AutoSpace: Neural Architecture Search with Less Human Interference

Daquan Zhou[1], Xiaojie Jin [2], Xiaochen Lian[2], Linjie Yang[2], Yujing Xue[1], Qibin Hou[1]*, Jiashi Feng[1]
[1]National University of Singapore,[2]ByteDance US AI Lab
{zhoudaquan21, xjjin0731, lianxiaochen, yljatthu, andrewhoux}@gmail.com
xueyj14@outlook.com, elefjia@nus.edu.sg

1. Implementation details

Search space generation When generating the search space, we set the batch size to be 256 and mutation frequency to be 40 iterations. We use 40 epochs for warm-up and 120 epochs for the evolution process. The population size is set to be 1000 and the team size for tournament selection is scheduled to increase from 30 to 250 every 30 epochs. The reference graph stop epoch is set to be 60.

Supernet construction Following ProxylessNAS [1], we set the total number of layers of the supernet to be 21 with one 3x3 convolution layer at the begining as the head and one fully connected layer at the end as the classifier. Each layer of the supernet is initialized with 9 randomly sampled cells from the population. The supernet is updated during tournament selection as illustrated in Fig. 2 in the main paper and Fig. 1 in the supplementary material. During evolution, we sample 9 cells from the population for each layer via tournament selection and use those selected cells to generate mutations and then replace the cells in the supernet.

Search space evaluation To verify the superiority of our auto-learned search space, we select the widely used IRB based search space [15, 1, 16] as a strong baseline. Different from the one used in MNasNet [15], we do not add in Squeeze and Excite(SE) [5] modules in the search space as it has been recognized as a widely used tricks for improving the model performance. Instead, we add in SE modules manually after finding the optimal model in the search space and compare the results with other methods that also has SE modules added in the similar manner for a fair comparison. We use the searching algorithms proposed in [1] as a standard architecture searching methods on both the baseline search space and AutoSpace for the performance comparison in the ablation study. When comparing with the baseline search space, we use the curve estimates methods as adopted in [6, 22]: we search for a handful of models within AutoSpace and the baseline search space respectively and then tracing the curves of accuracy $vs.$ model complexity. The learned search space is considered as superior than the baseline search space if every point in the AutoSpace's curve is higher than the baseline search space's curve.

*Corresponding author.

2. Algorithm details

The details of the proposed differentiable evolutionary algorithm (EDA) is shown in Alg. 1 and Alg. 2. Alg. 1 shows the initialization process and Alg. 2 shows the details of the evolution process as illustrated in Fig. 2 in the main paper. In Alg. 1, we abuse the notion of G to simplify the notions. We use G^l to denote the population of cells for layer l and G^l_k to denote the graph topology of cell d^l_k.

Algorithm 1 Differentiable evolutionary algorithm for search space generation (Initialization)

Input: Number of layers L, length of the mutation window
for $l = 1$ **to** L **do**
  Initialize Population G^l
  S^l = tournament select(G^l, K)
  for $k = 1$ **to** K **do**
    d^l_k = S^l_k
    Initialize α^l_k = 0
  end for
end for

3. COCO Object Detection

We further compare the proposed method with ProxylessNAS [1] and MobileNet [4] models on object detection to explore the task transfer capability of the searched model from our generated search space. Following [14], we report the results on COCO dataset [8] using SSDLite framework[14, 10]. Our implementation is based on PyTorch. Following the same configurations in [14], the first two layers of SSDLite are connected to the last pointwise convolutional layer with output stride of 16 and 32, respectively. The rest of SSDLite layers are added on top of the

Table 1: Comparison with baseline backbone on COCO object detection and instance segmentation. 'Cls' denotes the Top-1 classification accuracy on ImageNet. mAP denotes the mean average precision for objection detection on COCO. We report the computational cost on ImageNet dataset with image size of 224 $\times$ 224 for consistency with previous experiments.

Method	Backbone	Param. (M)	MAdds (M)	Cls (%)	AP	AP$_{50}$	AP$_{75}$	AP$_S$	AP$_M$	AP$_L$
SSDLite320	MobileNet	4.2	569	70.6	22.2	-	-	-		
SSDLite320	ProxylessNAS	7.17	470	74.88	21.4	36	21.5	1.9	19.4	42.7
SSDLite320	Ours	4.61	415	75.83	24.5	40.3	25.2	3.1	24.8	46.7

Algorithm 2 Differentiable evolutionary algorithm for search space generation (Evolution)

Input: Number of layers L, length of the mutation window, number of epochs for training E, mutation frequency f
for $e = 1$ **to** $E - 1$ **do**
  for iteration i, Mini-batch data pair (X,Y) in data loader **do**
    Calculate probability for each path: $\vec{p}$ = softmax($\vec{\alpha}$)
    Sample an active path according to the calculated probability
    Forward pass the supernet
    Update weights parameters
    Update fitness score in the supernet
    if (i%f) == 0 **then**
      for $l = 1$ **to** L **do**
        for $k = 1$ **to** K **do**
          Update fitness score α via Eqn.(5)
        end for
        S^l = tournament select(G^l, K)
      end for
      for $k = 1$ **to** K **do**
        Local mutate(d_k^l)
        while Madds(d_k^l) $>$ Madds$_{\max}$ or $r_H(G_k^l$,$G^{ref}) > \tau$ **do**
          Local mutate(d_k^l)
        end while
      end for
      Update supernet according to Sec.(3.2)
    end if
  end for
end for
Select top-K cells in the population for each layer: S^l = $G^l_{:K}$
Output: generated search space $\{S^1, S^2, ..., S^L\}$

last convolutional layer with output stride of 32. During the training, the batch size is set to be 256 and the synchronized batch normalization is used. We use the cosine learning schedule with an initial learning rate of 0.01 and train the models with 8 GPUs for 200,000 iterations. More detailed settings can be found in [14, 10]. In Tab. 1, we compare the results of different models on COCO 2017 validation set. Besides the AP score, we also report results in terms of AP$_{50}$, AP$_{75}$, AP$_S$, AP$_M$, and AP$_L$, respectively. As can be seen, with less computation cost, SSDLite equipped with our searched backbone network achieves better results on all metrics compared to SSDLite with MobileNet and ProxylessNAS. Above experiments demonstrate that our auto-learned search space contains models that have strong generality for other vision tasks like object detection, not limited in image classification.

4. More analysis

4.1. Gradient compensation

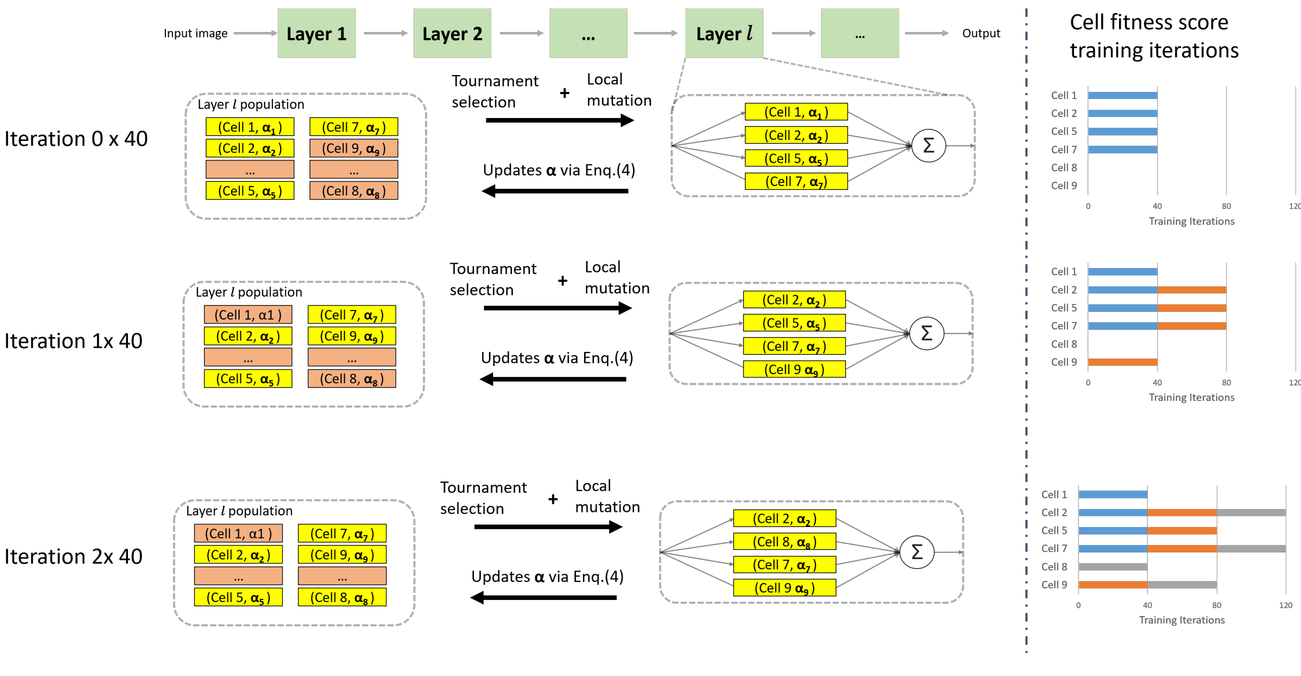
During evolution, in each layer, only K cells in the population are selected via tournament selection for a single round of fitness score updates as detailed in Alg. 2. This introduces an issue of imbalance gradient updates due to the randomness introduced in the tournament selection. We illustrate this imbalance in details in Fig. 1: we illustrate three rounds of tournament selections for the fitness score updates process. After three rounds of tournament selection, some of the cells are updated with more gradient steps. Motivated by [20], we propose to estimate the gradients of each cell based on the training iterations via Eqn. (4) in the main paper.

4.2. Model architecture for robustness analysis

As mentioned in Sec. 4.3 in the main paper, we run a set of experiments to study the robustness of the fitness scores to variations. We manually design three networks with the basic building blocks proposed in ResNet [3]. The detailed configurations for the three networks are shown in Tab. 2.

4.3. Weights sharing discussion

As shown in Fig. 7 in [19], the learned rankings of the models in the search space with a weight sharing based NAS algorithm may not be accurate and is fluctuating during the searching phases. However, the ranking difference among the candidate architectures are within a range. Thus, by selecting more candidate cell structures, the probability of containing the optimal solution in the searched space is higher than searching a single model. Thus, when exploring a large space, AutoSpace has higher probability to



...

Figure 1: Illustration of the gradient updates imbalance during tournament selection. In the figure, we show the fitness score updates for 120 training iterations with mutation frequency $f = 40$. On the right hand side of the figure, we plot the number of gradient updates steps for each cell in the population of layer l. As can be seen, the number of gradient updates steps are different for different cells' fitness score. To mitigate this gradient updates imbalance issue, we compensate the gradient update steps via Eqn.(5) in the main paper.

Table 2: Configurations of the three network architectures for fitness score robustness analysis. We use the the same basic building block as introduced in ResNet [3]. '1000-d fc' denotes the fully connected layer with 1000 output nodes. This is used as the classifier of the networks. The ground truth ranking of the three networks is Net1 $<$ Net2 $<$ Net3.

Layer name	Output size	Net 1	Net 2	Net 3
conv_1	112×112	$7 \times 7, 64$		
MaxPooling	56×56	3×3, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
	1×1	avgpool, 1000-d fc, softmax		

include the optimal solutions than the conventional NAS algorithms which search for a single model directly. As shown in Tab. 2 in the main paper, with less computational cost and memory footprint, the two step searching strategy used by AutoSpace achieves 2.1% top-1 classification accuracy improvement on ImageNet over the traditional one step searching algorithm.

4.4. Local mutation

To speed up the searching process, we implement a local mutation strategy as illustrated in Fig. 2: the starting and stopping nodes for mutation are only allowed to be in the same mutation window. The mutation window is a sliding window with a pre-defined length L.

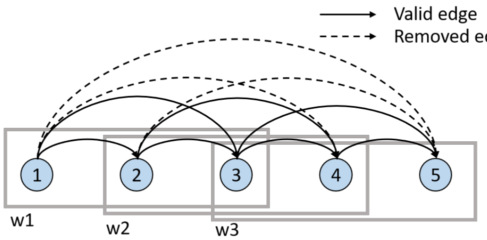


Figure 2: **Illustration on local mutations with the mutation window**. A 5-node fully connected DAG with a sliding mutation window of size 3 (covering 3 nodes). The solid line denotes the edges that are kept for mutation while the dotted lines are edges connecting the nodes outside the mutation window and are removed from mutation.

4.5. Architecture searching algorithms discussion

Generally, neural architecture searching algorithms can be divided into cell based searching methods [21, 12, 9, 7] and block based searching methods [1, 2, 15]. Cell based searching methods only search for several cell typologies and then stack them together to form the full network. Differently, the block-based searching algorithms allow different cell structures for different layers. Although with superior model performance, current block-based search spaces [15, 1] needs human interferes and requires expert knowledge on architecture properties. The search space are mainly based on the inverted residual block [14] with variable kernel sizes and expansion ratios. We also use block-wise searching algorithms in this work due to their high performance and efficiency. Differently, we resort to an differentiable evolutionary framework (DEA) to learn a set of search spaces. We only defines a set of basic operators and a fully connected DAG to minimize the manual efforts on graph topology design. In this manner, we are able to minimize the manual efforts while still enjoy the high searching efficiency of the block-wise searching algorithms.

In terms of the design choices in the search space, the searching algorithms can also be categorized into graph topology searching and structural hyper-parameter searching based on the definition of the search space. For graph topology searching, typically a general directed acyclic graph (DAG) is defined and the target is to search for the connection between nodes and the operations applied to each connection [11, 13, 12]. Structural hyper-parameter searching defines their search spaces based on a predefined graph structure, which search for structural hyper-parameters such as the kernel sizes, image resolutions and channel widths [18, 17, 15, 1]. Our method targets at graph topology searching and is orthogonal to filter structural hyper-parameter optimization. Once we found a promising graph topology, the optimal structural hyper-parameters can be applied to further enhance the performance.

References

[1] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[2] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[5] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[6] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[7] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

[8] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[9] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[11] 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*, pages 2736–2744, 2017.

[12] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[13] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[14] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018.

[15] 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*, pages 2820–2828, 2019.

[16] Mingxing Tan and Quoc V Le. Mixconv: Mixed depthwise convolutional kernels. *CoRR, abs/1907.09595*, 2019.

[17] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020.

[18] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, pages 10734–10742, 2019.

[19] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *ICLR*, 2020.

[20] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation for distributed deep learning. *arXiv preprint arXiv:1609.08326*, 2016.

[21] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[22] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.