AdaSearch: Many-to-One Unified Neural Architecture Search via A Smooth Curriculum

Chunhui Zhang*, Yongyuan Liang*, Yifan Jiang*

Brandeis University, Sun Yat-sen University, University of Texas at Austin
chunhuizhang@brandeis.edu, liangyy58@mail2.sysu.edu.cn, yifanjiang97@utexas.edu

Abstract

The cost of Neural Architecture Search (NAS) has been largely reduced, thanks to one-shot SuperNet methods that exploit the weight-sharing strategy as the proxy. However, challenges remain especially when we target for simultaneously deriving many high-performance, yet diverse models that can meet various resource constraints, by training just one SuperNet. If we treat SuperNet as a large *ensemble* of all those candidate neural networks with *highly varied complexity levels*, a **fundamental question** then arises: *how might we train those neural networks all (close) to their optimal performance, with their weights entangled due to sharing, under one unified optimization procedure?* To tackle this question, we propose **AdaSearch**, whose idea is inspired from *curriculum learning*. Towards our end goal of training all neural networks well, we start by focusing our SuperNet training on a "simple" subset of them, i.e., training a shallow SuperNet that consists of only low-complexity neural networks. We then iteratively grow and train the SuperNet, so that higher-complexity neural networks are gradually included and taken care of. For smoothly transiting the SuperNet curriculum, we also develop a key enabling technique called *SuperNet2SuperNet*, that is, using distillation to initialize the deeper SuperNet by inheriting knowledge from the shallower one each time. AdaSearch demonstrates state-of-the-art accuracy-efficiency trade-offs on ImageNet, while significantly trimming down search GPU hours and CO_2 emission by reducing **N** search times to **1** procedure. All our codes and pre-trained models will be publicly released upon paper acceptance.

Introduction

Improving neural architectures has been a key tenet in deep learning. From AlexNet (Krizhevsky, Sutskever, and Hinton 2012), VGG (Simonyan and Zisserman 2014), to ResNet (He et al. 2016): tremendous human efforts are spent on designing better architectures. The latest exciting wave is led by Neural Architecture Search (NAS) (Pham et al. 2018). Not only we see new searched models keep boosting accuracy, but also they are often more compact and efficient too. Early NAS works exploit reinforcement learning or evolutionary search, whose search costs are often prohibitively high (Zoph and Le 2016). Lately, the one-shot differentiable search (Liu, Simonyan, and Yang 2018; Dong and Yang 2019) introduce the continuous relaxation of the architecture space, allowing efficient search using gradient descent and drastically reducing the number of architecture evaluations required. Their demonstrated high efficiency in exploring a large search space, as well as high performance achieved, make them the current top workhorse in NAS.

Since the deployed platforms have various resource constraints (latency, memory, or energy), compared to discovering one single architecture, it is practically more desirable to obtain a series of models, that span over the accuracy-efficiency spectrum with different trade-offs. That constitutes new challenges for network design. The most naive solution is to design or search for a cell structure and repeatedly stack it to different depths, such as the ResNet family (He et al. 2016). An alternative goes for multiplying the channel width by different ratios, such as the MobileNet family (Howard et al. 2017). This design challenge also motivates the adoption of NAS techniques: for example, the EfficientNet family (Tan and Le 2019) searches for compound scaling ratios across different dimensions, to scale up a base architecture.

From the above model family studies, one meaningful inspiration to draw is that larger and smaller neural networks can share certain design patterns. Meanwhile, if we examine the most successful large and small networks designed by hands, each has its design uniqueness, and the inherent link between different complexity models appears to be more intriguing than simply repeating or plain scaling. We take DenseNet as an illustrative example: despite outperforming ResNets, it is high-latency and memory-hungry to run due to its heavier inter-connections than ResNet (Huang et al. 2017). As a result, state-of-the-art light-weight neural networks mostly follow the ResNet style while barely adopting dense connections.

Recognizing the discrepancy, one might lean towards another extreme, i.e., leveraging NAS to search for dedicated models for different complexity targets. Nevertheless, the resource consumption of doing so could become daunting. For instance, a typical one-shot differentiable search (Wu et al. 2019), with the weight-sharing proxy, takes 216 GPU hours on a small proxy dataset in order to obtain one target architecture. Moreover, placing a specific latency loss is prone to "architecture collapse" (Cheng et al. 2018; Chen

*Equal Contribution.

et al. 2019a), i.e., the sampled networks are of extremely low latency but with poor accuracy. Overcoming this demands carefully hand-tuning the penalty hyperparameter for each latency, which adds to the workload.

Our Contributions

The goal of this paper is to improve the training of SuperNet in one-shot differentiable neural architecture search, such that many high-performance models, yet with diverse complexity levels (e.g., FLOPS), can be derived simultaneously from the same **one** trained SuperNet. Ideally in this way, we could have models meeting various resource constraints, without sacrificing each best achievable performance (e.g., if searching dedicatedly for each complexity), while sharply trimming down the search cost.

We start addressing this challenge, by viewing SuperNet as a large *ensemble* of all those candidate networks with *highly varied complexity levels*. Our **fundamental question** then is: *how might we collectively train those networks all (close) to their optimal performance, with their weights entangled due to sharing, under one unified optimization procedure?* To handle this complicated training (meta-)set, we are inspired by the *curriculum learning* (**CL**) strategy (Bengio et al. 2009; Hacohen and Weinshall 2019): CL was adopted to train deep models, by first focusing training on an "easy" training subset (often adaptively selected), that is then gradually grown to the full set.

Following the CL idea, we consider all our different-complexity candidate networks in a SuperNet to be (meta-)training examples, naturally spanning over an easy-to-hard order. Initially, we focus SuperNet training only on a "simple" subset of all candidate networks, i.e., training a shallow SuperNet that consists of only low-complexity networks. Afterwards, we grow the search space to include more higher-complexity networks and train them; this *SuperNet curriculum* is repeated several times. At each transition between this shallow-to-deep curriculum, we further develop a key enabling technique called *SuperNet2SuperNet*, to smoothly transmit knowledge from the shallower SuperNet to initializing the deeper one, using knowledge distillation (Hinton, Vinyals, and Dean 2015).

By taking the above strategies, AdaSearch is applied to searching for a series of high-performance models of different complexities, derived from only one well-trained SuperNet. It demonstrates remarkable effectiveness, thanks to our novel curriculum SuperNet training as well as SuperNet2SuperNet transition. Our achieved state-of-the-art results include but are not limited to: (a) 0.3% higher top-1 accuracy with 5% fewer FLOPs than Efficient-B0; (b) 2.4% higher accuracy than MobileNet-V3-Large with comparable model size; and (c) similar accuracy/FLOP results with the latest FBNetV2 (Wan et al. 2020), but at sharply lower search costs.

Related Works

Efficient Neural Architecture Search An increasing amount of works are proposed to speed NAS. ENAS (Pham et al. 2018) introduces weight sharing among child models to avoid training them each from scratch. DARTS (Liu, Simonyan, and Yang 2018) accelerates NAS by turning into a continuous optimization that selects paths from a SuperNet. That is later improved further in GDAS by sampling a subnet from the SuperNet for training, using the Gumbel-softmax trick (Dong and Yang 2019).

Several works explore how to make NAS "progressive" and hence more efficient. PNAS (Liu et al. 2018) explored the search space progressively by searching for operations node-by-node within each cell, to reduce the number of architectures to evaluate. P-DARTS (Chen et al. 2019b) extends a similar search manner to differentiable search, enabling progressive search at the cell level to enlarge the depth of searched architectures gradually during the training procedure. Their motivations and algorithms are completely different from ours and not to confused with AdaSearch, whose focus is on training SuperNet with a smooth curriculum, and then multiple one-shot derivations can be done. A concurrent work (Guo et al. 2020) also involves the idea of curriculum learning, however, targets building efficient sampling strategy in the large search space. Different from that, AdaSearch aims to reduce the multiple search procedures into one time while obtaining several searched architectures serving for various deployed platforms.

Search for Efficient Neural Architectures MnasNet (Tan et al. 2019) considers the trade-off between accuracy and mobile latency as the joint optimization objective. FBNet (Wu et al. 2019), FBNetV2 (Wan et al. 2020), and ProxylessNAS (Cai, Zhu, and Han 2018) directly incorporate hardware feedbacks in the searching loop, to obtain practical hardware-efficient models for target platforms. Slimmable Network (Yu et al. 2018) proposes to train a single network to support different widths via switchable batch normalization, which was later extended to one-shot NAS for searching channel numbers (Yu and Huang 2019). Lately, Once for All (Cai, Gan, and Han 2019) trains a SuperNet and selects specialized sub-networks from the SuperNet for different accuracy-efficiency trade-offs, aided by a generalized pruning method during search. Note that all the above methods train their SuperNet/normal neural network completely from end to end, without any curriculum. AdaSearch presents a complementary effort to train SuperNet with a curriculum learning scheme, which can be potentially combined with those NAS methods.

Methodology

In this paper, we use differentiable NAS to optimize SuperNets with a designed curriculum searching process and smoothly bridge the gap across diverse resource constraints. Exploring in a more complex search space based on the prior knowledge from a compact SuperNet is a non-trivial work since it requires effective knowledge retaining techniques to avoid architecture search getting stuck with bad local minima. How to transfer the architecture and weight knowledge from the prior compact SuperNet to improve the more complex SuperNet training with much less cost than separately direct searching different models is our core problem. In brief, the goal is to progressively search for architectures that satisfy different hardware demands using the searched structural knowledge to improve both search efficiency and accuracy.

In this section, we introduce a curriculum SuperNet training (CST) framework to train SuperNets for target accuracy-efficiency trade-offs. Then, we propose two techniques that combined as CST to handle the common knowledge retaining challenge in SuperNet progressive training.

Table 1: AdaSearch-A0 largest search space. The input resolution for models is 224–by-224. "tbs" denotes layer type that needs to be searched and includes inverted residual block types as operation candidates. Tuples of three values under expansion rate and out channels filters represent the lowest value, highest, and growth step (low, high, step). n refers to the number of blocks. The search space grows up in the following stages while the details of other space settings will be shown in the Appendix.

Input	block	expansion rate	channel filters	num
$224^2 \times 3$	3x3	1	16	1
$112^2 \times 16$	tbs	1	(12, 16, 4)	1
$112^2 \times 16$	tbs	(0.75, 3.75, 0.5)	(16, 28, 4)	1
$56^2 \times 28$	tbs	(0.75, 3.75, 0.5)	(16, 40, 8)	2
$28^2 \times 40$	tbs	(0.75, 4.25, 0.5)	(48, 96, 8)	3
$14^2 \times 96$	tbs	(0.75, 4.5, 0.75)	(72, 128, 8)	2
$14^2 \times 128$	tbs	(0.75, 5.25, 0.75)	(112, 184, 8)	2
$7^2 \times 184$	tbs	(0.75, 5.25, 0.75)	(112, 184, 8)	1
$7^2 \times 184$	1x1	-	1280	1
1280	fc	-	1000	1

Table 2: The operation candidates of block-type searching in our framework.

op	kernel	se	activation
ir_k3	3	✗	relu
ir_k5	5	✗	relu
ir_k3_hs	3	✗	hswish
ir_k5_hs	5	✗	hswish
ir_k3_se	3	✓	relu
ir_k5_se	5	✓	relu
ir_k3_se_hs	3	✓	hswish
ir_k5_se_hs	5	✓	hswish

Architecture Space Our framework contains an extraordinary search space where block-type, channel filters, and expansion ratio are jointly searched. In the block-type search, we involve the kernel size, non-linear activation function, and squeeze-and-excite layer, details are shown in Table 2. In channel-level search, inspired by FBNetV2 (Wan et al. 2020), the number of output channels and the expansion rate of operation candidates in each layer can be searched with channel masking to constant the number of channel options. To balance the efficiency and effectiveness, only two paths in the SuperNet are sampled and trained each iteration to tackle the search memory cost.

The main search procedure of our framework starts from the very beginning stage, the size of the search space is shown in Table 1. While gradually moving into the next search period, the number of searchable layers, channel filters, and block expansion ratio are all growing up, targeting a more complex search space. The detailed hyperparameter settings for discovering the obtained architectures (AdaSearch-A_0 to A_N) are shown in the Appendix.

Curriculum SuperNet Training

In the curriculum SuperNet training, we design an easy-to-hard search space routine comprising multiple search stages as shown in 2. In detail, to flexibly bridge the gap across different hardware constraints, we propose to search SuperNets from small size to large size in a curriculum way. Firstly, we start by directly searching the smallest SuperNet with the minimum depth and width, named SuperNet-A0. The first initial search stage takes short search time and memory cost for the model with low-complexity. Then, we enlarge the search space by increasing SuperNet-A0's channels and layers. In the next search stage, via our novel knowledge retaining techniques, the weights and architecture of searched SuperNet-A0 are inherited by the larger SuperNet (named SuperNet-A1) with larger depth and width dimension and reused in SuperNet-A1 searching. The above process will loop until the largest architecture is searched. In deeper periods, we can use the structural knowledge of shallow architectures to optimize complex architectures that converge faster than direct searching.

We show that the knowledge of architecture and weights from a small but well-trained SuperNet can be used to speed up and stabilize the larger new search space exploring. In the following subsection, we describe the details of curriculum SuperNet training's two knowledge inheriting solutions for AdaSearch: *SuperNet2SuperNet* and *Progressive SuperNet Distillation* to consider the knowledge transfer between two curriculum searching stages.

SuperNet2SuperNet To search a new larger network in the further period, we enlarge the previous search space including scaling the depth dimension in each stage and width dimension in the convolution of each operation candidate. Since the depth and width of the SuperNets in the two search spaces do not match, the larger SuperNet cannot directly inherit the parameters of the previously searched SuperNet by simply loading the weights. Therefore, we present our *SuperNet2SuperNet*, which smoothly initializes a new and larger SuperNet using the previously searched SuperNet.

To place a layer with a wider layer, *Net2WiderNet* is used to preserve the functionality of operations in the layer (Chen, Goodfellow, and Shlens 2015). Considering a convolutional layer with weight matrix $\boldsymbol{W}_l$ and the next convolutional layer with weight matrix $\boldsymbol{W}_{l+1}$. The component of $\boldsymbol{W}_l$ is (w^l, h^l, i^l, o^l), where the w^l is filter width and h^l is filter height, while i^l and o^l denote the number of input and output channels. To replace this layer with a wider layer which has out channels number $\hat{o}^l > o^l$, a random remapping function f^l is introduced and set as:

$$f_l(j) = \begin{cases} j & 1 \leq j \leq o^l \\ \text{random sample from } \{1, \cdots, o^l\} & o^l < j \leq \hat{o}^l \end{cases} . \quad (1)$$

Then we wider the $\boldsymbol{W}_l$ to $\hat{\boldsymbol{W}}_l$ (with shape $(w^l, h^l, i^l, \hat{o}^l)$)

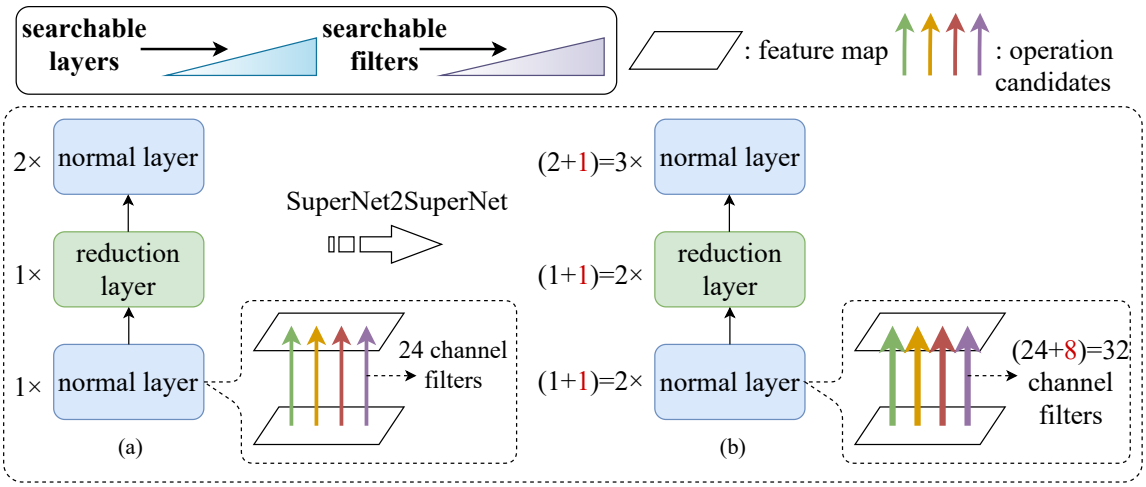

Figure 1: **SuperNet2SuperNet** (a) is a SuperNet with compact search space; (b) the search space grows up in number of layers and operation candidates' filters, which inherites parameters from (a) via SuperNet2SuperNet to smooth further search. Also, newly added layers and filters will still keep the functionality.

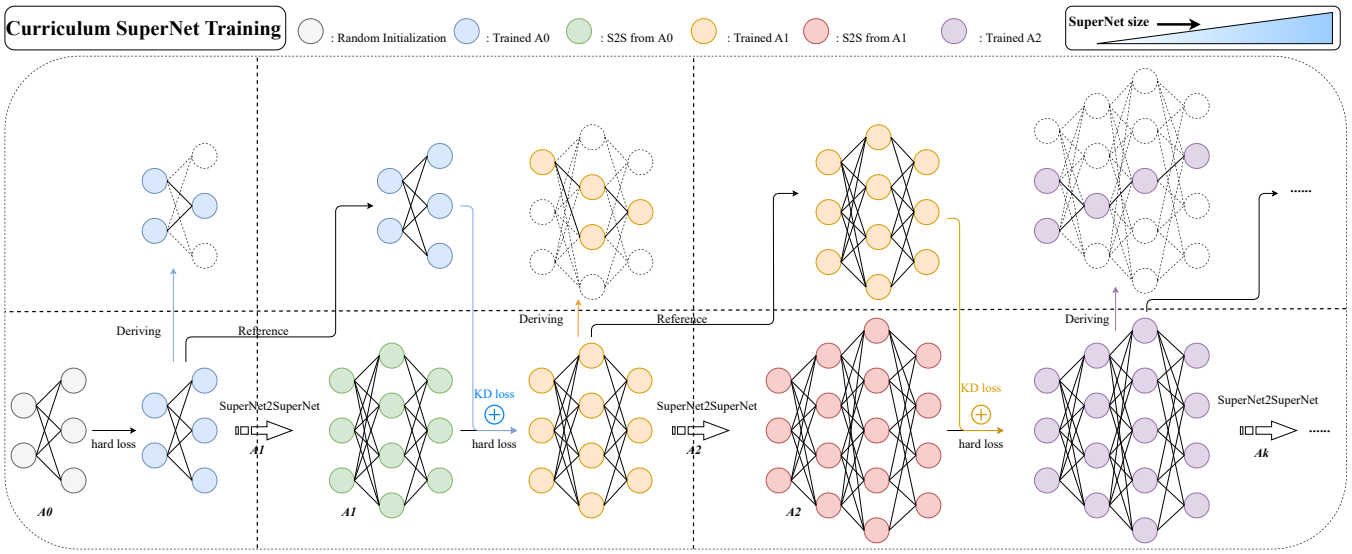

Figure 2: The overview of Curriculum SuperNet Training. Before the next search period's training, the parameters of student SuperNet are inherited from the past compact SuperNet by *SuperNet2SuperNet*. In the next search period, the soft output labels are adopted for the supervision of the student's exploring in more complex search space.

according to function f_l:

$$\hat{\boldsymbol{W}}_l[x, y, i, j] = \boldsymbol{W}_l[x, y, i, f_l(j)]. \tag{2}$$

Meanwhile, we modified the weight $\boldsymbol{W}_{l+1}$ of next convolutional layer by reshaping its input channel. The new weight matrix $\hat{\boldsymbol{W}}_{l+1}$ is given as:

$$\hat{\boldsymbol{W}}_{l+1}[x, y, j, k] = \frac{\boldsymbol{W}_{l+1}[x, y, f_l(j), k]}{\left|\{z|f_l(z) = f_l(j)\}\right|}. \tag{3}$$

We refer to original paper for the more descriptions of Equation 3 (Chen, Goodfellow, and Shlens 2015).

To deeper the network, we transform the existing network into a deeper one by adding new blocks at the end of every downsampling stage. Each candidate convolutional layer in a newly added block is initialized as an identity mapping by setting the kernel to be an identity matrix while preserving the operations between layers. To sum up, by the *SuperNet2SuperNet* mechanism, we combine wider and deeper operations to smoothly enlarge the architecture space and better inherited the searched SuperNet in further search periods.

Progressive SuperNet Distillation To more efficiently transfer knowledge of architectures and weights from trained compact SuperNets, we adopt soft targets from previous teacher SuperNet predictions to accelerate more complex student SuperNet searches in the search phase above AdaSearch-

A0. Soft targets contribute to a better entropy regularization, which guides the discovery of new higher capacity architectures. We introduce soft labels $\mathcal{L}_{soft}$ provided by the small but well-trained SuperNet in the prior period and use hard labels from the training dataset as defined in $\mathcal{L}_{hard}$. Then combine these two losses with a scaling ratio λ as the loss function for classification in larger architecture optimization:

$$\mathcal{L} = (1-\lambda)\mathcal{L}_{hard} + \lambda\mathcal{L}_{soft} \tag{4}$$

Moreover, in both update steps of architectures α and weights w, knowledge distillation in structural knowledge provides effective supervision for inheriting architecture knowledge and weight knowledge of the compact network into the new growing up network training. We initialize the λ as 0.5 and reduce it to 0.05 linearly. It also benefits to smooth the span of SuperNet optimization in two contiguous progressive searching periods.

Experiments

Implementation Details. We implement AdaSearch using PyTorch (Paszke et al. 2019) on 8 Tesla V100 GPUs and use a total batch size of 1024 on ImageNet 2012 classification dataset (Deng et al. 2009). Following previous common setting (Wu et al. 2019), we randomly sample 100 classes from the original 1000 classes to reduce search cost. We adopt differentiable search in our framework. When searching, weight parameters $\boldsymbol{w}$ is trained on $5/6$ of ImageNet-100 training set using SGD with momentum, and the architecture parameters $\boldsymbol{\alpha}$ is trained on the rest $1/6$ of ImageNet training set with Adam optimizer(Kingma and Ba 2014). The SuperNet-A0 is trained for 100 epochs, then in each further period, we sequentially grow up SuperNet-A1/A2/A3/A4 architecture space and gradually train these more complex SuperNets with 60 epochs following an easy-to-hard order adaptively until the final stage finished.

Compared to State-of-the-Art

FLOPs-efficiency We summarized the top-1 accuracy on ImageNet of models searched by AdaSearch, ranging from 100MFLOPs∼500MFLOPs and the comparison with state-of-the-art NAS methods in Table 3. Our models outperform all existing networks under the same computational FLOPs-efficiency constraints. As shown in Fig. 3, the models searched by AdaSearch achieve 0.3%, 1.2% and 4.3% absolute accuracy gains over EfficientNet-B0 (Tan and Le 2019), MobileNetV3 (Howard et al. 2019) and ProxylessNAS (Cai, Zhu, and Han 2018), respectively while maintaining similar FLOPs. Furthermore, AdaSearch achieves competitive accuracy performance compared to the previous SOTA method: FBNetv2 (Wan et al. 2020) yet saving much more search cost in GPU hours, energy and carbon emission, which will be detailedly reported in Table 4.

Parameter-efficiency Considering the edge-device storage efficiency in real-world applications, hardware has strict constraints in the model size. AdaSearch consistently minimizes the parameter, which improves the trade-off between accuracy and model-size by a significant margin compared with other NAS methods. Figure. 4 shows the detailed results in the parameter-efficiency. In the comparison of small size models (4M parameters), our model achieves 76.4% accuracy, shown significant improvements over other small models. With similar or much smaller model size in large models (7M∼8M parameters), AdaSearch yields 2.9% accuracy higher than MobileNetV3 (Howard et al. 2019) and 1.5% accuracy higher than FBNetV2 (Wan et al. 2020). Also, retraining an AdaSearch normal network only costs 180 epochs which are fewer than 360 epochs for FBNetV2 retraining.

Resource and Energy-efficiency From the perspective of environmental cost, we calculate the GPU hours, energy and CO_2 emission (following (Strubell, Ganesh, and McCallum 2019). Benefiting from our curriculum training framework, models for different efficiency constraints can be searched without superfluous computational cost and energy consumption while previous NAS methods require repeat searching to adapt diverse hardware demands. From a small model that has been searched, to search a larger model, AdaSearch only needs a considerable small environmental cost. Therefore, our total search cost increases linearly. Specifically, compared to FBNetV2 (Wan et al. 2020), we find the models maintaining similar or better performance with two thirds fewer total GPU hours, energy, and CO_2 emission, which is demonstrated in Table 4. The significant reduction in computational budgets greatly improves the practical impacts of AdaSearch in various deployment scenarios.

Ablation Experiments

As illustrated in Figure 5, we compare AdaSearch with compound scaling-up, naive progressive search, and separately direct search in accuracy-FLOPs trade-offs. For a fair comparison of the search framework, all the methods search under the same search space and use the same search epochs except direct search. In direct search, we separately trained SuperNet 100 epochs for each model.

By repeatedly scaling the width and depth of the searched SuperNet-A0 using a designed multiplier, it is shown that our AdaSearch substantially improves the flexibility of architecture structure compared to compound scaling-up with more than **2%** accuracy improvement. Compared to naive progressive search with randomly initializing new layers and directly inheriting prior parameters, thanks to our knowledge retaining techniques, AdaSearch achieves more than **1%** better accuracy than naive progressive search. With better supervision from the prior architectural knowledge, AdaSearch finds more accurate models with efficient guidance from the smaller searched model.

We further conduct an ablation study to evaluate the effectiveness of SuperNet2SuperNet and Progressive SuperNet Distillation, results are shown in Figure 6. We combine both techniques as CST and compare their performance under various FLOPs constraints separately for analyses. In the very beginning stage, those methods result in a similar performance. However, when the SuperNet grows up, the proposed technics gradually outperform the baseline method due to much more complexity of search space. These results demonstrate the superiority of both Progressive SuperNet Distillation and

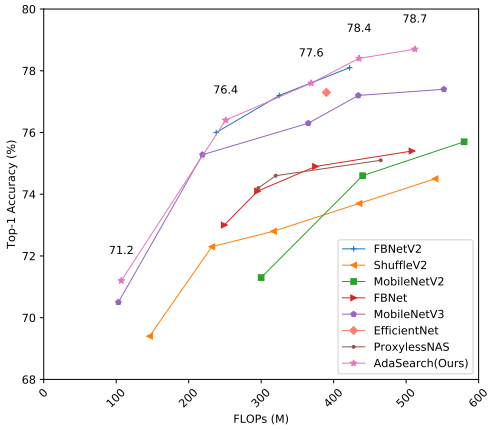

Figure 3: Accuracy vs. FLOPs

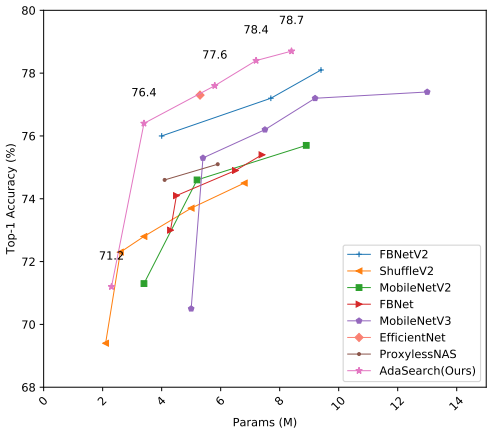

Figure 4: Accuracy vs. Params

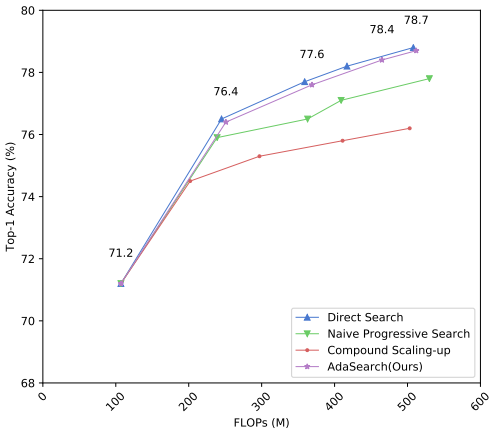

Figure 5: Ablation study about direct search, compound scaling-up, and naively progressive search.

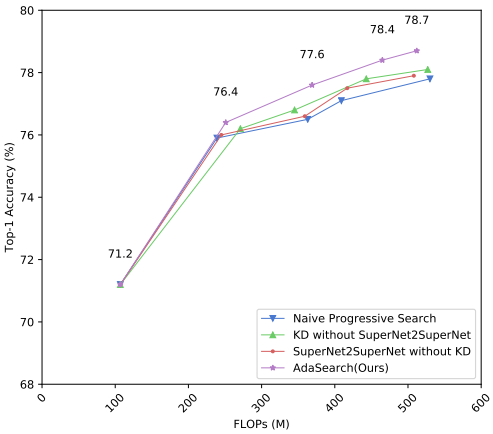

Figure 6: Ablation study for two techniques of CST.

SuperNet2SuperNet strategy.

Conclusion

We present AdaSearch, a flexible neural architecture search scheme that trains models to fit diverse hardware constraints via a shallow-to-deep training curriculum. Different from previous NAS methods, AdaSearch allows smoothly expanding search space using our SuperNet2SuperNet mechanism to adopt the deeper architecture search rather than naive progressive search. By inheriting structure knowledge from the prior shallow architecture with knowledge distillation, we achieve great performance and efficiency improvement in the higher-complexity architecture search. The strong empirical results on accuracy-efficiency trade-offs compare favorably against state-of-the-art methods, which demonstrate the effectiveness of AdaSearch. Moreover, attaching high importance to computational and environmental costs, our method greatly reduces the search cost compared to conventional NAS. We will extend the proposed approaches under a more diverse framework (e.g. no retrain required) in the future.

Appendix

References

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48.

Cai, H.; Gan, C.; and Han, S. 2019. Once for all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*.

Cai, H.; Zhu, L.; and Han, S. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.

Chen, T.; Goodfellow, I.; and Shlens, J. 2015. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*.

Chen, W.; Gong, X.; Liu, X.; Zhang, Q.; Li, Y.; and Wang, Z. 2019a. FasterSeg: Searching for Faster Real-time Semantic Segmentation. *arXiv preprint arXiv:1912.10917*.

Chen, X.; Xie, L.; Wu, J.; and Tian, Q. 2019b. Progressive differentiable architecture search: Bridging the depth gap

Model	Search		Params	FLOPs	Top-1
	Method	Space			Acc (%)
ShuffleNetV2 (Ma et al. 2018)	manual	-	2.1M	146M	69.4
MobileNetV3-small-1.3× (Howard et al. 2019)	scaling	-	5.0M	103M	70.5
AdaSearch-A0(ours)	**gradient**	**layer-wise**	**2.3M**	**107M**	**71.2**
ShuffleNetV2-1.1× (Ma et al. 2018)	scaling	-	2.6M	232M	72.3
FBNet-A (Wu et al. 2019)	gradient	layer-wise	4.3M	249M	73.0
FBNet-B (Wu et al. 2019)	gradient	layer-wise	4.5M	295M	74.1
FBNetV2-F4 (Wan et al. 2020)	gradient	layer-wise	4.0M	238M	76.0
AdaSearch-A1(ours)	**gradient**	**layer-wise**	**3.4M**	**251M**	**76.4**
ProxylessNAS-R (Cai, Zhu, and Han 2018)	gradient/RL	layer-wise	4.1M	320M	74.6
Single-path NAS (Guo et al. 2019)	EA	layer-wise	4.3M	365M	74.2
MobileNetV3-large-1.2×	scaling	-	7.5M	365M	76.2
EfficientNet-B0 (Tan and Le 2019)	RL	stage-wise	5.3M	390M	77.3
FBNetV2-L1 (Wan et al. 2020)	gradient	layer-wise	7.7M	325M	77.2
AdaSearch-A2(ours)	**gradient**	**layer-wise**	**5.8M**	**369M**	**77.6**
FBNet-1.2× (Wu et al. 2019)	scaling	-	6.5M	406M	74.6
ProxylessNAS-G (Cai, Zhu, and Han 2018)	gradient/RL	layer-wise	5.9M	465M	75.1
MobileNetV3-large-1.3× (Howard et al. 2019)	scaling	-	9.2M	434M	77.2
FBNetV2-L2 (Wan et al. 2020)	gradient	layer-wise	9.4M	422M	78.1
AdaSearch-A3(ours)	**gradient**	**layer-wise**	**7.2M**	**435M**	**78.4**
ShuffleNetV2-1.5× (Ma et al. 2018)	scaling	-	6.8M	540M	74.5
MnasNet-1.2× (Tan et al. 2019)	scaling	-	6.9M	553M	75.7
MobileNetV2-1.4× (Sandler et al. 2018)	scaling	-	8.9M	580M	75.7
MobileNetV3-large-1.4× (Howard et al. 2019)	scaling	-	13.0M	552M	77.4
AdaSearch-A4(ours)	**gradient**	**layer-wise**	**8.4M**	**512M**	**78.7**

Table 3: ImageNet classification performance. Compound scaling is applied on depth and width dimension.

Model	Search Cost			FLOPs
	GPU Hours(h)	Energy (kWh)	CO_2e (lbs)	
ProxylessNAS-R (Cai, Zhu, and Han 2018)	200.0	59.46	59.46	320M
ProxylessNAS-G (Cai, Zhu, and Han 2018)	200.0	59.46	59.46	465M
ProxylessNAS-G (Cai, Zhu, and Han 2018)	200.0	59.46	59.46	487M
Total-cost	600.0	178.38	178.38	-
FBNetV2-F4 (Wan et al. 2020)	216.0	64.21	61.26	238M
FBNetV2-L1 (Wan et al. 2020)	648.0	192.63	183.78	325M
FBNetV2-L2 (Wan et al. 2020)	648.0	192.63	183.78	422M
Total-cost	1512.0	449.47	428.82	-
AdaSearch-A0(ours)	**89.6**	26.75	25.52	107M
AdaSearch-A1(ours)	89.6+**74.4**=164.0	26.75+**22.19**=48.94	25.52+**21.17**=46.69	251M
AdaSearch-A2(ours)	164.0+**87.2**=251.2	48.94+**25.97**=74.91	46.69+**24.77**=71.46	369M
AdaSearch-A3(ours)	251.2+**96.8**=348.0	74.91+**28.94**=103.85	71.46+**27.61**=99.07	435M
AdaSearch-A4(ours)	348.0+**108.0**=456.0	103.85+**32.11**=135.96	99.07+**30.63**=129.70	512M
Total-cost (ours)	**456.0**	**135.96**	**129.70**	-

Table 4: Searching cost and energy consumption on ImageNet. AdaSearch obtains all architecture ($A0$-$A4$) in one procedure thus saves energy and searching time compared to others.

Table 5: AdaSearch-A1 largest search space.

Input	block	expansion rate	channel filters	num
$224^2 \times 3$	3x3	1	16	1
$112^2 \times 16$	tbs	1	(12, 16, 4)	1
$112^2 \times 16$	tbs	(0.75, 3.75, 0.5)	(16, 32, 4)	2
$56^2 \times 28$	tbs	(0.75, 3.75, 0.5)	(16, 48, 8)	3
$28^2 \times 40$	tbs	(0.75, 4.25, 0.5)	(48, 104, 8)	4
$14^2 \times 96$	tbs	(0.75, 4.5, 0.75)	(72, 136, 8)	3
$14^2 \times 128$	tbs	(0.75, 5.25, 0.75)	(112, 224, 8)	2
$7^2 \times 184$	tbs	(0.75, 5.25, 0.75)	(112, 256, 8)	1
$7^2 \times 184$	1x1	-	1280	1
1280	fc	-	1000	1

Table 6: AdaSearch-A2 largest search space.

Input	block	expansion rate	channel filters	num
$224^2 \times 3$	3x3	1	16	1
$112^2 \times 16$	tbs	1	(12, 16, 4)	1
$112^2 \times 16$	tbs	(0.75, 4.75, 0.5)	(16, 36, 4)	2
$56^2 \times 28$	tbs	(0.75, 4.75, 0.5)	(16, 56, 8)	3
$28^2 \times 40$	tbs	(0.75, 5.25, 0.5)	(48, 112, 8)	4
$14^2 \times 96$	tbs	(0.75, 5.25, 0.75)	(72, 144, 8)	4
$14^2 \times 128$	tbs	(0.75, 5.25, 0.75)	(112, 256, 8)	4
$7^2 \times 184$	tbs	(0.75, 5.25, 0.75)	(112, 256, 8)	2
$7^2 \times 184$	1x1	-	1280	1
1280	fc	-	1000	1

Table 7: AdaSearch-A3 largest search space.

Input	block	expansion rate	channel filters	num
$224^2 \times 3$	3x3	1	16	1
$112^2 \times 16$	tbs	1	(12, 16, 4)	1
$112^2 \times 16$	tbs	(0.75, 4.75, 0.5)	(16, 36, 4)	3
$56^2 \times 28$	tbs	(0.75, 5.25, 0.5)	(16, 64, 8)	3
$28^2 \times 40$	tbs	(0.75, 5.25, 0.5)	(48, 128, 8)	4
$14^2 \times 96$	tbs	(0.75, 5.25, 0.75)	(72, 160, 8)	5
$14^2 \times 128$	tbs	(0.75, 6, 0.75)	(112, 256, 8)	4
$7^2 \times 184$	tbs	(0.75, 6, 0.75)	(112, 336, 8)	2
$7^2 \times 184$	1x1	-	1280	1
1280	fc	-	1000	1

Table 8: AdaSearch-A4 largest search space.

Input	block	expansion rate	channel filters	num
$224^2 \times 3$	3x3	1	16	1
$112^2 \times 16$	tbs	1	(12, 16, 4)	1
$112^2 \times 16$	tbs	(0.75, 5.75, 0.5)	(16, 40, 4)	3
$56^2 \times 28$	tbs	(0.75, 5.75, 0.5)	(16, 64, 8)	3
$28^2 \times 40$	tbs	(0.75, 5.75, 0.5)	(48, 128, 8)	5
$14^2 \times 96$	tbs	(0.75, 6, 0.75)	(72, 160, 8)	5
$14^2 \times 128$	tbs	(0.75, 6, 0.75)	(112, 256, 8)	6
$7^2 \times 184$	tbs	(0.75, 6, 0.75)	(112, 336, 8)	3
$7^2 \times 184$	1x1	-	1280	1
1280	fc	-	1000	1

between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, 1294–1303.

Cheng, A.-C.; Lin, C. H.; Juan, D.-C.; Wei, W.; and Sun, M. 2018. InstaNAS: Instance-aware Neural Architecture Search. *arXiv preprint arXiv:1811.10201*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255.

Dong, X.; and Yang, Y. 2019. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1761–1770.

Guo, Y.; Chen, Y.; Zheng, Y.; Zhao, P.; Chen, J.; Huang, J.; and Tan, M. 2020. Breaking the curse of space explosion: Towards efficient nas with curriculum search. *arXiv preprint arXiv:2007.07197*.

Guo, Z.; Zhang, X.; Mu, H.; Heng, W.; Liu, Z.; Wei, Y.; and Sun, J. 2019. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*.

Hacohen, G.; and Weinshall, D. 2019. On the power of curriculum learning in training deep networks. *arXiv preprint arXiv:1904.03626*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, 1314–1324.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision*, 19–34.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Ma, N.; Zhang, X.; Zheng, H.-T.; and Sun, J. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision*, 116–131.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, 8026–8037.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Strubell, E.; Ganesh, A.; and McCallum, A. 2019. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; and Le, Q. V. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2820–2828.

Tan, M.; and Le, Q. V. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

Wan, A.; Dai, X.; Zhang, P.; He, Z.; Tian, Y.; Xie, S.; Wu, B.; Yu, M.; Xu, T.; Chen, K.; et al. 2020. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12965–12974.

Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; and Keutzer, K. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 10734–10742.

Yu, J.; and Huang, T. 2019. AutoSlim: Towards One-Shot Architecture Search for Channel Numbers. *arXiv preprint arXiv:1903.11728*.

Yu, J.; Yang, L.; Xu, N.; Yang, J.; and Huang, T. 2018. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*.

Zoph, B.; and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.