EPNAS: Efficient Progressive Neural Architecture Search

Yanqi Zhou[*1]
yanqiz@google.com

Peng Wang[3]
wangpeng54@baidu.com

Sercan Arik[*2]
soarik@google.com

Haonan Yu[*4]
haonanu@gmail.com

Syed Zawad[*5]
szawad@nevada.unr.edu

Feng Yan[5]
fyan@unr.edu

Greg Diamos[*6]
greg@landing.ai

[1] Google Brain
[2] Google Cloud AI
[3] Baidu Research
[4] Horizon Robotics
[5] University of Nevada
[6] Landing AI

Abstract

In this paper, we propose Efficient Progressive Neural Architecture Search (EPNAS), a neural architecture search (NAS) that efficiently handles large search space through a novel progressive search policy with performance prediction based on REINFORCE [57]. EPNAS is designed to search target networks in parallel, which is more scalable on parallel systems such as GPU/TPU clusters. More importantly, EPNAS can be generalized to architecture search with multiple resource constraints, *e.g.*, model size, compute complexity or intensity, which is crucial for deployment in widespread platforms such as mobile and cloud. We compare EPNAS against other state-of-the-art (SoTA) network architectures (*e.g.*, MobileNetV2 [39]) and efficient NAS algorithms (*e.g.*, ENAS [34], and PNAS [27]) on image recognition tasks using CIFAR10 and ImageNet. On both datasets, EPNAS is superior w.r.t. architecture searching speed and recognition accuracy.

1 Introduction

Deep neural networks have demonstrated excellent performance on challenging tasks and pushed the frontiers of impactful applications such as image recognition [40], image synthesis [22], language translation [46], speech recognition and synthesis [2, 45]. Despite all these advancements, designing neural networks remains as a laborious task, requiring extensive domain expertise. Motivated by automating the neural network design while achieving superior performance, neural architecture search (NAS) has been proposed [4, 27, 52, 55, 57].

Conventional NAS algorithms are performed with limited search spaces (*e.g.* small number of operation type) due to lack of efficiency, which hinders the application of NAS to various tasks. For example, [52] uses 800 GPUs, and takes 3 days to discover a model on a small dataset like CIFAR10 [24], which is infeasible to directly search models over larger datasets such as COCO [25] or ImageNet [11]. Therefore, it is crucial to improve searching efficiency of NAS to allow larger space of architecture variation, in order to achieve better model performance or handle multiple objective simultaneously.

Commonly, the efficiency of a NAS algorithm depends on two factors: 1) the number of models need to be sampled. 2) The time cost of training and evaluating a sampled model. Targeting at the two aspects, recent works [28, 34] have made significant progress. For example, ENAS [34], DARTS [28] or SNAS [49] try to share the architecture and parameters between different models trained during searching period, thus reduce the time cost of model training. However, their models are searched within a large pre-defined graph that need to be fully loaded into memory, therefore the search space is limited. EAS [9] and PNAS [27] progressively add or morph an operation based on a performance predictor or a policy network, thus can support larger operation set at search time and significantly reduce the number of models need to be sampled. This paper further explores the direction of progressive search, aiming to improve sample efficiency and generality of search space. We call our system efficient progressive NAS (EPNAS). Specifically, based on the framework of REINFORCE [47], to reduce necessary number of sampled models, we design a novel set of actions for network morphing such as group-scaling and removing, additional to the adding or widening operations as proposed in EAS [9]. It allows us to initiate NAS from a better standing point, *e.g.* a large random generated network [48], rather than from scratch as in PNAS or a small network as in EAS. Secondly, to reduce the model training time, we propose a strategy of aggressive learning rate scheduling and a general dictionary-based parameter sharing, where a model can be trained with one fifth of time cost than training it from scratch. Comparing to EAS or PNAS, as shown in Sec. 4, EPNAS provides $2\times$ to $20\times$ overall searching speedup with much larger search space, while providing better performance with the popular CIFAR10 datasets. Our model also generalizes well to larger datasets such as ImageNet.

In addition to model accuracy, deep neural networks are deployed in a wider selection of platforms (e.g., mobile device and cloud) today, which makes resource-aware architecture design a crucial problem. Recent works such as MNASNet [43] or DPPNet [12] extend NAS to be device-aware by designing a device-aware reward during the search. In our case, thanks to the proposed efficient searching strategy, EPNAS can also be easily generalized to multi-objective NAS which jointly considers computing resource related metrics, such as the model's memory requirement, computational complexity, and power consumption. Specifically, we transform those hard computational constraints to soft-relaxed rewards for effectively learning the network generator. In our experiments, we demonstrate that EPNAS is able to perform effectively with various resource constrains which fails random search.

Finally, to better align NAS with the size of datasets, we also supports two search patterns for EPNAS. For handling small dataset, we use an efficient layer-by-layer search that exhaustively modify all layers of the network. For handling larger dataset like ImageNet, similar with PNAS, we adopt a module-based search that finds a cell using small dataset and stack it for generalization. We evaluate EPNAS's performance for image recognition on CIFAR10 and its generalization to ImageNet. For CIFAR10, EPNAS achieves 2.79% test error when compute intensity is greater than 100 FLOPs/byte, and 2.84% test error when model size is less than 4M parameters. For ImageNet, we achieve 75.2% top-1 accuracy with 4.2M parameters. In both cases, our results outperform other related NAS algorithms

such as PNAS [27], and ENAS [33].

2   Related work

Neural architecture search (NAS).   NAS has attracted growing interests in recent years due to its potential advantages over manually-crafted architectures. As summarized by [13], the core issues lie in three aspects: efficient search strategy, large search space, and integration of performance estimation. Conventionally, evolutionary algorithms [3, 27, 29, 35, 36, 42] are one set of methods used for automatic NAS. NAS has also been studied in the context of Bayesian optimization [7, 21], which models the distribution of architectures with Gaussian process. Recently, reinforcement learning [4, 5, 52] with deep networks has emerged as a more effective method. However, these NAS approaches are computationally expensive, with relatively small search space w.r.t. single target of model accuracy, e.g., [52] used hundreds of GPUs for delivering a model comparable to human-crafted network on CIFAR10. To tackle the search cost, network morphism [44] for evolutionary algorithm, and efficient NAS approaches with parameter sharing such as ENAS [34] are proposed. Specifically, ENAS employs weight-sharing among child models to eschew training each from scratch until convergence. To handle larger datasets, one-shot NAS such as path selection [5] and operation selection such as DARTS [28] are also proposed. However, parameter sharing as ENAS and operation sharing as DARTS search within a pre-defined network graph, which limits the scaling of search space, *e.g.*, channel size and kernel size cannot be flexibly changed for each layer in order to reuse the operations or weights. However, **larger search space is crucial for discovering architectures especially when multiple objective or resource constraints are also considered**. Another set of methods for reducing search cost is to progressively adapt a given architectures such as EAS [9] and PNAS [27]. In each step, one may freely choose to add an operation from a set, which enables larger searching possibility for networks or a cell structure. Therefore, we follow this learning strategy and propose a novel progressive search policy with performance prediction, which brings additional efficiency and can reach higher accuracy as demonstrated in our experiments.

Resource-constraint NAS.   It used to be that most effective approaches for optimizing performance under various resource constraints rely on the creativity of the researchers. Among many, some notable ones include attention mechanisms [50], depthwise-separable convolutions [10, 18], inverted residuals [39], shuffle features [31, 51], and structured transforms [41]. There are common approaches that reduce model size or improve inference time as a post processing stage. For example, sparsity regularization [26], channel pruning [16], connection pruning [15], and weights/activation quantization [20] are common model compression approaches. Similarly, one may automatically design a resource constraint architecture through NAS. Most recently, AMC [17] adopts reinforcement learning to design policy of compression operations. MorphNet [14] proposes to morph an existing network, *e.g.* ResNet-101, by changing feature channel or output size based on a specified constraints. DPPNet [12] or MNASNet [43] propose to directly search a model with resource-aware operations based on PNAS [28] or NAS [53] framework respectively. In our work, as demonstrated in our experiments, EPNAS also effectively enables NAS with multiple resource requirements simultaneously thanks to our relaxed objective and efficient searching strategy.

3   Efficient progressive NAS with REINFORCE

In this section, for generality, we direct formulate EPNAS with various model constraints, as one may simply remove the constraints when they are not necessary. Then, we elaborate our
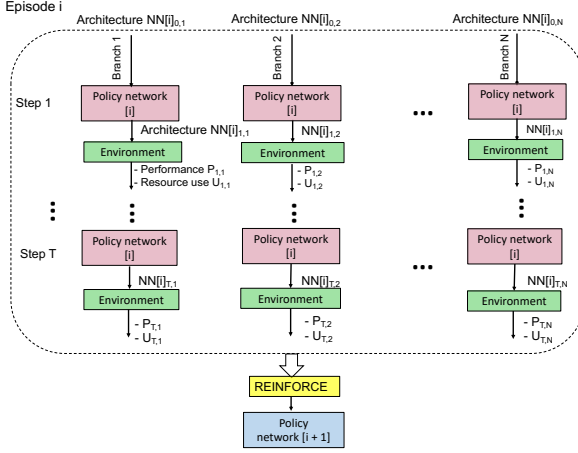
Figure 1: REINFORCE step for policy gradient. *N* is the number of parallel policy networks to adapt a baseline architecture at episode of *i*.

optimization and proposed architecture transforming policy networks.

As stated in Sec. 1, rather than rebuilding the entire network from scratch, we adopt a progressive strategy with REINFORCE [37] for more efficient architecture search so that architectures searched in previous step can be reused in subsequent steps. Similar ideas are proposed for evolutionary algorithms [27] recently, however, in conjunction with reinforcement learning (RL), our method is more sample efficient. This strategy can also be considered in the context of Markov Chain Monte Carlo sampling [8] (as the rewards represent a target distribution and the policy network approximates the conditional probability to be sampled from), which has been proven to be effective in reducing the sampling variation when dealing with high dimensional objective, yielding more stable learning of policy networks.

Formally, given an existing network architecture $\mathcal{X}$, a policy network $\pi_\theta(\mathbf{a}|\mathcal{X})$ generates action $\mathbf{a}$ that progressively change $\mathcal{X}$ from our search space $\mathcal{S}$, w.r.t. multiple resource constraints objective:

$$
\begin{aligned}
&\max_{\mathcal{X} \in \mathcal{S}} P(\mathcal{X}|\mathcal{D}) \\
&s.t.\ \mathbb{1}(U_i(\mathcal{X}) \in [C_{li}, C_{ui}]),\ \ \forall i \in \{1, \ldots, K\},
\end{aligned}
\tag{1}
$$

where $P(\mathcal{X}|\mathcal{D})$ is the performance of $\mathcal{X}$ under dataset $\mathcal{D}$. $\mathbb{1}()$ is an indicator function, and $U_i(\mathcal{X})$ is the i_{th} resource usage of the network, C_{li} and C_{ui} are the corresponding constraint, indicating the lower bound and upper bound for resource usage. Next, we show how to optimize the objective with policy gradient for progressive architecture generation.

3.1 Policy Gradient with Resource-aware Reward

Given the objective in Eqn. (1), we optimize it following the procedure to train a policy network $\pi_\theta(\cdot)$ using policy gradient [37], by taking the reward $r(\mathcal{X})$ as Lagrange function of Eqn. (1). We depict the optimization procedure in Fig. 1, where the policy networks are distributed to N branches. For a branch n, at episode e, we manipulate the network from an initial network $\mathcal{X}_{0,n}$, and at step t, the policy network generates the action $\mathbf{a}_{t,n}$, which transforms the network $\mathcal{X}_{t,n}$ to $\mathcal{X}_{t+1,n}$ that is evaluated using the reward. We adopt T steps for an episode, and accumulate all the rewards over all steps and all batches to compute the

gradients. Formally, the gradient for parameter θ is

$$g = \frac{1}{N}\sum_{n=0}^{N-1}\sum_{t=0}^{T-1}\bigtriangledown_\theta \log \pi_\theta(\mathbf{a}_{t,n}|\mathcal{X}_{t,n})\left(\sum_{t'=t}^{T-1}r_{t',n}-b(\mathcal{X}_t)\right).$$

where $\sum_{t'=t}^{T-1}r_{t',n}-b(\mathcal{X}_t)$ is used as an approximate estimate of the efficacy of the action $\mathbf{a}_{t,n}$. Ideally, we may train neural networks to convergence until the objective is fully optimized and all constraints are satisfied. However, empirically, hard constraints such as an indicator function in Eqn. (1) yields extremely sparse reward. We can not obtain any feedback if one constraint is violated. Thus, we relax the hard constraint from an indicator function to be a soft one using a hinge function, where models violating the constraint less are rewarded higher. Formally, we switch the binary constraint in Eqn. (1) to be a soft violation function:

$$\mathcal{V}(U_i) = p_i^{\max(\max(0,U_i/C_{ui}-1),\max(0,C_{li}/U_i-1))}, \tag{2}$$

where $p_i \in [0,1]$ is a base penalty hyperparameter used to control the strength of the soft penalty in terms of the degree of violation w.r.t. i_{th} constraint. [1]. Finally, we switch our reward function navigating the search in the constrained space as,

$$r_{t,n} = P_{t,n}\prod_{i=1}^{K}\mathcal{V}(U_i). \tag{3}$$

where K is the number of constraints. We train the policy network till convergence. Empirically, EPNAS always finds a set of models satisfying our original objective. We then select the best one, yielding a resource-constraint high-performance model. Later, we will introduce the designed policy network and the action space for progressive network generation.

3.2 Policy Network

Policy network, shown in Fig. 2, adapts any input network by progressively modifying its parameters (referred as the *scale action*), or by inserting/removing a layer (referred jointly as the *insert action*). At every training step t, rather than building the target network from scratch [33, 53], EPNAS modifies the architecture from preceding training step via previously described operations. This progressive search enables a more sample-efficient search.

We use a *network embedding* to represent the input neural network configuration. Initially, each layer of the input neural network is mapped to layer embeddings by a trainable embedding lookup table. Then, an LSTM layer (with a state size equal to the number of layers L) sequentially processes these layer embeddings and outputs a network embedding. Next, the network embedding is input to two different LSTMs that decide the scale, insert, and remove actions. The first LSTM, named as "scale LSTM", outputs the hidden states at every step t that correspond to a scaling action $\mathbf{a}_s$ from a scale lookup table which changes the filter width or change the number of filters, for the network. For example, in layer-by-layer search, we may partition the network into F parts, and for part f, the output $\mathbf{a}_{sf}$ can scale a group of filters simultaneously. In our experiments, this is more efficient than per-layer modification of ENAS [33], while more flexible than global modification of EAS [9]. The second LSTM, named as "insert LSTM", outputs hidden states representing actions $\mathbf{a}_i$,

---

[1]Each constraint may use a different base penalty hyperparameter in principle, but we simply use $p_i = 0.9$ for all architecture search experiments and show that it is effective to constrain the search for all constraints.
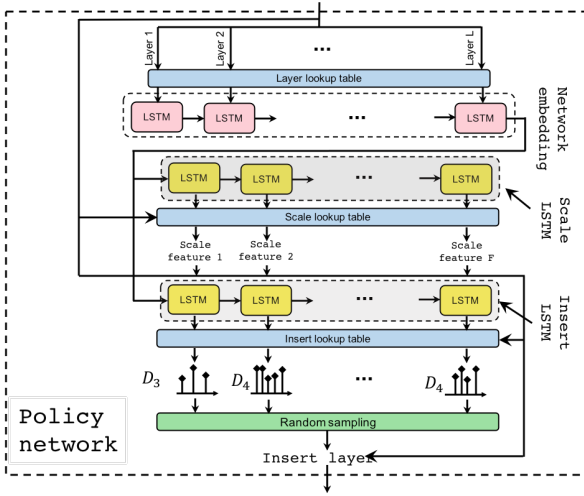
Figure 2: Policy Network of EPNAS. It is an LSTM-based network, which first generates network embedding, and then outputs actions to modify the network with a "Scale LSTM" and an "Insert LSTM". Details in Sec. 3.2.

which selects to do insert, remove, or keep a certain layer, *e.g.*, using a *conv* operation from the insert lookup table, at certain place l after the network is scaled. To encourage exploration of insert LSTM, *i.e.*, encourage inserting inside the network to explore inner structure, rather than always appending at the end of the network, we constrain the output distribution of inserting place l to be determinant on layer number of input network L using a check table. Formally, we let $\mathbf{P}(\mathbf{a}_i(p) = l) \sim \mathcal{D}_L$, where $\mathcal{D}_L$ stands for a discrete distribution with $L+5$ values. We use this prior knowledge, and found the architectures are more dynamically changed during search and able to find target model more efficiently.

3.3   Search patterns

Following common strategies [27, 53], EPNAS also supports two types of search patterns: (i) layer-by-layer search, which exhaustively modifies each layer of a network when dataset is not large, and (ii) module search, which searches a neural cell that can be arbitrarily stacked for generalization. The former is relatively slow, while can reach high performance, and the latter targets at architecture transformation applied on large datasets. We elaborate the two search patterns in this section.

In **Layer-by-layer search**, EPNAS progressively scales and inserts layers that are potentially with skip connection. Fig. 3 exemplifies an insert operation by the "Insert LSTM", which is applied to an input network. "Src1" determines the place to insert a given layer (a conv layer), and "Src2" indicates where to add a skip connection. Here, "Src2" can be *-1* to avoid a skip connection. Specifically, the search operations are chosen based on the problem domain. For example, one can include recurrent layers for discriminative speech models and convolution layers for discriminative image models. More details are explained in Section 4.

Module search aims to find an optimal small network cell which can be repeatedly stacked to create the overall neural network. It has relatively limited search space with multi-branch networks. The insert action by the "Insert LSTM" no longer inserts a layer, but inserts a "branch", and outputs the types of the operation and corresponding scaling parameters (filter width, pooling width, channel size, etc.). Similar to layer-by-layer search, each branch
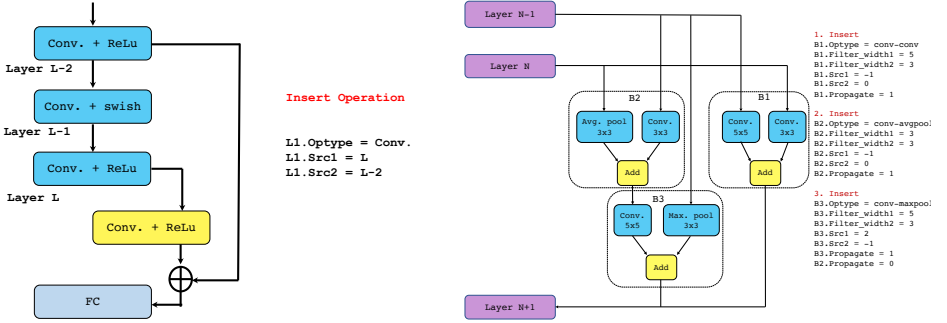
Figure 3: Left: A layer-by-layer search insert operation example. A conv operation is inserted after layer L, and has skip connection with layer L-2. Right: A module search insert operation example. When the branch 3 is inserted, one of its source value is from branch 2. After insertion, the connection between branch 2 and the next layer is cut off.

consists of two operations to be concatenated. We illustrate an insert operation in Fig. 3. Here, "Src1" and "Src2" determine where these two operations get input values from, and "propagate" determines whether the output of the branch gets passed to the next layer.

3.4 Speedup EPNAS with performance prediction

Training every sampled model till convergence can be time-consuming and redundant for finding relative performance ranking. We expedite EPNAS with a performance prediction strategy that provides efficient training and evaluation. Our approach is based on the observation that a similar learning pattern can be maintained as long as the learning rate and batch size are kept proportionally [58]. By scaling up the batch size, we can use a large learning rate with an aggressive decay, which enables accurate ranking with much fewer training epochs. Besides, since relative performance ranking is more important, learning rate and batch size ratio can be increased. Additionally, we also allow the generated networks to partially share parameters across steps. Differently, in our cases, we have additional operations rather than just insertion and widening in EAS. Therefore, we use a global dictionary that is shared by all models within a batch of searched models. The parameters are stored as key-value pairs, where the keys are the combination of the layer number, operation type, and the values are corresponding parameters. At the end of each step, if there is a clash between keys, the variables for the model with the highest accuracy is stored. For operations where the layer and operation types match, but the dimensions do not (e.g., after scaling a layer), the parameters with the closest dimensions are chosen and the variables are spliced or padded accordingly. Equipped with performance prediction, EPNAS can find the good models within much less time cost, which is critical for NAS with multiple resource constraints.

4 Experimental Evaluation

In this section, we evaluate the efficiency of EPNAS for handling large search space over two popular datasets: CIFAR-10 dataset [23] and ImageNet [1]. We first show model accuracy versus search time for five different search approaches used by EPNAS in Fig. 4. Then, we compare EPNAS with other related NAS algorithms in Tab. 4 and show its generalization to ImageNet. Finally, we compare EPNAS under three resource constraints, *i.e.* model size (sz), compute complexity ($c.c.$) and compute intensity ($c.i.$), with SoTA models in Tab. 5. Due to

Table 1: Search space of scale and insert actions

Table 2: Layer-by-layer search.

Feature	Search space
Layer type	[conv2d, dep-sep-conv2d, MaxPool2d, AvgPool2d, add]
Filter width	[1, 3, 5, 7]
Pooling width	[2, 3]
Channel size	[16, 32, 64, 96, 128, 256]
Nonlinear activation	["relu", "crelu", "elu", "selu", "swish"]
Src1 Layer	[i for i in range(MAX_LAYERS)]
Src2 Layer	[i for i in range(MAX_LAYERS)]

Table 3: Module search.

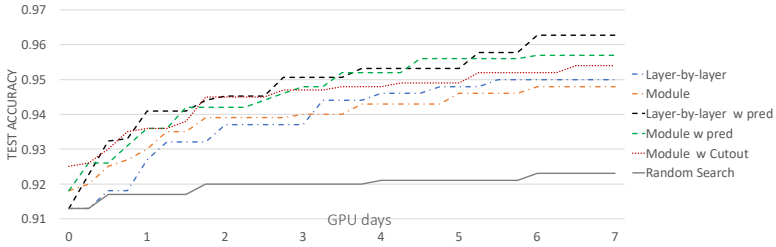| Feature	Search space
Branch type	[conv-conv, conv-maxpool, conv-avgpool, conv-none, maxpool-none, avgpool-none, 1×7-7×1-none]
Filter width	[1, 3, 5, 7]
Pooling width	[2, 3]
Channel size	[8, 12, 16, 24, 32]
Src1 Layer	[i for i in range(MAX_BRANCHES+1)]
Src2 Layer	[i for i in range(MAX_BRANCHES+1)]
Propagate	[0,1]

Figure 4: Accuracy VS. total search time for CIFAR-10. Note the accuracy reported here is only accuracy during searching. The final reported results in Tab. 4 and Tab. 5 are evaluated by training from scratch.

space limitation, we put the details of resource constraints, visualization of searched models for layer-by-layer and module search in our supplementary materials. In our supplementary files, we also experimented our algorithm to a language dataset, *i.e.* keyword spotting using Google Speech Commands dataset [1], and obtain SoTA results.

4.1 Experiments with CIFAR10

CIFAR-10 dataset contains 45K training images and 5K testing images with size of 32×32. Following PNAS [27], we apply standard image augmentation techniques, including random flipping, cropping, brightness, and contrast adjustments. For all our experiments, we start from a randomly generated architecture to fairly compare with other SoTA methods. We first illustrated the search space in Tables 1 for two search patterns, which is much larger than that proposed in EAS and PNAS, while we still have less GPU days for finding a good model.

Training details. Our policy network uses LSTMs with 32 hidden units for network embedding, while LSTMs with 128 hidden units for *scale* and *insert* actions. It is trained with the Adam optimizer with a learning rate of 0.0006, with weights initialized uniformly in [-0.1, 0.1]. In total, 8 branches are constructed for training. Each generated neural networks are trained for 20 epochs with batch size (BZ) of 128 using Nesterov momentum with a learning rate (LR) ($l_{max} = 0.05, l_{min} = 0.001, T_0 = 10, T_{mul} = 2$) following the cosine schedule[31]. With performance prediction (*p.p.*), we increase LR to 0.5 and BZ to 1024, and do early stop at 10 epochs for each model with parameter sharing. We use an episode step of 10 and 5 for layer-by-layer search and module search respectively, and select top 8 models from each episode to initialize the next. We run 15 episodes for searching the best model, which

Table 4: Comparison of EPNAS to automated architecture search literature for CIFAR-10. Note that ENAS GPU days are evaluated on our own server with author implementation, with the same system configuration with EPNAS's evaluation. M stands for million.

Model	Parameters	Test error (%)	Test error w cutout (%)	GPU days
AmoebaNet-B [56]	2.8 M	3.37	-	3150
NASNet-A [5]	3.3 M	3.41	2.65	1800
Progressive NAS [□]	3.2M	3.63	-	150
EAS (DenseNet on C10+) [□]	10.7 M	3.44	-	10
ENAS Macro Search [□]	21.3 M	4.23	-	1*
ENAS Micro Search [□]	4.6 M	3.54	2.89	1.5*
DARTS (first order) [□]	3.3 M	-	4.23	1.5
DARTS (second order) [□]	3.3 M	-	2.76	4
EPNAS Module Search	4.0 M	3.32	2.84	25
EPNAS Layer-by-Layer Search	3.4 M	3.87	-	16
EPNAS Module Search w prediction	4.3 M	3.2	2.79	8
EPNAS Layer-by-Layer Search w prediction	7.8 M	3.02	-	4

is retrained following PNAS, yielding our final performance.

Ablation study. Fig. 4 shows that EPNAS achieves the test accuracy up to 96.2% with $p.p.$ and 95% w/o $p.p.$ respectively after 6 GPU days, when started with a randomly generated model with a test accuracy of 91%. Both layer-by-layer search and module search significantly outperform random search, but layer-by-layer search slightly outperforms module search as it enables more fine-grained search space. Cutout marginally improves searched accuracy by 0.5-0.6%.

Quantitative comparison. Tab. 4 compares EPNAS (with model size constraint) to other notable NAS approaches. All techniques yield similar accuracy (within $\sim$0.5% difference) for comparable sizes. EPNAS with $p.p$ yields additional 1% accuracy gain compared to EPNAS w/o $p.p$ as it enables searching more models within the same search time budget. EPNAS significantly outperforms AmoebaNet, NASNet, PNAS, and EAS in search time with larger search space. Specifically, comparing with PNAS, our module search w $p.p$ train roughly 600 models (15 episodes 8 M40 GPUs and 5 steps for each episode) to obtain a good model using module search (20 min per model with 10 epochs), and PNAS trained 1160 models with longer time to reach similar performances. EPNAS slightly outperforms both ENAS Macro Search and ENAS Micro Search in terms of model accuracy and model size, but is slightly worse in search time.

Table 5: EPNAS with multiple resource constraints vs. the SoTA models on CIFAR10. sz means Model size (M). $c.i.$ means compute intensity (FLOPs/byte) and $c.c.$ means compute complexity (MFLOPs). Compute intensity is not compute complexity divided by model size. It reflects how models reuse data without loading data from slow memory. The higher compute intensity is, the better. MFLOPs stands for mega floating-point operations per second.

Model	Resource constraint	Test error (%)	Model size (sz)	Comp. intensity ($c.i.$) (FLOPs/byte)	Comp. complexity ($c.c.$) (MFLOPs)
ResNet50 [□]	-	6.97	0.86 M	32	10
DenseNet (L=40, k=12) [□]	-	5.24	1.02 M	43	21
DenseNet-BC (k=24) [□]	-	3.62	15.3 M	45	300
ResNeXt-29,8x64d [□]	-	3.65	34.4 M	58	266
MobileNetV2 [□]	-	5.8	1.9 M	10	10
DPPNet-PNAS [□]	-	5.8	1.9 M	10	10
MNASNet [□]	-	5.8	1.9 M	10	10
EPNAS: Module Search	$sz \leq 3M$	3.98	2.2 M	7.1	28
EPNAS: Layer-by-Layer Search	$sz \leq 2M,\ c.i. \geq 80$	4.31	1.7 M	97	26
EPNAS: Layer-by-Layer Search	$c.i. \geq 100\ \ c.c. \leq 200$	2.95	29 M	107	194
EPNAS: Layer-by-Layer Search	$sz \leq 8M,\ c.i. \geq 80,\ c.c. \leq 80$	3.48	7.7 M	92	72
EPNAS: Layer-by-Layer Search	$sz \leq 1M,\ c.i. \geq 30,\ c.c. \leq 15$	5.95	0.88 M	31	13

Table 6: Comparison of model generalization to ImageNet. Model error rate (%) is reported with Top-1 and Top-5 predictions.

Setting	Method	Top-1	Top-5	Params
Mobile	DPP-Net-Panacea [☐]	25.98	8.21	4.8M
	MnasNet-92 [☐]	25.21	7.95	4.4M
	PNASNet-5 [☐]	25.8	8.1	5.1M
	DARTS [☐]	26.7	8.7	4.7M
	EPNAS-mobile	**25.2**	**7.57**	4.2M
Large	DPP-Net-PNAS [☐]	24.16	7.13	77.16M
	EPNAS-large	21.49	5.16	78.33M

EPNAS with resource constraints. Table 5 compares EPNAS and other SoTA models with resource constraints. In particular, EPNAS is able to find model under 10M parameters with 3.48% test error with 92 FLOPs/Byte $c.i.$, and under 80 MFLOPs $c.c.$ With tight model size and FLOPs constraints, EPNAS is able to find model of similar size compared to ResNet50 but is about 1% more accurate. With relaxed model size, EPNAS finds SoTA model with 2.95% test error and high $c.i.$ of 107 FLOPs/byte, which outperforms ENAS macro search.

4.2   Results on ImageNet

We also have experiments which transfer optimal models from EPNAS module search in Tab. 4 to ImageNet borrowing the stacking structure following PNAS [27] with two settings: 1) *Mobile*: limiting the model size to be less than 5M parameters, 2) *Large*: without any model size constraint. For a fair comparison, we set the input image size to 224×224 for both settings. The upper part in Tab. 6 shows the results of *mobile* setting. We compare against other SoTA resource-aware NAS methods, DPPNet [12] and MNASNet [43], and DARTS [28]. Our model based on the searched module yields better results with similar model size. For *large* setting, our results is also significantly better than that reported in DPP-Net, while we omit that from PNAS since the input setting is different.

5   Conclusions

We propose efficient progressive NAS (EPNAS), a novel network transform policy with RE-INFORCE and an effective learning strategy. It supports large search space and can generalize well to NAS multiple resource constraints through a soft penalty function. EPNAS can achieve SoTA results for image recognition and KWS even with tight resource constraints.

References

[1] Launching the speech commands dataset. https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html.

[2] D. Amodei et al. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *arXiv:1512.02595*, December 2015.

[3] Peter J Angeline et al. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

[4] Bowen Baker et al. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[5] Bowen Baker et al. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

[6] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 549–558, 2018.

[7] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2188385.2188395.

[8] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

[9] Han Cai et al. Efficient architecture search by network transformation. AAAI, 2018.

[10] F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv: 1610.02357*, October 2016.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. *ECCV*, 2018.

[13] Thomas Elsken et al. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[14] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018.

[15] Song Han et al. Learning both weights and connections for efficient neural networks. NIPS, pages 1135–1143, 2015.

[16] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[17] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[18] Andrew G. Howard et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

[19] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[20] Itay Hubara et al. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv:1609.07061*, 2016.

[21] Kirthevasan Kandasamy et al. Neural architecture search with bayesian optimisation and optimal transport. *CoRR*, abs/1802.07191, 2018. URL http://arxiv.org/abs/1802.07191.

[22] T. Karras et al. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *arXiv: 1710.10196*, October 2017.

[23] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[24] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[25] 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.

[26] Baoyuan Liu et al. Sparse convolutional neural networks. In *CVPR*, pages 806–814, June 2015.

[27] Chenxi Liu et al. Progressive neural architecture search. In *ECCV*, pages 19–34, 2018.

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

[29] Hanxiao Liu et al. Hierarchical representations for efficient architecture search. *ICLR*, 2018.

[30] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. URL http://arxiv.org/abs/1608.03983.

[31] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164*, 2018.

[32] Renato Negrinho and Geoff Gordon. DeepArchitect: Automatically Designing and Training Deep Architectures. 2017. URL http://arxiv.org/abs/1704.08792.

[33] Hieu Pham et al. Efficient Neural Architecture Search via Parameter Sharing. 2018. URL http://arxiv.org/abs/1802.03268.

[34] Hieu Pham et al. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[35] Esteban Real et al. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[36] Esteban Real et al. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018.

[37] Williams R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. 1992.

[38] Chris Ying Quoc V. Le Samuel L. Smith, Pieter-Jan Kindermans. Don't decay the learning rate, increase the batch size. *ICLR*, 2018.

[39] M. Sandler et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv: 1801.04381*, January 2018.

[40] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556*, September 2014.

[41] V. Sindhwani et al. Structured Transforms for Small-Footprint Deep Learning. *arXiv:1510.01722*, October 2015.

[42] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[43] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

[44] Frank Hutter Thomas Elsken, Jan Hendrik Metzen. Multi-objective architecture search for cnns. *CoRR*, 2018. URL https://arxiv.org/abs/1804.09081.

[45] A. van den Oord et al. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. *arXiv:1711.10433*, November 2017.

[46] Y. Wu, Schuster, et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv:1609.08144*, 2016.

[47] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[48] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*, 2019.

[49] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: Stochastic neural architecture search. *ICLR*, 2019.

[50] K. Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *arXiv: 1502.03044*, February 2015.

[51] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.

[52] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. 2016. ISSN 1938-7228. doi: 10.1016/j.knosys.2015.01.010. URL http://arxiv.org/abs/1611.01578.

[53] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *CVPR*, 2018.