2021 IEEE/CVF International Conference on Computer Vision (ICCV) | 978-1-6654-2812-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCV48922.2021.00039

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

Abstract

Current neural architecture search (NAS) algorithms still require expert knowledge and effort to design a search space for network construction. In this paper, we consider automating the search space design to minimize human interference, which however faces two challenges: the explosive complexity of the exploration space and the expensive computation cost to evaluate the quality of different search spaces. To solve them, we propose a novel differentiable evolutionary framework named AutoSpace*, which evolves the search space to an optimal one with following novel techniques: a differentiable fitness scoring function to efficiently evaluate the performance of cells and a reference architecture to speedup the evolution procedure and avoid falling into sub-optimal solutions. The framework is generic and compatible with additional computational constraints, making it feasible to learn specialized search spaces that fit different computational budgets. With the learned search space, the performance of recent NAS algorithms can be improved significantly compared with using previously manually designed spaces. Remarkably, the models generated from the new search space achieve 77.8% top-1 accuracy on ImageNet under the mobile setting (MAdds≤500M), outperforming previous SOTA EfficientNet-B0 by 0.7%.* `https://github.com/zhoudaquan/AutoSpace.git`.

1. Introduction

Recently neural architecture search (NAS) algorithms are popularly explored and applied, yielding several state-of-the-art (SOTA) deep neural network architectures [32, 31, 15, 34]. Applying a NAS algorithm typically comprises three steps: (1) designing a search space by specifying its elementary operators; (2) developing a searching algorithm to explore the space and select operators from it to build the candidate model; and (3) implementing an evaluation strategy to validate the performance of searched models. Extensive studies have been devoted to the latter two steps, i.e. searching algorithms and evaluation strategies [33, 2, 15, 31, 35, 34, 33, 12, 32].

To reduce the complexity of the search space to explore, the common practice of recent NAS algorithms is to leverage human prior knowledge to design smaller search spaces, most of which are based on well performing hand-crafted building blocks and their variants, *e.g.*, the inverted residual block [2, 32, 33, 31, 14] and the channel shuffling block [12]. On one hand, using restricted search spaces indeed enables NAS algorithms to enjoy higher efficiency;

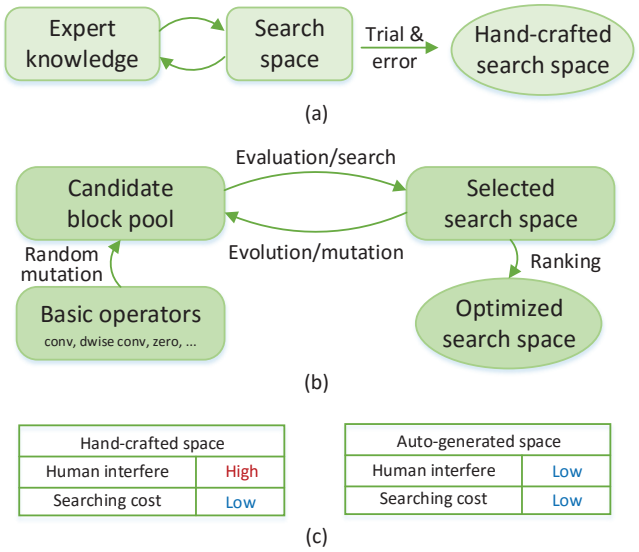

Figure 1: **Comparison of different search space construction schemes.** (a) Most existing NAS methods deploy handcrafted search spaces whose construction heavily relies on expertise and trial-and-error. (b) Our proposed method automatically builds and optimizes the search space by learning to form the basic operators into candidate building blocks and using an efficient approach to evolve and evaluate these building blocks. (c) Compared with existing schemes, our proposed one involves lower human effort and searching cost.

*Corresponding author.

978-1-6654-2812-5/21/$31.00 ©2021 IEEE
DOI 10.1109/ICCV48922.2021.00039

however on the other hand, due to such heavy human interference, the possibility to discover novel and better architectures is limited [9]. How to reduce human efforts in designing the search space and make the procedure automatic is still under-explored [27, 30].

In this work, we consider the open architecture space that consists of only basic operators with minimal human prior knowledge on cell graph typologies. Despite some early pilot investigations [41, 26], the progress is much hindered by several practical challenges. First of all, searching over the open space is unaffordably time consuming due to the combinatorial nature of the problem [27]. How to fast prune the unnecessary combinations of operators and lower the number of possible candidates to explore is thus necessary but remains an open question. Secondly, applying the RL-alike algorithms to search from scratch usually suffers poor exploration performance as they easily get stuck at sub-optimum. As a result, the model searched from the obtained space may perform no better than the ones from a manually designed search space.

In view of the above challenges, we then wonder whether it is possible to maximally reduce human interference in search space construction in a way such that the algorithms can effectively explore over the large space within an acceptable time and computation cost budget? To this end, we develop a novel differentiable AutoSpace framework to automatically evolve the full search space to an optimal subspace for the target applications. Our main insight is that a one-time searching for an optimal subspace at first and further performing NAS within it would provide higher exploration capability and searching efficiency at the same time, while avoiding getting stuck at the sub-optimum. Figure 1 illustrates the differences between our space evolving strategy and the previous ones for designing search spaces.

Concretely, AutoSpace starts with an open space that comprises all the possible combinations of basic operators (*e.g*. convolution, pooling, identity mapping). Then a differentiable evolutionary algorithm (DEA) is developed to evolve the search space to a subspace of high-quality cell structures. The subspace can then be adopted in any NAS algorithms seamlessly to find the optimal model architectures. To reduce the potential high cost of subspace searching and evaluation, AutoSpace introduces a couple of new techniques to remove the redundant cell structures and improve the parallelism of the evolution process as detailed in Section 3.

We verify the superiority of the search space from AutoSpace on the ImageNet [18] dataset. With the same NAS algorithm, AutoSpace provides much more accurate models than the previous SOTA models searched from the manually designed spaces. Besides, by combining the cell structures discovered with AutoSpace to EfficientNet, we successfully improve the Top-1 accuracy on ImageNet by 0.7%.

In summary, we make the following contributions:

- We are among the first to explore the automatic learning of search spaces in NAS algorithms. Compared to searching for network architectures on a manually designed search space, searching for a search space is more challenging due to the larger exploration space/computational complexity.
- We propose a novel learning framework that takes advantage of both the high exploration capability of evolutionary algorithms and the high optimization efficiency of gradient descend methods. The proposed framework can be seamlessly integrated with popular neural architecture searching algorithms.
- By directly replacing the original search space with the learned search space, the top-1 classification accuracy of previous SOTA NAS algorithms can be improved significantly at different model sizes. Specifically, at 200M MAdds, the performance of the searched model is improved by more than 1.8% on ImageNet.

2. Related Work

Most of the previous neural architecture search (NAS) works focus on better searching algorithms while the design of the search space is less studied. This is primarily due to the unaffordable computation cost for automatically searching a search space. For example, a recent method of evaluating search spaces uses empirical distribution function (EDF) [24] where each evaluation iteration takes 25k GPU hours[1] even on a small dataset of CIFAR10 [17].

Thus, most of the NAS algorithms use manually designed search spaces. An early work NAS-RL [41] defines its space via macro and micro architectures. The macro architecture is used for connections between layers and the micro architecture space includes the structural hyper-parameters for each filter within a layer. Such a huge space is extremely hard or even impractical to enumerate each candidate within it for evaluation. Thereafter, most NAS methods change to adopt size-reduced search spaces to improve their searching efficiency. For example, the cell based methods [42, 25, 21, 20, 23, 36, 19, 3] achieve affordable cost by only searching two types of cell structures, *i.e*., a normal cell and a reduction cell, which are shared across all the layers for constructing a neural network. However, those methods can only search on a small proxy dataset due to high memory cost. Most recent NAS algorithms [31, 33, 32, 2, 15, 35, 7, 34, 6, 40] employ well handcrafted inverted residual blocks (IRB) with varying kernel sizes and expansion ratios as search space candidates. Though offering good efficiency, such a constrained

[1]The work [24] evaluates the distribution on 50k models and the reported training speed is 2 models per GPU hour.

Table 1: **Search space design choices comparison.** The number of design choices in AutoSpace eclipse the previous algorithms' search space. AutoSpace automatically finds an optimized subspace of comparable size to [2] and [31]. "Layer Variety" indicates if the searching algorithms allow different cell structures at different model layers; "# Models (log)" denotes the log10 value of the total number of architectures included in the search space for a 21-layer model.

Algo.	Layer Variety	Search on ImageNet	Space Design	# Models (log)
DARTS [21]	✗	✗	Manual	2.38
ENAS [23]	✗	✗	Manual	3.70
PNAS [20]	✗	✗	Manual	5.74
Amoeba [26]	✗	✗	Manual	5.74
NASNet [42]	✗	✗	Manual	7.85
MNasNet [31]	✓	✗	Manual	52.72
SPOS [12]	✓	✓	Manual	12.64
ProxylessNAS [2]	✓	✓	Manual	17.74
AutoSpace (ours)	✓	✓	Auto	104.37

search space severely limits the exploration capability of NAS algorithms to search for more powerful network structures.

In this work, we propose a simple and efficient method for automating the search space design in two steps. First, we conduct a one-time searching for the optimal subspace from a full search space on the target dataset. Using the search space obtained, a typical NAS algorithm can be applied to search for the final network architecture. In this way, we can not only minimize the human interference in the search space design but also improve the network performance by searching in a better space. A comparison of the design choices of our proposed method and previous SOTAs are listed in Table 1. More discussions are deferred to supplementary materials.

3. Method

3.1. Problem Formulation

We consider the NAS problem of searching for a model architecture N from a search space S that consists of multiple basic building blocks (cells) d. Different from previous NAS work using a manually pre-defined search space, we aim to learn a proper search space automatically. We formulate the neural architecture search as a two-phase problem: first searching for an optimized search space and then searching network architectures using the optimized search space.

Formally, given an open search space S with minimal human prior knowledge on the network architecture, we aim at finding a subset of the full space (called subspace) $S^{\star}_{\text{sub}} \subset S$ which contains cells with optimized structures such that the constructed model architecture can achieve maximal accuracy Acc on the target dataset $\mathcal{D}$ at affordable computation cost (in MAdds). The objective can be formulated as a bi-level optimization problem:

$$\begin{aligned} S^{\star}_{\text{sub}} &= \arg\max_{S_{\text{sub}} \subset S} \text{Acc}(N(d, S_{\text{sub}}), \mathcal{D}), \\ \text{s.t.} \quad d &= \arg\max_{d \in S_{\text{sub}}} \text{Acc}(N(d, S_{\text{sub}}), \mathcal{D}), \\ & \text{MAdds}(d_i) < \text{MAdds}_{\max}, \forall d_i, \end{aligned} \tag{1}$$

where $N(d, S_{\text{sub}})$ denotes the network searched from the subspace S_{sub} and $d = \{d_1, d_2, \ldots\}$ denotes the set of selected cells for all the layers. $\text{MAdds}_{\max}$ denotes the upper limit of the allowed MAdds of cell d_i. In particular, we consider searching for different subspaces at different layers of an L-layer model architecture, *i.e.*, $S_{\text{sub}} = \prod_{i=1}^{L} S^i$ and $d_i \in S^i (i = 1, \ldots, L)$. In the following, we use d^l_k to denote the k^{th} candidate cell in the generated subspace of layer l, with $d^l_k \in S^l, |S^l| = K$.

Following [42, 21], each cell in the subspace is represented by a directed acyclic graph (DAG) G. Each node x_i of G is the output feature representation from a certain edge and each directed edge G_{ij} is associated with some operation that transforms the input x_i to x_j. In particular, o denotes the set of basic operators defined below and $o(G_{ij})$ denotes the selected operation between node i and j.

Basic operators To minimize the human prior knowledge required on designing the network architecture, we do not specify any constraints on the node connection topology for the cells except for specifying two nodes as input and output of its DAG respectively. The edge connections are learned with our proposed learning framework as detailed in Section 3.2. For each edge, similar to most previous works [2, 12, 35], we consider five basic operations: 1×1 convolution, 3×3 convolution, depth-wise convolution, identity mapping, and the zero operation. The input and output nodes are used as dimension adjustment nodes and following [35, 2], the ratios between the input channels and the output channels $r \in \{1, 3, 6\}$ are learned through our proposed method which will be detailed in Section 3.2.1. The aggregation function o_f over the output features from different operators is selected from {addition, dot product}.

3.2. AutoSpace Method

Our AutoSpace learns the subspace in three steps as illustrated in Figure 2. Following previous works [31, 2], we construct an L-layer classification super-network (supernet) with a 3×3 convolutional layer as the head and one fully-connected layer for score prediction. Each layer in between has K parallel paths and each path is associated with a candidate cell d^l_k in the corresponding layer-wise search space S^l. The population of cells with different structures will be updated and evaluated via differentiable evolutionary algorithm (DEA). After the evolution process, for each layer, the

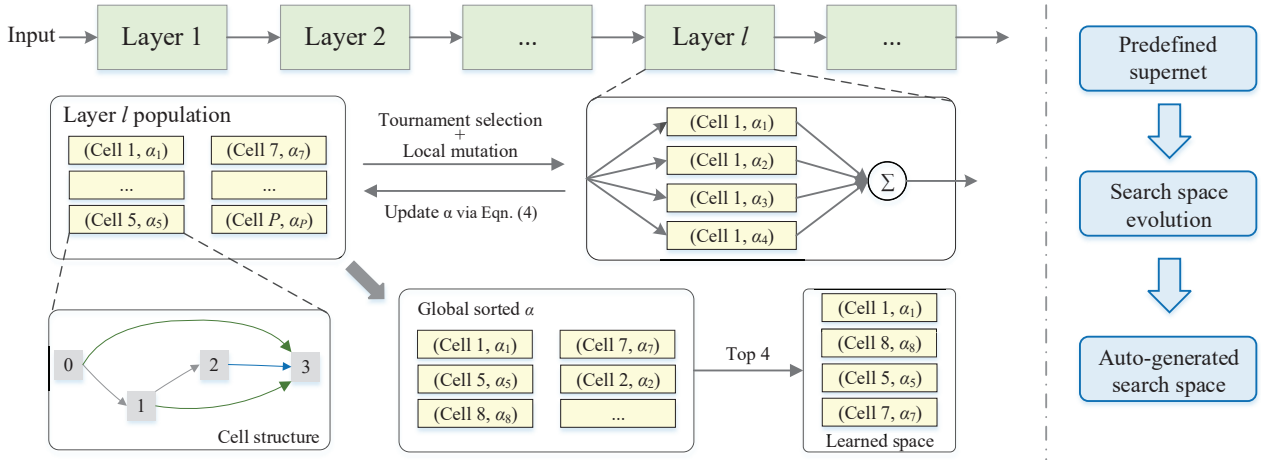

Figure 2: **Illustration on the search space generation process of AutoSpace**. Given a target network with a specified layer number and channel dimension, AutoSpace aims to learn an optimal layer-wise search space to be used in following NAS algorithms. The entire process is performed by iterative sampling-and-update. For each layer l, we first sample K cells ($K = 4$ in this example) from a randomly initialized population through tournament selection [11, 10]. α is the fitness score for each cell. Then a layer is represented by stacking K cells in parallel. The layer output is calculated as the weighted sum of the outputs of each single cell whthin the layer (ref. Eqn. (3)). In this way, the network can be trained on the target dataset with α updated efficiently via gradient back-propagation (ref. Eqn. (4)). In the end of each iteration, a cell's fitness score is updated via Eqn. (5). After the network converges, the top K cells with the largest fitness scores in each layer are output as the final search space.

top-K performing cells from the population are selected to form the search space S^l.

3.2.1 Automatic search space generation

A typical evolutionary algorithm consists of three steps [1]. The first step is to generate a population of cells with different structures (*i.e.*, different DAG connection topologies). We maintain a population of cells, denoted as G^l, for each layer of the constructed supernet. The second step is to apply a scoring function to evaluate the individual cell via tournament selection, where the winner cells with the largest K fitness scores are allowed to generate off-springs via mutation. The third step is to generate new off-springs via mutation between the selected cells to enrich and improve the population. The last two steps are performed in an iterative manner. At the end of evolution, the top K performing cells from the population will be used as the layer-wise subspace S^l.

However, the above EA method cannot be directly used to solve Eqn. (1) and search for a subspace, due to the following challenges. First, a large redundancy exits in the full space (population), incurring high computation overhead. Secondly, a one-by-one evaluation for the subspace is extremely time consuming. Thirdly, an EA process usually starts from scratch and is slow to converge. To solve these challenges, we propose to leverage a reference DAG to assist in removing the redundant cells, speedup the convergence, and increase the parallelism for subspace evaluation.

Reference DAG Instead of evolving from scratch as convention, we propose to speedup the evolution process by inheriting the prior knowledge of a reference graph set at initialization. We use G^{ref} to denote the DAG of a verified well performing cell structure, and use it as the reference DAG. Hamming distance is harnessed as a measure of similarity between DAGs [8]. After each mutation, we calculate the hamming distance between the generated DAG and the reference DAG as follows:

$$r_H(G, G^{ref}) = \sum_{i,j} \frac{|G_{i,j} - {G^{ref}}_{i,j}|}{V(V-1)}. \tag{2}$$

The mutation process will repeat until the distance is below the threshold τ. Note that this reference graph regularization is only applied for the first half iterations of the entire training process to avoid adversely affecting learning of a better search space.

3.2.2 Differentiable scoring function

As our approach allows different layers to select different cell structures, previous cell-based evaluation methods [26, 25] cannot be applied here due to the explosive increase in size of the search space, and there is no one-to-one correspondence between network performance and quality of the cell structure. To evaluate different cell structures, we propose to learn the fitness score α_k^l for each individual cell structure d_k^l via gradient optimization to make the search process fully differentiable.

As illustrated in Fig. 2, we maintain a layer-wise population G^l for layer l of the supernet. Each layer is initialized with K randomly sampled cell structures $\{G_1^l, G_2^l, ..., G_K^l\}$ from the corresponding population G^l. The output of each sampled cell structure from the population will be weighted by its fitness score in the constructed supernet:

$$f_{S^l} = \sum_{k=1}^{K} p_k d_k(x) = \sum_{k=1}^{K} \frac{\alpha_k^l}{\sum_j \alpha_j^l} d_k(x), \tag{3}$$

where p_k is the weight for each cell structure d_k in the supernet and f_{S^l} is evaluated by computing its classification loss over the provided training dataset $\mathcal{D}$. In this manner, multiple individuals at different layers can be evaluated concurrently. To save the computation memory, we employ the binary gating function as proposed in [2] when learning the fitness score for each selected cell.

The fitness scores for the selected K cells in certain layer's population will be updated by gradient back-propagation when training the supernet to minimize the cross-entropy loss on the target dataset. To alleviate the imbalanced gradient updates on the fitness scores in each cell population due to the random sampling in tournament selection, we compensate the gradient update step of the fitness score via a scaling factor based on the training iteration number:

$$\frac{\partial L}{\partial \alpha_i^l} \approx \sum_{k=1}^{K} \frac{\partial L}{\partial g_k} p_k (\delta_{i,k} - p_i) \frac{n(d_i^l)}{n'(d_i^l)}, \tag{4}$$

where g_k is the binary gates as introduced in [2] and $\delta_{i,k} = 1$ if $i = k$ and 0 otherwise. $n(d_i^l)$ denotes the accumulative training iteration number in the supernet and $n'(d_i^l)$ denotes that for d_i^l in the whole population. More discussions can be found in the supplementary materials.

After training the supernet for a few iterations, the updated fitness scores in the supernet, denoted as $\alpha^\star$, will be used to update those of corresponding cell structures in the population as below:

$$\alpha_k^{l\,(t)} = \epsilon \alpha_k^{l\,\star} + (1 - \epsilon) \alpha_k^{l\,(t-1)}, \tag{5}$$

where $\epsilon \in [0, 1]$ is the momentum hyperparameter for updating the fitness score, and $\alpha^{(t-1)}$ denotes the old fitness score recorded in the population before the updates.

The supernet will be re-generated every f iterations. When sampling the new search space, the fitness scores and the cell structures are sampled in pairs. The fitness scores for each selected cell structure in each population will be updated iteratively via Eqn. (4) and Eqn. (5) until the supernet converges. After the evolution, the top-K cell structures at each layer will be selected to form the searched space. In this way, each training of the supernet will evaluate K^L network structures concurrently and thus the evolution process is sped up by K^L times compared to sequential evaluation.

3.3. Architectures search on the generated space

Our search space searching process only needs to be run once for a target dataset and the generated search space can be used by any searching algorithms. To verify the effectiveness of AutoSpace, we allow different cell structures for different layers and directly search on ImageNet dataset to remove the transfer-ability issue. To speed up the training process and save computation memory, we implement the weights sharing scheme as introduced in ENAS [23] such that each time we re-build the supernet, the weights are inherited from previous runs. We select three previous SOTA architecture searching algorithms based on gradient optimization (G), reinforcement learning (RL) and random sampling (RS) methods respectively from ProxylessNAS (G and RL) [2] and SPOS (RS) [12]. For both searching algorithms, we add MAdds as a regualrization term to the loss function following previous NAS works [2, 31]:

$$Loss = Loss_{CE} + \lambda \text{MAdds}(N), \tag{6}$$

where $Loss_{CE}$ is the cross entropy loss and MAdds(N) is the number of MAdds operation times in the selected network. As will be shown in experiments, under all the three different algorithms, our auto-generated search space outperforms manually-designed ones significantly.

4. Experiments

4.1. Setup

Searching algorithm The learned search space from AutoSpace can be applied to various NAS algorithms. To evaluate the searched space, we use the differentiable searching method ProxylessNAS [2] to search model architectures because it allows layer variety and supports direct model searching on large datasets with high computation efficiency. Following [2, 31], we use MAdds-based regularization to control the computation cost of the searched models. Besides, to verify the generalization ability of our learned search space in different NAS algorithms, we also evaluate its performance with reinforcement learning based [2] and random sampling based [12] NAS algorithms. More details can be found in our supplementary material.

Model evaluation We train the searched model on the ImageNet dataset [28] with an initial learning rate of 0.1 and cosine learning rate decay policy for 250 epochs. The batch size and weight decay are set to 512 and $1e{-}4$, respectively. We use 10 epochs for learning rate warmup. Following [2], we do not apply extra data augmentation, like AutoAugment [4, 5] for a fair comparison. When comparing with state-of-the-art efficient models, we follow the same set of training hyper-parameters as them and use RandAugment [5] with default hyper-parameters during training.

Manually designed search spaces for comparison We choose the inverted residual block (IRB) [29] based search space as the baseline for comparison, which has been adopted by many recent SOTA NAS algorithms [2, 31, 33, 35]. It includes six variants of IRB, *i.e.*, IRB with kernel sizes of 3, 5, 7 and expansion ratios of 3, 6 respectively. It also includes a zero operator for skipping certain layers. Throughout the experiments, we refer to this baseline search space as "manually-designed".

4.2. Ablation Study

Effectiveness of AutoSpace We first validate our proposed strategy, *i.e.*, generating a subspace from the open space first and then searching for models within the subspace, via comparison with the following two popular NAS strategies. The first is to use the evolutionary algorithm to directly search for *a model in the same open space* as ours, which consists of all possible combinations of basic operators; the other is to search for *a model in the manually-designed search space*.

The performance of the searched models using the above two strategies on ImageNet are summarized in Table 2. As can be seen, the model searched by our method outperforms the baseline models significantly in terms of both Top-1 accuracy and computational efficiency, which delivers two interesting findings. First, the superiority over the direct searching strategy demonstrates that our proposed strategy can generate a high-quality search space that enables the subsequent NAS algorithm to search for better network architectures. Second, compared with the handcrafted search space, our search space is optimized jointly with the model architecture on the target dataset. This allows our method to learn better task-specific models and benefit from the end-to-end training pipeline.

Table 2: **Performance comparison of different searching strategies.** 'S.S' denotes search space. 'Full' denotes full search space as ours. 'Manual' denotes manually-designed search space.

S.S.	Search Algo.	Param. (M)	MAdds (M)	Acc. (%)
Full	EA	4.2	480	73.7
Manual	RL	7.2	470	74.9
AutoSpace	RL	4.6	**415**	**75.8**

Comparison to handcrafted search space To comprehensively compare our learned search space and the handcrafted one, we first run AutoSpace to generate layer-wise search spaces on ImageNet. Then, we run the ProxylessNAS gradient-based (G) searching algorithms [2] with different MAdds regularizations, obtaining models of sizes spanning 100-200M (100M+), 300-400M (300M+), 400-500M (400M+), and 500-700M (500M+), respectively.

From the results in Table 3, for each model size constraint, the searched model from our auto-generated search space always outperforms the one from the manually-designed space by a large margin with the same searching algorithm. We argue that the advantage of our method is that our auto-generated search space is optimized for each layer on the target dataset, which is more likely to contain superior model structures than the manually designed search space that includes only identical operators across layers. Above results evidently validate the advantage of AutoSpace over the manually-designed search spaces.

Table 3: **Comparison among the searched models based on our search space and the manually-designed ('manual') search space used by ProxylessNAS.** 'Acc.' denotes the top-1 classification accuracy on ImageNet.

Budget	Search Space	Param. (M)	MAdds (M)	Acc. (%)
100M+ MAdds	Manual [2]	3.3	190	69.2
	Ours	3.3	175	71.1
300M+ MAdds	Manual [2]	3.7	320	74.1
	Ours	4.3	340	75.3
400M+ MAdds	Manual [2]	5.3	465	74.8
	Ours	4.7	430	75.7
500M+ MAdds	Manual [2]	6.9	590	76.6
	Ours	6.4	570	77.0
	Ours	7.2	650	77.2

Applications to different searching algorithms To verify the generalizability of the search space learned by AutoSpace, we run different searching algorithms on it as well as on the baseline search space to compare the resulting models. We choose two representative searching algorithms which are based on reinforcement learning (RL) and random sampling (RS) respectively. For the RL searching algorithm, we choose the widely-used single path sampling method proposed in [2]. For the RS algorithm, we use [12]. The results are shown in Table 4. Under different computation budgets, the model searched using search spaces from AutoSpace significantly outperforms the ones from the baseline search space. For example, in the group with 300M+ Madds, our model outperforms competing models searched by RL and RS by 0.8% and 0.4% respectively. These results further verify the superiority of our auto-learned search space.

4.3. Algorithm Analysis

Analysis on search space size We use 6 nodes for each DAG including the two IO nodes. The IO nodes are only used to adjust the dimensions. Thus, there are totally 6 edge connections within each DAG for evolution. With the above definition, for an L layer network, the total number of architectures included in each DAG is $5^6 \times 3 \times 2 = 93750$. As we allow a variety for different layers, the total number of

Table 4: **Performance of our auto-generated search space with different searching algorithms.** 'Manual' denotes the manually-designed search space. 'RL' denotes reinforcement learning based algorithms as used in ProxylessNAS [2]. 'RS' denotes the random searching algorithm as used in SPOS [12]. 'Cls' denotes the Top-1 classification accuracy on ImageNet.

Group	Search Algo.	Param. (M)	MAdds (M)	Cls (%)
300M+	Manual-RL	4.1	330	74.3
	Ours-RL	4.0	360	75.1
	Manual-RS	4.0	350	73.5
	Ours-RS	4.2	360	73.9
400M+	Manual-RL	7.2	470	74.9
	Ours-RL	4.6	415	75.8
	Manual-RS	4.3	420	74.4
	Ours-RS	4.6	430	74.9

networks that could be formed via those DAG is 93750^L. A typical value of L is 21 [2, 31], and using it as an example, the size of the full search space of AutoSpace is $2^{325}\times$ larger than the DARTS search space. A more detailed comparison is shown in Table 1 in the main paper.

Reference graph speedups convergence We further study the effects of using reference graph (described in Section 3.2) for search space optimization. We use the reference graph regularization for the first 60 epochs and disable it for the rest of the evolution iterations. The implementation of reference graph could speed up the convergence of the evolution of the supernet significantly as shown in Fig. 3. The overall searching time is reduced by $5.2\times$ as detailed in Tab. 5.

Search Cost Analysis For the search space evolution process, we use 8 GPUs and run for 5 days. As we evaluate 9 cells for each fitness score update in Eqn. (5), there are in total 9^{21} different numbers of models in the constructed supernet. Thus, this evaluation approach is roughly $9^{21}\times$ faster than the conventional EA process with the same search space size. More importantly, the searching on the large space only needs to be run once on a given dataset and computation budget. A detailed comparison of searching cost with other NAS algorithms is shown in Table 5.

Robustness to variations of fitness scores Each time we update the layerwise subspace with the differentiable evolutionary algorithm (DEA), the fitness score for each cell will be updated together with the cell structures based on the records in the population. Thus, during training, it is expected that the fitness score can accurately reflect the relative advantages among selected cell structures. To study the ranking accuracy of the differentiable fitness scoring function, following previous works [38, 37], we measure the stability of the generated rankings of three pre-defined networks with our proposed differentiable scoring functions

Table 5: **Searching cost analysis on ImageNet.** 'AutoSpace-G' denotes the searching cost on our learned search space with gradient optimization method proposed in [2]. The search space design only needs to be run once on the target dataset.

Methods	Search Space Design Cost (GPU hours)	Searching Cost (GPU hours)
NasNet-A [42]	Manual	48,000
MNasNet [31]	Manual	40,000
AmoebaNet-A [25]	Manual	75,600
ProxylessNAS [2]	Manual	200
AutoSpace-G (w/o ref)	5000	200
AutoSpace-G (w/ ref)	960	200

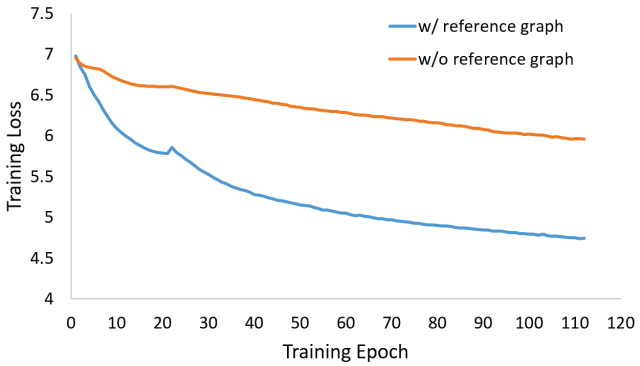

Figure 3: The training loss curve of the evolution process with and without reference DAG (blue vs. red line). The convergence is faster with the reference graph.

Table 6: **Fitness score robustness of the differentiable scoring function.** We list the initial fitness scores for the three predefined networks. 'Epochs' denotes the starting number of epochs for the fitness score, reflecting the correct ranking of the networks.

S/N	$\alpha_{\text{Net 1}}$ $(\cdot 10^{-3})$	$\alpha_{\text{Net 2}}$ $(\cdot 10^{-3})$	$\alpha_{\text{Net 3}}$ $(\cdot 10^{-3})$	Epochs
1.	-0.296	2.670	-0.141	5
2.	0.023	-0.577	0.705	3
3.	0.426	-0.752	-1.180	4
4.	-1.950	-0.061	-0.334	5
5.	0.304	1.220	0.774	6

and use this stability as a measurement of the robustness. Specifically, we manually design three networks with the basic building blocks proposed in ResNet [13], denoted as Net1, Net2, and Net3. The three networks are using the same building block with different depth and the detailed configurations for these three networks can be found in our supplementary material. We first train the three networks on ImageNet to get the ground truth rankings of their classification accuracy: Net1 $<$ Net2 $<$ Net3. Then, we use the proposed differentiable scoring function as introduced in Section 3.2 to learn the fitness scores with different initial values. The results are shown in Table 6. Based on

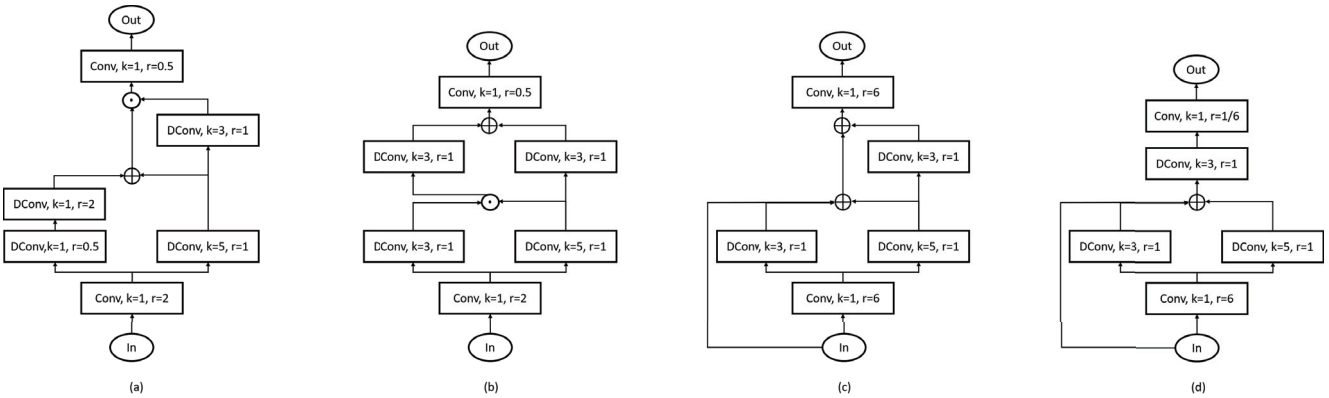

Figure 4: **Illustration of the learned cell structures.** We list four most frequently selected cell structures learned via AutoSpace. 'DConv' denotes depthwise convolution; 'k' denotes the kernel size; 'r' denotes the ratio between the output channel and the input channel of a convolution layer. '$\odot$' denotes dot product of two feature maps and '$\oplus$' denotes element-wise addition of two feature maps. Among the four cell structures, (d) is selected the most often.

Table 7: **Comparison of our proposed AutoSpace with previous SOTA NAS algorithms.** Our model is obtained by applying ProxylessNAS onto the auto-generated search space. For fair comparison, we split the methods into two groups, *i.e.*, without SE [16] or with SE (labeled with '†'). '$\star$' denotes the EfficientNet-b0 model improved with our searched cell structure as shown in Fig. 4(d).

Method	Param. (M)	MAdds (G)	Top-1
MobileNetV2 [29]	3.5	300	72.0
ShuffleNetV2-1.5 [22]	-	299	72.6
DARTS [21]	4.7	574	73.3
FBNet-C [35]	5.5	375	74.9
ProxylessNAS-M [2]	4.1	330	74.4
PNAS [20]	5.1	588	72.7
AmoebaNet-A [25]	5.1	555	74.5
ChamNet-B [7]	5.2	323	73.8
MNasNet-B1 [31]	4.4	326	74.2
MobileNeXt [39]	3.5	300	74.0
Ours	4.3	340	75.3
MobileNeXt† [39]	6.1	590	76.1
MNasNet-A3† [31]	5.2	403	76.7
EfficientNet-B0† [32]	5.3	390	77.1
MixNet-M† [33]	5.0	360	77.0
Ours†	5.7	380	77.5
Ours*	5.2	420	77.8

the results, with different initial values, the proposed differentiable scoring function is always able to learn the correct rankings of the three networks' fitness scores within 6 epochs.

4.4. Comparison to the State-of-the-Arts

Classification on ImageNet We follow ProxylessNAS [2] to use the single path searching algorithm for searching model architectures from the auto-learned search space. Then, we compare our model with other models obtained by manual design or popular NAS algorithms, the results of which are shown in Table 7. For fair comparison, we divide compared models into two groups: models without SE modules [16] and models with SE and extra data augmentation techniques. In each group, our model outperforms other models in terms of top-1 accuracy with less or comparable computation cost. As aforementioned, we believe this improvement comes from the high-quality cell structures learned with our proposed AutoSpace method. To further verify this, we summarize common characteristics appeared in the learned search space, as illustrated in Fig. 4. Among all the learned cell structures, we observe that (d) is the most frequently selected. To further investigate the performance of structure (d), we replace all the DEA learned cells with (d) and add in SE modules [16] in the same manner as EfficientNet-B0 [32]. Surprisingly, without using excessive data augmentation methods such as AutoAugmentation [4], the modified model achieves 77.8% Top-1 classification accuracy on ImageNet [18], outperforming the EfficientNet-B0 model by 0.7%.

5. Conclusion

In this work, we present the first affordable approach to generate search spaces for NAS algorithms automatically, which targets at alleviating human effort in search space design. The search space generation is based on a substantially improved evolutionary algorithm with three novel techniques to enhance the efficiency. Extensive experiments show that direct replacement of the manually-crafted search space with our auto-generated one could significantly improve performance of searched models in image classification. We believe the proposed method will motivate the research along the line of automatic search space design for NAS algorithms.

References

[1] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

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

[3] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *arXiv preprint arXiv:2102.11535*, 2021.

[4] ED Cubuk, B Zoph, D Mane, V Vasudevan, and QV Le. Autoaugment: Learning augmentation policies from data. arxiv 2018. *arXiv preprint arXiv:1805.09501*.

[5] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[6] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. Fbnetv3: Joint architecture-recipe search using neural acquisition function. *arXiv preprint arXiv:2006.02049*, 2020.

[7] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pages 11398–11407, 2019.

[8] Claire Donnat and Susan Holmes. Tracking network dynamics: A survey of distances and similarity metrics. *arXiv preprint arXiv:1801.07351*, 2018.

[9] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[10] Yongsheng Fang and Jun Li. A review of tournament selection in genetic programming. In *International Symposium on Intelligence Computation and Applications*, pages 181–192. Springer, 2010.

[11] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier, 1991.

[12] 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.

[13] 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.

[14] Qibin Hou, Daquan Zhou, and Jiashi Feng. Coordinate attention for efficient mobile network design. *arXiv preprint arXiv:2103.02907*, 2021.

[15] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[16] 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.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] 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.

[20] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, pages 19–34, 2018.

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

[22] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, pages 116–131, 2018.

[23] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[24] Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1882–1890, 2019.

[25] 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.

[26] 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.

[27] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

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

[30] Mrunal S Surve and Rahul L Paikrao. An approach for generating utility pattern by reducing search space. *INTERNATIONAL JOURNAL*, 3(8), 2018.

[31] 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.

[32] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

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

[34] 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.

[35] 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.

[36] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. *arXiv preprint arXiv:1907.05737*, 2019.

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

[38] Yuge Zhang, Zejun Lin, Junyang Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. Deeper insights into weight sharing in neural architecture search. *arXiv preprint arXiv:2001.01431*, 2020.

[39] Daquan Zhou, Qibin Hou, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. Rethinking bottleneck structure for efficient mobile network design. In *Eur. Conf. Comput. Vis.*, 2020.

[40] Daquan Zhou, Xiaojie Jin, Qibin Hou, Kaixin Wang, Jianchao Yang, and Jiashi Feng. Neural epitome search for architecture-agnostic network compression. *arXiv preprint arXiv:1907.05642*, 2019.

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

[42] 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.