Searching for A Robust Neural Architecture in Four GPU Hours

Xuanyi Dong[1,2], Yi Yang[1]
[1]University of Technology Sydney [2]Baidu Research
xuanyi.dong@student.uts.edu.au, yi.yang@uts.edu.au

Abstract

Conventional neural architecture search (NAS) approaches are based on reinforcement learning or evolutionary strategy, which take more than 3000 GPU hours to find a good model on CIFAR-10. We propose an efficient NAS approach learning to search by gradient descent. Our approach represents the search space as a directed acyclic graph (DAG). This DAG contains billions of sub-graphs, each of which indicates a kind of neural architecture. To avoid traversing all the possibilities of the sub-graphs, we develop a differentiable sampler over the DAG. This sampler is learnable and optimized by the validation loss after training the sampled architecture. In this way, our approach can be trained in an end-to-end fashion by gradient descent, named Gradient-based search using Differentiable Architecture Sampler (GDAS). In experiments, we can finish one searching procedure in four GPU hours on CIFAR-10, and the discovered model obtains a test error of 2.82% with only 2.5M parameters, which is on par with the state-of-the-art.

1. Introduction

Designing an efficient and effective neural architecture requires substantial human effort and takes a long time [7, 9, 10, 12, 14, 20, 37, 44]. Since the birth of AlexNet [20] in 2012, human experts have conducted a huge number of experiments, and consequently devised several useful structures, such as attention [7] and residual connection [12]. However, the infinite possible choices of network architecture make the manual search unfeasible [1]. Recently, neural architecture search (NAS) has increasingly attracted the interest of researchers [1, 6, 17, 22, 25, 31, 46]. These approaches learn to automatically discover good architectures. They can thus reduce the labour of human experts and find better neural architectures than the human-invented architectures. Therefore, NAS is an important research topic in machine learning.

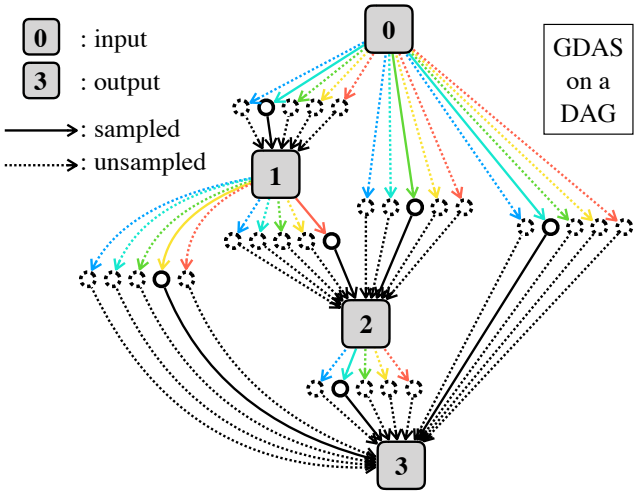

Figure 1. We utilize a DAG to represent the search space of a neural cell. Different operations (colored arrows) transform one node (square) to its intermediate features (little circles). Meanwhile, each node is the sum of the intermediate features transformed from the previous nodes. As indicated by the **solid** connections, the neural cell in the proposed GDAS is a sampled sub-graph of this DAG. Specifically, among the intermediate features between every two nodes, GDAS samples one feature in a differentiable way.

Most NAS approaches apply evolutionary algorithms (EA) [32, 23, 33] or reinforcement learning (RL) [46, 47, 3] to design neural architectures automatically. In both RL-based and EA-based approaches, their searching procedures require the validation accuracy of numerous architecture candidates, which is computationally expensive [47, 32]. For example, the typical RL-based method utilizes the validation accuracy as a reward to optimize the architecture generator [46]. An EA-based method leverages the validation accuracy to decide whether a model will be removed from the population or not [33]. These approaches use a large amount of computational resources, which is inefficient and unaffordable. This motivates researchers to reduce the computational cost.

*Part of this work was done when Xuanyi Dong was a research intern with Baidu Research. Code is publicly available on GitHub: https://github.com/D-X-Y/GDAS.

†Corresponding Author: Yi Yang

In this paper, we propose a **G**radient-based searching approach using **D**ifferentiable **A**rchitecture **S**ampling (GDAS). It can search for a robust neural architecture in four hours with a single V100 GPU. GDAS significantly improves efficiency compared to the previous methods. We start by searching for a robust neural "cell" instead of a neural network [46, 47]. A neural cell contains multiple functions to transform features, and a neural network consists of many copies of the discovered neural cell [22, 47]. Fig. 1 illustrates our searching procedure in detail. We represent the search space of a cell by a DAG. Every grey square node indicates a feature tensor, numbered by the computation order. Different colored arrows indicate different kinds of operations, which transform one node into its intermediate features. Meanwhile, each node is the sum of the intermediate features transformed from the previous nodes. During training, the proposed GDAS samples a sub-graph from the whole DAG, indicated by solid connections in Fig. 1. In this sub-graph, each node only receives one intermediate feature from every previous node. Specifically, among the intermediate features between every two nodes, GDAS samples one feature in a differentiable way. In this way, GDAS can be trained by gradient descent to discover a robust neural cell in an end-to-end fashion.

The fast searching ability of GDAS is mainly due to the sampling behavior. A DAG contains hundreds of parametric operations with millions of parameters. Directly optimizing this DAG [24] instead of sampling a sub-graph leads to two disadvantages. First, it costs a lot of time to update numerous parameters in one training iteration, increasing the overall training time to more than one day [24]. Second, optimizing different operations together could make them compete with each other. For example, different operations could generate opposite values. The sum of these opposite values tends to vanish, breaking the information flow between the two connected nodes and destabilizing the optimization procedure. To solve these two problems, the proposed GDAS samples a sub-graph at one training iteration. As a result, we only need to optimize a part of the DAG at one iteration, which accelerates the training procedure. Moreover, the inappropriate competition is avoided, which makes the optimization effective.

In summary, GDAS has the following benefits:

1. Compared to previous RL-based and EA-based methods, GDAS makes the searching procedure differentiable, which allows us to end-to-end learn a robust searching rule by gradient descent. For RL-based and EA-based methods, feedback (reward) is obtained after a prolonged training trajectory, while feedback (loss) in our gradient-based method is instant and is given in every iteration. As a result, the optimization of GDAS is potentially more efficient.

2. Instead of using the whole DAG, GDAS samples one sub-graph at one training iteration, accelerating the searching procedure. Besides, the sampling in GDAS is learnable and contributes to finding a better cell.

3. GDAS delivers a strong empirical performances while using fewer GPU resources. On CIFAR-10, GDAS can finish one searching procedure in several GPU hours and discover a robust neural network with a test error of 2.82%. On PTB, GDAS discovers a RNN model with a test perplexity of 57.5. Moreover, the networks discovered on CIFAR and PTB can be successfully transferred to ImageNet and WT2.

2. Related Work

Recently, researchers have made significant progress in automatically discovering good architectures [46, 47, 23, 22, 33]. Most NAS approaches can be categorized in two modalities: macro search and micro search.

Macro search algorithms aim to directly discover the entire neural networks [5, 4, 38, 46, 21]. To search convolutional neural networks (CNNs) [20], typical approaches apply RL to optimize the searching policy to discover architectures [1, 5, 46, 31]. Baker et al. [1] trained a learning agent by Q-learning to sequentially choose CNN layers. Zoph and Le [46] utilized long short-term memory (LSTM) [13] as a controller to configure each convolutional layer, such as the filter shape and the number of filters. In these macro search algorithms [1, 5, 46], the number of possible networks is exponential to the depth of a network, e.g., a depth of 12 can result in more than 10^{29} possible networks [31]. It is difficult and ineffective to search networks in such a large search space, and, therefore, these macro search methods [31, 46, 5] usually limit the CNN models to be shallow, e.g., a depth is less than 12. Since macro-discovered networks are shallower than deep CNNs [12, 14], their accuracies are limited. In contrast, our GDAS allows the network to be much deeper by stacking tens of discovered cells [47] and thus can achieve a better accuracy.

Micro search algorithms aim to discover neural cells and design a neural architecture by stacking many copies of the discovered cells [47, 32, 33, 31]. A typical micro search approach is NASNet [47], which extends the approach of [46] to search neural cells in the proposed "NASNet search space". Following NASNet [47], many researchers propose their methods based on the NASNet search space [22, 24, 5, 32]. For example, Real et al. [32] applied EA algorithm with a simple regularization technique to search neural cells. Liu et al. [22] proposed a progressive approach to search cells from shallow to deep gradually. These micro search algorithms usually take more than 100 GPU days [22, 47]. Even though some of them reduce the searching cost, they still take more than one GPU day [24]. Our GDAS is a also micro search algorithm, focusing on search cost reduction. In experiments, we can find a robust network within fewer GPU hours, which is 1000$\times$

less than the standard NAS approach [47].

Improving Efficiency. Since NAS algorithms usually require expensive computational resources [46, 47, 32], an increasing number of researchers focus on improving the architecture search speed [5, 22, 31, 38, 24]. A variety of techniques have been proposed, such as progressive-complexity search stages [22], accuracy prediction [2], HyperNet [4], Net2Net transformation [5], and parameter sharing [31]. For instance, Cai et al. [5] reused weights of previously discovered networks to amortize the training cost. Pham et al. [31] shared parameters between different child networks to improve the efficiency of the searching procedure. Brock et al. [4] utilized a network to generate model parameters given a discovered network, avoiding fully training from scratch. Liu et al. [24] relaxed the search space to be continuous, so that they can use gradient descent to effectively search cells. Though these approaches successfully accelerate the architecture search procedure, several GPU days are still required [22, 5]. Our GDAS samples individual architecture in a differentiable way to effectively discover architecture. As a result, GDAS can finish the search procedure in several GPU hours on CIFAR-10, which is much faster than these efficient methods.

Contemporary to this work, Xie et al. [39] applied a similar technique to relax the discrete candidate sampling as ours. They focus on fixing the inconsistency between the loss of attention-based NAS [24] and their objective. In contrast, we focus on making the sampling procedure differentiable and accelerating the searching procedure.

3. Methodology

3.1. Search Space as a DAG

We search for the neural cell in the search space and stack this cell in series to compose the whole neural network. For CNN, a cell is a fully convolutional network that takes output tensors of previous cells as inputs and generates another feature tensor. For recurrent neural network (RNN), a cell takes the feature vector of the current step and the hidden state of the previous step as inputs, and generates the current hidden state. For simplification, we take CNN as an example for the following description.

We represent the cell in CNN as a DAG $\mathcal{G}$ consisting of an ordered sequence of B computational nodes. Each computational node represents one feature tensor, which is transformed from two previous feature tensors. This procedure can be formulated as shown in Eq. (1) following [47].

$$\boldsymbol{I}_i = f_{i,j}(\boldsymbol{I}_j) + f_{i,k}(\boldsymbol{I}_k) \quad \text{s.t.} \quad j < i \ \ \& \ \ k < i, \qquad (1)$$

where $\boldsymbol{I}_i$, $\boldsymbol{I}_j$, and $\boldsymbol{I}_k$ indicate the i-th, j-th, and k-th nodes, respectively. $f_{i,j}$ and $f_{i,k}$ indicate two functions from the candidate function set $\mathbb{F}$. We denote the computational nodes of a cell as B. Taking $B = 4$ as an example, a cell contains 7 nodes in total, i.e., $\{\boldsymbol{I}_i | 1 \leq i \leq 7\}$. $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ nodes are the cell outputs in the previous two layers. $\boldsymbol{I}_3$, $\boldsymbol{I}_4$, $\boldsymbol{I}_5$, and $\boldsymbol{I}_6$ nodes are the computational nodes calculated by Eq. (1). $\boldsymbol{I}_7$ indicates the output tensor of this cell, which is the concatenation of the four computational nodes, i.e., $\boldsymbol{I}_7 = \boldsymbol{I}_3 \frown \boldsymbol{I}_4 \frown \boldsymbol{I}_5 \frown \boldsymbol{I}_6$. In GDAS, the candidate function set $\mathbb{F}$ contains the following 8 functions: (1) identity, (2) zeroize, (3) 3x3 depth-wise separate conv, (4) 3x3 dilated depth-wise separate conv, (5) 5x5 depth-wise separate conv, (6) 5x5 dilated depth-wise separate conv, (7) 3x3 average pooling, (8) 3x3 max pooling. We use the same candidate function set $\mathbb{F}$ as [24], which is similar to [47] but removes some unused functions and adds some useful functions.

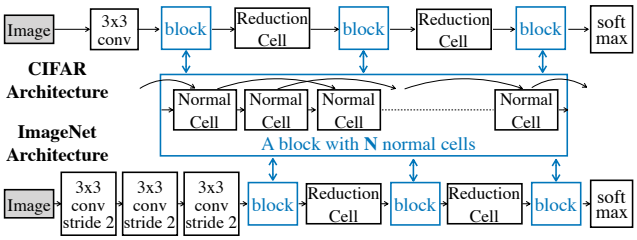

Figure 2. The strategy to design CIFAR architecture (top) and ImageNet architecture (bottom) based on the discovered cell. We use the same block structure (middle) in two cases. Both normal and reduction cells receive the outputs of two previous cells as inputs, as illustrated in the middle.

From cell to network. We search for two kinds of cells, i.e., a normal cell and a reduction cell. For the normal cell, each function in $\mathbb{F}$ has the stride of 1. For the reduction cell, each function in $\mathbb{F}$ has the stride of 2. Once we discover one normal cell and one reduction cell, we stack many copies of these discovered cells to make up a neural network. As shown in Fig. 2, for the CIFAR architecture, we stack N normal cells as one block. Given an image, it first forwards through the network head part, i.e., one 3 by 3 convolutional layer. It then forwards through three blocks with two reduction cells in between. The ImageNet architecture is similar to the CIFAR architecture, but the network head part consists of three 3 by 3 convolutional layers. We follow [24] to setup these two overall structures.

3.2. Searching by Differentiable Model Sampling

Formally, we denote a neural architecture as α and the weights of this neural architecture as ω_α. The goal of NAS is to find an architecture α, which can achieve the minimum validation loss after being trained by minimizeing the training loss, as shown in Eq. (2).

$$\begin{aligned} &\min_{\alpha} \mathbb{E}_{(x',y')\sim\mathbb{D}_V} - \log \Pr(y'|x'; \alpha, \omega_\alpha^*), \\ &\text{s.t. } \omega_\alpha^* = \arg\min_\omega \mathbb{E}_{(x,y)\sim\mathbb{D}_T} - \log \Pr(y|x; \alpha, \omega_\alpha), \end{aligned} \qquad (2)$$

where ω_α^* is the best weight of α and achieves the minimum training loss. We use the negative log likelihood as the training objective, i.e., $-\log \Pr$. $\mathbb{D}_T$ and $\mathbb{D}_V$ indicate the training set and the validation set, respectively. (x, y) and (x', y') are the data associated with its label, which are sampled from $\mathbb{D}_T$ and $\mathbb{D}_V$, respectively.

An architecture α consists of many copies of the neural cell. This cell is sampled from the search space represented by $\mathcal{G}$. Specifically, between node$_i$ and node$_j$, we sample one transformation function from $\mathbb{F}$ from a discrete probability distribution $\mathcal{T}_{i,j}$. During the search, we calculate each node in a cell as:

$$\boldsymbol{I}_i = \sum_{j=1}^{i-1} f_{i,j}\left(\boldsymbol{I}_j; \boldsymbol{W}_{f_{i,j}}\right) \quad \text{s.t.} \quad f_{i,j} \sim \mathcal{T}_{i,j}, \tag{3}$$

where $f_{i,j}$ is sampled from $\mathcal{T}_{i,j}$ and $\boldsymbol{W}_{f_{i,j}}$ is its associated weight. The discrete probability distribution $\mathcal{T}_{i,j}$ is characterized by a learnable probability mass function as in Eq. (4):

$$\Pr(f_{i,j} = \mathbb{F}_k) = \frac{\exp(\boldsymbol{A}_{i,j}^k)}{\sum_{k'=1}^{K} \exp(\boldsymbol{A}_{i,j}^{k'})}, \tag{4}$$

where $\boldsymbol{A}_{i,j}^k$ is the k-th element of a K-dimensional learnable vector $\boldsymbol{A}_{i,j} \in \mathbb{R}^K$, and $\mathbb{F}_k$ indicates the k-th function in $\mathbb{F}$. K is the cardinality of $\mathbb{F}$, i.e., $K = |\mathbb{F}|$. Actually, $\boldsymbol{A}_{i,j}$ encodes the sampling distribution of the function between node$_i$ and node$_j$. As a result, the sampling distribution of a neural cell is encoded by all $\boldsymbol{A}_{i,j}$, i.e., $\mathcal{A} = \{\boldsymbol{A}_{i,j}\}$.

Given Eq. (3) and Eq. (4), we can obtain α and ω, and thus can calculate $\Pr(y|x; \alpha, \omega)$ in Eq. (2). However, since Eq. (3) needs to sample from a discrete probability distribution, we cannot back-propagate gradients through $\boldsymbol{A}_{i,j}$ in Eq. (4) to optimize $\boldsymbol{A}_{i,j}$. To allow back-propagation, we first use the Gumbel-Max trick [11, 27] to re-formulate Eq. (3) as Eq. (5), which provides an efficient way to draw samples from a discrete probability distribution.

$$\boldsymbol{I}_i = \sum_{j=1}^{i-1} \sum_{k=1}^{K} \boldsymbol{h}_{i,j}^k \, \mathbb{F}_k(\boldsymbol{I}_j; \boldsymbol{W}_{i,j}^k), \tag{5}$$

$$\text{s.t.} \;\; \boldsymbol{h}_{i,j} = \text{one\_hot}(\arg\max_k(\boldsymbol{A}_{i,j}^k + \boldsymbol{o}_k)), \tag{6}$$

where $\boldsymbol{o}_k$ are i.i.d samples drawn from Gumbel (0,1)[1], and $\boldsymbol{h}_{i,j}^k$ is the k-th element of $\boldsymbol{h}_{i,j}$. $\boldsymbol{W}_{i,j}^k$ is the weight of $\mathbb{F}_k$ for the transformation function between node$_i$ and node$_j$. Then, we use the softmax function to relax the $\arg\max$ function so as to make Eq. (5) being differentiable [16, 26]. Formally, we use Eq. (7) to approximate Eq. (5).

$$\tilde{\boldsymbol{h}}_{i,j}^k = \frac{\exp((\boldsymbol{A}_{i,j}^k + \boldsymbol{o}_k)/\tau)}{\sum_{k'=1}^{K} \exp((\boldsymbol{A}_{i,j}^{k'} + \boldsymbol{o}_{k'})/\tau)}, \tag{7}$$

[1] $\boldsymbol{o}_i = -\log(-\log(u))$ with $u \sim \text{Unif}\,[0, 1]$

Algorithm 1 Searching Algorithm based on AOS

Input: the training set $\mathbb{D}_T$, and the validation set $\mathbb{D}_V$
randomly initialized $\mathcal{A}$ and $\mathcal{W}$, and the batch size n
while not converge **do**
  Sample batch of data $\mathbb{D}_t = \{(x_i, y_i)\}_{i=1}^n$ from $\mathbb{D}_T$
  Calculate $L_T = \sum_{i=1}^n \ell(x_i, y_i)$ based on Eq. (8)
  Update $\mathcal{W}$ by gradient descent: $\mathcal{W} = \mathcal{W} - \nabla_{\mathcal{W}} L_T$
  Sample batch of data $\mathbb{D}_v = \{(x_i, y_i)\}_{i=1}^n$ from $\mathbb{D}_V$
  Calculate $L_V = \sum_{i=1}^n \ell(x_i, y_i)$ based on Eq. (8)
  Update $\mathcal{A}$ by gradient descent: $\mathcal{A} = \mathcal{A} - \nabla_{\mathcal{A}} L_V$
end while

where τ is the softmax temperature. When $\tau \to 0$, $\tilde{\boldsymbol{h}}_{i,j}^k = \boldsymbol{h}_{i,j}^k$. When $\tau \to \infty$, each element in $\tilde{\boldsymbol{h}}$ will be the same and the approximated distribution will be smooth. To be noticed, we use the $\arg\max$ function in Eq. (5) during the forward pass but the $\text{soft}\max$ function in Eq. (7) during the backward pass to allow gradient back-propagation.

Training. Reviewing the objective of NAS in Eq. (2), the main challenge is learning to find architecture α. By utilizing Eq. (7), we can make the sampling procedure differentiable and learn a distribution of neural cells (representing architectures). However, it is still intractable to directly solve Eq. (2), because the nested formulation in Eq. (2) needs to calculate high order derivatives. In practice, to avoid calculating high order derivatives, we apply the alternative optimization strategy to update the sampling distribution $\mathcal{T}_{\mathcal{A}}$ and the weights of all functions $\mathcal{W}$ in an iterative way. Given one data sample x and its associated label y, we calculate the loss as:

$$\ell(x, y) \;\; = -\log \Pr(y|x; \alpha, \omega_\alpha), \tag{8}$$

$$\text{s.t.} \;\; \alpha \sim \mathcal{T}_{\mathcal{A}} \;\; \& \;\; \omega_\alpha \subset \mathcal{W}, \tag{9}$$

where $\mathcal{T}_{\mathcal{A}}$ is the distribution encoded by $\mathcal{A}$ and $\mathcal{W} = \{\boldsymbol{W}_{i,j}^k\}$ represents the weights of all functions in all cells of the network. Note that, for one data sample, it first samples α from $\mathcal{T}_{\mathcal{A}}$ and then calculates the network output only on its associated weight ω_α, which is a part of $\mathcal{W}$. As shown in Alg. 1, we apply the alternative optimization strategy (AOS) to update $\mathcal{A}$ based on the validation losses from $\mathbb{D}_V$ and update $\mathcal{W}$ based on the training losses from $\mathbb{D}_T$. It is essential to train $\mathcal{W}$ on $\mathbb{D}_T$ and $\mathcal{A}$ on $\mathbb{D}_V$, because (1) this strategy is theoretically sound with the objective Eq. (2); and (2) this strategy can improve the generalization ability of the searched structure.

Architecture. After training, we need to derive the final architecture from the learned $\mathcal{A}$. Each node$_i$ connects with T previous nodes. Following the previous works, we use $T = 2$ for CNN [47, 24] and $T = 1$ for RNN [31, 24]. Suppose Ω is the candidate index set, we derive the final architecture by the following procedure: (1) define the importance of the connection between node$_i$ and node$_j$ as:

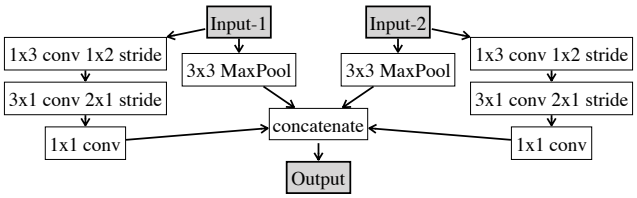

Figure 3. The designed reduction cell. "1x3 conv 1x2 stride" indicates a convolutional layer with 1 by 3 kernel and 1 by 2 stride.

$\max_{k\in\Omega} \Pr(f_{i,j} = \mathbb{F}_k)$. (2) for each node$_i$, retain T connections with the maximum importance from the previous nodes. (3) for the retained connection between node$_i$ and node$_j$, we use the function $\mathbb{F}_{\arg\max_{k\in\Omega} \Pr(f_{i,j}=\mathbb{F}_k)}$. Ω is $\{1, ..., K\}$ by default.

Acceleration. In Eq. (5), $\boldsymbol{h}_{i,j}$ is a one-hot vector. As a result, in the forward procedure, we only need to calculate the function $\mathbb{F}_{\arg\max(\boldsymbol{h}_{i,j})}$. During the backward procedure, we only back-propagate the gradient generated at the $\arg\max(\tilde{\boldsymbol{h}}_{i,j})$. In this way, we can save most computation time and also reduce the GPU memory cost by about $|\mathbb{F}|$ times. Within one training batch, each sample produces a different $\boldsymbol{h}_{i,j}$, and, therefore, each element in $\boldsymbol{A}_{i,j}$ has a high possibility of being updated with gradients.

One benefit of this acceleration trick is that it allows us to directly search on the large-scale dataset (e.g., ImageNet) due to the saved GPU memory. We did some experiments to directly search on ImageNet using the same hyper-parameters as on the small datasets, however, failed to obtain a good performance. Searching on a large-scale dataset might require different hyper-parameters and needs careful tuning. We will explore this in our future work.

3.3. Discussion on the Reduction Cell

Revisiting state-of-the-art architectures designed by human experts, AlexNet [20] and VGGNet [36] use the max pooling to reduce the spatial dimension; ResNet [12] uses a convolutional layer with stride of 2; and DenseNet [14] uses a 1 by 1 convolutional layer followed by average pooling to reduce dimension. These human-designed reduction cells are simple and effective. The automatically discovered reduction cells are also usually similar and simple [47, 31, 24]. For example, the reduction cell discovered by [24] only has max pooling and identity operations.

Most human-designed and automatically discovered reduction cells are simple and can achieve a high accuracy. Moreover, compared to searching one normal cell, jointly searching a normal cell and a reduction cell will greatly increase the search space and make the optimization difficult. We hope to find a better network by fixing the reduction cell. Inspired by [36, 24], we design a fixed reduction cell as shown in Fig. 3. In the experiments, with this human-designed reduction cell, GDAS finds a better architecture, yielding fewer parameters and higher accuracy.

4. Experimental Study

4.1. Datasets

CIFAR-10 and **CIFAR-100** [19] consist of 50K training images and 10K test images. CIFAR-10 categorizes images into 10 classes, while CIFAR-100 has 100 classes.

ImageNet [34] is a large-scale and well-known benchmark for image classification. It contains 1K classes, 1.28 million images for training, and 50K images for validation.

Penn Treebank (PTB) [28] is a corpus consisting of over 4.5 million words of American English words. We preprocess PTB following [30, 29].

WikiText-2 (WT2) [30] is a collection of 2 million tokens from the set of verified Good and Featured articles on Wikipedia. The training set contains 600 articles with 2,088,628 tokens. The validation set contains 60 articles with 217,646 tokens. The test set contains 60 articles with 245,569 tokens.

4.2. Search for CNN

CNN Searching Setup. The neural cells for CNN are searched on CIFAR-10 following [46, 47, 22, 31]. We randomly split the official training images into two groups, with each group containing 25K images. One group is used as the training set $\mathbb{D}_T$ in Alg. 1, and the other is used as the validation set $\mathbb{D}_V$ in Alg. 1. The candidate function set $\mathbb{F}$ has 8 different functions as introduced in Sec. 3.1. The default hyper-parameters for each function in $\mathbb{F}$ are the same in [24, 47, 22]. By default, we set the number of initial channels in the first convolution layer C as 16; set the number of computational nodes in a cell B as 4; and the number of layers in one block N as 2. We train the model by 240 epochs in total. For ω, we use the SGD optimization. We start with a learning rate of 0.025 and anneal it down to 1e-3 following a cosine schedule. We use the momentum of 0.9 and the weight decay of 3e-4. For α, we use the Adam optimization [18] with the learning rate of 3e-4 and the weight decay of 1e-3. The τ is initialized as 10 and is linearly reduced to 1. To search the normal cell and the reduction cell on CIFAR-10, our GDAS takes about five hours to finish the search procedure on a single NVIDIA Tesla V100 GPU. As discussed in Sec. 3.3, we also run experiments to only search the normal cell and fix the reduction cell as shown in Fig. 3, denoted as GDAS (FRC). When we use GDAS (FRC) to search on CIFAR-10, it takes less than four hours to finish one search procedure. *Following [24], we run GDAS 4 times with different random seeds and pick the best cell based on its validation performance. This procedure can reduce the high variance of the searched results, especially when searching the RNN structure.*

Type	Method	Venue	GPUs	Times (days)	Params (million)	Error on CIFAR-10	Error on CIFAR-100
Human expert	ResNet + CutOut [12]	CVPR16	–	–	1.7	4.61	22.10
	DenseNet-BC [14]	CVPR17	–	–	25.6	3.46	17.18
Macro search space	MetaQNN [1]	ICLR17	10	8-10	11.2	6.92	27.14
	Net Transformation [5]	AAAI18	5	2	19.7	5.70	–
	SMASH [4]	ICLR18	1	1.5	16.0	4.03	–
	NAS [46]	ICLR17	800	21-28	7.1	4.47	–
	NAS + more filters [46]	ICLR17	800	21-28	37.4	3.65	–
	ENAS [31]	ICML18	1	0.32	38.0	3.87	–
Micro search space	Hierarchical NAS [23]	ICLR18	200	1.5	61.3	3.63	–
	Progressive NAS [22]	ECCV18	100	1.5	3.2	3.63	19.53
	NASNet-A [47]	CVPR18	450	3-4	3.3	3.41	–
	NASNet-A + CutOut [47]	CVPR18	450	3-4	3.3	**2.65**	–
	ENAS [31]	ICML18	1	0.45	4.6	3.54	19.43
	ENAS + CutOut [31]	ICML18	1	0.45	4.6	2.89	–
	DARTS (1st) + CutOut [24]	ICLR19	1	0.38	3.3	3.00	–
	DARTS (2nd) + CutOut [24]	ICLR19	1	1	3.4	2.82	**17.54**†
	GHN + CutOut [41]	ICLR19	1	0.84	5.7	2.84	–
	NAONet [25]	NeurIPS18	200	1	10.6	3.18	–
	AmoebaNet-A + CutOut [32]	AAAI19	450	7	3.1	3.12	18.93†
	GDAS [C=36,N=6]	CVPR19	1	0.21	3.4	3.87	19.68
	GDAS [C=36,N=6] + CutOut	CVPR19	1	0.21	3.4	2.93	18.38
	GDAS (FRC) [C=36,N=6]	CVPR19	**1**	**0.17**	**2.5**	3.75	19.09
	GDAS (FRC) [C=36,N=6] + CutOut	CVPR19	**1**	**0.17**	**2.5**	2.82	18.13

Table 1. Classification errors of GDAS and baselines on CIFAR. † indicates the results trained using our setup. "FRC" indicates that we fix the reduction cell and only search the normal cell. Note that researchers might run their algorithms on different kinds of machines. The searching costs are derived from the original papers, and we did not normalize them across different GPUs. Our experiments are based on the V100 GPU; and if we run on Titan 1080Ti, the searching cost will increase to about seven GPU hours.

Clarifications on the searching cost (GPU days) of different methods. The searching costs listed in Tab. 1 and Tab. 2 are **not normalized** across different GPU devices. Different algorithms might run on different machines, and we simply refer the searching costs reported in their papers.[2] If we use other GPU devices, the searching cost of "GDAS (FRC)" could be a different number. For example, if we use Titan 1080Ti, the search cost will increase to about seven GPU hours.

Discussion on the acceleration step. If we do not apply the acceleration step introduced in Sec. 3.2, each iteration will cost $|\mathbb{F}|$=8$\times$ more time and GPU memory than GDAS. In the same time, without this acceleration step, it requires less training epochs to converge but still costs more time than applying the acceleration step.

Results on CIFAR. After the searching procedure, we use C=36, B=4, and N=6 to form a CNN. Following the previous works [24, 31, 47], we train the network by 600 epochs in total. We start the learning rate of 0.025 and reduce it to 0 with the cosine learning rate scheduler. We set the probability of path dropout as 0.2 and the auxiliary tower with the weight of 0.4 [46]. We use the standard preprocessing and data augmentation, i.e., randomly cropping, horizontally flipping, normalization, and CutOut [8, 43].

We compare the models discovered by our approach with other state-of-the-art models in Tab. 1. The models discovered by the macro search algorithms obtain a higher error than the models discovered by the micro search algorithms. Using GDAS, we discover a model with 3.3M parameters, which achieves 2.93% error on CIFAR-10. Using GDAS (FRC), we discover a model with only 2.5M parameters, which achieves 2.82% error on CIFAR-10. NASNet-A achieves a lower error rate than ours, but it contains more than 80% of the parameters than the model discovered by GDAS (FRC). Notably, our GDAS discovers a comparable model with the state-of-the-art, whereas the searching cost of our approach is much less than the others. For example, GDAS (FRC) takes less than 4 hours on a single V100 GPU, which is about 0.17 GPU days. It is faster than NASNet by almost 10^4 times. ENAS is a recent work that focuses on accelerating the searching procedure. ENAS is very efficient, whereas our GDAS (FRC) is three times faster than ENAS.

Results on ImageNet. Following [47, 24, 31, 35], we

[2]It is difficult for us to run all algorithms on the same GPU.

Type	Method	Venue	GPUs	Times (days)	Test Error (%)		Params (million)	+× (million)
					Top-1	Top-5		
Human expert	Inception-v1 [37]	CVPR15	–	–	30.2	10.1	6.6	1448
	MobileNet-V2 [35]	CVPR18	–	–	28.0	–	3.4	300
	ShuffleNet [42]	CVPR18	–	–	26.3	–	∼5	524
Micro search space	Progressive NAS [22]	ECCV18	100	1.5	25.8	8.1	5.1	588
	NASNet-A [47]	CVPR18	450	3-4	26.0	8.4	5.3	564
	NASNet-B [47]	CVPR18	450	3-4	27.2	8.7	5.3	488
	NASNet-C [47]	CVPR18	450	3-4	27.5	9.0	4.9	558
	DARTS (2nd) [24]	ICLR19	1	1	26.9	9.0	4.9	595
	GHN [41]	ICLR19	1	0.84	27.0	8.7	6.1	569
	AmoebaNet-A [32]	AAAI19	450	7	25.5	8.0	5.1	555
	AmoebaNet-B [32]	AAAI19	450	7	26.0	8.5	5.3	555
	AmoebaNet-C [32]	AAAI19	450	7	**24.3**	**7.6**	6.4	570
	GDAS [C=50,N=4]	CVPR19	1	0.21	26.0	8.5	5.3	581
	GDAS (FRC) [C=52,N=4]	CVPR19	**1**	**0.17**	27.5	9.1	**4.4**	**497**

Table 2. Top-1 and top-5 errors of GDAS and baselines on ImageNet. +× indicates the number of multiply-add operations. We refer results reported in [24] for Progressive NAS, NASNet, and AmoebaNet.

Architecture	Perplexity		Params (M)	Search Cost (GPU days)
	val	test		
V-RHN [45]	67.9	65.4	23	–
LSTM [29]	60.7	58.8	24	–
LSTM + SC [29]	60.9	58.3	24	–
LSTM + SE [40]	58.1	56.0	22	–
NAS [46]	–	64.0	25	10^4
ENAS [31]	60.8	58.6	24	0.5
DARTS (1st) [24]	60.2	57.6	23	**0.13**
DARTS (2nd) [24]	**58.1**	**55.7**	**23**	0.25
GDAS	59.8	57.5	**23**	0.4

Table 3. Comparison *w.r.t.* the perplexity of different language models on PTB (lower perplexity is better). V-RHN indicates Variational RHN [45]. LSTM + SC indicates LSTM with skip connection [29]. LSTM + SE indicates LSTM with 15 softmax experts [40]. The first four models are designed by human experts, and the last four models are automatically searched by machine.

Architecture	Perplexity		Params (M)	Search Cost (GPU days)
	val	test		
LSTM + AL [15]	91.5	82.0	28	–
LSTM [29]	69.1	65.9	33	–
LSTM + SC [29]	69.1	65.9	23	–
LSTM + SE [40]	66.0	63.3	33	–
ENAS [31]	72.4	70.4	**33**	0.5
DARTS (2nd) [24]	71.2	69.6	**33**	**0.25**
GDAS	**71.0**	**69.4**	**33**	0.4

Table 4. Comparison with different language models on WT2 (lower perplexity is better). LSTM + AL indicates LSTM with augmented loss [15]. Other notation is the same as in Tab. 3.

use the ImageNet-mobile setting, in which the input size is 224×224 and the number of multiply-add operations is restricted to be less than 600M. We train models by SGD with 250 epochs and use the batch size of 128. We initialize the learning rate of 0.1 and reduce it by 0.97 after each epoch.

We compare our results on ImageNet with the other methods in Tab. 2. Most algorithms in Tab. 2 take more than 1000 GPU days to discover a good CNN cell. DARTS [24] uses minimum resources among the compared algorithms, whereas ours is even faster than DARTS [24] by more than 10 times. For GDAS (FRC), we use C=52 and N=4 to construct the model following the setting in [24]. For GDAS, if we use C=52 and N=4, the number of multiply-add operations will be larger than 600 MB, and thus we use C=50 to restrict it to be less than 600MB. Our model, GDAS (FRC) [C=52,N=4], costs about 20% less multiply-add operations than [24] but obtains the same top-5 error. AmoebaNet-A and Progressive NAS achieve a slightly lower test error than ours. However, their methods cost a prohibitive amount of GPU resources. The results in Tab. 2 show the discovered cell on CIFAR-10 can be successfully transferred to ImageNet and achieve competitive performance.

4.3. Search for RNN

RNN Searching Setup. The neural cells for RNN are searched on PTB with the splits following [24, 31] The candidate function set $\mathbb{F}$ contain 5 functions, i.e., zeroize, Tanh, ReLU, sigmoid, and identity. We use B=9 to search the RNN cell. The RNN model consists of one word embedding layer with a hidden size of 300, one RNN cell with a hidden size of 300, and one decoder layer. We train the model by 200 epochs with a batch size of 128 and a BPTT length of 35. We optimize ω by Adam with a learning rate

Architecture	Params	Error on CIFAR-10
GDAS-N + GDAS-R	3.3	2.93
GDAS-N + FIX-R	3.0	2.87
FRC-N + GDAS-R	2.9	2.84
FRC-N + FIX-R	**2.5**	**2.82**

Table 5. Different normal and reduction cell combinations. GDAS-N and GDAS-R indicate the normal and reduction cells discovered by GDAS, respectively. FRC-N and FIX-R mean the normal cell from GDAS (FRC) and our designed reduction cell, respectively.

of 20 and a weight decay of 5e-7. We optimize α by Adam with a learning rate of 3e-3 and a weight decay of 1e-3. Other setups are the same in [31, 24].

Results on PTB. We evaluate the RNN model formed by the discovered recurrent cell on PTB. We use a batch size of 64 and a hidden size of 850. We train the model using the A-SGD by 2000 epochs. The learning rate is fixed as 20 and the weight decay is 8e-7. DARTS [24] and ENAS [31] greatly reduce the search cost compared to previous methods. Our GDAS incurs a lower search cost than all the previous methods. Note that our code is not heavily optimized and the theoretical search cost should be less than the one reported in Tab. 3.

We compare different RNN models in Tab. 3. The model discovered by GDAS achieves a validation perplexity of 59.8 and a test perplexity of 57.5. The performance of our discovered RNN is on par with the state-of-the-art models in Tab. 3. LSTM + SE [40] obtains better results than ours, but it is an ensemble method using mixture of softmax. By applying the SE technique [40], GDAS can achieve the lower perplexity without doubt. LSTM [29] is an extensively tuned model, whereas our automatically discovered model is superior to it. Compared to other efficient approaches, the search cost of GDAS is the lowest.

Results on WT2. To train the model on WT2, we use the same experiment settings as PTB, but we use a hidden size of 700 and a weight decay of 5e-7. We train the model in 3000 epochs in total. Tab. 4 compares different RNN models on WT2. Our approach achieves competitive results among all automatically searching approaches. GDAS is worse than "LSTM + SC" [29]. Since our model is searched on a small dataset PTB, and the transferable ability of the discovered model might be a little bit weak. If we directly search the RNN model on WT2, we could obtain a better model and improve the transferable ability.

4.4. Discussion

We visualize the discovered cells in Fig. 4. These automatically discovered cells are complex and hard to be designed manually. Moreover, networks with these discovered cells can achieve more superior performance than hand-crafted networks. This demonstrates that automated neural architecture search is the future of architecture design.

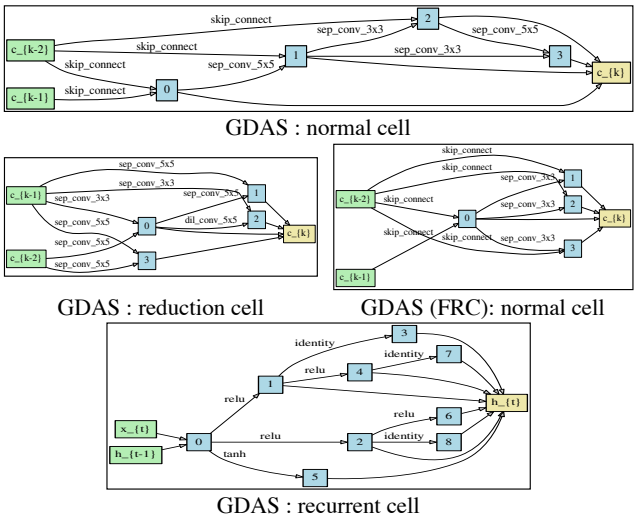

Figure 4. The top block and the middle-left block show the normal cell and the reduction cell discovered by GDAS, respectively. The middle-right block shows the discovered normal cell when we fix the reduction cell. The bottom block shows the discovered recurrent cell.

Revisiting Sec. 3.3, we propose a new reduction cell as a replacement for automated reduction cell. With this reduction cell, we can more effectively search neural cells. For further analysis, we use the normal cell found by GDAS and the proposed reduction cell to construct a new CNN, denoted as "GDAS-N + FIX-R" in Tab. 5. The accuracy of this network on CIFAR-10 is similar to "GDAS-N + GDAS-R" and "FRC-N + FIX-R" in Tab. 5. This result implies that the reduction cell might have a negligible effect on the performance of networks and the hand-crafted reduction cell could be on par with the automatically discovered one.

Most recent NAS approaches search neural networks on the small-scale datasets, such as CIFAR, and then transfer the discovered networks to the large-scale datasets, such as ImageNet. The obstacle of directly searching on ImageNet is the huge computational cost. GDAS is an efficient NAS algorithm and gives us an opportunity to search on ImageNet. We will explore this research direction in our future work.

5. Conclusion

In this paper, we propose a Gradient-based neural architecture search approach using Differentiable Architecture Sampler (GDAS). Our approach is efficient and reduces the search cost of the standard NAS approach [47] by about 10^4 times. Moreover, both CNN and RNN models discovered by our GDAS can achieve competitive performance compared to state-of-the-art models.

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] 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.

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

[4] 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.

[5] 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.

[6] Liang-Chieh Chen, Maxwell D Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8713–8724, 2018.

[7] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

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

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

[10] Xuanyi Dong, Shoou-I Yu, Xinshuo Weng, Shih-En Wei, Yi Yang, and Yaser Sheikh. Supervision-by-Registration: An unsupervised approach to improve the precision of facial landmark detectors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 360–368, 2018.

[11] Emil Julius Gumbel. Statistical theory of extreme values and some practical applications. *NBS Applied Mathematics Series (AMS)*, 33, 1954.

[12] 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.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] 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.

[15] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. In *International Conference on Learning Representations (ICLR)*, 2017.

[16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

[17] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2020–2029, 2018.

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

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

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1097–1105, 2012.

[21] Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[22] 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.

[23] 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.

[24] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[25] Renqian Luo, Fei Tian, Tao Qin, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 7827–7838, 2018.

[26] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

[27] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3086–3094, 2014.

[28] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[29] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations (ICLR)*, 2018.

[30] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*, 2017.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

[37] 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.

[38] Tom Vniat and Ludovic Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3492–3500, 2018.

[39] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[40] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. In *International Conference on Learning Representations (ICLR)*, 2018.

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

[42] 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.

[43] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

[44] Linchao Zhu, Zhongwen Xu, and Yi Yang. Bidirectional multirate reconstruction for temporal modeling in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2653–2662, 2017.

[45] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutnık, and Jürgen Schmidhuber. Recurrent highway networks. In *International Conference on Machine Learning (ICML)*, pages 4189–4198, 2017.

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

[47] 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.