RAPDARTS: Resource-Aware Progressive Differentiable Architecture Search

Sam Green[§], Craig M. Vineyard[‡], Ryan Helinski[‡], Çetin Kaya Koç[*]
[§]Semiotic AI, Los Altos, California, USA
sam@semiotic.ai
[‡]Sandia National Laboratories, Albuquerque, New Mexico, USA
{cmviney, rhelins}@sandia.gov
[*]University of California Santa Barbara, Santa Barbara, California, USA
cetinkoc@ucsb.edu

***Abstract*—Early neural network architectures were designed by so-called "grad student descent". Since then, the field of Neural Architecture Search (NAS) has developed with the goal of algorithmically designing architectures tailored for a dataset of interest. Recently, gradient-based NAS approaches have been created to rapidly perform the search. Gradient-based approaches impose more structure on the search, compared to alternative NAS methods, enabling faster search phase optimization. In the real-world, neural architecture performance is measured by more than just high accuracy. There is increasing need for efficient neural architectures, where resources such as model size or latency must also be considered. Gradient-based NAS is also suitable for such multi-objective optimization. In this work, we extend a popular gradient-based NAS method to support one or more resource costs. We then perform in-depth analysis on the discovery of architectures satisfying single-resource constraints for classification of CIFAR-10.**

I. Introduction

THE optimal design of a neural architecture depends on 1) the target dataset, 2) the set of *primitive operations* (e.g. convolutional filters, skip-connections, nonlinearity functions, pooling), 3) how the primitive operations are composed into a neural architecture and optimized, and 4) resource constraints like hardware cost, minimum accuracy, or maximum latency. In this paper, we assume the target dataset has been provided, and we provide guidelines and analysis for searching for neural architectures under one or more hardware resource constraints.

Convolutional layers and fully-connected layers are parameter-heavy operations. Those, along with other lighter primitive operations, like pooling layers or batch normalization, may be composed into an endless variety of neural architectures. But what is the optimal neural architecture for a given dataset? There is no existing closed-form solution to that question.

Historically, the highest performing neural architectures have been found by applying heuristics and a large amount of compute. Some well known examples of modern hand-crafted architectures include AlexNet [1], VGG16 [2], ResNet [3], and the Inception series [4], [5], [6]. None of these examples consider hardware, and they pursue classification performance at all cost.

[§] Research performed while author was employed by Sandia National Laboratories.

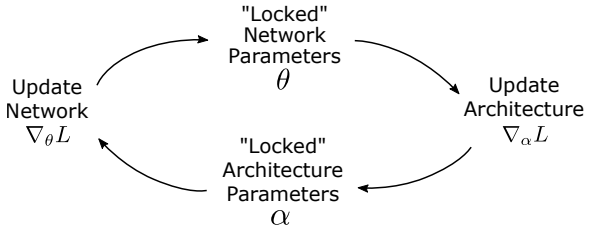


Fig. 1. Gradient-based Neural Architecture Search (GBNAS) methods maintain two sets of parameters. *Neural network parameters* are represented by $\boldsymbol{\theta}$ and *architecture parameters* are represented by $\boldsymbol{\alpha}$. GBNAS algorithms leverage differentiable functions, parameterized by architecture parameters, to design deep neural networks, which are parameterized by network parameters. First-order optimization alternates between "locking" one set of parameters and updating the other.

Neural Architecture Search (NAS) methods automate strategies for discovery of high performing neural architectures. A reinforcement learning-based (RL) approach was the first post-AlexNet NAS method with state-of-the-art performance on CIFAR-10 [7], [8]. The RL approach was quickly followed by a high performance Evolutionary Strategy (ES) based method [9]. While both the RL and ES methods discovered high performance architectures, their use came at the cost of thousands of GPU hours.

Gradient-based NAS (GBNAS) methods have the benefit of being directly optimized through gradient descent and consequently complete the search faster than other NAS methods. The basic idea of GBNAS is given in Figure 1. The search process alternates between temporarily fixing one set of parameters, i.e. assuming they are constants, and updating the other set of parameters. This approach has no convergence guarantees, but it works well in practice.

Because neural models are now widely deployed on systems like edge devices, in cars, and running in servers, available hardware resources also have an impact on what may be considered an "optimal" neural architecture design. Hardware resource constraints are often summarized as size, weight, and power (SWaP). Resource constraints could also include maximum latency, minimum throughput, or a manufacturing budget which will determine if a custom ASIC is an option, if a COTS device is sufficient, or if something semi-custom, like an FPGA, is an option. For example, during the design

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

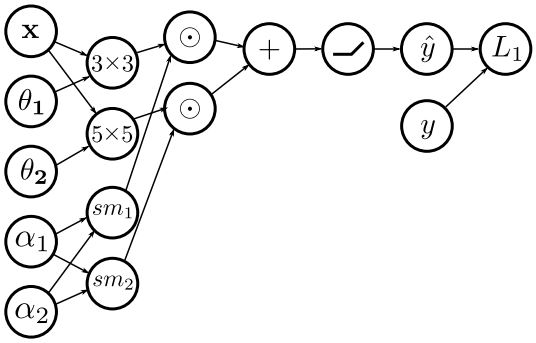


Fig. 2. The function of DARTS architecture parameters is to scale the output of primitive operations. In this illustration the primitive operations include 3×3 and 5×5 convolutional filters parameterized by tensors $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ respectively. The output feature maps of the primitive operations are element-wise scaled $\odot$ by the softmax (sm) of architecture parameters α_1 and α_2. The scaled output feature maps are then added, thereby creating a *mixed operation*. This notional illustration shows a network with only two primitive operations, followed by a nonlinearity, producing an output prediction $\hat{y}$. In practice, there may be many mixed operations, each containing many primitive operations, forming a deep network.

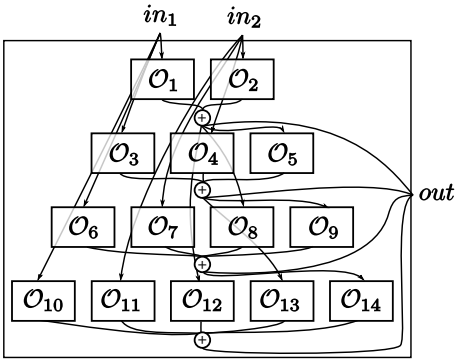


Fig. 3. The DARTS cell architecture has 14 mixed operations (represented by $\mathcal{O}_i$) distributed among four steps with skip-connections between each step. At each step, the outputs of the mixed operations are element-wise added. The sum is then passed as an input to a mixed operation in the next step. All element-wise sums are concatenated as the cell output and fed forward to the next cell in the network.

of Google's TPUv1, architects were given a budget of 7 ms per inference (including server communication time) for user-facing workloads [10].

Recent efforts described below implement NAS strategies incorporating hardware resource constraints into the search. GBNAS methods capture hardware resource constraints within a differentiable loss function. This approach enables the architecture search to yield network architectures biased toward satisfying resource constraints.

In this work we have modified P-DARTS [11], which in-turn is based on another popular gradient-based NAS algorithm, DARTS [12], to support resource costs. We use our modified GBNAS algorithm to search for many neural architectures under various resource consumption penalties. We then use our results and observations to answer the following questions:

- What is the computational cost of searching for satisficing architectures?
- What heuristics can be used to guide the search and training process to reduce compute time?
- How reproducible are search results under random initial conditions?

II. Related Work

The first competitive NAS approach applied to modern image classification tasks was based on reinforcement learning (RL) [7]. In this work, an LSTM-based RL agent was trained to output primitive operations which were then chained together into a directed acyclic graph. After training and evaluating the graph, the agent was then encouraged or discouraged, via a positive or negative reward derived from classification accuracy, to generate similar graphs in the future or to explore and make new graphs.

The reinforcement learning NAS approach worked well and was able to achieve high accuracy, but at unheard of computational expense. It required 3,150 GPU-days to discover one of their published architectures.

Related approaches to sampling neural architectures include Markov chain Monte Carlo methods [13], evolutionary strategies [14], and genetic algorithms [15]. Similar to RL approaches, all of these optimization methods generate populations of neural architectures. The populations are then trained and a fitness value is derived from the classifier's final test performance. The fitness value is used to encourage or discourage the design of the next population of architectures.

Reinforcement learning, Markov chain Monte Carlo methods, evolutionary strategies, and genetic algorithms discover high-performance architectures, but they are incredibly expensive. These methods often require $100\times$ to $1000\times$ more compute than gradient-based methods [16].

Gradient-based neural architecture search has recently become popular because of its efficiency [12], [17], [18], [11]. GBNAS methods maintain two sets of parameters: *network parameters* $\boldsymbol{\theta}$ and *architecture parameters* $\boldsymbol{\alpha}$. Previous GBNAS methods have introduced various methods to optimize and use the two parameter sets. In the simplest case, optimization is achieved by optimizing one set of parameters and then the other. This first-order optimization approach is further explained and illustrated in Figure 1.

Differentiable Architecture Search (DARTS) is a GBNAS technique that uses *mixed operations* to compute multiple primitive operations in parallel, followed by element-wise summation [12]. The mixed operations are scaled by architecture parameters prior to summation. For example, as illustrated in Figure 2, a 3×3 convolutional filter and a 5×5 convolutional filter can be designed such that both receive the same input feature map and both generate additively conformable output feature maps.

Extending this technique, DARTS composes 14 mixed operations into a *cell*. Eight cells are then chained to create the network. Each cell shares the same connectivity and architecture parameters ($\boldsymbol{\alpha}$) for mixed operations, but the network parameters ($\boldsymbol{\theta}$) are learned independently in each primitive operation and in each cell. An illustration of the DARTS cell connectivity is given in Figure 3.

DARTS has a limitation which requires the entire neural

network (i.e. all cells and all mixed operations) to fit in GPU memory. This limits the depth of the neural network as well as the batch size during training. Progressive Differentiable Architecture Search (P-DARTS) mitigates the memory limitation of DARTS by 1) gradual growth in the depth of the neural network, and simultaneously 2) gradual reduction in number of primitive operations per mixed operation, thus reducing model size [11].

ProxylessNAS also extended DARTS [18]. ProxylessNAS treats the architecture parameters of each mixed operation as a probability distribution. ProxylessNAS stores a large overparameterized network in system memory, because the network is too large to fit on a GPU. During evaluation, a subnetwork is sampled and transferred to the GPU for evaluation. Gradients are calculated and used to update the shared-weights of the overparameterized network.

Addressing the need to search for architectures which not only strive for high accuracy, but also meet additional performance constraints, hardware-aware NAS techniques have been pursued. ProxylessNAS is particularly relevant for hardware-aware GBNAS, because it formalizes the approach to incorporating resource costs during the search. In the context of classification, ProxylessNAS creates a loss function that incorporates both a cross-entropy loss for the classification accuracy as well as a resource loss for latency.

In this work we augment P-DARTS with a ProxylessNAS-style resource loss and analyze its impact on architectures discovered during the search phase.

III. Method

A. Resource-Aware Differentiable Neural Architecture Search

When training a convolutional neural network for classification, the goal is to obtain a model that best predicts labels from observations drawn from an underlying distribution of interest. Fitting a neural model to an underlying distribution is achieved by finding optimal network parameters $\boldsymbol{\theta}^*$ that minimize expected prediction error on an available dataset:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \big[J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}} L_1(f(\boldsymbol{x};\boldsymbol{\theta}), y) \big], \quad (1)$$

where J is the objective function, $\boldsymbol{x}$ are dataset observations, y are dataset labels, $\hat{p}_{data}$ is the empirical distribution, L_1 is a prediction error loss function, and f is the neural network parameterized by $\boldsymbol{\theta}$.

Gradient-based NAS methods introduce another set of *architecture parameters* $\boldsymbol{\alpha}$, producing:

$$g(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\alpha}). \quad (2)$$

We refer to g as a directed acyclic graph, or simply *graph*, to highlight that it is composed of a neural network whose control flow is modified by other non-network architecture parameters. Note the distinction between f used in Equation 1, which is only parameterized by network parameters, and g used in Equation 2, which is parameterized by both network and architecture parameters.

Architecture parameters, like network parameters, are scalar-valued tensors. Architecture parameters are used to control either the weight of primitive operations, as in [12], [11], or the probability primitive operations will take place, as in [19], [18]. In both cases, the scalar values are at least interpreted as one or more probability distributions through processing by the softmax function. In our case, the probability distribution is then used for evaluation of a mixed operation.

A mixed operation is illustrated in Figure 2, and it is formalized as:

$$\mathcal{O}(\boldsymbol{x}) = \mathbb{E}\big[o(\boldsymbol{x})\big] \approx \sum_{i=1}^{N} \frac{\exp(\alpha_i)}{\sum_j \exp(\alpha_j)} o_i(\boldsymbol{x}) = \sum_{i=1}^{N} p_i o_i(\boldsymbol{x}), \quad (3)$$

where $o_i(\boldsymbol{x})$ is a primitive operation, and $\mathcal{O}(\boldsymbol{x})$ is equivalent to the expected value of the primitive operations. This formalism extends the mixed operation to the inclusion of N primitive operations that are evaluated in parallel and designed such that their outputs are additively conformable. In practice many mixed operations are used, with unique subsets of $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ used for the calculation of each expected value, but we show only a single mixed operation here for clarity.

The inclusion of architecture parameters implies there are now two objective functions to be optimized:

$$\begin{aligned} J_1(\boldsymbol{\theta}) &= \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}} L_1(g(\boldsymbol{x},\boldsymbol{\alpha};\boldsymbol{\theta}), y), \\ J_2(\boldsymbol{\alpha}) &= \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}} L_1(g(\boldsymbol{x},\boldsymbol{\theta};\boldsymbol{\alpha}), y). \end{aligned} \quad (4)$$

The graph evaluations in Equation 4 are now denoted $g(\boldsymbol{x},\boldsymbol{\alpha};\boldsymbol{\theta})$ and $g(\boldsymbol{x},\boldsymbol{\theta};\boldsymbol{\alpha})$. This notation highlights that in the case of $J_1(\boldsymbol{\theta})$ the graph is evaluated at input and architecture parameter constants $(\boldsymbol{x},\boldsymbol{\alpha})$ and optimized using network parameters $\boldsymbol{\theta}$. In the second case of $J_2(\boldsymbol{\alpha})$ the graph is evaluated at input and network parameter constants $(\boldsymbol{x},\boldsymbol{\theta})$ and optimized using architecture parameters $\boldsymbol{\alpha}$. Therefore the following bilevel optimization must be solved:

$$\begin{aligned} \boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \big[J_1(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}} L_1(g(\boldsymbol{x},\boldsymbol{\alpha}^*;\boldsymbol{\theta}), y) \big], \\ \boldsymbol{\alpha}^* &= \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \big[J_2(\boldsymbol{\alpha}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}} L_1(g(\boldsymbol{x},\boldsymbol{\theta}^*;\boldsymbol{\alpha}), y) \big]. \end{aligned} \quad (5)$$

When using first-order differentiable methods, this bilevel optimization is solved by alternatingly "locking" one set of parameters and updating the other with gradient descent. Second-order optimization methods, which involve calculation of the Hessian, are also possible and slightly better in terms of accuracy, but this comes at significant computational cost. However, it is possible to approximate the second-order optimization with reduced computational cost [12].

Our method extends P-DARTS to discover neural architectures biased toward the satisfaction of resource constraints. We do this by including one or more "expected resource cost" loss terms. As mentioned previously, each of the primitive operations in a mixed operation is associated with a unique architecture parameter. P-DARTS uses 14 mixed operations in the search phase of cell architecture discovery, and there are eight primitive operations per mixed operation, so there are $14 \times 8 = 112$ architecture parameters total.

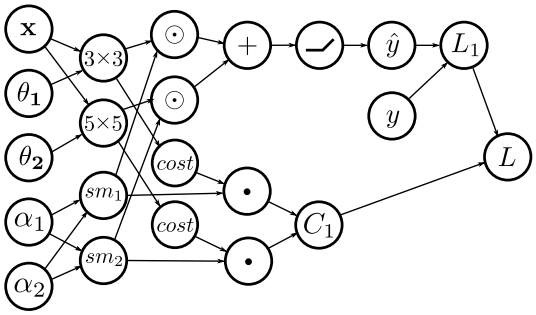


Fig. 4. P-DARTS may be extended with the calculation of an expected resource cost (C_1) for each mixed operation. When the gradient of the expected resource cost is calculated, the more expensive primitive operations are penalized more heavily than the less expensive operations, but the penalty is balanced by how much the primitive operation contributes to classification accuracy.

The expected value of a single mixed operation was given in Equation 3. We temporarily make index values of the mixed operation explicit here for clarity:

$$\mathcal{O}_k(\boldsymbol{x}_k) = \sum_{i=1}^{8} p_{k,i} \cdot o_{k,i}(\boldsymbol{x}_k), \tag{6}$$

where k is the mixed operation index. Note here that the probability distributions, $p_{k,i}$, are now tied to a particular mixed operation. This calculation is equivalent to the addition node in Figure 2.

As introduced in ProxylessNAS, the probabilities used in the mixed operation calculation are also conducive to calculation of the expected value of various resource costs. For example, if there is a cost function that takes as input the description of each primitive operation (including the input feature map dimension information) and outputs a resource cost, it may be used for the calculation of an expected resource cost of the mixed operation:

$$\mathbb{E}\big[cost(\mathcal{O}_k(\boldsymbol{x}_k))\big] \approx \sum_{i=1}^{8} p_{k,i} \cdot cost(o_{k,i}(\boldsymbol{x}_k)). \tag{7}$$

The cost function may be an analytical function, e.g. number of bytes required by the model, or the cost function could be based on a simulation or a surrogate model trained from data collected from a physical device.

The expected cost of the mixed operation is differentiable with respect to the mixed operation's architecture parameters. Accordingly, the partial derivative of the expected resource cost with respect to architecture parameter α_i is given as:

$$\begin{aligned}\frac{\partial \mathbb{E}\big[cost(\mathcal{O}(\boldsymbol{x}))\big]}{\partial \alpha_i} &\approx \frac{\partial \big[p_1 c_1 + p_2 c_2 + \cdots + p_8 c_8\big]}{\alpha_i}, \\ &= \sum_{l=1}^{8} \frac{\partial \left[\frac{\exp(\alpha_l)}{\sum_j \exp(\alpha_j)} \cdot c_l\right]}{\partial \alpha_i}, \\ &= \sum_{l=1}^{8} c_l p_l (\delta_{i,l} - p_i). \end{aligned} \tag{8}$$

where we have abbreviated $cost(o_i(\boldsymbol{x}))$ as c_i, $\delta_{i,l} = 1$ if i equals l and 0 otherwise, and we have dropped the mixed operation index k for brevity.

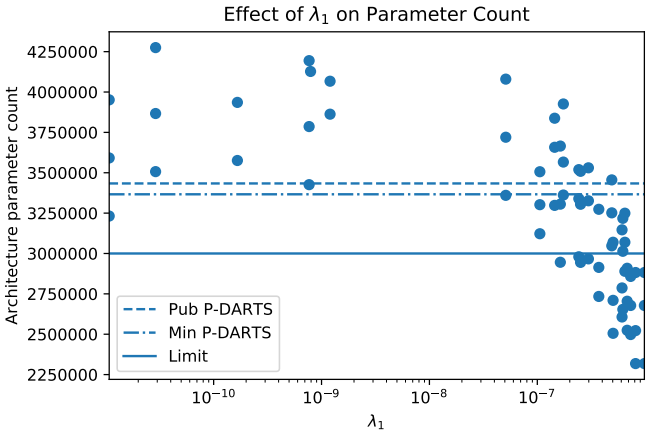


Fig. 5. Coarse-search for resource expected parameter count hyperparameter λ_1. As λ_1 grows beyond 10^{-7}, RAPDARTS increasingly identifies architectures that require less than 3 M parameters. The publish P-DARTS architecture is marked with the dashed line. The minimum P-DARTS architecture found by us is marked with the dash-dot line. Our self-imposed budget is marked with the solid line.

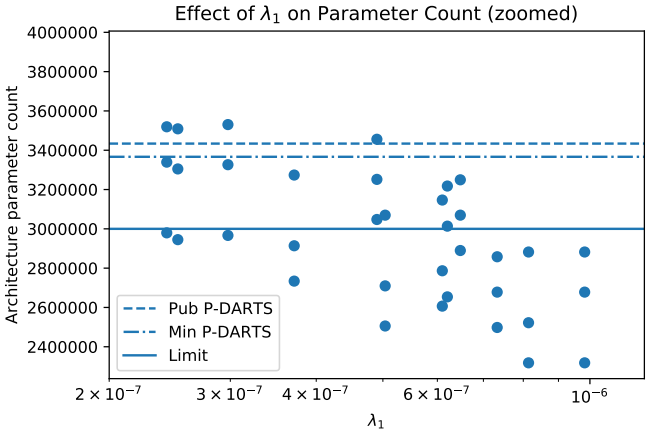


Fig. 6. Fine-search focused $2 \times 10^{-7} < \lambda_1 < 10^{-6}$. At around $\lambda_1 = 6 \times 10^{-6}$ architectures are frequently generated which meet the 3 M parameter constraint.

We denote the sum of expected mixed operation costs as:

$$C_m = \sum_{k=1}^{14} \mathbb{E}\big[cost_m(\mathcal{O}_k(\boldsymbol{x}_k))\big], \tag{9}$$

Note that unique m correspond to unique resource costs, e.g. C_1 could be the sum of expected mixed operation parameter sizes, and C_2 could be the sum of expected mixed operation latencies.

We denote the sum of the classification and resource losses as:

$$L = L_1 + \sum_{m=1}^{M} \lambda_m C_m, \tag{10}$$

where M is the number of resource costs to satisfy, and λ_m is the resource-cost hyperparameter and controls how important the resource cost m is compared to accuracy as well as other resource costs.

The bilevel optimization in Equation 5 may now be slightly rewritten as:

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[J_1(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}} L(g(\boldsymbol{x}, \boldsymbol{\alpha}^*; \boldsymbol{\theta}), y)\right], \\ \boldsymbol{\alpha}^* &= \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \left[J_2(\boldsymbol{\alpha}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}} L(g(\boldsymbol{x}, \boldsymbol{\theta}^*; \boldsymbol{\alpha}), y)\right],\end{aligned} \tag{11}$$

where only L_1 has been replaced by L. As before, this may be optimized using first or second-order approaches. For intuition on the continued use of a single loss function L, consider Figure 4. Under the assumption that a change in network parameters $\boldsymbol{\theta}$ creates no change in cost (given a fixed input feature map and primitive operation), the gradient of C_1 with respect to $\boldsymbol{\theta}$ is zero. On the other hand, a change in architecture parameters $\boldsymbol{\alpha}$ creates a change in both L_1 and C_1. So calculating the gradient of $L = L_1 + \lambda_1 C_1$ with respect to both $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$ results in the correct values.

Using the method above, we created *Resource-Aware P-DARTS* (RAPDARTS). Practically, the modification to P-DARTS requires the total expected resource cost be returned during the forward pass of an input tensor. To achieve this, during calculation of each mixed operation (Equation 6), we also calculate the expected resource cost (Equation 7). The expected cost for all mixed operations is accumulated and added to the classification loss (Equation 9). If multiple costs are required, e.g. model size and latency, each cost requires its own version of Equation 7, and must be accumulated individually from other costs.

IV. Experiments and Results

We use RAPDARTS to search for CIFAR-10 neural architectures. We follow the architecture discovery algorithm of P-DARTS and search for cell architectures containing the same primitive operations as used by DARTS and P-DARTS, namely:

- Zero*
- Skip-Connect*
- Avg-Pool 3×3*
- Max-Pool 3×3*
- Seperable 3×3 Conv.
- Seperable 5×5 Conv.
- Dialated 3×3 Conv.
- Dialated 5×5 Conv.

All of the above primitive operations are standard convolutional layers except Zero which allows a cell to learn *not* to pass information. Skip-connect is a parameter-free operation which allows information to pass through the mixed operation without modification. Parameter-free primitive operations are marked with an asterisk.

In an effort to simulate a real-world constraint, we restrict ourselves such that discovered CIFAR-10 architectures must have less than 3×10^6 parameters. This constrained optimization problem may be captured as:

$$\begin{aligned}&\underset{\boldsymbol{\theta},\boldsymbol{\alpha}}{\text{minimize}} \quad L_1(g(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\alpha}), y) \\ &\text{subject to} \quad \text{Parameter count} < 3 \times 10^6.\end{aligned} \tag{12}$$

We perform NAS adhering to this constraint using the RAP-DARTS framework above.

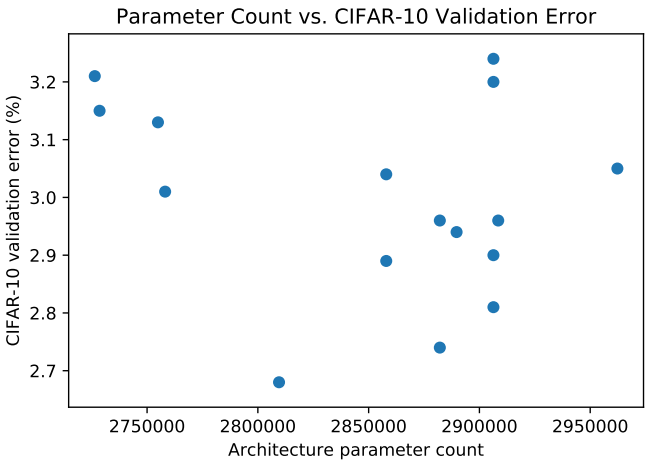


Fig. 7. Relationship between RAPDARTS model size and trained validation error appears uncorrelated. Indicating that at this variation of model capacity, model size is not a predictor of final classifier performance.

For the purpose of baseline calculations, we first consider the unconstrained results from P-DARTS. The authors of P-DARTS provided a reference architecture discovered through their algorithm [20]. We trained and evaluated that architecture eight times using the latest version of the P-DARTS code [21]. We then used the results from the repeated training to obtain performance statistics of the published architecture.

The resulting trained models achieved $2.60 \pm .13\%$ error on the CIFAR-10 validation dataset. Additionally, the published P-DARTS architecture requires 3.4×10^6 parameters.

We then executed the P-DARTS architecture search code four times to test the ability to rediscover architectures with the performance of the published architecture. The four searches resulted in nine architectures. However, per the P-DARTS algorithm, we eliminated one architecture with more than two skip-connections in the *normal* cell (see P-DARTS paper for details on the two cell types).

None of the eight valid architectures were the same as the official P-DARTS CIFAR-10 architecture, but this is not surprising, given the size of the P-DARTS architecture search space. Because of this, we compare our results to the statistics of various architectures discovered during our search, instead of the statistics of the single published architecture. The resulting trained models achieved $2.72 \pm .22\%$ error on CIFAR-10. The architectures required $3.9 \pm .3$ M parameters. The smallest P-DARTS model required 3.4 M parameters.

We now explore the impact of different hyperparameter values on the unconstrained multi-objective version of Equation 12:

$$L = L_1 + \lambda_1 C_1, \tag{13}$$

where C_1 is the sum of expected number of parameters in the model. As introduced in Equation 10, the λ_1 scalar is a hyperparameter which determines the relative importance of the resource cost explicitly and the relative importance of the accuracy of the network implicitly.

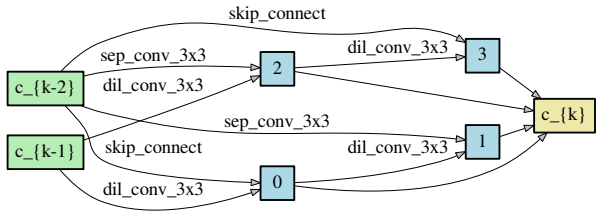
As stated in this section's introduction, our self-imposed resource budget is 3 M parameters. The default P-DARTS

Architecture	C10 Test Err (%) Best	Avg	Params (M)	Search Cost (GPU-days)	Search Method
AmoebaNet+B + cutout [22]	N/A	$\mathbf{2.55 \pm 0.05}$	2.8	3150	evolution
ASAP-Small [23]	**1.99**	N/A	2.5	.2	gradient-based
ASHA [24]	2.85	3.03 ± 0.13	2.2	9	random
DARTS [12]	2.94	N/A	2.9	.4	gradient-based
DSO-NAS [25]	N/A	2.84 ± 0.07	3.0	1	gradient-based
SNAS + moderate constraint + cutout [17]	2.85	N/A	2.3	1.5	gradient-based
RAPDARTS + cutout (ours)	2.68	2.83 ± 0.05	2.8	12	gradient-based

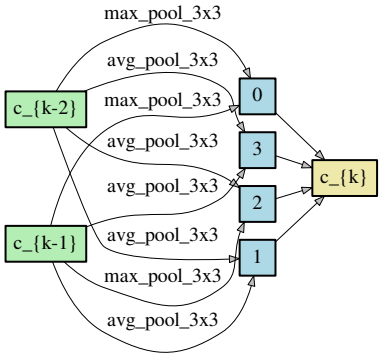
TABLE I
RAPDARTS CIFAR-10 ERROR RATE VERSUS OTHERS FOR MODELS WITH LESS THAN 3×10^6 PARAMETERS. WE ALSO INCLUDE NAS RESULTS FROM RANDOMLY SEARCHED ARCHITECTURES [24] AS WELL AS RECENT RESULTS [23]. FOR RAPDARTS, SEARCH COST INCLUDES ACTUAL COST FOR ALL EXPERIMENTS FOR FINDING THE 2.68% MODEL. IN TOTAL, THE SEARCH AND TRAIN PHASES REQUIRED 26 GPU-DAYS.

search does not generate models that small, however, by using RAPDARTS we are able to satisfy this constraint. To achieve this, we need to discover a λ_1 value to guide the architecture search. That is accomplished by finding a coarse range of suitable λ_1s and then identifying a refined λ_1.

The coarse λ_1 is identified by performing various architecture searches with λ_1s sampled randomly from a uniform distribution $\mathcal{U}([10^{-11}, 10^{-6}])$. Each search requires .3 GPU-days.



(a) Normal Cell



(b) Reduce Cell

Fig. 8. Cells found by RAPDARTS achieving 2.68% CIFAR-10 validation error. All primitive operations are low-cost operations.

Results from the coarse-search are shown in Figure 5. At approximately $\lambda_1 > 10^{-7}$, architectures begin to be generated which meet the 3×10^6 parameter count constraint. Parameter counts reduce dramatically as λ_1 approaches 10^{-6}, but we have observed that models with higher capacity tend to perform better than models with lower capacity, so it is unlikely that architectures derived from $\lambda_1 > 10^{-6}$ are preferred over those closer to the 3 M parameter threshold.

Figure 6 "zooms in" on the previous figure, focusing on λ_1 sampled uniformly from $\mathcal{U}[(2 \times 10^{-7}, 10^{-6})]$. Near $\lambda_1 = 6 \times 10^{-7} \approx 1 \times 10^{-6.2}$, architectures are generated that often require less than 3 M parameters.

One final search is then performed on λ_1 sampled uniformly from $U([10^{-6.24}, 10^{-6.2}])$. This test resulted in 48 valid architectures with resulting models between 2.1 M and 2.96 M parameters. We then trained the 16 largest resulting architectures. The resulting best model achieved 2.68% CIFAR-10 validation error and required 2.8 M parameters. The results for all 16 trained models are plotted in Figure 7. As can be seen, there is no linear relationship at this scale between parameter count and CIFAR-10 accuracy. For statistical confidence, we retrained the best model eight times with different seeds and obtained $2.83\% \pm .05$ validation error.

The discovered cells corresponding to the 2.68% CIFAR-10 validation are shown in Figure 8. The DARTS-based algorithms use two cell types: a "normal" cell, which maintains input and output feature map dimensionality, and a "reduce" cell, which decrease the output feature maps dimensionality.

The cell architectures discovered by RAPDARTS are noteworthy in several respects. First, the normal cell has discovered a "deep" design, similar to that discovered by P-DARTS, but only lightweight convolutional operations are used. Second, all pooling operations have been moved to the reduce cells.

Table I compares the RAPDARTS architecture with the performance of recent architectures with parameter counts less than 3 M. RAPDARTS competes favorably with the others.

We report the actual number of hours spent searching for our winning architecture, not merely the search time for a single architecture. Including both the coarse and fine-search phases, 40 different λ_1 values were used. This took a total of 12 GPU-days to compute.

We trained 16 of the fine-search phase models to completion. Each model required less than 20 hours to train, so the 16 fine-search models took less than 14 GPU-days total to train. All experiments were performed using an NVIDIA V100 GPU.

V. CONCLUSION AND FUTURE WORK

Classification accuracy achieved by neural architecture search methods now surpass hand-designed neural models.

First-generation NAS methods include those based on evolutionary search and reinforcement learning. Second generation NAS methods use gradient-based optimization. In this work we present RAPDARTS, which augments a popular gradient-based NAS method with the ability to target neural architectures meeting specified resource constraints. We use RAPDARTS to identify a neural architecture achieving 2.68% test error on CIFAR-10. This is competitive with other existing results for models with less than 3 M parameters.

We believe third-generation methods will be gradient-based and attempt to make more aspects of the search differentiable. For example, the P-DARTS (and RAPDARTS) search begins with five cells, then grows the search network to 11 cells, and finally 17 cells. At the same time, as the network grows, less important primitive operations are dropped. The "gradual" adjustments introduced by this technique enable architecture parameters learned by gradient-descent in one phase to be useful in another. It would be preferable to make these changes even more gradually. We leave that for future work.

In conclusion, we have presented an example that optimizes two objectives: minimizing accuracy loss while keeping the number of model parameters below a resource constraint threshold. A limitation of our work is that the number of parameters required by our discovered models may not optimize other constraints, e.g. minimum latency. To address this concern, future work will focus on multiple resource constraints guided by more hardware-specific costs.

ACKNOWLEDGMENT

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

The views expressed in the article do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

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

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[5] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[6] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

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

[8] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[9] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2902–2911.

[10] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[11] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," *arXiv preprint arXiv:1904.12760*, 2019.

[12] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[13] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, "Neural networks designing neural networks: multi-objective hyper-parameter optimization," in *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 2016, p. 104.

[14] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *arXiv preprint arXiv:1804.09081*, 2018.

[15] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: neural architecture search using multi-objective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2019, pp. 419–427.

[16] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," *arXiv preprint arXiv:1905.01392*, 2019.

[17] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," *arXiv preprint arXiv:1812.09926*, 2018.

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

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

[20] "P-darts published cifar-10 genotype," https://github.com/chenxin061/pdarts/blob/b1575e101aedb7396a89d8a7f74d0318877a1156/genotypes.py, accessed: 2019-10-24.

[21] "P-darts source code," https://github.com/chenxin061/pdarts/tree/05addf3489b26edcf004fc4005bbc110b56e0075, accessed: 2019-10-24.

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4780–4789.

[23] A. Noy, N. Nayman, T. Ridnik, N. Zamir, S. Doveh, I. Friedman, R. Giryes, and L. Zelnik-Manor, "Asap: Architecture search, anneal and prune," *arXiv preprint arXiv:1904.04123*, 2019.

[24] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," *arXiv preprint arXiv:1902.07638*, 2019.

[25] X. Zhang, Z. Huang, and N. Wang, "You only search once: Single shot neural architecture search via direct sparse optimization," *arXiv preprint arXiv:1811.01567*, 2018.