# One-Shot Neural Architecture Search via Self-Evaluated Template Network

Xuanyi Dong[†‡] and Yi Yang[‡]
[†]Baidu Research, [‡]ReLER, University of Technology Sydney
xuanyi.dong@student.uts.edu.au, yi.yang@uts.edu.au

## Abstract

*Neural architecture search (NAS) aims to automate the search procedure of architecture instead of manual design. Even if recent NAS approaches finish the search within days, lengthy training is still required for a specific architecture candidate to get the parameters for its accurate evaluation. Recently one-shot NAS methods are proposed to largely squeeze the tedious training process by sharing parameters across candidates. In this way, the parameters for each candidate can be directly extracted from the shared parameters instead of training them from scratch. However, they have no sense of which candidate will perform better until evaluation so that the candidates to evaluate are randomly sampled and the top-1 candidate is considered the best. In this paper, we propose a Self-Evaluated Template Network (SETN) to improve the quality of the architecture candidates for evaluation so that it is more likely to cover competitive candidates. SETN consists of two components: (1) an evaluator, which learns to indicate the probability of each individual architecture being likely to have a lower validation loss. The candidates for evaluation can thus be selectively sampled according to this evaluator. (2) a template network, which shares parameters among all candidates to amortize the training cost of generated candidates. In experiments, the architecture found by SETN achieves the state-of-the-art performance on CIFAR and ImageNet benchmarks within comparable computation costs.*

先前one-shot的总结

## 1. Introduction

Representation learning [5] is a fundamental research problem in computer vision, because it is beneficial to a variety of computer vision applications, such as detection and segmentation. Due to the success of deep learning, it has undergone a transition from "feature engineering" [27, 10] to "architecture engineering" [35, 19, 34, 26, 18, 12]. However, a large amount of expert knowledge and ample computational resources are still required to secure an architecture for good feature representations [42]. Fortunately, neural architecture search (NAS) brings hope to deep learning researchers and alleviate their labours [43, 29].

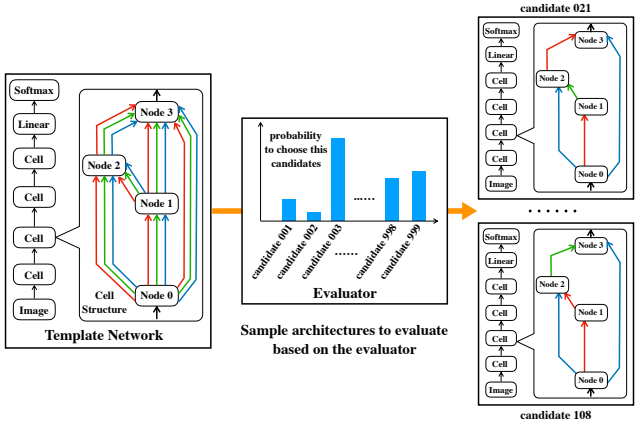


Figure 1. An overview of the self-evaluated template network (SETN). SETN consists of a template network and an evaluator. Architectures for evaluation are generated by sampling candidates in the template network using the evaluator. The template network shares the parameters for different candidates as indicated by connections of arrow lines in different colors. The evaluator learns the distribution of the architectures being likely to have a lower validation loss.

The goal of NAS is to discover an optimal network in the search space, which can maximize the validation accuracy after training. Typical algorithms apply reinforcement learning (RL) [42, 43, 29, 9] or evolutionary strategy (EA) [24, 30] to solve this problem, and most are computationally expensive, e.g., 500 GPUs over four days [43]. Such huge computational costs motivate the researchers to focus on efficient architecture search algorithms. Recently, several works reduced the computational cost through weight sharing [29, 7], weight generation [6, 4], accuracy prediction [2, 21], progressive strategy [23], etc.

One-shot NAS approaches stand out among these efficient NAS approaches [4, 6], because they can significantly squeeze the tedious training process by sharing parameters across architecture candidates. A typical one-shot NAS paradigm is to (1) randomly sample hundreds of archi-

SMASH,Understand One-Shot

*This work was done when Xuanyi Dong was a research intern with Baidu Research.

tecture candidates from the parameter-shared network; (2) evaluate these candidates; and (3) find the candidate with the highest validation accuracy. We observed that most of these randomly sampled candidates are useless, since most of them have a poor performance. Besides, these sampled candidates are only a small portion of the whole search space; therefore the probability for inclusion of the best architecture is extremely low.

To solve the above problem, we propose *Self-Evaluated Template Network (SETN)* for one-shot NAS. SETN equips a template network with an evaluator. The *template network* contains all possible candidate convolutional neural networks (CNNs) and shares its parameters (template parameters) with all candidates. We train this template network in *a stochastic strategy*: during one iteration, we uniformly sample one candidate and only optimize partial template parameters for this sampled candidate. In this way, after training, each candidate CNN can directly use the corresponding template parameters without additional training. Some previous methods coupled the shared parameters with a learnable distribution of architectures, such as [25, 14, 29]. These methods will introduce the bias into the template parameters. Since some shallow and light-weight network will quickly converged, the learnable distribution will bias to these "simple" networks, and other networks may not have a chance to be updated. In contrast to them, our uniformly stochastic training strategy allows each candidate to be treated equally. In this way, each candidate with shared template parameters would be trained fully, and their validation accuracy would be closer to the ground truth validation accuracy.

The *estimator* learns to indicate the probability of each individual candidate CNN being likely to have a lower validation loss. We train this estimator on the validation data with the assistance of the template network. After training, according to the learned probability, our estimator can pick up low-validation-loss candidates for one-shot evaluation. As a result, the probability of including the best CNN in the search space can be dramatically improved. Compared to previous random sampling algorithms [4, 6], SETN can potentially search for a CNN with a higher accuracy.

In experiments, SETN can discover a superior CNN on CIFAR-10 within two GPU days. This SETN-searched architecture achieves state-of-the-art performance on three benchmarks, i.e., CIFAR-10, CIFAR-100, and ImageNet.

## 2. Related Work

Automatically discovering effective networks has attracted more and more researchers [6, 1, 42, 24, 15, 13]. Various kinds of searching algorithms have been proposed, such as RL-based [42, 43, 29], EA-based [30, 31], gradient-based [25, 28] approaches. Each of these algorithm has its unique advantage. For simplicity, we summarize some ubiquitous algorithms in Table 1 regarding five aspects. Our SETN has advantages when compared to them, and we will introduce these related works at below. Note that we focus on searching CNN models, and thus approaches about recurrent neural network or downstream applications are out of the scope of this paper.

| | Search | Eff. | Share | Eval. | Gen. |
|---|---|---|---|---|---|
| ResNet [18] | Manual | × | – | – | – |
| MetaQNN [1] | RL | × | × | – | – |
| NASNet [43] | RL | × | × | – | – |
| AmoebaNet [30] | EA | × | × | – | – |
| H-NAS [24] | EA | × | × | – | – |
| EAS [7] | RL | √ | √ | – | – |
| PNAS [23] | SMBO | × | × | slow | – |
| SMASH [6] | Gradient | √ | × | quick | random |
| Understand [4] | Gradient | √ | √ | quick | random |
| DARTS [25] | Gradient | √ | √ | – | – |
| GDAS [14] | Gradient | √ | √ | – | – |
| Our SETN | Gradient | √ | √ | quick | selective |

Table 1. We compare different algorithms with five aspects. "Search" shows the search algorithm type. "Eff." indicates efficient, and we consider the algorithm that can discover CNN within five GPU days as efficient. "Share" indicates whether the algorithm shares parameters over different candidate networks or not. "Eval." means whether the algorithm is able to quickly evaluate a network. "Gen." indicates the network sampling strategy during *the one-shot evaluation procedure*. "-" indicates not available, which means those methods do not have these steps.

Early approaches train a large amount of candidate networks by tens epochs and use the validation accuracy of these networks as the supervisory signal [24, 30, 42, 43, 31]. For example, Zoph et al. [42, 43] learn an RL policy to sample networks with high accuracy. Real et al. [31, 30] utilize EA algorithms to mutate low-quality networks to high-quality networks. Unfortunately, these approaches cost too much computational resources. Recent approaches aim to solve the searching problem in an affordable computation cost. Liu et al. [23] progressively search CNN from simple to complex. Barker et al. [2] accelerate the searching procedure by performance prediction. Pham et al. [29] share the parameters of different networks. Liu et al. [25] propose DARTS, allowing efficient search of the architecture using gradient descent. DARTS [25] and GDAS [14] choose the best architecture using the $\arg\max$ over a continues architecture representation, while the performance of an architecture cannot be correctly estimated without fully training it. Our SETN can more accurately estimate the performance (validation loss) of all candidates without separately training each of them one by one.

Our SETN is one-shot NAS, and is closely related to previous one-shot approaches [40, 4, 6]. Brock et al. [6] train hypernetworks [16] to generate suitable weights for

every network in the search space. Zhang et al. [40] encode the network as a computation graph and use a graph neural network to predict weights. Bender et al. [4] deliver thorough experimental analysis for one-shot architecture search. These approaches can estimate the network performance correctly without additional training, while the architecture candidates for evaluation are randomly picked with uneven quality [4, 6]. Our approach can selectively sample low-validation-loss architecture candidates. In this way, our sampled candidates would have much lower validation loss (better performance) than that of the random strategy [4, 6], and thus we can potentially discover a more effective CNN architecture.

## 3. Background

Early works search for the whole CNN structure [42, 3], whereas recent works propose that finding a good neural cell is more effective than finding a whole CNN [23, 25, 40, 14]. Therefore, we also search for a good cell instead of a full CNN model. As shown in Figure 1, a cell is a fully convolutional structure, mapping a tensor $\boldsymbol{I}_{in} \in \mathcal{R}^{H\times W\times C}$ to another tensor $\boldsymbol{I}_{out} \in \mathcal{R}^{H'\times W'\times C'}$. If we use the stride of 1, the cell is named normal cell, which has $(H', W', F')$=$(H, W, F)$; and if we use the stride of 2, the cell is named reduction cell, which has $(H', W', F')$=$(\frac{H}{2}, \frac{W}{2}, 2F)$. Each cell contains $B$ nodes, where each of them is specified as a quadruple $(\boldsymbol{I}_1, \boldsymbol{I}_2, f_1, f_2)$ [23]. Specifically, the $i$-th node in the $c$-the cell takes two inputs $\boldsymbol{I}_1, \boldsymbol{I}_2 \in \mathcal{I}_i^c$ and generates a tensor $\boldsymbol{H}_i^c = f_1(\boldsymbol{I}_1) + f_2(\boldsymbol{I}_2)$. $f_1, f_2 \in \mathcal{O}$ are transformation functions to apply to inputs. The output of the $c$-th cell is the concatenation of each intermediate output tensor from each node, denoted as $\boldsymbol{H}^c$.

The set of possible inputs $\mathcal{I}_i^c$ is the output set of all previous nodes adding the outputs of two previous cells: $\mathcal{I}_i^c = \{\boldsymbol{H}_1^c, ..., \boldsymbol{H}_{i-1}^c, \boldsymbol{H}^{c-1}.\boldsymbol{H}^{c-2}\}$. The candidate function set $\mathcal{O}$ contains several pre-defined functions. In this paper, we apply our SETN on the candidate function set $\mathcal{O}$ following previous methods [25, 14] as follows:

- 3x3 max pooling
- 3x3 avg pooling
- skip connection
- 3x3 separable conv
- 5x5 separable conv
- 1x3 & 3x1 conv

We set the number of nodes in a cell as $B = 4$. Therefore, the number of candidates in the search space of $\mathcal{O}$ is $(1 \times 3 \times 6 \times 10 \times (6^2)^4)^2 = 9.1 \times 10^{16}$.

Once we obtain the topology structures of the normal cell and the reduction cell, we follow previous works to construct the overall CNN [23, 25, 14]. For CIFAR, the overall CNN is [image] → [N-Cell]×N → [R-Cell] → [N-Cell]×N → [R-Cell] → [N-Cell]×N → [Softmax]; and for ImageNet, the overall CNN is [image] → [a pair of 3x3 Conv] → [3x3 Conv] → [N-Cell]×N → [R-Cell] → [N-Cell]×N → [R-Cell] → [N-Cell]×N → [Softmax], where [N-Cell] and [R-Cell] indicate the normal and reduction cells, respectively.

## 4. Methodology

### 4.1. Template Network

The template network contains all candidate CNNs in the search space. The parameters of each candidate CNN are shared by a single template network. It is non-trivial to make billions of candidate CNNs perform well after optimizing one template network. To achieve this goal, we introduce a stochastic training strategy to optimize the template network by stochastically selecting the operations and inputs as below:

$$\boldsymbol{H}_i^c = f_1(\boldsymbol{I}_1) + f_2(\boldsymbol{I}_2), \tag{1}$$

$$\text{s.t. } \{\boldsymbol{I}_1, \boldsymbol{I}_2\} = \text{UniformSample}(\mathcal{I}_i^c, 2)^1, \tag{2}$$

$$\{f_1, f_2\} = \text{OrderedUniformSample}(\mathcal{O}, 2)^2, \tag{3}$$

where , at the $i$-th node, we randomly sample two inputs $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ from the set $\mathcal{I}_i^c$ with replacement; random sample two functions $f_1$ and $f_2$ from the set $\mathcal{O}$ by restricting the index of $f_1$ in $\mathcal{O} \leq$ the index of $f_2$. This sample strategy can avoid the redundant candidates in the search space. For example, ($f_1$=3x3 conv,$f_2$=5x5 conv,$\boldsymbol{I}_1$=$\boldsymbol{H}_1^c$,$\boldsymbol{I}_2$=$\boldsymbol{H}_1^c$) is the same architecture as ($f_1$=5x5 conv,$f_2$=3x3 conv,$\boldsymbol{I}_1$=$\boldsymbol{H}_1^c$,$\boldsymbol{I}_2$=$\boldsymbol{H}_1^c$), but these two combinations are considered as different architectures during searching in some previous works [25, 29, 14].

At each training iteration, the template network uniformly samples a candidate CNN, decided by Eq. (1)∼Eq. (3), and then it only optimizes template parameters of this sampled CNN. This strategy allows us to optimize each candidate with equal opportunity, thus avoiding the Matthew effect. As a result, each candidate CNN is more likely to be fully trained compared to that in previous joint optimization strategies [25, 14, 8]. We use "the Matthew effect" to refer that some quickly-converged candidates will get more chances to be further optimized in some NAS algorithms [25, 14, 8, 29]. Besides, if we increase the cardinality of the function set $|\mathcal{O}|$, the search space will grow exponentially, but the size of the template network will grow only linearly. This property allows us to search over large search space but only using a relatively small template network.

### 4.2. Estimator

We only optimize the template parameters on the trraining data, and a candidate CNN would thus be considered to

[2] $\text{UniformSample}(S, N)$ indicates a set of $N$ elements chosen randomly from set $S$ with replacement via a uniform distribution.

[3] $\text{OrderedUniformSample}(S, N)$ indicates a set of $N$ elements chosen randomly from set $S$ via a uniform distribution, in the mean time, the index of a later sampled element should be not less than the index of former element.

**Algorithm 1** The Searching Algorithm of SETN

**Input:** the whole available training data
a template network with $\omega$ and an estimator with $\alpha$
Split the whole available training data into the training set $\mathcal{D}_{train}$ and the validation set $\mathcal{D}_{val}$ for searching
**while** not converge **do** ▷ Optimize $\omega$ and $\alpha$
Sample training batch $\mathcal{D}_t = \{(x_i, y_i)\}_{i=1}^{batch}$ from $\mathcal{D}_{train}$
Calculate $\ell_{train} = \sum_{\mathcal{D}_t} \ell(x_i, y_i)$ based on Eq. (1)
Update $\omega$ via gradients from the training loss $\ell_{train}$
Sample validation batch $\mathcal{D}_v = \{(x_i, y_i)\}_{i=1}^{batch}$ from $\mathcal{D}_{val}$
Calculate $\ell_{val} = \sum_{\mathcal{D}_v} \ell(x_i, y_i)$ based on Eq. (10)
Update $\alpha$ via gradients from the validation loss $\ell_{val}$
**end while**
After the above steps, we obtain the optimized $\omega$ and $\alpha$.
Initialize $\mathcal{A} = \phi$ ▷ Obtain Low-Validation-Loss Candidates
**for** i=1; i $\leq T$; i++ **do**
Sample an architecture $\boldsymbol{a}$ using Eq. (4) to Eq. (9)
$\mathcal{A} = \mathcal{A} \supset \{\boldsymbol{a}\}$
**end for**
Evaluate all candidates in $\mathcal{A}$ with parameters extracted from $\omega$
Select the candidate with the lowest validation loss
**Output:** the final selected candidate

generalize well if it can use learned template parameters to yield a low validation loss. To find the best CNN, a trivial solution is traversing all candidates and evaluate them one by one, yet it would cost unaffordable computation time to cross over $10^{16}$ candidates. Some one-shot methods [4, 6] uniformly select a small amount of candidates to evaluate, where most uniformly selected candidates are useless. To solve these issues, we design an estimator to indicate the probability of each individual candidate CNN being likely to have a lower validation loss. To represent this probability of each candidate, we encode one candidate CNN as a set of quadruples, and then define the probabilities over these quadruples.

We introduce the encoding approach for the $i$-th node in the $c$-th cell, and one can easily infer the steps for other nodes. For simplicity, suppose we only search for one neural cell. At first, we encode the choices of $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$. Based on Eq. (2), there are $|\mathcal{I}|$ choices for $\boldsymbol{I}_1$ and $|\mathcal{I}|$ choices for $\boldsymbol{I}_1$ (we omit the subscript and superscript for simplicity). We thus use two vectors $\boldsymbol{f} \in \mathcal{R}^{|\mathcal{I}|}$ and $\boldsymbol{g} \in \mathcal{R}^{|\mathcal{I}|}$ to indicate the categorical choice for $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$, and use its softmax-normalized value as the choice probability, which can be formulated as:

$$\hat{\boldsymbol{f}} = \text{softmax}(\boldsymbol{f}) \ ; \ \hat{\boldsymbol{g}} = \text{softmax}(\boldsymbol{g}), \tag{4}$$

$$t \sim \mathcal{T}(\hat{\boldsymbol{f}}) \ \ ; \ \ u \sim \mathcal{T}(\hat{\boldsymbol{g}}), \tag{5}$$

$$\boldsymbol{I}_1 = \mathcal{I}_{(t)} \ \ ; \ \ \boldsymbol{I}_2 = \mathcal{I}_{(u)}, \tag{6}$$

where $\mathcal{T}(\hat{\boldsymbol{f}})$ represents the categorical distribution drawn by the vector $\boldsymbol{f}$, so as $\mathcal{T}(\hat{\boldsymbol{g}})$. $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ are chosen as the $t$-th and $u$-th element in $\mathcal{I}$. Similarly, there are $|\mathcal{O}|(|\mathcal{O}|+1)/2$ combination choices for $f_1$ and $f_2$, and therefore, we leverage a vector $\boldsymbol{h} \in \mathcal{R}^{|\mathcal{O}|(|\mathcal{O}|+1)/2}$ to indicate the categorical choice, which is formulated as:

$$\hat{\boldsymbol{h}} = \text{softmax}(\boldsymbol{h}) \ \ \rightarrow \ \ r \sim \mathcal{T}(\hat{\boldsymbol{h}}), \tag{7}$$

$$r1 = \min\left\{n \in [\,|\mathcal{I}|\,] : \sum_{k=1}^{n} k \geq r\right\} ; \ r2 = r - \sum_{k=1}^{r1\text{-}1} k, \tag{8}$$

$$f_1 = \mathcal{O}_{(r1)} \ ; \ f_2 = \mathcal{O}_{(r2)}, \tag{9}$$

where $\mathcal{T}(\hat{\boldsymbol{h}})$ represents the categorical distribution drawn by the vector $\hat{\boldsymbol{h}}$. $\mathcal{O}_{(t)}$ and $\mathcal{O}_{(u)}$ are the $t$-th and $u$-th functions in $\mathcal{O}$, respectively. Eq. (8) guarantees the indexes $r2 \leq r1$, which is consistent with Eq. (3). Eq. (4) to Eq. (9) can sample one quadruple $(r1, r2, t, u)$ for one node based on the probabilities $\hat{\boldsymbol{f}}$, $\hat{\boldsymbol{g}}$, and $\hat{\boldsymbol{h}}$, which are encoded by $\boldsymbol{f}$, $\boldsymbol{g}$, and $\boldsymbol{h}$. The set of quadruples for all nodes $\{(r1, r2, t, u)\}$ can represent one candidate architecture in the search space.

To enable the estimator being able to indicate whether a candidate could result in a low validation loss, we need to optimize the parameters of this estimator $\alpha = \{(\boldsymbol{h}, \boldsymbol{f}, \boldsymbol{g})\}$ on the validation set. Since Eq. (4) to Eq. (9) are discrete, we use continues relaxation to calculate the output $\boldsymbol{H}_i^c$ of a node, as follows:

$$\boldsymbol{H}_i^c = \sum_{r=1}^{\frac{(|\mathcal{O}||\mathcal{O}|+1)}{2}} \hat{\boldsymbol{h}}_{(r)}$$

$$\times(\sum_{t=1}^{|\mathcal{I}|} \hat{\boldsymbol{f}}_{(r1)} \mathcal{O}_{(r1)}(\mathcal{I}_{(t)}) + \sum_{u=1}^{|\mathcal{I}|} \hat{\boldsymbol{g}}_{(u)} \mathcal{O}_{(r2)}(\mathcal{I}_{(u)})), \tag{10}$$

$$\text{s.t.} \ \ \text{calculate } r1 \text{ and } r2 \text{ as Eq. (8)}, \tag{11}$$

Based on Eq. (10), we can back-propagate gradients through the architecture encoding $\alpha$. To enforce the learned estimator being able to reflect the validation loss of an architecture candidate, our objective for this estimator is to minimize the validation loss. Specifically, we forward the validation images through the template network with assistance of the estimator via Eq. (10), and backward the validation loss[3] to the estimator's parameters. In this way, after optimizing the estimator, candidates sampled by the learned probabilities (Eq. (5) and Eq. (7)) would be more likely to result in a high performance on the validation set.

### 4.3. The Searching Algorithm of SETN

We use $\omega$ to denote the parameters of the template network, i.e., the parameters of candidate functions in each node of each cell. The searching algorithm of SETN should (1) optimize parameters of the template network $\omega$ and parameters of the estimator $\alpha$; (2) sample $T$=1000 low-validation-loss architecture candidates via the estimator; and (3) evaluate these sampled candidates with the template parameters $\omega$ and choose the candidate with the lowest validation loss.

[3] the validation loss could be a simple softmax with cross-entropy loss, and could also integrate other constrains about latency or memory size [8]

We show the overall searching algorithm in Algorithm 1, where $\ell(x, y)$ indicates the standard classification loss based on an input image $x$ with its label $y$. We optimize $\alpha$ based on Eq. (10) and $\omega$ based on Eq. (1) in an alternative way. The parameters of the template network are optimized on the training set, while the parameters of the estimator are optimized on the validation set to guarantee the generalization ability of the searched model [25].

**Note that** we use different forward procedures for the template network and the estimator: the template network uses Eq. (1) and can enable each candidate perform well with the shared template network; the estimator uses Eq. (10) can help squeeze the candidate set for evaluation. After optimizing $\alpha$ and $\omega$, we sample $T$=1000 low-validation-loss candidates and select the one with the best one-shot performance.

### 4.4. Connections with Other NAS Approaches

Our proposed SETN generalized over DARTS [25] and one-shot NAS approaches [4, 6]. DARTS directly can pick the best architecture, while this network capacity can not be correctly estimated in the validation set without fully training it. Besides, the architecture found by DARTS yields a performance with a high variance. Comparatively, one-shot NAS can estimate the network performance correctly without additional training, while the architecture candidates are randomly picked with uneven quality instead of generating the "best" candidate [4, 6]. Our SETN is a new framework, which generalizes the above two typical streams and assimilates their benefits of accurate evaluation and high-quality architecture candidates selection.

To analyze the difference of technique details between our approach and others [25], we consider the following variants of our methods.

**SETN**: our proposed search algorithm.

**SETN-LR**: use a stochastic strategy to train the template network with less randomness, where the indexes of the sampled inputs/functions are the same for different cells.

**SETN-NON**: optimize the template network without randomness, in which $\boldsymbol{H}_i^c$ is the weighted sum of all possible function and input combinations as Eq. (10).

**SETN-RAND**: randomly sample candidates for evaluation as previous one-shot approaches.

SETN-NON is the same strategy as [25], and SETN-RAND is the same strategy as [4, 6]. We will show that SETN is superior to SETN-LR, SETN-NON, and SETN-RAND in experiments.

## 5. Experiments

### 5.1. Experimental Setup

**Datasets.** CIFAR-10 [22] contains 60,000 images categorized into 10 classes. The training set has 5000 images per class, 50,000 images in total. The test set contains 1000 images per class, 10,000 images in total. CIFAR-100 [22] is similar to CIFAR-10. It contains 50,000 training and 10,000 test images, categorized into 100 classes. All images are 32x32 colored ones. ImageNet [32] is a large-scale image classification dataset, containing 1000 classes, 1.28 million training images and 50,000 validation images.

**Searching Setup.** We search the normal CNN cell and the reduction CNN cell on CIFAR-10. For searching, the official training images are randomly split into the searching training set $\mathcal{D}_{train}$ and the search validation set $\mathcal{D}_{val}$ in Algorithm 1. $\mathcal{D}_{train}$ contains 50% of the official CIFAR-10 dataset, i.e., 25,000 images. $\mathcal{D}_{val}$ contains the rest images. The candidate function set $\mathcal{O}$ has 6 different operations as introduced in Section 3. The hyper-parameters to construct the whole CNN model are: the number of nodes in a cell $B$, the initial channels of the first layer $C$, the number of repeated normal cells $N$. By default, we set $C = 16$, $B = 4$, and $N = 2$ to search the CNN cells. Note that, the number of operations in $\mathcal{O}_s$ is the same as ENAS [29] and DARTS [25].

We train the template network and the estimator with the batch size of 64 in 400 epochs. To optimize the parameters $\omega$ of template network, we use the SGD optimization. We start the learning rate of 0.025 and anneal it down to 0 following a cosine schedule. We use the momentum of 0.9 and the weight decay of 3e-4. We set the probability of path dropout as 0.1. To optimize the parameters $\alpha$ of the estimator, we use the Adam optimization [20] with the learning rate of 3e-3 and the weight decay of 1e-3. To avoid the gradient explosion, we clip the gradient for both $\mathbb{W}$ and $\mathbb{A}$ by 10 during training.

**Computational Costs.** Our SETN would take about 40 hours to optimize both template network and estimator on a single NVIDIA Tesla V100 GPU. Evaluating $T$=1000 candidates costs less than three hours on a single GPU (about 13 seconds per candidate). Therefore, it requires about 43 GPU hours to obtain the final CNN structure. Note that when different NAS methods report their searching costs, they may use different hardware, e.g., NVIDIA GTX 1080Ti [25] and NVIDIA P100 [43]. We did not normalize the GPU cost of compared methods across different devices, and we use the numbers from their original papers.

### 5.2. Compared with State-of-the-art Approaches

**Experiments on CIFAR.** After we discover outstanding cells in the search space, we use the discovered topology with $C = 36$ and $N = 6$ to construct the CNN model for CIFAR-10 and CIFAR-100 following [25, 28]. We train this network by 600 epochs with the initial learning rate of 0.025. We anneal the learning rate down to 0 with the cosine schedule. The batch size is 96; the momentum is 0.9; and the weight decay is 5e-4. The ratio of drop path is 0.2.

| Method | GPU Days | M | C | Parameters | Error on CIFAR-10 (%) | Error on CIFAR-100 (%) |
|---|---|---|---|---|---|---|
| DenseNet-BC [19] | – | – | – | 25.6 MB | 3.46 | 17.18 |
| PyramidNet [17] | – | – | – | 26.0 MB | 3.31 | 16.35 |
| MetaQNN [1] | >80 | – | – | 11.2 MB | 6.92 | 27.14 |
| Net Transformation [7] | 10 | – | – | 19.7 MB | 5.70 | – |
| SMASH [6] | 1.5 | – | – | 16.0 MB | 4.03 | – |
| Hierarchical NAS [24] | 300 | 6 | 64 | – | 3.75 | 20.3 |
| Progressive NAS [23] | 150 | 11 | 48 | 3.2 MB | 3.63 | 19.53 |
| NASNet-A [43] | 2000 | 20 | 32 | 3.3 MB | 3.41 | 19.70 |
| AmoebaNet-A [30] | 3150 | 20 | 36 | 3.2 MB | 3.34 | – |
| ENAS [29] | 0.45 | 20 | 36 | 4.6 MB | 3.54 | 19.43 |
| NAONet [28] | 200 | 20 | 36 | 10.6 MB | 3.18 | – |
| NASNet-A + CutOut [43] | 2000 | 20 | 32 | 3.3 MB | **2.65** | 17.81† |
| PNAS + CutOut [23] | 150 | 11 | 48 | 3.2 MB | – | 17.63 |
| DARTS + CutOut [25] | 4 | 20 | 36 | 3.4 MB | 2.83 | – |
| GHN + CutOut [40] | 0.84 | 18 | 32 | 5.7 MB | 2.84 | – |
| ENAS + CutOut [29] | 0.45 | 20 | 36 | 4.6 MB | 2.89 | 18.91† |
| GDAS + CutOut [14] | 0.84 | 20 | 36 | 3.4 MB | 2.93 | 18.38 |
| SETN-LR ($T$=1K) + CutOut | 1.8 | 20 | 36 | 5.5 MB | 2.81 | 17.88 |
| SETN-NON ($T$=1K) + CutOut | 1.8 | 20 | 36 | 3.7 MB | 3.12 | 18.27 |
| SETN ($T$=1) + CutOut | 1.7 | 20 | 36 | 4.5 MB | 3.41 | 18.12 |
| SETN ($T$=1K) | 1.8 | 20 | 36 | 4.6 MB | 3.56 | 19.38 |
| SETN ($T$=1K) + CutOut | 1.8 | 20 | 36 | 4.6 MB | 2.69 | **17.25** |

Table 2. We compare SETN and other algorithms on CIFAR-10 and CIFAR-100. The top block presents state-of-the-art architectures designed by human experts. The bottom block presents architectures that are automatically discovered by machine. "$M$" indicates the total number of cells in the CNN, and "$C$" denotes the number of the filter channel in the first cell. "CutOut" indicates the data argumentation approach [11]. † denotes the results reproduced by ourself. The bottom five lines show results for different variants of our approach. We run each model three times and report the mean error (lower is better).

Following [23, 42, 43, 25, 14], we use an auxiliary tower with the weight of 0.4 to train the network. We train each network three times and report the mean error.

**Comparison with the state-of-the-art on CIFAR-10 and CIFAR-100.** We compare the results of the found network with other state-of-the-art networks in Table 2. "SETN ($T$=1K)" indicates the network found by our approach. First, our SETN is one of the most efficient algorithms, in which we complete the search procedure in 1.8 GPU days. Second, among those efficient NAS approaches [40, 29, 25] (less than five GPU days), the network found by SETN achieves the lowest error with similar or fewer parameters. Other NAS approaches need more than 100 times computational costs than ours, whereas models found by most approaches have higher error with more parameters than our SETN. On CIFAR-100, our network achieves the best performance (a error of 17.25%) among all compared methods. On CIFAR-10, our network is slightly worse than NASNet-A (2.69% test error vs. 2.65% test error), however, NASNet-A needs more than 1000× computational costs than SETN.

**Comparison with other SETN variants.** In Table 2, we compare several variants of SETN introduced in Section 4.4. SETN-NON is a straightforward approach to optimize the estimator, however, the model found by SETN-NON leads to the highest error. SETN-LR finds the model with more parameters but is inferior to the model found by SETN. In conclusion, the proposed SETN is superior to its baselines, i.e., SETN-LR and SETN-NON. "SETN ($T$=1)" indicates that we directly choose the best architecture from the estimator via $\arg\max$ over $\boldsymbol{f}/\boldsymbol{h}$ as [25]. From Table 2, "SETN ($T$=1)" finds a small CNN, however, this CNN yields a relatively higher error than our SETN. Therefore, the stochastic training strategy and the final candidate evaluation strategy are necessary to find a good architecture.

**Scalability.** We compare the search cost between the small space using the candidate function set $\mathcal{O}$ and the large space using another candidate function set $\mathcal{O}_l$. This set $\mathcal{O}_l$ adds two more functions into $\mathcal{O}_l$: 3x3 dilated conv and 5x5 dilated conv. It has eight functions in total, and its search space has about $9.1\times10^{18}$ candidates. With the large search space, training SETN costs 50 GPU hours using the default hyper-parameters, and evaluating $T$=1K candidates takes less than three hours. Therefore, even though the large search space is 100× larger than the small one, SETN needs only about 18% more GPU days to complete

| | Method | GPU days | Parameters | $+\times$ (million) | Top-1 Accuracy | Top-5 Accuracy |
|---|---|---|---|---|---|---|
| Human Expert | Inception-v1 [35] | – | 6.6 MB | 1448 | 69.8% | 89.9% |
| | ResNet [18] | – | 11.7 MB | 1814 | 69.8% | 89.1% |
| | MobileNet-v2 [33] | – | 3.4 MB | 300 | 72.0% | – |
| | ShuffleNet [41] | – | $\sim$5 MB | 524 | 73.7% | – |
| NAS with more than 100 GPU days | Progressive NAS [23] | 150 | 5.1 MB | 588 | 74.2% | 91.9% |
| | NASNet-A [43] | 2000 | 5.3 MB | 564 | 74.0% | 91.6% |
| | NASNet-B [43] | 2000 | 5.3 MB | 488 | 72.8% | 91.3% |
| | NASNet-C [43] | 2000 | 4.9 MB | 558 | 72.5% | 91.0% |
| NAS with less than 5 GPU days | DARTS [25] | 4 | 4.9 MB | 595 | 73.1% | 91.0% |
| | GHN [40] | 0.84 | 6.1 MB | 569 | 73.0% | 91.3% |
| | SNAS [38] | 1.5 | 4.3 MB | 522 | 72.7% | 90.8% |
| | GDAS [14] | 0.84 | 5.3 MB | 581 | 74.0% | 91.5% |
| | SETN (N=1 & C=73) | 1.8 | 5.2 MB | 597 | 73.3% | 91.4% |
| | SETN (N=2 & C=58) | 1.8 | 5.3 MB | 600 | 74.3% | 91.6% |
| | SETN (N=3 & C=49) | 1.8 | 5.3 MB | 584 | 74.1% | 91.9% |
| | SETN (N=4 & C=44) | 1.8 | 5.4 MB | 599 | 74.3% | 92.0% |

Table 3. We compare networks found by SETN and other approaches on ImageNet. We report the model size, the computation cost, the top-1 accuracy, and the top-5 accuracy. The top block shows the manually designed CNNs. The bottom two blocks indicate the automatically design CNNs. $+\times$ indicates the number of multiply-add operations.

the search procedure. Besides, the network found with $\mathcal{O}_l$ achieves a similar performance compared to that of $\mathcal{O}$. This shows that our SETN can be successfully applied to much larger search space.

**Experiments on ImageNet.** We use the same cell structures found on the CIFAR-10 dataset to construct the CNN for ImageNet. We adjust hyper-parameters $N$ and $C$ to make the network align with the ImageNet-mobile setting, i.e., under 600M FLOPs. We train the network with a batch size of 256 over four GPUs in 250 epochs totally. We warm-up at the first five epochs, start the learning rate with 0.1, and decrease it to 0 via the cosine scheduler. We set the momentum as 0.9 and the weight decay as 3e-5. Besides, the label smoothing is applied with a epsilon of 0.1. An auxiliary tower with the weight of 0.4 is applied during training.

**Comparison with the state-of-the-art on ImageNet.** Since the training procedure of SETN does not use any ImageNet images, this experiment can investigate the transferability of the discovered network. We use the same CNN structure found on CIFAR-10 with different $N$ and $C$ configurations. These networks strictly matche the ImageNet-mobile setting. We show the top-1 and top-5 accuracy in Table 3. "SETN (N=2 & C=58)" achieves a top-1 accuracy of 74.3% on ImageNet. Our network obtains competitive accuracy compared to efficient NAS approaches [25, 38, 40]. AmoebaNet [30] achieves a similar accuracy than ours, but it costs 3150 GPU days, which is 1750$\times$ more than ours. In sum, our network is competitive to state-of-the-art networks, whereas SETN needs acceptable search costs.

### 5.3. Ablation Studies

In Section 5.2, we deliver a brief comparison between SETN and its variants w.r.t. the model size and the model accuracy. In this section, we will give a more comprehensive analysis for different aspects of SETN.

**The quality of estimated candidates.** There are four options to generate candidate CNNs. (a) the proposed SETN. (b) SETN-RAND [4]. (c) SETN-LR, sharing the sampled indexes of operations and inputs for different cells (d) SETN-NON, directly training the template network in the standard classification fashion as [25]. We use these four methods to generate 1000 candidate CNN and count their one-shot accuracy in a histogram. In each sub-figure of Figure 2, the x-axis indicates the one-shot validation accuracy and y-axis indicates the number of candidate CNNs. Several conclusion can be made. First, randomly generated candidates have much lower accuracy than all other compared strategies. Second, SETN-NON is better than the random approach but still inferior to SETN and SETN-LR. Third, the performance of SETN generated candidates is similar to that of SETN-LR. However, taking a close look at the histogram, SETN generate more accurate candidates than SETN-LR, e.g., there are more candidates with the accuracy between 85%$\sim$90%.

**Can the validation accuracy with template parameters reflect the ground truth validation accuracy?** We want to investigate whether the validation accuracy using the template parameters can provide a robust relative ranking of different networks or not. To achieve, we randomly sample 2000 networks, i.e., 1000 pairs. We first evaluate these 2000 networks by using the template parameters

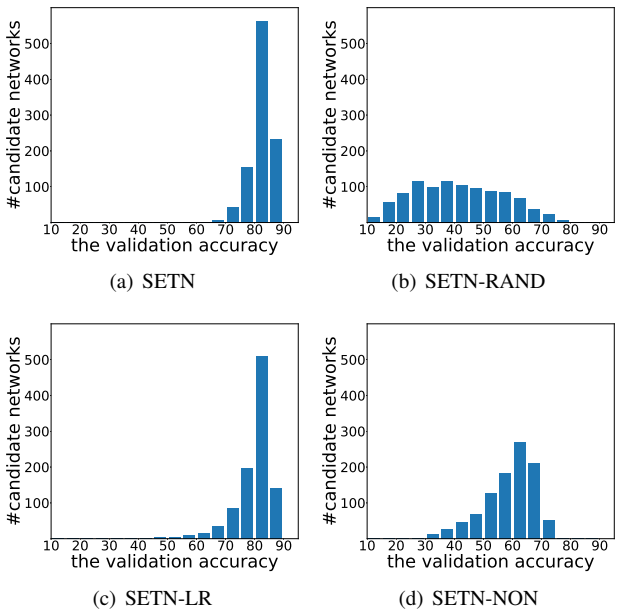


Figure 2. Comparison of the candidate networks generated by different strategies. 2(a) shows the generated candidates from the proposed SETN. 2(c), 2(d), and 2(b) show statistics of other three SETN variants. In each sub-figure, the x-axis and y-axis indicate the number of candidate networks and the validation accuracy, respectively.

on $\mathcal{D}_{val}$. We denote the obtained validation accuracy as $acc_t$. Then we retrain these 2000 networks from scratch from scratch by 100 epochs on $\mathcal{D}_{train}$, and other hyper-parameters are the same as experiments in Table 2. We evaluate these re-trained networks on $\mathcal{D}_{val}$ and denote the accuracy as $acc_r$, indicating the ground truth accuracy. For each pair, if the comparison of $acc_t$ is the same with the comparison of $acc_r$, we count this pair is good; otherwise, we count this pair is bad. Finally, we obtain more than 80% pairs are good.

The above two analysis paragraphs did not directly evaluate (1) how good the candidate sampled by the estimator is and (2) how accurate the one-shot accuracy is. To answer these two questions, we need a NAS dataset with ground truth accuracy of each candidate, where NAS-Bench-101 [39] is the only one. However, our SETN can not be directly evaluated on NAS-Bench-101 [39] due to its limitation. We would investigate these open questions once a suitable NAS dataset being public.

**The effect of the number of generated candidates.** In Table 2, we show that using $T$=1 finds a model with a higher error than using $T$=1K. A small number of $T$, e.g., 10, will cause a high variance of the accuracy of discovered CNN. If we increase the number of $T$, we could reduce the variance. In our experiments, $T$=1K is enough to discover a good network. If we increase $T$ to 10K, we could potentially find better networks, but the search cost will be more than five

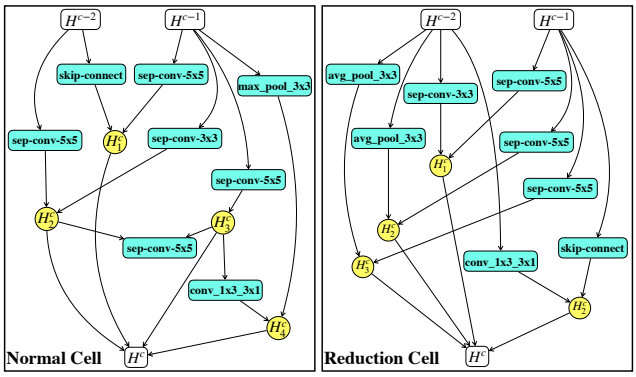


Figure 3. The left figure is the normal CNN cell that SETN discovered on CIFAR-10. The right figure is the reduction CNN cell that SETN discovered on CIFAR-10.

GPU days. Some approaches apply a progressive strategy to select the final CNN [4, 40]. For example, they first select the top 10 networks (ranked by the validation accuracy) and then retrain these networks with more epochs to get a precise validation accuracy. These strategies are able to further reduce the performance variance of the discovered CNN.

**Visualization.** We visualize the discovered cells in Figure 3. Compared to manually designed cells [18, 36, 37], the automatically discovered cells are much more complex and difficult to be designed by human experts. Given the superior performance of NAS-discovered networks, it is necessary to devote more effort on this topic.

**Discussions.** One limitation of SETN is that when the search space is very large, e.g., $10^{30}$ candidates, a small number of $T$ may not be able to find a good model. In this case, we have to increase the number of $T$, which will also increase the corresponding evaluation cost. A more efficient training strategy for the template network and the estimator can alleviate this problem. We leave such interesting extensions for future work.

## 6. Conclusion

We propose the self-evaluated template network (SETN) to search for the CNN with higher accuracy. Compared to previous one-shot NAS approaches, SETN significantly improves the quality of architecture candidates for the one-shot evaluation procedure. In this way, the sampled candidates of SETN can cover better architectures, and thus can finally find an architecture with higher performance. In experiments, SETN can complete the search procedure within two GPU days. It finds a good CNN from more than $10^{16}$ network possibilities, and this CNN achieves state-of-the-art performance on three benchmarks.

## References


[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017. 2, 6

[2] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. In *International Conference on Learning Representations (ICLR) Workshop*, 2018. 1, 2

[3] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, pages 459–468, 2017. 3

[4] 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 (ICML)*, pages 549–558, 2018. 1, 2, 3, 4, 5, 7, 8

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(8):1798–1828, 2013. 1

[6] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. SMASH: one-shot model architecture search through hypernetworks. In *International Conference on Learning Representations (ICLR)*, 2018. 1, 2, 3, 4, 5, 6

[7] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2787–2794, 2018. 1, 2, 6

[8] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*, 2019. 3, 4

[9] Yukang Chen, Gaofeng Meng, Qian Zhang, Shiming Xiang, Chang Huang, Lisen Mu, and Xinggang Wang. RENAS: Reinforced evolutionary neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4787–4796, 2019. 1

[10] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 886–893, 2005. 1

[11] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 6

[12] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5840–5848, 2017. 1

[13] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. *arXiv preprint arXiv:1905.09717*, 2019. 2

[14] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770, 2019. 2, 3, 6, 7

[15] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. *arXiv preprint arXiv:1906.09607*, 2019. 2

[16] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations (ICLR)*, 2017. 2

[17] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6307–6315, 2017. 6

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 2, 7, 8

[19] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. 1, 6

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 5

[21] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with Bayesian neural networks. In *International Conference on Learning Representations (ICLR)*, 2017. 1

[22] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 5

[23] 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 *European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 1, 2, 3, 6, 7

[24] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations (ICLR)*, 2018. 1, 2, 6

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. 2, 3, 5, 6, 7

[26] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Prototype propagation networks (PPN) for weakly-supervised few-shot learning on category graph. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 3015–3022, 2019. 1

[27] David G Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1150–1157, 1999. 1

[28] Renqian Luo, Fei Tian, Tao Qin, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 7816–7827, 2018. 2, 5, 6

[29] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning (ICML)*, pages 4095–4104, 2018. 1, 2, 3, 5, 6

[30] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019. 1, 2, 6, 7

[31] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning (ICML)*, pages 2902–2911, 2017. 2

[32] 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. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 5

[33] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018. 7

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 1

[35] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 1, 7

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. 8

[37] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017. 8

[38] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. 7

[39] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning (ICML)*, pages 7105–7114, 2019. 8

[40] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. 2, 3, 6, 7, 8

[41] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018. 7

[42] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017. 1, 2, 3, 6

[43] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710, 2018. 1, 2, 5, 6, 7