett.*, vol. 26, no. 12, pp. 1723–1727, Dec. 2019.

[6] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 295–307, Feb. 2016.

[7] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep Laplacian pyramid networks for fast and accurate super-resolution," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 624–632.

[8] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 286–301.

[9] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. Int. Conf. Mach. Learn.*, 2019, *arXiv:1802.03268*.

[10] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 8697–8710.

[11] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," presented at the Int. Conf. Learn. Representations, San Diego, CA, USA, 2018.

[12] Y. Guo *et al.*, "Nat: Neural architecture transformer for accurate and compact architectures," in *Proc.Adv. Neural Inf. Process. Syst.*, 2019, pp. 735–747.

[13] Y. Guo *et al.*, "Breaking the curse of space explosion: Towards efficient nas with curriculum search," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020.

[14] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2017.

[15] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Representations*, 2019.

[16] M. Tan, *et al.*"Mnasnet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 2820–2828.

[17] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Representations*, 2020.

[18] Z. Zhuang *et al.*, "Discrimination-aware channel pruning for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 875–886.

[19] J. Liu *et al.*, "Discrimination-aware network pruning for deep model compression," 2020, *arXiv:2001.01050*.

[20] Y. Xu *et al.*, "PC-DARTS: Partial channel connections for memory-efficient architecture search," in *Proc. Int. Conf. Learn. Representations*, 2020.

[21] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE Int. Conf. Comput. Vision*, 2019, pp. 1294–1303.

[22] M. Haris, G. Shakhnarovich, and N. Ukita, "Deep back-projection networks for super-resolution," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 1664–1673.

[23] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," 2016, *arXiv:1610.07629*.

[24] M. D. Zeiler, G. W. Taylor, R. Fergus , "Adaptive deconvolutional networks for mid and high level feature learning." in *Proc. IEEE Int. Conf. Comput. Vision*, 2011, pp. 2018–2025.

[25] W. Shi *et al.*, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 1874–1883.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3/4, pp. 229–256, 1992.

[28] X. Chu, B. Zhang, R. Xu, and J. Li, "Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search," 2019, *arXiv:1907.01845*.

[29] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 1646–1654.

[30] J. Kim, J. Kwon Lee, and K. Mu Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 1637–1645.

[31] Y. Tai, J. Yang, and X. Liu, "Image super-resolution via deep recursive residual network," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 3147–3155.

[32] J.-S. Choi and M. Kim, "A deep convolutional neural network with selection units for super-resolution," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. Workshops*, 2017, pp. 154–160.

[33] N. Ahn, B. Kang, and K.-A. Sohn, "Fast, accurate, and lightweight super-resolution with cascading residual network," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 252–268.

[34] X. Chu, B. Zhang, R. Xu, and H. Ma, "Multi-objective reinforced evolution in mobile neural architecture search," 2019, *arXiv:1901.01074*.

[35] X. Chu, B. Zhang, H. Ma, R. Xu, J. Li, and Q. Li, "Fast, accurate and lightweight super-resolution with neural architecture search," 2019, *arXiv:1901.07261*.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 770–778.

[37] R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, and L. Zhang, "Ntire 2017 challenge on single image super-resolution: Methods and results," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. Workshops*, 2017, pp. 114–125.

[38] M. Bevilacqua, A. Roumy, C. Guillemot, and M. L. Alberi-Morel, "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," in *Proc. 23rd Brit. Mach. Vision Conf. (BMVC)*, 2012, pp. 135.1–135.10.

[39] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse-representations," in *Proc. Int. Conf. Curves Surf.*, 2010, pp. 711–730.

[40] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. Eighth IEEE Int. Conf. Comput. Vision. (ICCV)*, 2001, pp. 416–423.

[41] J.-B. Huang, A. Singh, and N. Ahuja, "Single image super-resolution from transformed self-exemplars," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 5197–5206.

[42] A. Fujimoto, T. Ogawa, K. Yamamoto, Y. Matsui, T. Yamasaki, and K. Aizawa, "Manga109 dataset and creation of metadata," in *Proc. 1st Int. Workshop Comics Anal., Process. Understanding*, 2016, pp. 1–5.