Trilevel Neural Architecture Search for Efficient Single Image Super-Resolution (Supplementary Matrial)

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

In the supplementary material, we first provide a detailed description of the proposed trilevel SISR NAS algorithm's coding implementation as outlined in the main paper with the coding platform details, such as hardware and software used in our implementation. Further, detailed information on the optimal architecture obtained after trilevel optimization is tabulated. Finally, we supply more experimental results on Set5 [1], Set14 [6], and DIV2K [4] valid sets as well as some ablation study, showing our approach's clear superiority over state-of-the-art methods.

1. Implementation Details

Our approach is implemented in Python 3.6 with Pytorch 1.3.1 library. The complete search process takes 8 GPU days on a single NVIDIA Tesla V100 (16GB RAM), and the training of a PSNR-oriented and visualization-oriented SR model takes 0.5 and 1.5 GPU days, respectively.

Algorithm 1 presents the **pseudocode of our TrilevelNAS SISR algorithm** implementation. It mainly consists of two stages, *Search* and *Train from search*, and the following paragraphs further describe the details about these steps respectively for both the visualization-oriented SR model and PSNR-oriented SR model.

The proposed NAS optimization algorithm contains three steps: *search*, *model discretization* and *train from scratch*. Due to the sparsity property of sparsestmax, we suggest *Train from Search* without the need for discretization step, which reduces the model discrepancy between supernet and the discretized architecture.

• **Search:** We follow AGD[1] [2] to apply the same strategy as follows. The search process comprises two phases: a pre-train phase and a search phase. The pre-train phase uses half of the training data to update the network weights for 100 epochs with only content loss L_c. The search phase alternately updates the network weights and architecture weights for 100 epochs on two equally split training data. The PSNR-oriented model search is optimized with the content loss L_c, and the visualization-oriented model search is tuned with the perceptual loss for better visual quality. We implement our search phase using DIV2K and Flickr2K datasets [4].

(a) Visualization-oriented: Here, we took the visualization-oriented ESRGAN model as a teacher model for the SR model search task. Subsequently, we perform the search task in two phases: *(i) Pretrain*: using half of the training data, we optimize the content loss L_c and train the supernet weights without updating architecture parameters. We train for 100 epochs using Adam optimizer, and at each optimization step, 3 patches of size 32×32 are randomly cropped. For architecture parameters, we use a constant learning rate 3×10^{-4}, and for network weights, we set the initial learning rate as 1×10^{-4}, decayed by 0.5 at 25^{th}, 50^{th}, 75^{th} epoch. *(ii) Search*: Optimizing content loss L_c and perceptual loss L_p, we alternatively update the network weights and architecture weights for 100 epochs on two equally split training data. The optimizer follows the same setting as in the pre-train phase.

(b) PSNR-oriented: For this, we took the PSNR-oriented ESRGAN model as teacher model for SR model search and followed the similar procedure i.e., *Pretrain* and *Search*, and however, here we optimize only the content loss L_c in the two phases.

• **Train from search.** As we do not need a model discretization step with a completely converged supernet after the search process, we can inherit the pre-trained network weights from the search stage as a good network initialization and continue to train the converged architecture. Following the same training process [2], we train the PSNR-oriented SR model with the content loss L_c for 900 epochs. For the visualization-oriented SR model, we con-

[1]We follow AGD [2] to use the same computational constraint (i.e., FLOPS-based one) that has been well-studied, and apply the single run to conduct the evaluations.

Algorithm 1 The Proposed TrileveNAS SISR Algorithm

Input: dataset $\chi = \{x_i\}_{i=1}^{N}$, pretrained generator G_0, search space and supernet G, epochs to pretrain (T_1), search (T_2) and train-from-scratch (T_3)
Output: trained efficient generator G^*
Equally split χ into χ_1 and χ_2 and initialize supernet weight w and architecture parameters $\{\alpha, \beta, \gamma\}$ with uniform distribution
 # First Step: Pretrain
for $t \leftarrow 1$ to T_1 **do**
  Get a batch of data X_1 from χ_1
  for γ in $[\gamma_{\max}, \gamma_{\min}, \gamma_{\text{random1}}, \gamma_{\text{random2}}]$ **do**
    $g_w^{(t)} = \nabla_w d(G(X_1, \alpha, \beta, \gamma), G_0(X_1))$
    $w^{(t+1)}$ = update($w^{(t)}$, $g_w^{(t)}$)
  end for
end for
 # Second Step: Search
for $t \leftarrow 1$ to T_2 **do**
  Get a batch of data X_1 from χ_1
  $g_w^{(t)} = \nabla_w d(G(X_1, \alpha, \beta, \gamma), G_0(X_1))$
  $w^{(t+1)}$ = update($w^{(t)}$, $g_w^{(t)}$)
  Get a batch of data X_2 from χ_2
  $g_\alpha^{(t)} = \nabla_\alpha d(G(X_2, \alpha, \beta, \gamma), G_0(X_2)) + \lambda\omega_1 \nabla_\alpha F(\alpha|\beta, \gamma)$
  $g_\beta^{(t)} = \nabla_\alpha d(G(X_2, \alpha, \beta, \gamma), G_0(X_2)) + \lambda\omega_1 \nabla_\beta F(\beta|\alpha, \gamma)$
  $g_\gamma^{(t)} = \nabla_\gamma d(G(X_2, \alpha, \beta, \gamma), G_0(X_2)) + \lambda\omega_2 \nabla_\gamma F(\gamma|\alpha, \beta)$
  $\alpha^{(t+1)}$ = update($\alpha^{(t)}$, $g_\alpha^{(t)}$)
  $\beta^{(t+1)}$ = update($\beta^{(t)}$, $g_\beta^{(t)}$)
  $\gamma^{(t+1)}$ = update($\gamma^{(t)}$, $g_\gamma^{(t)}$)
end for
#Third Step: Train from search
Derive the searched architecture G^* with maximal $\{\alpha, \beta, \gamma\}$ for each layer and inherit weight w.
for $t \leftarrow 1$ to T_3 **do**
  Get a batch of data X from χ
  $g_w^{(t)} = \nabla_w d(G^*(X), G_0(X))$
  $w^{(t+1)}$ = update($w^{(t)}$, $g_w^{(t)}$)
end for

tinue to finetune for 1800 epochs with a perceptual loss L_p. We inherit the weights from the derived supernet and train with DIV2K and Flickr2K datasets [4] at the training phase. Patches of size 32×32 are randomly cropped, and the batch size is set to 16.

(a) Visualization-oriented: SR model training is conducted in two steps: *(i) Pretrain* by minimizing content loss L_c for 900 epochs. Adam optimizer is used with initial learning rate 1×10^{-4} and learning rate decays by 0.5 at 225^{th}, 450^{th}, 675^{th} epoch. *(ii) Fine-tune* with perceptual loss L_p for 1800 epochs. We also train using Adam optimizer with initial learning rate 1×10^{-4}, decayed by 0.5 at 225^{th}, 450^{th}, 900^{th}, 1350^{th} epoch.

(b) PSNR-oriented: SR model is trained by minimizing the content loss L_c for 900 epochs, and the optimizer follows the same setting as the pretrain phase in visualization-oriented model training. Note that, for the challenge setup, we only train our proposed model with DIV2K (without using extra data), and merely use a teacher for data distillation instead of using the ground truths for strong supervision. This might lead to marginally inferior performance, but it is closer to real-world scenarios that generally lacks of ground truths. We first crop LR images in DIV2K into sub-images of size 120×120. We use Adam to train for 300 epochs, where the learning rate decays by 0.5 at 75^{th}, 150^{th}, 225^{th} epoch.

2. Architecture Details

Table 1-4 present our derived network-level (path), cell-level (5 searchable operations in each RiR block) and kernel-level (output channel number) architectures. Table 1-2 demonstrate the SR architecture models obtained using visualization-oriented search strategy on DIV2K and Flick2K datasets. Table 3-4 provide the SR architectures obtained by our proposed method using PSNR-oriented search strategy.

The obtained network-level path is shown in first row of the respective table. The entry '**0**' and '**1**' in the path-index row-vector indicates RiR block and upsampling/convolution layer respectively. In addition, we lists the cell-level structures (i.e., the selection of operations in OP1-OP5 in each RiR block) and the kernel-level structures (i.e., the selection of output channels from the full 64 channels) in Table 3-4. For each OP, the format (A, B) indicates selecting the operation A with its kernel width being B. Regarding the types of different OPs, DwsBlock, ResBlock, Conv symbolises depthwise convolution block, residual block and convolution block respectively.

3. More Experimental Results

PSNR Oriented Result. We implemented the PSNR-oriented SR model search on DIV2K and Flickr2K datasets and used the PSNR-oriented ESRGAN model as our teacher model. We have observed a clear FLOPs advantage of our new backbone (TrilevelNAS-B) over the original backbone (TrilevelNAS-A) in previous experiments. Here, we focus on the Trilevel search on our new backbone and aim for a more efficient SR model. We compare our derived model against our competitors in Table 5. With comparable performance, the derived PSNR-oriented SR model on the new backbone prunes a convolution layer, and it is clearly better in terms of FLOPS and parameter size than the competitors.

Further, we compared our model with the winner and runner-up of the recent AIM challenge [7]. Following the challenge setup, we train and validate our model with 800 DIV2K training images and 100 valid images respectively. Table 6 provides the quantitative results of our method with competing methods, which include challenge winner (NJU_MCG), runner-up (AiriA_CG), and NAS-based SR models. It can be inferred from the statistics that our method, despite being significantly lighter, gives PSNR

Path	**[0, 0, 0, 1, 0, 1]**				
RiR Block ID	OP1	OP2	OP3	OP4	OP5
0	(Conv 1×1, 64)	(Conv 3×3, 24)	(Conv 1×1, 24)	(Conv 1×1, 24)	(Conv 3×3, 64)
1	(DwsBlock, 32)	(DwsBlock, 32)	(DwsBlock, 64)	(Conv 3×3, 24)	(ResBlock, 64)
2	(Conv 1×1, 64)	(Conv 1×1, 64)	(Conv 3×3, 40)	(Conv 1×1, 32)	(DwsBlock, 64)
3	(Conv 3×3, 24)	(Conv 3×3, 64)	(Conv 1×1, 32)	(Conv 3×3, 32)	(Conv 3×3, 64)

Table 1. Visualization-oriented SR model architecture searched on DIV2K and Flickr2K dataset [4] with original AGD backbone (TrilevelNAS-A). We present the network-level architecture (i.e., path of stacking RiR blocks and upsampling layers), the cell-level structures (i.e., the selection of operations in OP1-OP5 in each RiR block) and the kernel-level structures (i.e., the selection of output channels from the full 64 channels). In the path, '0' indicates a RiR block and the '1' indicates an upsamling layer. For each OP, the format (A, B) indicates selecting the operation A with its kernel width being B.

Path	**[0, 0, 0]**				
RiR Block ID	OP1	OP2	OP3	OP4	OP5
0	(DwsBlock, 32)	(Conv 1×1, 32)	(DwsBlock, 24)	(DwsBlock, 40)	(DwsBlock, 64)
1	(DwsBlock, 24)	(Conv 1×1, 56)	(Conv 3×3, 24)	(DwsBlock, 24)	(Conv 3×3, 64)
2	(ResBlock, 64)	(Conv 1×1, 24)	(Conv 1×1, 24)	(Conv 1×1, 24)	(Conv 1×1, 64)

Table 2. Visualization-oriented SR model architecture searched on DIV2K and Flickr2K dataset [4] with our proposed new backbone (TrilevelNAS-B). We present the network-level architecture (i.e., path of stacking RiR blocks and convolutional layers), the cell-level structures (i.e., the selection of operations in OP1-OP5 in each RiR block) and the kernel-level structures (i.e., the selection of output channels from the full 64 channels). In the path, '0' indicates a RiR block and the '1' indicates a convolutional layer. For each OP, the format (A, B) indicates selecting the operation A with its kernel width being B.

Path	**[0, 0, 0, 0, 0, 1]**				
RiR Block ID	OP1	OP2	OP3	OP4	OP5
0	(Conv 1×1, 32)	(Conv 1×1, 24)	(DwsBlock, 64)	(DwsBlock, 24)	(Conv 1×1, 64)
1	(DwsBlock, 32)	(ResBlock, 40)	(DwsBlock, 56)	(Conv 3×3, 40)	(Conv 3×3, 64)
2	(Conv 3×3, 64)	(Conv 1×1, 64)	(Conv 3×3, 24)	(Conv 1×1, 32)	(DwsBlock, 64)
3	(DwsBlock, 64)	(DwsBlock, 56)	(Conv 3×3, 24)	(Conv 3×3, 24)	(DwsBlock, 64)
4	(DwsBlock, 32)	(Conv 3×3, 64)	(Conv 3×3, 64)	(Conv 3×3, 24)	(Conv 3×3, 64)

Table 3. PSNR-oriented SR model architecture searched on DIV2K and Flickr2K [4] with our new backbone (TrilevelNAS-B). We present the network-level architecture (i.e., path of stacking RiR blocks and convolutional layers), the cell-level structures (i.e., the selection of operations in OP1-OP5 in each RiR block) and the kernel-level structures (i.e., the selection of output channels from the full 64 channels). In the path, '0' indicates a RiR block and the '1' indicates a convolutional layer. For each OP, the format (A, B) indicates selecting the operation A with its kernel width being B.

Path	**[0, 0, 1, 1]**				
RiR Block ID	OP1	OP2	OP3	OP4	OP5
0	(Conv 1×1, 40)	(Conv 1×1, 24)	(Conv 1×1, 56)	(ResBlock, 56)	(Conv 1×1, 64)
1	(DwsBlock, 64)	(DwsBlock, 56)	(Conv 3×3, 32)	(Conv 3×3, 24)	(Conv 1×1, 64)

Table 4. PSNR-oriented SR model architecture searched on DIV2K and Flickr2K [4] with our proposed TrilevelNAS-B on AIM challenge [7]. We present the network-level structures (i.e., path of stacking RiR blocks and convolutional layers), the cell-level structures (i.e., the selection of operations in OP1-OP5 in each RiR block) and the kernel-level structure (i.e., the selection of output channels from the full 64 channels). In the path, '0' indicates a RiR block and the '1' indicates a convolutional layer. For each OP, the format (A, B) indicates selecting the operation A with its kernel width being B.

Method	Path	Params (M)	GFLOPS 256×256	PSNR Set5	PSNR Set14	Type
ESRGAN [5]	-	16.70	1176.6	32.70	28.95	Manual
HNAS^{+}	Up: 9^{th} layer	1.69	330.7	31.94	28.41	Bi-level NAS
AGD [2]‡	[0,0,0,0,0,1,1]*	0.90	140.2	31.85	28.40	Bi-level NAS
AGD-AutoDeepLab†	[0,0,0,1,0,0,1]	0.71	165.8	31.83	28.38	Tri-level NAS
TrilevelNAS-B	[0, 0, 0, 0, 0, 1]	**0.51**	**33.3**	31.62	28.26	Tri-level NAS

Table 5. Quantitative results of PSNR-oriented SR models with scaling factor 4. The listed Paths use '0' to indicate a *RiR* block, and '1' to denote a *UpConv* layer. The following symbols indicate: $^{+}$ Reproduced HNAS for PSNR-oriented $\times$4 SR tasks; ‡ Reproduced with AGD official setup and implementation; † Transferred from the visualization-oriented model; * Fixed path. Clearly, our method performs better with all the evaluation metrics combined.

Method	Params (M)	GFLOPS	PSNR [Val]	Extra Data	Ground Truth
NJU_MCG	0.43	27.10	29.04	✓	✓
AiriA_CG	0.687	44.98	29.00	✓	✓
HNAS [3]	1.69	330.74	28.86	×	×
AGD [2]	0.45	110.9	28.66	×	×
AGD-AutoDeepLab	0.71	165.8	28.83	×	×
TrilevelNAS-B	**0.27**	**17.33**	28.52	×	×

Table 6. Quantitative results on AIM 2020 Challenge dataset.

Method	Path	Params (M)	GFLOPS	PSNR Set5	PSNR Set14
Softmax	[0,0,1,0,1]	0.47	154.80	30.34	27.28
Sparsestmax	[0,0,0,1,0,1]	**0.34**	**117.39**	30.34	27.29

Table 7. Quantitative results comparison of visualization-oriented SR models searched using softmax and sparsestmax supernet.

λ	Path	Params (M)	GFLOPS	PSNR Set5	PSNR Set14
0	[0,0,1,0,0,0,1]	0.58	184.08	30.32	27.21
0.01	[0,0,1,0,0,1]	0.52	169.69	30.34	27.20
0.1	[0,0,0,1,0,1]	0.34	117.39	30.34	27.29

Table 8. Quantitative results of visualization-oriented SR models with scaling factor 4 for different ordering constraint strengths. In the listed Path, '0' indicate a *RiR* block and '1' a *UpConv* layer.

value comparable to competing approaches.

Note that the winner and runup methods are both trained with extra Flickr2K dataset and take ground-truth HR images for strong supervision, which can be unrealistic for real-world application. Consequently, we adhere to use a teacher model for knowledge distillation. For that, we follow AGD, using the pretrained ESRGAN generator for the supervision. We can infer from Table 6 that TrilevelNAS being much lighter gives comparable results to the winning methods, which is desirable for mobile devices.

Ablation Study. Below studies the priority of the suggested use of sparsestmax, sorted sparsestmax, trilevel NAS, and train from search scheme against their direct competitors.

(a) Softmax vs. Sparsestmax We study the supernet optimization with softmax and sparsestmax combination, respectively, keeping the same ordering constraint strategy. From Table 7, we can observe that with weight ordering constraints, softmax and sparsestmax can prune one or two RiR blocks and have comparable performance in PSNR. However, sparsestmax converges to a more efficient model with a smaller model size and less flops consumption.

Method	Path	Params (M)	GFLOPS	PSNR Set5	PSNR Set14
BilevelNAS	[0,0,0,0,0,1,1]*	0.52	115.39	30.32	27.22
TrilevelNAS	[0,0,0,1,0,1]	**0.34**	117.39	30.34	27.29

Table 9. Quantitative results comparison of BilevelNAS (kernel- and cell-level) and TrilevelNAS (kernel-, cell- and network-level). Here $*$ indicates that the path is fixed rather than being searched.

(b) Sorted Sparsestmax The proposed sorted sparsestmax approach allow for a shallow and efficient SR model without much loss in performance. In Eq. 2 of the main paper, λ controls the trade-off between the major constraint and the ordering constraint. A large ordering constraint leads to a shallow network. We set λ as 0, 0.01, 0.1, respectively, and examine its effects. Table 8 shows the derived paths and the corresponding model performance. We see that without a weight ordering constraint ($\lambda = 0$), TrellisNAS prefers a full network path with 5 RiR blocks and 2 upsampling blocks. When we impose ordering constraint with λ=0.01, 0.1, the tail RiR block is seen to be pruned, which yields more efficient SR models with small model size and FLOPs without loss in performance.

(c) Bilevel vs. Trilevel NAS To show the necessity and advantage of introducing our network-level search in addition to cell- and kernel-level search, we compare the BilevelNAS, which has a fixed network-level design (i.e., path: [0, 0, 0, 0, 0, 1, 1]) with our TrilevelNAS. For BilevelNAS, we use AGD [2], and for a fair comparison, we replace the original softmax combination in AGD with sparsestmax combination. From Table 9, we can conclude that the additional network-level search in TrilevelNAS enables us to derive a lighter model with comparable performance. It can be inferred from the Table 9 statistics that the number of parameters in our approach is significantly smaller as compared to BilevelNAS, whereas the PSNR value is slightly better with comparable GFLOPS on Set5 and Set14.

(d) Train from Scratch vs. Train from Search Lastly, we study the effect of inheriting weights from the search phase. Fig. 1 shows the valid PSNR evolution of training from search and training from scratch, respectively. We can observe a clear advantage of inheriting weights from the search phase. With training from search strategy, we can converge to a better PSNR performance in fewer epochs.

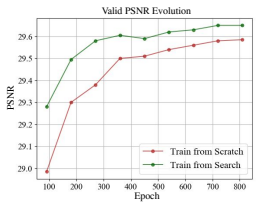

Figure 1. Train from Search vs Train from Scratch study.

Figure 2 shows the visual comparison of our approach against other competing methods. The last two columns in the figure show the results obtained using our TrilevelNAS

Figure 2. Visualization results of different SR models on Set5, Set14 and DIV2K valid set. Our TrilevelNAS-A and TrilevelNAS-B achieve comparable visual quality with very light model.

algorithm with TrilevelNAS-A and TrilevelNAS-B backbone design, respectively. Clearly, our method supplies a significantly lighter model and provides a super-resolved image that is perceptually as good as, if not better, than other approach results.

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 *British Machine Vision Conference (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 *International Conference on Machine Learning*, pages 3292–3303. PMLR, 2020.

[3] Yong Guo, Yongsheng Luo, Zhenhao He, Jin Huang, and Jian Chen. Hierarchical neural architecture search for single image super-resolution. *IEEE Signal Processing Letters*, 27:1255–1259, 2020.

[4] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 114–125, 2017.

[5] 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 *Proceedings of the European conference on computer vision (ECCV) workshops*, pages 0–0, 2018.

[6] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International conference on curves and surfaces*, pages 711–730. Springer, 2010.

[7] Kai Zhang, Martin Danelljan, Yawei Li, Radu Timofte, Jie Liu, Jie Tang, Gangshan Wu, Yu Zhu, Xiangyu He, Wenjie Xu, et al. Aim 2020 challenge on efficient super-resolution: Methods and results. In *European Conference on Computer Vision*, pages 5–40. Springer, 2020.