IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 17, 2021, accepted July 22, 2021, date of publication August 2, 2021, date of current version August 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3101975

Exploring Neural Architecture Search Space via Deep Deterministic Sampling

KEITH G. MILLS[1,2], MOHAMMAD SALAMEH[2], DI NIU[1], FRED X. HAN[2], SEYED SAEED CHANGIZ REZAEI[2], HENGSHUAI YAO[2,3], WEI LU[2], SHUO LIAN[4], AND SHANGLING JUI[4]

[1]Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2R3, Canada
[2]Huawei Technologies Canada Company Ltd., Edmonton, AB T6G 2C8, Canada
[3]Department of Computing Science, University of Alberta, Edmonton, AB T6G 2R3, Canada
[4]Huawei Kirin Solution, Shanghai 201206, China

Corresponding author: Keith G. Mills (kgmills@ualberta.ca)

ABSTRACT Recent developments in Neural Architecture Search (NAS) resort to training the supernet of a predefined search space with weight sharing to speed up architecture evaluation. These include random search schemes, as well as various schemes based on optimization or reinforcement learning, in particular policy gradient, that aim to optimize a parametric architecture distribution and the shared model weights simultaneously. In this paper, we focus on efficiently exploring the important region of a neural architecture search space with reinforcement learning. We propose Deep Deterministic Architecture Sampling (DDAS) based on deep deterministic policy gradient and the actor-critic framework, to selectively sample important architectures in the supernet for training. Through balancing exploitation and exploration, DDAS is designed to combat the disadvantages of prior random supernet warm-up schemes and optimization schemes. Gradient-based NAS approaches require the execution of multiple short experiments in order to combat the random stochastic nature of gradient descent, while still only producing a single architecture. Contrary to this approach, DDAS employs a reinforcement learning-based agent and focuses on discovering a Pareto frontier containing many architectures over the course of a single experiment requiring 1 GPU day. Experimental results for CIFAR-10 and CIFAR-100 on the DARTS search space show that DDAS can depict in a single search, the accuracy-FLOPs (or model size) Pareto frontier, which outperforms random sampling and search. With a test accuracy of 97.27%, the best architecture found on CIFAR-10 outperforms the original second-order DARTS while using 600M fewer parameters. Additionally, DDAS finds an architecture capable of achieving 82.00% test accuracy on CIFAR-100 while using only 3.14M parameters and outperforming GDAS.

INDEX TERMS Neural architecture search, reinforcement learning, differentiable optimization.

I. INTRODUCTION

Manual neural architecture design demands laborious efforts accompanied by time-consuming experimentation for model evaluation. Neural Architecture Search (NAS) mitigates the hurdles of hand-crafted designs by using algorithms to search for the best architecture, given a particular task. Despite showing remarkable improvements in image classification and language modeling tasks, NAS algorithms that rely on Evolutionary Algorithms [1] or Reinforcement learning (RL) [2] suffer from the evaluation bottleneck. The evaluation process of architectures generated by such algorithms is computationally expensive and requires the use of a large number of GPUs.

Recent developments have given rise to one-shot architecture search [3], [4], in which a *one-shot model* or *supernet* as a superposition of all candidate architectures is trained by sharing weights among all architectures. One-shot architecture search has reduced the search cost down to one or a few GPU days, as the evaluation of individual architectures is now converted to an alternative process of training a single supernet and validating individual architectures based on the shared weights inherited from the supernet.

Various methods have been proposed to optimize a parametric architecture distribution while updating the shared weights. DARTS [5] parameterizes the search space through

The associate editor coordinating the review of this manuscript and approving it for publication was Valentina E. Balas.

a set of differentiable operation weights α and relies on gradient-based optimization algorithms to simultaneously optimize the operation weights and the shared model weights, in a process known as bi-level optimization. Gradient-based optimization algorithms have further been used for architecture sampling based on Gumbel Softmax (e.g., SNAS [6]) and architecture distribution binarization (e.g., ProxylessNAS [7]). On the other hand, ENAS [8] uses an RNN-based policy to sample discrete architectures for training, and adjusts the policy toward increased validation accuracy using the policy gradient method in reinforcement learning.

However, there have been concerns that the policies of popular NAS algorithms and the architectures they produce are not decisively better than those of Random Search [9], [10], which samples architectures uniformly at random. Moreover, it has been found that most cell-based NAS algorithms tend to produce wide and shallow architectures [11]. The class of cells mentioned above are not superior to deep and narrow architectures. Rather, by allowing for easier gradient flow, feature smooth gradient landscapes, they train faster during the search phase and are thus preferred by existing algorithms at the cost of a thorough exploration. Therefore, an important trade-off any weight sharing NAS schemes face is between exploitation—the ability to locate and fully train the best architecture, and exploration—the ability to explore large swathes of a search space to allow the better architectures to be discovered.

The problem is further complicated by the need for hardware-friendly architecture search, e.g., by budgeting the number of FLOPS,[1] inference time, or the total number of parameters that a model can have. Schemes such as SNAS [6], RC-DARTS [12] and ProxylessNAS [7] address the problem by introducing constraints or penalty into their optimization formulations. In practical deployment, however, the need for depicting a Pareto frontier of architectures necessitates repeated executions of these algorithms under different constraints, which is costly.

In this paper, we use Reinforcement Learning to efficiently explore an architecture search space and propose Deep Deterministic Architecture Sampling (DDAS), a weight sharing NAS algorithm based on Deep Deterministic Policy Gradient (DDPG) [13], a continuous, off-policy reinforcement learning algorithm [13], in order to efficiently generate a Pareto frontier of architectures in terms of the accuracy and FLOPS (or number of parameters) and find the best architecture, all in a single run. Specifically, we make the following contributions.

First, we model NAS as a continuous control problem in a high-dimensional search space. Similar to DARTS [5], SNAS [6] and ProxylessNAS [7], we parameterize the search space using a set of continuous weights of operations that connect latent vectors. However, instead of updating these weights of operations with gradient descent or bi-level optimization, we rely on the ability of DDPG to explore and sample them in an actor-critic framework. Empirical evidence shows that DDPG performs well in high-dimensional control tasks with continuous actions (e.g., robotic control) [14].

Second, previous reinforcement learning schemes proposed for NAS, e.g., ENAS, mainly use an RNN controller, which is essentially a stochastic policy, to sample architectures for training and evaluation. However, due to the stochastic nature of the policy, the same policy may sample a large number of different architectures, entailing Monte Carlo gradient estimates which have large variances. In contrast, we use a deterministic policy in DDAS, since the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient, as is shown in DPG [15] and DDPG [13].

Third, we judiciously design the reward and strike a balance between exploitation and exploration when updating the actor and critic networks in DDAS, such that the agent will maintain a high reward while being allowed to explore a potentially large space of candidate architectures, which gradient-based optimization methods fail to fully explore. In fact, in the pursuit of a higher validation performance, gradient-based optimization schemes may often converge to a single large architecture, which is repeatedly selected for training, preventing these schemes from sampling other architectures in the search space and generating the Pareto frontier. In DDAS, the ability of exploration over diverse architectures in the search space is achieved by the use of a combination of noise-based exploration schemes in a continuous space.

A series of experimental results on CIFAR-10 and CIFAR-100 [16] suggest that the Pareto frontiers generated by DDAS show a clear superiority over that of Random Search over a randomly warm-started supernet [9]. In the meantime, the test accuracy of the best architectures found by DDAS is comparable to a range of related NAS algorithms that rely on weight sharing.

II. RELATED WORK

Originally, NAS is a resource-intensive task, since every candidate architecture must be fully trained to retrieve its performance, which is then used to guide the search. For instance, NASNet [17] and AmoebaNet [18] spend over 2000 GPU days to find the best architecture. To reduce this cost, succeeding works like ENAS [8] and DARTS [5] adopt a weight-sharing scheme by training a supernet containing all possible operations/connections. Each architecture then inherits the corresponding weights from the supernet.

Current NAS techniques mainly branch into two categories, ones that rely on random search, and ones that attempt to incrementally learn a distribution of the best architectures. Under the first category, [3] train an one-shot model once and then sample architectures from a fixed distribution for performance estimation and search. [4] sample a single path uniformly from a supernet to train the shared weights. Similarly, [9] randomly sample a child network and only update its corresponding shared weights. [10], [19] recently provide

[1] Floating point operations used to forward pass a single data sample.

more insights on the ability of shared weights in the supernet to represent the true weights of individual architectures.

In comparison, gradient-based methods like DARTS [5] construct a continuous relaxation of the search space and learn the degree of contribution of each operation. DARTS has given rise to many off-shoots, including P-DARTS [20], an exploitation-driven scheme which aims to bridge the gap between search and evaluation. It does so by gradually decreasing the size of the search space while increasing the size of the search model. However, training the supernets used by both algorithms requires a large amount of GPU memory. To this end, SNAS [6] and GDAS [21] instead learn a sampling distribution for each modifiable operation in the search space of DARTS. ProxylessNAS [7] learns the probabilities to binarize the operations per edge using BinaryConnect [22].

Our proposed algorithm is related to both DARTS and sampling-based methods including SNAS, GDAS and ProxylessNAS, searching for architectures parameterized by differentiable architecture weights. However, instead of solving bi-level optimization (as in DARTS) or using Gumbel Softmax (e.g., as in SNAS, GDAS) or BinaryConnect (as in ProxylessNAS) to handle the continuous-to-discrete conversion, we use DDPG [13] to generate architecture weights. Our experimental results show that DDPG achieves better exploration than optimization-based methods.

Closely related to our work are NAS algorithms based on Reinforcement Learning, e.g., ENAS [8], NASNet [17], MNASNet [23], which use either REINFORCE [24] or Proximal Policy Optimization [25] to learn to sample child networks in a stochastic manner. These algorithms operate in an episodic manner [26]. A single architecture is constructed per episode over multiple time steps and can only be evaluated upon completion. A key departure of our work from the existing RL-based NAS is the use of a deterministic policy instead of stochastic policies, which results in less variance in the updates of architecture distributions via policy gradient. DDAS generates and evaluates one architecture per step, resulting in a continuing problem with an infinite horizon. Another difference is that we focus on using the exploration mechanisms of DDPG to enhance the search of Pareto frontiers instead of a single best architecture.

Hardware constraints, such as FLOPS, model size and inference time, are considered by a number of NAS schemes [23], [27], [28]. Using an Evolutionary-based algorithm [29] approximates the Pareto frontier of architectures under multiple objectives. SNAS [6] and ProxylessNAS [7] handle hardware-friendly objectives, i.e., latency, to tailor the search for specific devices, i.e., CPU, GPU, or Mobile, by adding regularizers to the loss. Instead of introducing penalty terms, which necessitates repeated search runs, DDAS generates the Pareto frontier in a single search by judiciously striking a balance between exploitation and exploration. A similar one-shot Pareto frontier search scheme is presented in [30], which decouples supernet training and search, and uses a progressive shrinking trick to combat interference between child models. In contrast, DDAS solves the supernet training and architecture search as a holistic problem, relying on the ability of DDPG to discover and train important architectures on the Pareto frontier in a continuous search space.

III. METHODOLOGY

In this section, we present the detailed mechanisms of the proposed DDAS. We first define our search space, from which we form and train the supernet. We then present our DDPG agent and environment, followed by a description of the training procedure and methods to balance exploitation and exploration.

A. THE SUPERNET ENVIRONMENT

Formally speaking, a supernet is the superposition of all the possible architectures in a search space. Our environment is a supernet that is similar to the Convolutional Neural Network supernet in DARTS [5]. It consists of a stem layer that performs several preliminary convolutions, followed by a sequence of stacked *cells*, and finally, a head that performs the classification. As in DARTS, two types of cells are considered in our search process: a normal cell and a reduction cell. Reduction cells contain convolution operations with strides of 2 and are responsible for halving the width and height of data tensors while doubling the number of channels. Normal cells do not modify the dimension of input data and contain operations with strides of 1. All networks contain two reduction cells, positioned one and two thirds into the entire network, respectively. All other cells are normal cells.

The search is conducted for cell architectures. A cell is defined as a directed acyclic graph (DAG) of N ordered nodes, including two input nodes and one output node, along with an edge set E. Each node represents a latent vector. A pair of nodes (i, j) is connected by a directed edge if $i < j$, which represents a set of predefined operations. Let x_i denote the latent vector corresponding to node i, $\mathcal{O}$ be the set of predefined operations, and $\alpha \in \mathbb{R}^{|E|\times|\mathcal{O}|}$ represent the weights for the operations on the edges of the DAG.

For each directed edge (i, j), we compute a weighted sum $f_{i,j}(x_i)$ of all possible operators $o(.) \in \mathcal{O}$ applied onto x_i, i.e., $o(x_i)$:

$$f_{i,j}(x_i) = \sum_{o\in\mathcal{O}} \alpha_{(i,j),o}\, o(x_i). \qquad (1)$$

The latent vector x_j for each intermediate node j is then computed as the sum of outputs from all its preceding nodes, i.e., $x_j = \sum_{i<j} f_{i,j}(x_i)$.

The two input nodes of a cell are connected to the output nodes of the previous two cells, respectively. The output node of a cell is obtained by concatenating the latent vectors of all the intermediate nodes in the cell.

In the following we will present the actor-critic framework of the DDPG algorithm and its application to Deep Deterministic Architecture Sampling.

B. THE DDPG AGENT

In RL, at time-step t, an agent in state s_t interacts with an environment by executing an action a_t. The environment in turn returns a reward, r_t, and the agent observes the next state, s_{t+1}. The goal of the agent is to maximize its *return* $R = \sum_{t=0}^{T} \gamma^t r_t$ over T time steps subject to a discounting factor γ. Generally, in continuing tasks [26] such as ours with an infinite horizon, $T = \infty$ and $\gamma \in [0, 1)$.

DDPG adopts an actor-critic framework. The actor $\mu(.)$ is a neural network that takes a state s_t as its input, and produces an action

$$a_t = \mu(s_t) + Z_t, \tag{2}$$

where Z_t is a noise added to the actor's output to encourage exploration of architectures. The critic $Q(s_t, a_t)$ is a neural network that is trained to maximize the return by predicting the action value of a state-action pair (s_t, a_t). On the other hand, the actor learns the optimal policy necessary to maximize the return.

A replay buffer is used to store the interactions of the agent with the environment, the supernet, in a form of experience tuples (s_t, a_t, r_t, s_{t+1}). *Experiences* are randomly sampled with replacement to train the actor and critic.

C. INTERACTION WITH THE ENVIRONMENT

We now describe the DDPG agent's interaction with the environment through the action, state and reward. We split all the available training data into two non-overlapping sets, the training data $\mathcal{D}_T$ and validation data $\mathcal{D}_V$; the first is used for training the supernet weights w while the latter is used to evaluate the performance of a given architecture. The DDAS procedure is illustrated in Figure 1.

We first initialize the environment by setting every element of α to one to obtain the supernet with all the operations present. Then, we *warm up* the supernet with several epochs of training [3], [4]. The accuracy of the warmed-up one shot model *OS* is denoted by *Acc(OS)*.

The DDPG agent interacts with the environment in an iterative process. In particular, the actor network of DDAS, will output the action $a_t = \alpha_t$, where α_t represents the α generated at time step t. We then use Algorithm 1 to discretize α_t to obtain $\alpha_t^d \in \{0, 1\}^{|E|\times|\mathcal{O}|}$. Recall the notation used in Section III-A and Equation 1, specifically. Algorithm 1 follows the procedure DARTS [5] use to discretize a single architecture at the *end* of a search experiment. That is, for a given intermediate node j, we select the top 2 edges with the highest operation weights incoming from all its predecessor nodes i; $i < j$. Then we discretize the two edges by setting the index of the operation with highest weight on each edge to 1. All other entries are set to 0. Following Equation 1, the operation-edge entries set to 1 are allowed to perform computation uninhibited, while all others are effectively disabled. When discretizing each subsequent node $j+1$, we must consider an additional edge, stemming from all the nodes we had to consider when discretizing node j, as well as the edge between nodes j and $j+1$. Although the number of edges to consider increases with the number of nodes, the number of edges to be discretized per node is always 2.

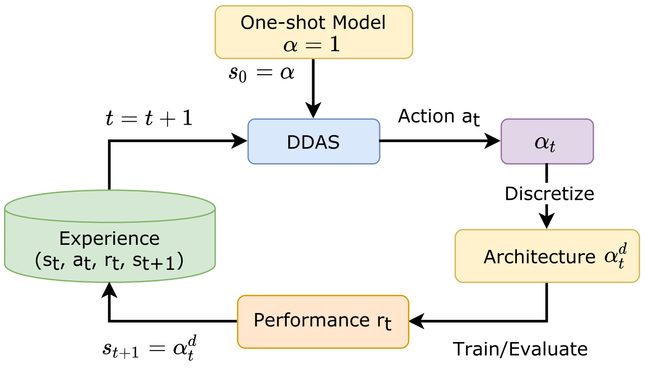


FIGURE 1. An illustration of one DDAS step. Starting from one-shot model training, DDAS selects a continuous action for discretization into a discrete architecture. The architecture is then fine-tuned and evaluated to obtain the accuracy and loss, which are used to compute the reward. The state, action, reward and next state are stored as an experience.

In fact, α_t^d corresponds to a single deterministic architecture, with a controlled complexity of only $2|N|$ edges that can perform operations. Only the corresponding weights of this architecture will be updated by SGD.

Next, using the supernet as well as the training and evaluation datasets, the discretized architecture α_t^d will be fine-tuned and evaluated by the environment according to Algorithm 2 to obtain the reward r_t and next state s_{t+1}.

To calculate the reward, we first compute the incremental changes in accuracy $\text{Acc}(\alpha_t^d)$ and loss $\mathcal{L}_V(\alpha_t^d)$ as compared to the *one-shot* model and previously selected architecture, respectively, as

$$\begin{aligned} \Delta\text{Acc}_t &= \text{Acc}(\alpha_t^d) - \text{Acc}(\text{OS}), \\ \Delta\mathcal{L}_t &= -\mathcal{L}_V(\alpha_t^d) + \mathcal{L}_V(\alpha_{t-1}^d). \end{aligned} \tag{3}$$

Next, we define the reward r_t for time step t as

$$r_t = \frac{\text{Acc}_t + \mathcal{L}_t}{2}. \tag{4}$$

The accuracy term encourages the DDAS agent to select well-performing architectures, while the validation loss term (e.g., cross-entropy loss in the case of classification)

Algorithm 1 Discretize

```
1:  **Input:** Continuous $\alpha_t \in \mathbb{R}^{|E|\times|\mathcal{O}|}$
2:  **Output:** Discrete $\alpha_t^d \in \{0, 1\}^{|E|\times|\mathcal{O}|}$
3:  Start $= 0$, $n = 1$
4:  $\alpha_t^d = 0^{|E|\times|\mathcal{O}|}$
5:  **for** $j = 0, 1, .., |N| - 1$ **do**        ▷ $N$ intermediate nodes
6:      End $=$ Start $+ n$
7:      $A = \alpha_t[\text{Start} : \text{End}, :]$        ▷ Edges of node $j$
8:      $((i,j)_1, o_1) = \arg\max_{((i,j),o)} A_{(ij),o}$        ▷ $i < j$
9:      $((i,j)_2, o_2) = \arg\max_{((i,j),o):(i,j)\neq(i,j)_1} A_{(ij),o}$
10:     $\alpha_t^d[\text{Start} + (i,j)_1, o_1] = 1$
11:     $\alpha_t^d[\text{Start} + (i,j)_2, o_2] = 1$
12:     Start $=$ End $+ 1$
13:     $n = n + 1$
14: **end for**
15: **Return** $\alpha_t^d$
```

Algorithm 2 Architecture Sampling and Evaluation

```
 1: Input: Action $\alpha_t$                                   ▷ Eq. 2
 2: Input: Supernet $S$
 3: Input: Datasets $\mathcal{D}_T, \mathcal{D}_V$
 4: $\alpha_t^d \leftarrow$ Discretize($\alpha_t$)             ▷ Algorithm 1
 5: Assign architecture $\alpha_t^d$ to $S$ to get $S(\alpha_t^d)$
 6: for $M$ minibatches do
 7:     Sample a minibatch $m$ from $\mathcal{D}_T$
 8:     Update $S(\alpha_t^d)$ using $m$                          ▷ Eq. 1
 9: end for
10: Evaluate $S(\alpha_t^d)$ on $\mathcal{D}_V$ to get Acc($\alpha_t^d$) and $\mathcal{L}_V(\alpha_t^d)$
11: Compute $r_t$ from Acc($\alpha_t^d$) and $\mathcal{L}_V(\alpha_t^d)$        ▷ Eq. 4
12: $s_{t+1} = \alpha_t^d$
13: $t \leftarrow t + 1$
14: Return $r_t, s_{t+1}$
```

encourages the agent to constantly improve. Moreover, the addition of loss in the reward is empirically critical to addressing concerns raised by [10]; that the policies of popular NAS algorithms become indistinguishable from random search. As Figure 2 shows, without a loss component, the actor policy eventually degenerates into random search.

Finally, the agent sets the next state to the selected architecture, i.e., $s_{t+1} = \alpha_t^d$, and continues the process to find a better architecture.

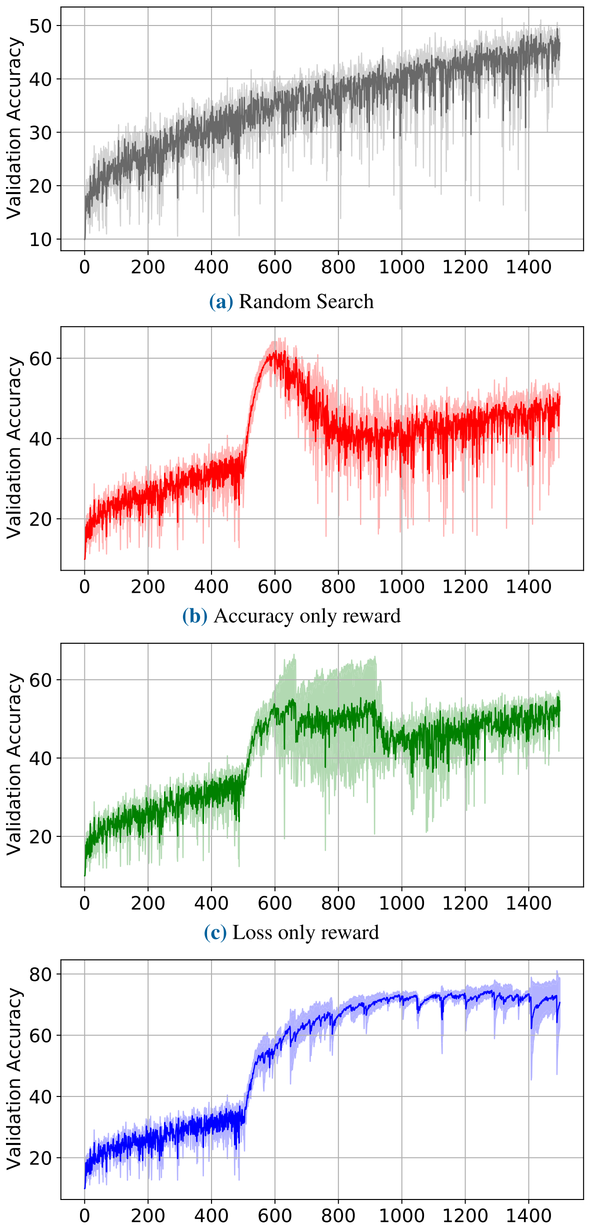
D. EXPLORATION AND EXPLOITATION

The goal of DDAS is to generate a Pareto frontier of architectures in terms of accuracy and FLOPS through a single run of the algorithm. Intuitively speaking, we can also obtain the Pareto frontier by warming up the supernet and then applying random search or evolutionary algorithms over architectures that inherit weights from the supernet. In contrast, optimization schemes such as DARTS, SNAS, etc., are not capable of depicting the Pareto frontier in one run, as gradient descent will drive these schemes to train a single or a few large architectures fully in order to minimize the validation loss of the selected architecture(s).

The ability of DDAS to discover a better Pareto frontier in one run critically depends on a balance between exploration and exploitation processes. DDPG splits exploration and exploitation into two sequential phases. Since DDPG is *off-policy*, it benefits from the use of an experiential replay buffer. In the first phase, neither the actor nor the critic is used or updated. Instead, the agent accumulates a diverse collection of state transitions in its replay buffer by sampling actions from a random distribution. In the second phase, i.e., the exploitation-centered phase, actions are generated by the actor using Equation 2. The agent samples a random batch B of experiences from its replay buffer and uses them to update the networks. First, the discounted estimation of future rewards [13] for an arbitrary step i is computed as,

$$r_i' = \gamma Q'(s_{i+1}, \mu'(s_{i+1})), \tag{5}$$

where Q' and μ' are the *target networks* used to aid in the training procedure. We refer the reader to [13], [31] for further details. The following loss is then used to update the critic,



(a) Random Search
(b) Accuracy only reward
(c) Loss only reward
(d) Full reward

FIGURE 2. Comparison of variants of DDAS with Random Search (a) on CIFAR-10 under different reward functions. Variants include DDAS using only the accuracy term (b), only the loss term (c) and the full reward (d). After 500 initial steps of random sampling, DDAS becomes unstable and nearly indistinguishable from random search when either the loss or accuracy terms are removed from the reward. We run each variant 3 times and plot the mean and standard deviation.

$$\mathcal{L}_{Critic} = \frac{1}{|B|}\sum_{i\in B}(r_i + r_i' - Q(s_i, a_i))^2. \tag{6}$$

The actor network is then updated using a sampled policy gradient from the critic,

$$\mathcal{L}_{Actor} = \frac{1}{|B|}\sum_{i\in B} Q(s_i, \mu(s_i)). \tag{7}$$

One caveat of Equation 6 is that given our definition of the reward, the actor will learn to sample the same (and most likely large) architecture repeatedly regardless of the state in order to train this architecture fully to increase the validation performance. In DDAS, we introduce exploration in architecture sampling through the use of two types of noise.

First, we introduce exploration during the exploitation phase by adding a Gaussian noise to the actor's output in every step, as in Equation 2. However, it may not be strong enough to completely randomize the actions. Rather, it perturbs α such that when discretized into α^d, it is in the same neighborhood of the actor's output. If the agent samples from a small neighborhood repeatedly, the validation performance will be guaranteed to improve, as the shared weights are repeatedly updated.

To further encourage exploration in DDAS, we introduce a new phase to follow the normal exploitation phase, where we replace the Gaussian noise by the Ornstein-Uhlenbeck process [32], which is more effective than Gaussian noise at overwriting actions [13]. Thus, we do not add Ornstein-Uhlenbeck process to the actor output in every step. When the agent detects that α has been stagnant, measured by observing minimal changes from step-to-step, for a number of steps T_{stag}, the new noise is added to the actor's output for the next T_{stag} steps. When the noise is off, the agent will focus on a small number of architectures that the critic deems worthwhile and continuously train their weights, driving up the validation performance. On the other hand, when the Ornstein-Uhlenbeck process is temporarily introduced, the newly selected architectures will become radically different, yet still having a few shared weights overlapped with previously selected architectures. This overlap of shared weights can be used to boost the performance of the newly selected architectures.

Through a combined use of the above two types of noise, the DDAS agent can switch attention to seldom sampled architectures including the smaller architectures, so that the Pareto front in terms of validation accuracy and FLOPS can be uplifted.

E. COMPUTATIONAL COMPLEXITY

The time complexity of differentiable NAS algorithms [5], [6], [20], [21], [33] is linearly bound by the number of epochs the search algorithm will execute for, which itself is a hyperparameter. While this bound provides a simple means to estimate the time it will take an experiment to run, the time to execute a single epoch can vary depending on the type of algorithm used. For example, DARTS [5] performs search using first and second-order gradient descent, the latter optimization being the more computationally expensive and slower of the two.

In contrast, the time complexity of DDAS can be measured using three metrics: The number of one-shot training epochs, the total number of RL time-steps T, and the number of minibatch updates per step M as in Algorithm 2. Given that supernets can be trained once and re-used multiple times, the computational cost of the first factor is seldom incurred. T is analogous to the number of epochs that differentiable NAS algorithms run for, as it is the number that directly quantifies the search time of the algorithm. However, most differentiable NAS algorithms run for less than 100 epochs, each epoch representing one whole pass through the training dataset. Meanwhile, one time step does not constitute one whole pass through the dataset. Rather, the M batches of data used per step constitute a small fraction of the entire dataset. This allows the search algorithm to report the performance of more architectures as many time steps can be executed in the time it takes to execute a full epoch.

IV. EXPERIMENTAL RESULTS

In this section we present and discuss the experimental results of the proposed DDAS. We first elaborate on our experimental setup in terms of dataset, one-shot supernet models as well as enumerate on several algorithm configurations to be tested. We then perform search and evaluation experiments. To illustrate the effect of different algorithm configurations, we provide plots of accuracy growth over the course of a search experiment as well as Pareto frontiers of the best architectures found during search and evaluation.

A. EXPERIMENTAL SETUP

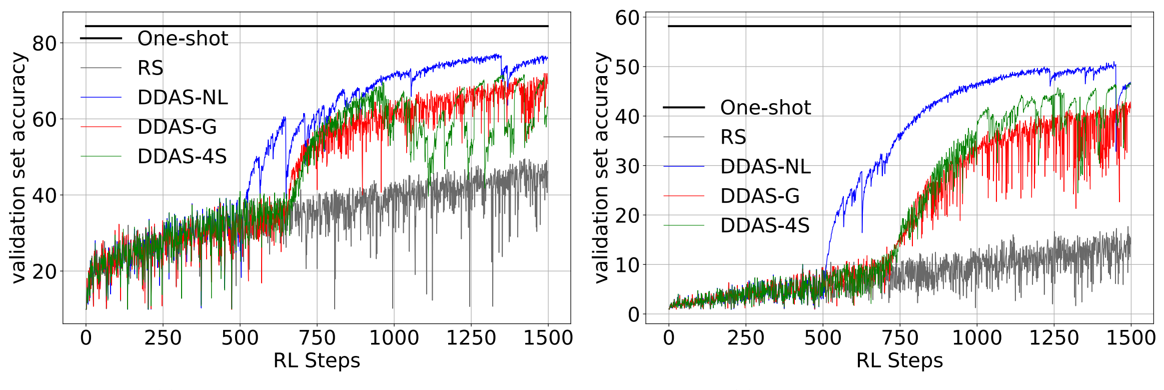
We perform our experiment on two image classification datasets, CIFAR-10 and CIFAR-100 [16]. Both contain 60k images each, of dimension size 32×32, with ten classes for CIFAR-10 and one hundred classes for CIFAR-100. Architecture search is performed on a data split similar to DARTS, resulting in a training set $\mathcal{D}_T$, validation set $\mathcal{D}_V$, and test set with sizes 25k, 25k, and 10k samples, respectively. Further evaluation of the best architectures found involves training on the official CIFAR-10 and CIFAR-100 splits that partition the data into 50k training samples and 10k testing samples.

1) WARMED-UP SUPERNET

We warm up all our architecture search experiments by training a 6-cell (with 4 normal cells and 2 reduction cells) one-shot supernet for 75 epochs on $\mathcal{D}_T$ with all elements of α set to one. Each cell contains 7 nodes, of which there are 2 input nodes, 1 output node and 4 intermediate nodes. There are 8 operations and 14 edges. Supernet training typically takes less than 6 GPU hours.

2) ARCHITECTURE SAMPLING WITH DDAS

We initialize DDAS with the warmed up supernet and start the architecture sampling process. For every sampled architecture, the supernet is trained for 25 batches on $\mathcal{D}_T$ to fit the supernet's weights to their new architecture configuration. We evaluate a total of 4 methods, consisting of a Random Search baseline and 3 different DDAS configurations. Given that the additive noise Z_t added to actions "can be chosen to suit the environment" [13], these configuration methods are primarily differentiated by the choice of Z_t and how it is applied. All 4 methods are provided below:



(a) CIFAR-10 validation curves (b) CIFAR-100 validation curves

FIGURE 3. Search validation curves on CIFAR-10 and CIFAR-100 as a function of RL steps. The 'One-shot' line represents the best validation accuracy the warmed-up supernet obtained prior to architecture search. It is horizontal, as the value corresponds to the *Acc(OS)* scalar used in Equation 3.

1) *Random Search (RS)*: Following [9], we search on the supernet by randomly sampling architectures from `Uniform(0, 1)`$^{|\mathbb{E}|\times|\mathcal{O}|}$ before performing discretization.
2) *Noiseless (DDAS-NL)*: After an initial 500 steps of exploration, DDAS enters an almost purely deterministic exploitation phase where $Z_t = U(-10^{-5}, 10^{-5})$.
3) *Gaussian (DDAS-G)*: Same as *DDAS-NL*, we engender further exploration during the exploitation phase by disrupting the actor's output a_t with a noise sampled from a Normal distribution, $Z_t = \mathcal{N}(0, 0.05)$.
4) *4-Stage (DDAS-4S)*: Behaves like *DDAS-G* for the first 500 steps of the exploitation phase. In the last 500 steps of the experiment, the additive noise is turned off by default, $Z_t = 0$, then re-enabled sporadically. The key difference is that the agent keeps track of selected architectures. Algorithm 1 reduces the number of operation-performing edges per cell type from 14 to 8, for a total of 16 across both cell types. The agent considers two architectures to be similar if less than 6 of the 16 activated operation-edge pairs between the normal and reduction cells are different. If the agent detects that it has been selecting a similar architecture for $T_{stag} = 32$ steps in a row, then a large, [32] noise will be added to the actor output for the next T_{stag} steps.

Each method runs for 1,500 steps and takes 1 GPU day to finish. For each experiment we obtain the best architecture found by DDAS with the highest validation accuracy on a given dataset. For every architecture sampled by DDAS, we calculate both the number of FLOPS and model parameters assuming the architecture was instantiated on a 6-cell network. We construct the Pareto frontier from each sampled architecture's validation accuracy on the supernet, constrained by the number of FLOPS/parameters on the 6-cell network.

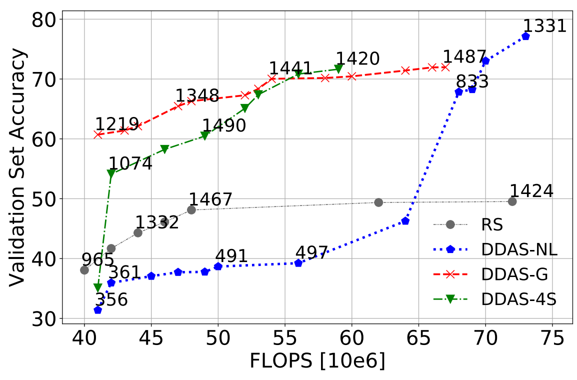
In the second half of our experiments, we forwarded many of the architectures found on the FLOPS Pareto frontiers for further evaluation on larger models for 600 epochs each. The number of cells used were 10 and 20 for CIFAR-10 and CIFAR-100, respectively.

Lastly, we took the absolute best performing architecture from each experimental setting and compared their test accuracies against those of several related NAS algorithms. For comparisons on CIFAR-10, we re-trained these architectures using 20 cell models in order to perfectly match the hyperparameter choices of DARTS [5].

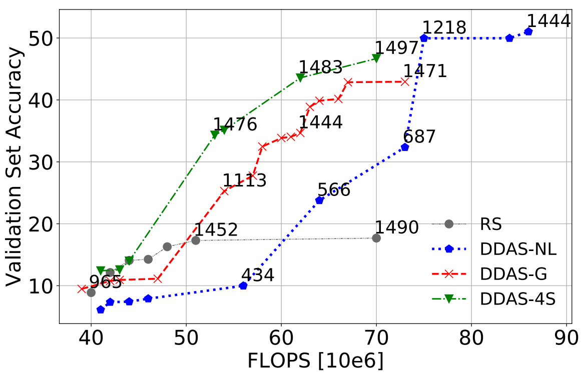
B. EVALUATION AND COMPARISON

Search validation curves for all experiments are illustrated by Figure 3. All variants of DDAS demonstrate a clear superiority over random search. The performance of *DDAS-NL* is the quickest to rise following the initial exploration steps. Moreover, the behaviour of DDAS-NL in Figure 3 on both datasets is consistent with Figure 2(d). A sharp rise in accuracy occurs after the initial steps of random actions before performance tapers off as it approaches the one-shot accuracy. *DDAS-G* and *DDAS-4S* take a few hundred additional steps before they surpass random search. Additionally, dips and rises in the plots of *DDAS-4S* clearly denote the time steps where a large noise is added to the actor output. The validation Pareto frontiers found by our search experiments, in terms of FLOPS, are presented in Figure 4. Architectures on these curves were selected for further evaluation through larger models. Note how well *DDAS-NL* appears to outperform all other methods in terms of validation performance over time, Pareto frontier regions corresponding to smaller FLOPS are dominated by *DDAS-G* and *DDAS-4S*.

We adopt the definition introduced by [11] for measuring the width and depth of NAS cells. These metrics are a means of quantifying the degree of exploration the search algorithm is performing in terms of the cell topologies selected among high-performing architectures. A narrow distribution of cell widths centered around a high number indicates a systemic and undesirable preference for shallower architectures and therefore low exploration. Denoted with 'c', the width is the average number of edges originating from the input nodes,

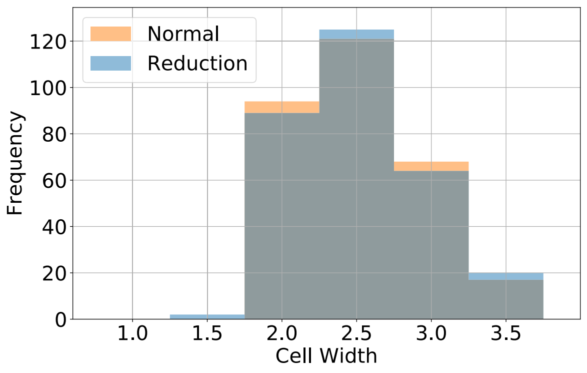


(a) CIFAR-10 FLOPS Pareto frontiers

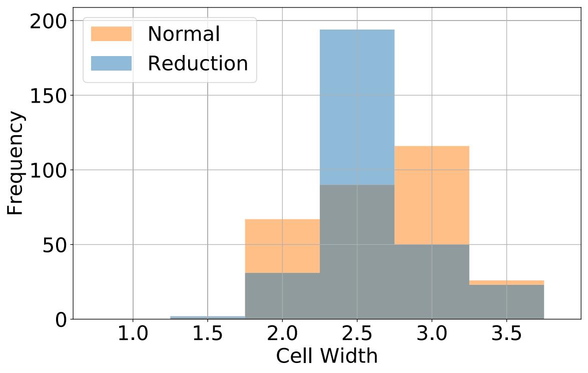


(b) CIFAR-100 FLOPS Pareto frontiers

FIGURE 4. Search Pareto frontiers constraining accuracy against FLOPS. Each line is generated by one of the four search methods. Numerical annotations denote the step *t* where an architecture was sampled. Any step below 500 is guaranteed to be generated from a uniform random distribution.



(a) CIFAR-10 Cell Width Histogram



(b) CIFAR-100 Cell Width Histogram

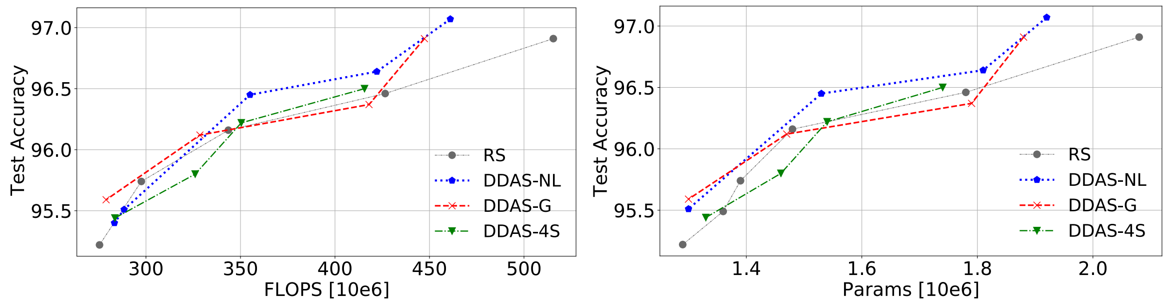
FIGURE 5. Search histograms of normal and reduction cell width distributions on CIFAR-10 and CIFAR-100 across architectures in the top 5% accuracy percentile. Width is defined as the average number of edges (operations) originating from the two input nodes, and can take values between 0.5 and 4.

while the depth is the length of the longest path between the input and output nodes. We exclude the 'none' operation from these calculations. Put quantitatively, in the case of DARTS, a 'wide cell' has a width of approximately 3c or more, corresponding to at least 6 of the 8 edges in E originating from one of the input nodes, rather than linking one intermediate node to another. More specifically, the normal cell found by the second-order DARTS [5] has a width and depth of 3.5c and 3, respectively, while the reduction cell has a width of 2.5c and a depth of 3.

Figure 5 displays the histograms of cell widths for cells in the top 5% accuracy percentile for all experiments on both datasets. The distribution of architectures for both datasets resembles that of a Gaussian distribution centered around 2.5; corresponding to 2-3 edges per input node, in the case of CIFAR-10. For CIFAR-100, the distribution of normal cells more closely resembles a uniform distribution bounded between 2 and 4. Reduction cell widths follow a narrow Gaussian centered around 2.5. Regardless of the distribution, it is clear that respectable accuracy metrics can be found across a spectrum of cell widths—high accuracies are not limited to a narrow range of cells with large widths. These findings corroborate our claim that NAS algorithms should incorporate a higher degree of exploration and avoid being biased toward a specific type of topologies.

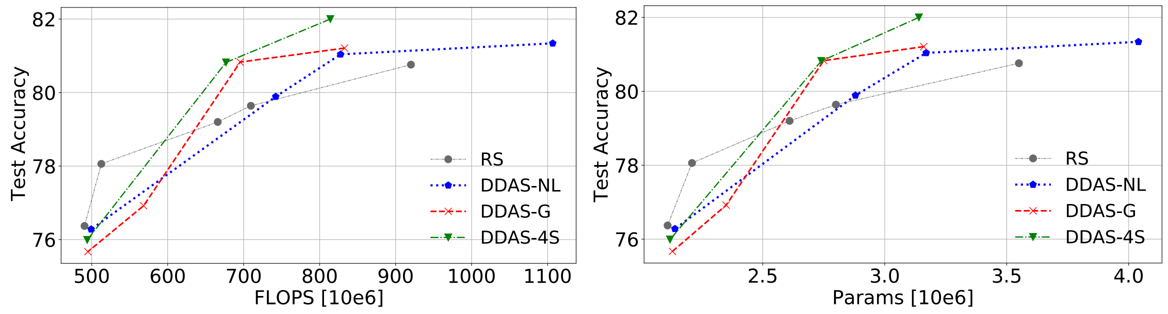
Test set Pareto frontiers, in terms of both FLOPS and total number of parameters on CIFAR-10 and CIFAR-100, are given by Figures 6 and 7, respectively. By test set accuracy, the Pareto frontiers of all three DDAS configurations are higher than those of *RS* in at least one region. This reflects the search curves where their architectures were chosen from. *DDAS-NL* is the sole exception to this observation. *DDAS-NL* produced the highest test score on CIFAR-10 and the highest validation scores on both datasets. According to Figure 4, the only architectures *DDAS-NL* chose that had a small number of FLOPS were sampled during the initial 500-step exploration phase, or shortly afterward. When comparing *DDAS-G* to *DDAS-4S* we observe that their evaluation Pareto frontiers almost identically match the ones generated during the search. On CIFAR-10, *DDAS-G* is better at sampling low-FLOPS architectures, but is eventually overtaken by *DDAS-4S*. Meanwhile, on CIFAR-100, the *DDAS-4S* Pareto frontiers completely dominate *DDAS-G* on both search and evaluation.

Our best cell architectures for CIFAR-10 and 100 are given by Figures 8 and 9, respectively. With exception to the normal



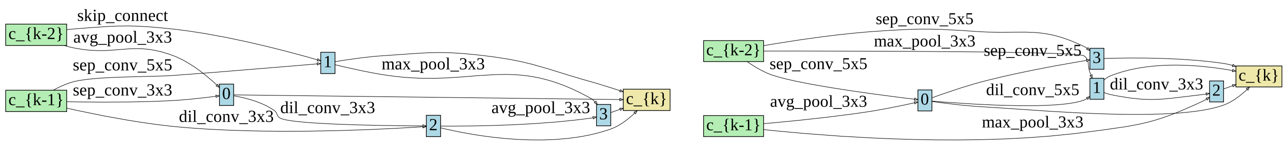
(a) FLOPS Pareto frontiers
(b) Parameter Pareto frontiers

FIGURE 6. Test set evaluation Pareto frontiers for CIFAR-10 constraining accuracy against FLOPS or parameters. Points correspond to architectures present on the search Pareto frontier for a given search scheme. All models were trained using 10 cells - 8 normal and 2 reduction.



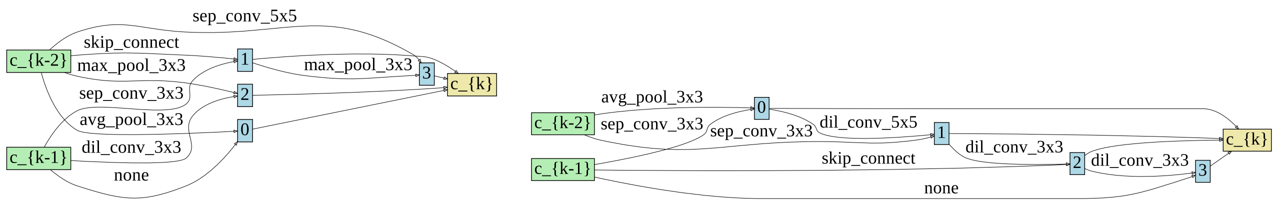
(a) FLOPS Pareto frontiers
(b) Parameter Pareto frontiers

FIGURE 7. Test set evaluation Pareto frontiers for CIFAR-100 constraining accuracy against FLOPS or parameters. Points correspond to architectures present on the search Pareto frontier for a given search scheme. All models were trained using 20 cells, including 18 normal cells and 2 reduction cells.



(a) Normal Cell, 2.5c, 4
(b) Reduction Cell, 2.5c, 4

FIGURE 8. Best set of cells found on CIFAR-10, annotated with width (c) and depth according to [11]. Green nodes represent input from the two previous cells in the network. Edges between intermediate nodes (blue) and the cell output (yellow) do not perform operations and are not considered a part of *E*. These cells were found using the noiseless configuration (DDAS-NL) at step 833.



(a) Normal Cell, 3c, 3
(b) Reduction Cell, 2c, 5

FIGURE 9. Best set of cells found on CIFAR-100, annotated with width (c) and depth according to [11]. Green nodes represent input from the two previous cells in the network. Edges between intermediate nodes (blue) and the cell output (yellow) do not perform operations and are not considered a part of *E*. These cells were found using the four-stage configuration (DDAS-4S) at step 1483.

cell for CIFAR-100, no cell has a width above 3 nor a depth smaller than 3. This demonstrates that DDAS is not prone to the same issue as cells found by other NAS algorithms as listed by [11]. That is, the layout of the cells do not resemble a wide, shallow neural network; each input is not simply passed to each node independently before being aggregated at the output. Instead, the inputs are subject to a series of sequential operations as they are passed from one node onto the next.

Next, we compare the test performance of the best architectures found by all four of our methods to those reported by

TABLE 1. Comparison of DDAS schemes with related NAS weight sharing algorithms in terms of test accuracy, FLOPS and number of parameters.

	CIFAR-10			CIFAR-100		
Architecture	**FLOPS [G]**	**Params [M]**	**Test Acc. (%)**	**FLOPS [G]**	**Params [M]**	**Test Acc. (%)**
DARTS First Order	1.022	3.65	97.00 ± 0.14	1.022	3.77	82.37
DARTS Second Order	1.078	3.83	97.24 ± 0.09	1.078	3.95	82.65
ENAS	–	4.60	97.11	-	-	-
ProxylessNAS-G	–	5.70	97.92	–	–	–
GDAS	–	3.40	97.07	–	3.40	81.62
GDAS (FRC)	–	2.50	97.18	–	2.50	81.87
SNAS (Mild Const.)	–	2.90	97.02	–	–	–
SNAS (Moderate Const.)	–	2.80	97.15	–	–	–
RS	1.024	3.67	97.16	0.920	3.55	80.76
DDAS-NL	*0.876*	*3.23*	*97.27*	1.106	4.04	81.34
DDAS-G	0.839	3.10	96.81	0.916	3.47	80.88
DDAS-4S	0.842	3.07	96.74	*0.814*	*3.14*	*82.00*

several related NAS algorithms that use weight sharing and rely on a few GPUs. The results are given in Table 1. We manually evaluated the publicly available architectures found by DARTS first-order and second-order on CIFAR-100.

Table 1 provides evidence that DDAS is superior to ENAS [8], GDAS [21] and SNAS [6], where the latter two employs exploration in the form of Gumbel Softmax. The only architectures whose scores are higher than DDAS are ProxylessNAS [7] on CIFAR-10 and DARTS [5] on CIFAR-100. Both methods achieve their high accuracy metrics at the cost of substantially larger model sizes.

Comparing our experimental configurations against each other, we observe the superiority of *DDAS-NL* and *Random Search* over *DDAS-G* and *DDAS-4S* on CIFAR-10. Both of these algorithms favored architectures with a much higher number of parameters than *DDAS-G* and *DDAS-4S*. Most notably *Random Search* is the more inefficient of the two. Moreover, the situation is partially true on CIFAR-100, where *DDAS-G* and *DDAS-4S* reign supreme with fewer parameters.

DDAS-NL is most comparable to gradient-based NAS algorithms due to a low, almost negligible amount of exploration during exploitation. Conversely, *DDAS-4S* encorporates mechanisms that allow it to actively fight against the sampled policy gradient of its critic, while *DDAS-G* does not heavily depart from the original specification of DDPG given by [13]. On CIFAR-100 *DDAS-4S* completely outperformed *DDAS-NL*, both in terms of performance and parameter efficiency. CIFAR-100 is inherently more difficult to classify than CIFAR-10 due to having the same number of samples but 10 times as many classes and therefore 10 times fewer samples per class. Thus, it can be said that *DDAS-4S* demonstrates the benefits of modifying RL algorithms beyond the scope of their original theory for use in NAS problems. In addition, we approximated the slope of accuracy against FLOPS or parameters using linear regression. For CIFAR-10, we found that test accuracy increased at rates of 2.86% per gigaFLOPS and 2.123% per million parameters, both with linear correlations over 0.93. For CIFAR-100, these values are higher at 4.03% per gigaFLOPS and 3.196% per million parameters, linearly correlated over 0.86. These metrics quantify the small loss of accuracy entailed by downsizing model size and indicate the ability of DDAS to find resource-efficient architectures for practical deployment.

TABLE 2. Spearman correlation coefficients between validation and evaluation accuracies for Pareto front cells.

Setting	**CIFAR-10**	**CIFAR-100**
RS	0.964	1.000
DDAS-NL	0.881	0.826
DDAS-G	0.886	0.810
DDAS-4S	0.886	0.600

We also computed the ranking correlation between the validation and evaluation scores of all Pareto frontier architectures we evaluated. The Spearman coefficients are given in Table 2. Random Search achieves the highest correlation on both datasets as it performs a uniform scan of the search space and does not focus on specific regions. As shown in Figures 4, Random Search performance struggles to improve past 45 megaFLOPS, resulting in only a handful of architectures being selected in high FLOPS regions. Thus, most of the Random Search architectures are located in low FLOPS regions where small increases in FLOPS have a greater impact on accuracy. On both datasets, all three DDAS configurations achieve high correlation coefficients that exceed 0.5. This is because DDAS is a guided algorithm that searches disproportionately and focuses on learning where high-performance architectures are likely to be found, and therefore finds larger architectures where the choice of operation and topology play a larger role in determining accuracy.

Finally, the search cost of DDAS is relatively comparable to DARTS. On a single RTX 2080 Ti GPU, DDAS takes approximately 6 GPU hours to train a one-shot model which only needs to be pre-trained once and can be re-used in multiple searches. Search itself costs approximately 1 GPU day to run for 1,500 steps. It is worth noting that DARTS, and GDAS ran their search experiments four or three times with different random seeds in order to pick the best architecture according to the validation accuracy. Repeated searches are a mechanism to encourage exploration. In contrast, DDAS is designed to explore, train and identify a range of good architectures in the same search run.

V. CONCLUSION

In this paper, we introduce Deep Deterministic Architecture Search (DDAS), an algorithm based on deep deterministic policy gradient (DDPG) in Reinforcement Learning, to thoroughly explore a neural architecture search space and perform neural architecture search by sampling and training architectures on a weight-sharing supernet. Unlike prior reinforcement learning schemes for NAS which use stochastic policy gradient to sample architectures, DDAS uses a deterministic policy and leverages the ability of DDPG to handle high-dimensional control in a continuous space. Coupled with a loss-based reward function, the policy of DDAS is distinct from random search and can learn to focus on important regions of the search space.

Furthermore, DDAS addresses the lack-of-exploration issue present in recent optimization-based NAS frameworks via several exploration schemes. Unlike gradient-based NAS schemes such as DARTS or GDAS, which perform multiple search runs to produce a single architecture, DDAS instead performs one long search experiment which produces a Pareto frontier containing a spectrum of architectures. As a result, DDAS is capable of generating architectures for flexible deployment on target hardware where FLOPS or model size may be constrained, without the need to incorporate a specific resource penalty into the reward. Additionally, the cells produced by DDAS are not always wide and shallow or biased toward a specific type of topologies. We performed extensive experiments on CIFAR-10 and CIFAR-100 in a wide range of experimental settings. With a test accuracy of 97.27%, experimental results have shown that DDAS is capable of generating architectures that outperform the original DARTS with a lower number of parameters on CIFAR-10. On CIFAR-100, DDAS finds an architecture that is capable of achieving 82.00% test accuracy with only 3.14M parameters, outperforming GDAS. In addition, in a single search algorithm run for 1 GPU day, DDAS can produce Pareto frontiers that outperform random search based on a warm-started supernet, demonstrating its superior capability to automatically explore and discover important regions of a neural architecture search space.

APPENDIX

A. OPERATIONS

The operation set $\mathcal{O}$ used in the DARTS search space consists of the following:

1) None (Zero input tensor)
2) Maximum Pooling 3×3
3) Average Pooling 3×3
4) Skip Connection
5) Separable Convolution 3×3
6) Separable Convolution 5×5
7) Dilation Convolution 3×3
8) Dilation Convolution 5×5

We make no change to these operations relative to how they are implemented by DARTS [5] and allow each of them to be selected by Algorithm 1.

B. HYPERPARAMETERS IN SEARCH

Our weight-sharing search models, modified from DARTS [5], all have 6 cells (4 normal and 2 reduction cells). Data enters through a head which applies a channel multiplier of 16 as well as a few preliminary convolution operations, before being passed on to the cells. A batch size of 64 is used at all times, and each supernet is trained over the course of 75 epochs on the 25k training set, $\mathcal{D}_T$. We followed the precedent set by DARTS [5] and utilized a stochastic gradient descent optimizer with momentum. During one-shot supernet training, the initial learning rate is set to 2.5×10^{-2}, but is annealed down to 10^{-3} by a cosine schedule without restarts [34]. When searching for an architecture using DDAS, we set the learning rate to a constant value of 10^{-3}. For reproducibility, all experiments are initialized with the same random seed values of 2 for search and 0 for evaluation. Random seeds values of 0, 1 and 2 were used to generate Figure 2.

C. HYPERPARAMETERS USED IN EVALUATION

Once a cell architecture is found and sent for evaluation (testing), the tested network consists of 10 or 20 cells for CIFAR-10 and CIFAR-100, respectively. The channel multiplier present at the beginning of a network is increased to 36. The same cosine annealed SGD with momentum optimizer is used here, except now the learning rate is annealed down to a value of 0 over the course of every experiment, all of which lasted 600 epochs with a batch size of 96. Finally, we also made use of DARTS path dropout feature, with a probability of 0.2, and an auxiliary head with a weight of 0.4.

When further evaluating the best CIFAR-10 architectures for Table 1, we re-ran the evaluation experiments with 20 cells. This allowed us to directly compare our results with those of DARTS [5]. In all experiments, we made use of Cutout [35] using the recommended lengths for CIFAR-10 and CIFAR-100.

D. REINFORCEMENT LEARNING HYPERPARAMETERS IN DDAS

We first describe the hyperparameters common to all versions of DDAS, before listing the hyperparameter discrepancies among different DDAS versions in Table 3. Our RL code is based off of [36].

The actor and critic networks of DDAS are both MLPs with 3 hidden layers and 256 neurons in each layer that receive vectorized α matrices as input. Both networks are trained using Adam [37] with its default parameters of $\vec{\beta} = (0.9, 0.99)$ and learning rates of 10^{-4} and 10^{-3}, respectively. ReLU [38] is used as the internal activation function for both the actor and the critic. However, the actor's final layer uses a sigmoid activation (σ) to truncate the output into the range (0, 1). The critic does not utilize any final activation at all, because it produces a scalar. The target networks (see DDPG [13] for details) are synchronized at every step using a mixing coefficient of 10^{-3}. The replay buffer is truncated to only hold experiences from the last 500 time steps during

TABLE 3. Hyperparameters specific to different versions of DDAS algorithms.

Setting	Explore Steps	Exploit Steps	Final Steps	$Z_{Exploit}$	Z_{4S}
DDAS-NL	500	1,000	0	$U(-10^{-5}, 10^{-5})$	N/A
DDAS-G	500	1,000	0	$\mathcal{N}(0, 0.05)$	N/A
DDAS-4S	500	500	500	$\mathcal{N}(0, 0.05)$	[29]

Phase 4. The size of the buffer is 10^6 at all other times. The number of experiences, $|B|$, sampled from the replay buffer is always 64. The discount factor γ is set to 0.99.

DDAS uses a Gaussian noise $\mathcal{N}(0, 0.05)$ during its exploitation phase (DDAS-G) before adopting the Ornstein-Uhlenbeck [32] process for its final, fourth stage (DDAS-4S). Unlike DDPG [13], the actor and critic networks are completely separate with no overlap between their parameters. We do not apply any regularization to either network.

E. COMPUTING PLATFORMS

Workstations used to run our experiments were equipped with Threadripper 2990WX processors, with two exceptions: One computer used a Ryzen 9 3900X, and the other was equipped with a Intel Core i9-9900X. All systems were equipped with dual RTX 2080 Ti GPUs.

REFERENCES

[1] D. R. So, C. Liang, and Q. V. Le, "The evolved transformer," 2019, *arXiv:1901.11117*. [Online]. Available: http://arxiv.org/abs/1901.11117

[2] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[3] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 550–559.

[4] 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," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 544–560.

[5] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[6] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," 2018, *arXiv:1812.09926*. [Online]. Available: http://arxiv.org/abs/1812.09926

[7] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[8] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2018, pp. 4095–4104.

[9] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in Artificial Intelligence*. 2020, pp. 367–377.

[10] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *Proc. ICLR*, 2020.

[11] Y. Shu, W. Wang, and S. Cai, "Understanding architectures learnt by cell-based neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[12] X. Jin, J. Wang, J. Slocum, M.-H. Yang, S. Dai, S. Yan, and J. Feng, "RC-DARTS: Resource constrained differentiable architecture search," 2019, *arXiv:1912.12814*. [Online]. Available: http://arxiv.org/abs/1912.12814

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2016.

[14] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1329–1338.

[15] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2014, pp. 387–395.

[16] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[17] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[18] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[19] Y. Zhang, Z. Lin, J. Jiang, Q. Zhang, Y. Wang, H. Xue, C. Zhang, and Y. Yang, "Deeper insights into weight sharing in neural architecture search," 2020, *arXiv:2001.01431*. [Online]. Available: http://arxiv.org/abs/2001.01431

[20] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1294–1303.

[21] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1761–1770.

[22] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[23] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2820–2828.

[24] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[27] B. Wu, K. Keutzer, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, and Y. Jia, "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10734–10742.

[28] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-Path NAS: Designing hardware-efficient convnets in less than 4 hours," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. New York, NY, USA: Springer, 2019, pp. 481–497.

[29] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via Lamarckian evolution," 2018, *arXiv:1804.09081*. [Online]. Available: http://arxiv.org/abs/1804.09081

[30] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *Proc. NIPS Deep Learn. Workshop*, 2013. [Online]. Available: https://www.cs.toronto.edu/ vmnih/docs/nipsdlw2013.bib

[32] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Phys. Rev.*, vol. 36, p. 823, Sep. 1930.

[33] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, "PC-DARTS: Partial channel connections for memory-efficient architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[34] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. ICLR*, 2017.

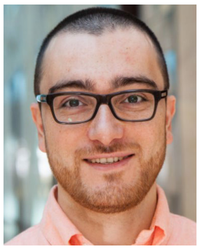
[35] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017, *arXiv:1708.04552*. [Online]. Available: http://arxiv.org/abs/1708.04552

[36] S. Zhang. (2018). *Modularized Implementation of Deep RL Algorithms in PyTorch*. [Online]. Available: https://github.com/ShangtongZhang/DeepRL

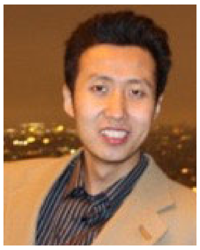
[37] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, Dec. 2014.

[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.

KEITH G. MILLS received the B.Sc. (Hons.) and M.Sc. degrees in computer engineering from the University of Alberta, in 2018 and 2020, respectively. He is currently pursuing the Ph.D. degree under the supervision of Dr. Di Niu. He is currently an Associate Researcher and an Intern at Huawei Technologies Canada Company Ltd., Markham, Canada. His research interest includes simplifying and generalizing neural architecture search problems.



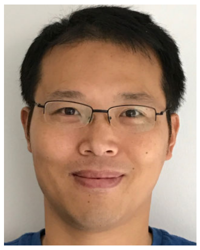
MOHAMMAD SALAMEH received the Ph.D. degree from the University of Alberta under the supervision of Dr. Greg Kondrak and Dr. Colin Cherry, with a main focus on statistical machine translation and sentiment analysis. He is currently a Senior Researcher at Huawei Technologies Canada Company Ltd. He is also working on neural architecture search with a focus on gradient-based and reinforcement learning approaches. He co-organized Determining Sentiment Intensity in Tweets (SemEval2016) and Affects in Tweets (SemEval2018) shared tasks.



DI NIU received the B.Eng. degree from Sun Yat-sen University, in 2005, and the M.Sc. and Ph.D. degrees from the University of Toronto, in 2009 and 2013, respectively. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Alberta, specialized in the interdisciplinary areas of distributed systems, data mining, machine learning, text mining, and optimization algorithms. He was a recipient of the Extraordinary Award of the CCF-Tencent Rhino Bird Open Grant 2016 for his research on natural language processing and machine learning for web document understanding at scale.

FRED X. HAN received the M.Sc. degree in electrical and computer engineering from the University of Alberta, in 2019, with specialization in software engineering and intelligent systems. He is currently a Research Associate at Huawei Technologies Canada Company Ltd. His research interests include deep learning, reinforcement learning, data mining, knowledge discovery, and automated machine learning.

SEYED SAEED CHANGIZ REZAEI received the Ph.D. degree in graduate studies from the University of Waterloo, with a focus on network information theory and combinatorics and optimization. He has been working as a Senior Machine Learning Researcher at Huawei Technologies Canada Company Ltd., since April 2019. Before Joining Huawei, he was a Researcher in optimization, graph theory, and machine learning at 1QBit Information Technology, and also held a postdoctoral position with the Department of Mathematics, Simon Fraser University.



HENGSHUAI YAO received the Ph.D. degree in reinforcement learning from the Reinforcement Learning and Artificial Intelligence (RLAI) Laboratory, Department of Computing Science, University of Alberta, in 2014. His thesis is on model-based reinforcement learning with linear function approximation. During his Ph.D. studies, he worked with Csaba Szepesvari, Rich Sutton, Dale Schuurmans, and Davood Rafiei, where he working on reinforcement learning theory, algorithms, and web applications. He joined NCSoft Game Studio, San Francisco, in 2016, where he working on reinforcement learning in games. He moved back to Canada and joined Huawei, in 2017. He was recently appointed as an Adjunct Professor with the Department of Computing Science, University of Alberta.

WEI LU (ROBIN) received the master's degree in electrical engineering from Southeast University, China. After he graduated from university, he worked at Huawei as an IC Engineer in mobile chipset design for two years. He then transferred to Huawei Kirin Solution as a Software Engineer, where he was a Senior Manager of User Experience Team, where he mainly focused on power and performance. He is currently the Director of Huawei Technologies Canada Company Ltd., Edmonton Research Center, and leading two AI research teams in Edmonton.

SHUO LIAN received the Ph.D. degree from the School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China. He is currently a Technical Team Leader with Model Optimization Team, Huawei Kirin Solution, Shanghai, China. His research interests include deep learning, model optimization, efficient algorithms, network measurement, and hardware for computation-intensive AI applications.

SHANGLING JUI is the Chief AI Scientist for Huawei Kirin Chipset Solution. He is an expert in machine learning, deep learning, and artificial intelligence. Previously, he was the President of the SAP China Research Center and the SAP Korea Research Center, responsible for 2400 employees and 150 million USD research and development annual budget. He was also the CTO of Pactera, leading innovation projects based on cloud and big data technologies. He is currently an Expert Reviewer of the Project Committee for China-EU Science and Technology Co-Operation and a Guest Professor of the Software Institute of Beijing University. He has published various books and articles about the Chinese software industry and big data analytics in China, U.K., Australia, and the USA. He has 27 years of working experience in Germany, the USA, and China. He received the Magnolia Award from the Municipal Government of Shanghai, in 2011.