L^{2}NAS: Learning to Optimize Neural Architectures via Continuous-Action Reinforcement Learning

Keith G. Mills[1†], Fred X. Han[2], Mohammad Salameh[2], Seyed Saeed Changiz Rezaei[2]
Linglong Kong[1], Wei Lu[2], Shuo Lian[3], Shangling Jui[3], Di Niu[1]
[1]University of Alberta, Edmonton, AB, Canada
[2]Huawei Technologies, Edmonton, AB, Canada
[3]Huawei Kirin Solution, Shanghai, China

ABSTRACT

Neural architecture search (NAS) has achieved remarkable results in deep neural network design. Differentiable architecture search converts the search over discrete architectures into a hyperparameter optimization problem which can be solved by gradient descent. However, questions have been raised regarding the effectiveness and generalizability of gradient methods for solving non-convex architecture hyperparameter optimization problems. In this paper, we propose L^{2}NAS, which learns to intelligently optimize and update architecture hyperparameters via an actor neural network based on the distribution of high-performing architectures in the search history. We introduce a quantile-driven training procedure which efficiently trains L^{2}NAS in an actor-critic framework via continuous-action reinforcement learning. Experiments show that L^{2}NAS achieves state-of-the-art results on NAS-Bench-201 benchmark as well as DARTS search space and Once-for-All MobileNetV3 search space. We also show that search policies generated by L^{2}NAS are generalizable and transferable across different training datasets with minimal fine-tuning.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Reinforcement learning*; Supervised learning by classification; • **Theory of computation** → *Continuous optimization.*

KEYWORDS

Neural Architecture Search; Deep Deterministic Policy Gradient

ACM Reference Format:
Keith G. Mills[1†], Fred X. Han[2], Mohammad Salameh[2], Seyed Saeed Changiz Rezaei[2] and Linglong Kong[1], Wei Lu[2], Shuo Lian[3], Shangling Jui[3], Di Niu[1]. 2021. L^{2}NAS: Learning to Optimize Neural Architectures via Continuous-Action Reinforcement Learning. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482360

†Work done during an internship at Huawei Technologies Canada.

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482360

1 INTRODUCTION

Neural architecture search (NAS) automates neural network design and has achieved state-of-the-art performance in computer vision tasks [22], [30], [2], [19]. A NAS scheme usually consists of a search strategy and a performance estimation strategy for candidate architectures. Typical search strategies include random search [20], Bayesian optimization [16], and reinforcement learning [26, 46].

Differentiable architecture search (DARTS) [22] and the variants based on optimization, e.g., [2, 4, 5, 19, 35, 38] have dramatically reduced the search cost in NAS by converting search in an exponentially large space into optimization performed on a weight-sharing *supernet*, where all candidate architectures are mixed through the architecture hyperparameters α. The architecture hyperparameters α can then be optimized with gradient descent to minimize the meta loss of the supernet on a validation set.

However, differentiable architecture search has brought about a few issues. First, during search DARTS aims to minimize the validation loss of the mixed supernet, whereas the final evaluation is performed on a discrete architecture extracted from the supernet based on the largest architecture hyperparameters. Such a discrepancy leads to the optimization gap known as the discretization error [36, 41] in NAS literature. Second, although stochastic gradient descent (SGD) has been remarkably effective for training neural network weights [34], it is unclear whether gradient descent extends to the bi-level optimization of both architecture hyperparameters and model weights, where the former can lie in non-Euclidean domains [19]. Recent work shows that DARTS is unstable and unable to generalize to different search spaces, some of which contain dummy operators that DARTS fails to rule out during the search [42].

Another common problem facing differentiable NAS approaches is a lack of exploration mechanisms. By design, gradient-based methods seek the nearest local loss minima as quickly as possible. Consequently, gradient-based NAS approaches are reported to be driven toward regions of the search space where the supernet can train rapidly. This results in wide and shallow architectures being selected over deeper and narrower networks [28]. The above issues have given motivation to developing more advanced schemes than gradient descent for differentiable neural architecture search.

In this paper, we propose L^{2}NAS which given a search space of candidate architectures, learns *how* to optimize the architecture hyperparameters α in differentiable NAS, in order to achieve generalizable hyperparameter optimization policies in NAS that are transferable across training datasets. Rather than always descending in the direction of gradients, L^{2}NAS updates α through

a trainable actor neural network taking as input the statistical information learned from the search history. In other words, L^{2}NAS learns to learn (optimize) a search policy for α. In particular, we make the following contributions:

First, L^{2}NAS learns to generate architecture hyperparameters α in a continuous domain via an actor network based on MLP. The hyperparameters α then undergo a many-to-one deterministic mapping into a discrete architecture, based on which the reward of α can be obtained from either a weight-sharing supernet or from other performance estimation mechanisms such as querying true performance, if available. We design a novel state representation of the optimization landscape by performing average pooling of top K architectures found in the search history, which will be fed into the actor to determine α in the next step. Thus, L^{2}NAS can also be understood as replacing the gradient descent update rule with a meta-learned architecture hyperparameter optimizer represented by the actor.

Second, we propose an effective and efficient procedure to train L^{2}NAS through a quantile-driven loss. Leveraging the Actor-Critic framework, we introduce a critic network to predict the reward given α. Due to the stochastic nature of the reward model, the critic is specifically trained by the check loss in quantile regression [18] to predict the 90th or greater percentile of performance instead of expected mean performance under each α to track high potential regions in the search domain. The actor network is then updated through an efficient gradient ascent algorithm based on the critic output, following a similar framework in continuous-action reinforcement learning [21, 24].

Third, we incorporate several exploration strategies into L^{2}NAS, including a noise-based exploration mechanism and the ϵ-greedy strategy, to ensure that the search space be adequately explored to avoid premature convergence.

Through extensive experiments, we show that L^{2}NAS achieves state-of-the-art performance on the public benchmark set NAS-Bench-201 [12] in terms of accuracy constrained by the number of architecture evaluations incurred, which converts to search cost in reality. In DARTS [22] and Once-for-All (OFA) [2] search spaces, we show that L^{2}NAS can find models that achieve competitive accuracies on CIFAR-10 and ImageNet, as compared to existing methods operating in the same search spaces. Furthermore, we show that the architecture optimization policy pre-trained on a simpler dataset, e.g., CIFAR-10, can be transferred to search for competitive architectures that perform well on more complex or higher-resolution datasets, e.g., CIFAR-100 and ImageNet, after a small amount of fine-tuning, further reducing the cost of customizing architecture searches per dataset.

2 PROPOSED METHOD

Neural architecture search can be formulated as a black-box optimization problem [6, 7]. Suppose that x is a neural architecture. Let $S(x)$ be a real-valued function representing its performance after fully training the network on a given dataset, i.e., the accuracy of x. In NAS, the goal is to maximize $S(x)$ subject to $x \in \mathcal{X}$, where $\mathcal{X}$ denotes a predefined search space of neural architectures. The black-box function S is not available in a closed form, but can be evaluated at any query point $x \in \mathcal{X}$. In other words, S can only be observed through point-wise observations.

Since the size of $\mathcal{X}$ grows exponentially in terms of the number of operators considered, differentiable architecture search [22] relaxes $\mathcal{X}$ into a continuous space of architecture representations, α, a.k.a. *architecture hyperparameters*, converting the search in $\mathcal{X}$ into an optimization problem over α in a continuous domain. That is, one can define a deterministic mapping function `Discretize` such that $x = \texttt{Discretize}(\alpha)$, and maximizing $S(x)$ is converted to maximizing $S(\alpha)$.

2.1 Continuous Relaxation of Discrete Architectures

We now describe how the continuous relaxation of a neural architecture can be achieved using DARTS [22] as an example. A neural network can be represented by a directed acyclic graph (DAG) $G = (N, E)$ with $|N|$ numbered nodes $1, 2, \ldots, |N|$ and a set of directed edges E connecting them. The nodes are latent data representations in a neural network. Let x_i denote the latent representation on node i. Let O denote the predefined set of candidate operations.

In a discrete architecture in DARTS, each edge represents one operation in O. But in its continuous relaxation G, a.k.a. *the supernet*, each edge (i, j) performs a weighted sum of all candidate operations applied onto x_i. Specifically, let $\alpha_{(i,j),o} \in \mathbb{R}$ be the raw weight value of operation o on edge (i, j), the computation performed on the edge (i, j) from node i to node j $(i < j)$ is defined as:

$$f_{i,j}(x_i) = \sum_{o \in O} \frac{\exp(\alpha_{(i,j),o})}{\sum_{o' \in O} \exp(\alpha_{(i,j),o'})} \cdot o(x_i).$$

The result of edge (i, j) will be aggregated with other incoming edges at node j, if any, to yield

$$x_j = \sum_{i:(i,j) \in E} f_{i,j}(x_i).$$

We could then define $\alpha = \{\alpha_{(i,j),o}\} \in \mathbb{R}^{|E| \times |O|}$ as the hyperparameters matrix, which is a relaxed representation of all possible networks in the search space. By optimizing α and supernet weights w to increase $S(\alpha)$, i.e., validation accuracy of the supernet under α, α would eventually converge to the region of better performing architectures. DARTS achieves this by updating α and w using gradient descent in a bi-level optimization setup:

$$\min_{\alpha} \quad S(\alpha) = \mathcal{L}_{\text{val}}(w^*(\alpha), \alpha) \tag{1}$$

$$\text{s.t.} \quad w^*(\alpha) = \arg\min_{w} \mathcal{L}_{\text{train}}(w, \alpha), \tag{2}$$

where all supernet weights w are fit to the training set given α, while α is found by minimizing the validation loss (of the entire supernet) on a separate validation set.

Unlike DARTS, we set $S(\alpha) = S(\texttt{Discretize}(\alpha))$ in L^{2}NAS to evaluate α directly by the performance of the individual architecture derived from α via discretization. Similarly, we train supernet weights w in L^{2}NAS by randomly sampling α for every input batch of data and discretizing it into an architecture, and only updating the corresponding supernet weights used in the sampled architecture. By doing so, L^{2}NAS avoids the generalization gap of DARTS

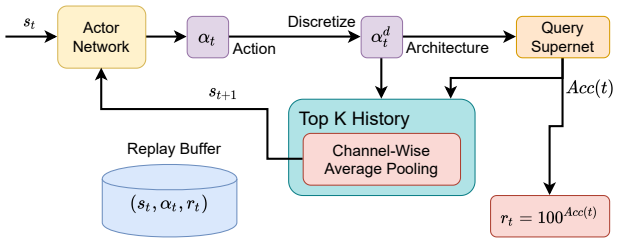


Figure 1: A High-Level Illustration of L^{2}NAS.

caused by evaluating supernet performance during the search while evaluating individual architectures in the test.

As will be demonstrated in experiments, L^{2}NAS can operate on many search spaces other than DARTS that allow continuous relaxation. Furthermore, L^{2}NAS can work regardless of how the performance $S(\alpha)$ is queried, whether from a predictor or a weight-sharing supernet.

2.2 Learning to Optimize Architecture Hyperparameters

We now present the operating mechanisms of the proposed algorithm, L^{2}NAS. Traditionally, in differentiable architecture search, the architecture hyperparameters α are updated via gradient descent with a first-order or second-order approximation of the derivative of validation loss over α. The goal of L^{2}NAS, however, is to learn an update rule μ through performing the following iterations:

$$\begin{aligned} \alpha_t &= \mu(s_t), \\ r_t &\sim p(r|\texttt{Discretize}(\alpha_t)), \\ s_{t+1} &= \phi(\alpha_t, r_t, s_t), \end{aligned}$$

where s_t can be seen as a state recording important statistics in the optimization process up to time step t. At time step t, an action α_t is generated given the state s_t. We then discretize α_t into a neural architecture to query its performance r_t. Finally, the state s_{t+1} is updated with the newly queried α_t and r_t. A high-level overview of our scheme is illustrated in Figure 1.

Specifically, at step t, L^{2}NAS produces an action that serves as continuous architecture hyperparameters, i.e., $\alpha_t \in \mathbb{R}^{|E|\times|O|}$, through a neural network μ, which is a multi-layer perceptron (MLP). Each continuous action α_t is deterministically mapped into a discrete α_t^d. We further elaborate on this `Discretize` process in Section 3, as it is search space dependent. Each α^d corresponds to an individual architecture in the search space. The reward of taking the query α_t is defined as

$$r_t = 100^{Acc(\alpha_t^d)}, \tag{3}$$

where Acc is the measured accuracy of the architecture $\alpha_t^d = \texttt{Discretize}(\alpha_t)$.

Throughout the optimization process, L^{2}NAS keeps track of the top-K architectures in terms of the accuracy seen so far. That is, we store a history tensor, $h_t \in \{0,1\}^{K\times|E|\times|O|}$ of top-K α_t^d seen so far and define the state, $s_t \in \mathbb{R}^{|E|\times|O|}$, as the averaging of h_t over the first dimension. The state s_t is meant to provide statistical information regarding the search space.

Given the above definition, each entry of the state matrix s_t represents the sample probability that a specific operation is present on an edge in the top-K architectures seen so far. A higher value on an entry indicates that the corresponding operation-edge pair is favoured by top performing architectures.

2.3 Policy Training Procedure

L^{2}NAS adopts an *Actor-Critic* framework in reinforcement learning to train the update policy μ. L^{2}NAS produces continuous actions, which makes its training procedure similar to the Deep Deterministic Policy Gradient (DDPG) [21] framework. In continuous-action RL, the agent interacts with an environment over an infinite number of steps, $t = 0, 1, ...,$, by performing an action a_t at each step, receiving a reward r_t in return, followed by transitioning to a state s_{t+1}. Nevertheless, to solve our black-box optimization problem, specifically neural architecture search, there are several key differences in the policy training of L^{2}NAS from DDPG, including a redefined critic network, a quantile-driven loss in critic training, and the use of epsilon-greedy strategy to reduce the number of queries (architecture evaluations).

In particular, we maintain two neural networks in L^{2}NAS: The actor, $\mu(s_t)$, which generates an action α_t given s_t; and the critic, $Q(\alpha_t)$, which predicts the action value. Note that as opposed to DDPG, we remove the dependency of the critic on s_t. We also maintain a replay buffer, R, which stores triplets in the form of (s_t, α_t, r_t).

The actor network is given by,

$$\alpha_t = \mu(s_t) + z_t, \tag{4}$$

where $z_t = \texttt{Uniform}(-\xi, \xi)$ is a small noise following a uniform distribution added to the actor output to encourage exploration. Furthermore, we introduce additional exploration strategies in the form of taking random actions drawn from a uniform distribution, $\texttt{Uniform}(0,1)^{|E|\times|O|}$, instead of being determined by the actor network in Equation 4. We apply two different exploration strategies depending on the search space:

- **ϵ-greedy**: At every step, the actor will take a random action with probability ϵ. We initialize ϵ to a high value and anneal it to a minimum value over time.
- **Random warm-up**: The agent takes random actions in the first W steps. Actions taken during all remaining steps $t > W$ are determined by Equation 4.

The original DDPG critic $Q(s_t, a_t)$ calculates a discounted estimate of future rewards [21] based on state transitions. In contrast, L^{2}NAS uses a critic $Q(\alpha_t)$ to directly approximate the reward r_t. It differs from DDPG in that the critic network does not take the state s_t as an input. The critic network of L^{2}NAS is also an MLP with 3 hidden layers.

Furthermore, considering the stochastic nature of r_t, if the critic were trained by an L_2 loss as in DDPG, it would predict the conditional mean reward,

$$r_t = \mathbb{E}[r|\alpha_t].$$

In contrast, we use a *check loss* to train the critic in L^{2}NAS. In quantile regression [18], the τ-th quantile of a random variable Z, denoted by $Q_\tau(Z)$, is defined as the value such that Z is no more than $Q_\tau(Z)$ with probability τ and no less than $Q_\tau(Z)$ with

probability $1-\tau$. When $\tau = 0.5$, $Q_{0.5}(Z)$ is the median of Z. It is shown [18] that

$$Q_\tau(Z) = \arg\min_c \mathbb{E}_Z[\rho_\tau(Z-c)],$$

where ρ_τ is the *check loss* function defined as

$$\rho_\tau(x) = x(\tau - \mathbf{1}(x \le 0)),$$

where $\mathbf{1}(x \le 0)$ is one if $x \le 0$ and zero otherwise, and $\tau \in [0,1]$ corresponds to the desired quantile level.

L^{2}NAS is trained off-policy, which means that the experience (s_t, α_t, r_t) of a step t is not immediately used to update the critic and actor networks. Instead, all experiences are stored in a replay buffer R and randomly sampled in batches with replacement to train the actor and critic. At every time step t, we randomly sample a batch B_R from the experience replay buffer and use it to update the critic network using the *check loss*:

$$\mathcal{L}_{\text{Critic}} = \frac{1}{|B_R|} \sum_{i \in B_R} \rho_\tau(r_i - Q(\alpha_i)) \tag{5}$$

The actor network is then updated directly based on critic outputs with the following loss:

$$\mathcal{L}_{\text{Actor}} = \frac{1}{|B_R|} \sum_{i \in B_R} Q(\mu(s_i)). \tag{6}$$

A critic trained by the check loss as in Equation 5 learns to predict the τth conditional quantile of the reward

$$r_t = Q_\tau(r|\alpha_t) = \arg\min_c \mathbb{E}_r[\rho_\tau(r-c)|\alpha_t].$$

The key advantage is that the proposed critic is capable of picking up and dealing with the best architectures while the traditional L_2 loss is only able to handle average architectures. Put succinctly, the check loss function ensures that the critic predicts the tail of the reward, which corresponds to the accuracy of the best architectures selected. This knowledge is then used in actor update in Equation 6.

Finally, we can accelerate policy training by updating the actor and critic networks with C batches of samples pulled from the replay buffer per step, instead of one batch per step as in DDPG. This is useful in situations where the number of steps allowed is tightly budgeted. We determine the number of batches C used per step by $C = \min(\left\lfloor \frac{|R|}{|B_R|} \right\rfloor, C_{\max})$, where $|R|$ is the total number of samples in the replay buffer and $C_{\max}$ is an upperbound on C. The actor and critic networks start training once the experience replay buffer has accumulated $|B_R|$ samples.

2.4 Transferability

It is possible to train an agent on one dataset, e.g. CIFAR-10, and then transfer the pretrained agent to another dataset, e.g., CIFAR-100 or ImageNet, where both the critic and actor networks can be fine-tuned with a low number of steps in order to search for high-performance architectures based on the new dataset. Transferability of search policy can be achieved when the action space is held constant.

For this purpose, we introduce another reward function such that the agent can generalize across different datasets and accuracy ranges. For example, the highest validation accuracies achieved by NAS-Bench-201 [12] on CIFAR-10 and CIFAR-100 are 94.37% and 73.51%, respectively, which fall in different ranges. If a search algorithm is trained on CIFAR-10 and then transferred to CIFAR-100, it will need to be rescaled such that it learns a dataset-agnostic understanding on what constitutes a high-performance architecture. In L^{2}NAS, we accomplish this by rescaling the reward function to

$$r_t = \frac{100^{Acc(\alpha_t^d)/Acc(Env)}}{100} - 1, \tag{7}$$

where $Acc(Env)$ is an environment-dependent accuracy measure used to scale the accuracy of architectures generated by the agent to fit to the target dataset.

Algorithm 1 `Discretize()` in NAS-Bench-201/OFA

Input: $\alpha \in \mathbb{R}^{|E| \times |O|}$
Output: $\alpha^d \in \{0,1\}^{|E| \times |O|}$
$\alpha^d = 0^{|E| \times |O|}$
for $k = 0, 1, .., |E| - 1$ **do**
  $A = \alpha[k, :]$
  $i_k = \arg\max_i A_i$
  $\alpha^d[k, i_k] = 1$
end for
Return α^d

3 EXPERIMENTAL SETUP

In this section we enumerate our three candidate search spaces, NAS-Bench-201, DARTS and Once-for-All in detail. Specifically, we describe the internal layout, explain how discretization is performed and elaborate on our training details.

3.1 NAS-Bench-201

The NAS-Bench-201 (NB) search space is based on NASNet [46]. Many cells are stacked repeatedly to form a neural network. There are two types of cells: *normal cells*, which do not modify the dimensions of input tensors and *reduction cells*, which are responsible for halving the height and width of input tensors while doubling the number of channels. The internal structure of reduction cells is fixed, and there are two in total, residing 1/3 and 2/3 through the network, respectively. All other cells are normal and the structure of normal cells is variable, facilitating architecture search.

The internal structure of the normal cells is represented by a DAG with an input node, an output node and $|N|_{\text{NB}} = 3$ intermediate nodes. The input node of cell k receives data from the output node of cell $k-1$. Each intermediate node is connected to the output node by an edge. Additionally, $|E|_{\text{NB}} = 6$ edges connect the input and intermediate nodes. These edges are responsible for performing operations within the cell. The pre-defined operator set, $|O|_{\text{NB}} = 5$, consists of the following: None (Zero), Skip Connection, Average Pooling 3×3, `Nor_Conv`[1] 1×1 and `Nor_Conv` 3×3.

This layout gives rise to $\alpha_{\text{NB}} \in \mathbb{R}^{6\times 5}$ for a total of 15,625 architectures. Algorithm 1 describes the discretization process for NAS-Bench-201. Essentially, the process consists of an `argmax` operation performed on each row of α_{NB} to select the operators.

We do not need to train networks from scratch to evaluate L^{2}NAS on NAS-Bench-201 as the accuracy of each architecture is provided in the form of an API and lookup table.

[1] `Nor_Conv` refers to a sequence consisting of *ReLU-Conv-BN* (batch normalization) [12].

Algorithm 2 `Discretize()` in DARTS/PC-DARTS

Input: $\alpha \in \mathbb{R}^{|E|\times|O|}$
Output: $\alpha^d \in \{0,1\}^{|E|\times|O|}$
Start = 0, $n = 1$
$\alpha^d = 0$
for $k = 0, 1, .., |N| - 1$ **do**
  End = Start + n
  $A = \alpha$[Start : End, :]
  $(i_1, j_1) = \arg\max_{(i,j)} A_{ij}$
  $(i_2, j_2) = \arg\max_{(i,j):i\neq i_1} A_{ij}$
  α^d[Start + i_1, j_1] = 1
  α^d[Start + i_2, j_2] = 1
  Start = End + 1
  $n = n + 1$
end for
Return α^d

3.2 DARTS

The DARTS (DA) topology is also based on NASNet [46], but more complex than NAS-Bench-201 and contains approximately 10^{18} [29] architectures. Rather than simply receiving input from the previous cell, each cell k receives input from the previous two cells, $k-1$ and $k-2$, respectively. There are $|N|_{\text{DA}} = 4$ intermediate nodes forming $|E|_{\text{DA}} = 14$ edges as each intermediate node can receive data from all previous intermediate nodes and both input nodes. There are $|O|_{\text{DA}} = 7$ operations consisting of the following: Maximum Pooling 3×3, Average Pooling 3×3, Skip Connection, Separable Convolution 3×3, Separable Convolution 5×5, Dilation Convolution 3×3 and Dilation Convolution 5×5. Note that compared to the original DARTS, we omit the 'None' operation. Although DARTS allows it during search, unlike NAS-Bench-201, it cannot be selected to form a discrete architecture once search is complete.

Unlike NAS-Bench-201, the reduction cell structure is not fixed and is searchable independently of the normal cell architecture. These properties give rise to architectures represented by two matrices, $\alpha^{\text{N}}_{\text{DA}} \in \mathbb{R}^{|E|\times|O|} = \mathbb{R}^{14\times7}$ and $\alpha^{\text{R}}_{\text{DA}} \in \mathbb{R}^{|E|\times|O|} = \mathbb{R}^{14\times7}$, corresponding to the normal and reduction cells, respectively.

Discretization for DARTS is more complex than it is for NAS-Bench-201. Similar to [24], each intermediate node in N will receive input from only 2 of the directed edges that feed into it. Each of these selected edges shall perform a single operation. The choice of which edges are chosen, and which operators will occupy these edges are determined using the magnitude of their architecture distribution parameters, where higher is better. This means that while the cell may contain up to $|E| = \sum_{i=1}^{|N|}(i+1)$ edges during search, only $2|N|$ edges are allowed after discretization. Algorithm 2 describes the process of discretizing either α_{DA} matrix.

We train a weight-sharing supernet using the PC-DARTS framework, which uses the same topology and operation set as DARTS, but incorporates several memory-saving features [38]. We refer the reader to PC-DARTS for more details, but note that our use of Algorithm 2 allows us to omit the use of 'β'.

Our supernet training strategy is inspired by random search [20] and the single-path uniform sampling scheme used by [13]. DARTS supernets are only used for search; evaluation is performed by training found architectures from scratch. Moreover, this sampling method saves GPU memory cost [3, 11].

For each batch of training data, we first sample a random matrix $\alpha \in \mathbb{R}^{|E|\times|O|}$, which is then discretized. The corresponding weights of the discrete architecture are then updated with the batch of data. This process is equivalent to selecting random individual architectures from the supernet to update per each batch of data.

We train a supernet for CIFAR-10. CIFAR-10 consists of 50k training samples and 10k test samples split across 10 classes. We split the original training set in half into equally sized training and validation partitions. The new training set is used to train the supernet. The validation set is used to query the supernet during model search. The supernet consists of 8 cells (6 normal, 2 reduction) with an initial channel multiplier of 16. The initial learning rate is set to 0.025 and is annealed down to $1e^{-3}$ by cosine schedule. Stochastic gradient descent with a momentum factor of 0.9 optimizes the weights. We use Cutout [10] using the recommended length for CIFAR-10 and train for 10k epochs with a batch size of 250. On a single RTX 2080 Ti, supernet training takes around 3 GPU days.

At the end of each epoch, we gauge the validation set accuracy of the supernet. We record the highest validation accuracy observed during supernet training. This allows the supernet to be used in conjunction with Equation 7.

To facilitate transferability, we train additional supernets on CIFAR-100 and ImageNet-32-120[2] [8] as it has been shown that ImageNet subsets can be used as efficient proxies [35]. CIFAR-100 has the same number of training and test samples as CIFAR-10, only split across 100 classes instead of 10. Likewise, the supernet training scheme for CIFAR-100 follows that of CIFAR-10. ImageNet-32-120 consists of 155k training and 6k test samples split across 120 classes, respectively. We further split the training set into separate training and validation sets like the CIFAR datasets. The supernet trains for 5k steps using a batch size of 750 with the same optimizer and learning rate scheduler as the CIFAR datasets.

After any search on DARTS is complete, we select the best architecture according to validation accuracy, and evaluate it by training it from scratch and obtaining its accuracy on the test set. For CIFAR datasets, DARTS evaluation is performed using models with 20 cells, 18 of which are normal while the remaining 2 are reduction. We set the initial channel size to 36. We utilize Cutout [10] with the recommended lengths and an auxiliary head with a weight of 0.4. The initial learning rate is 0.025, which is annealed down to 0 following a cosine schedule [23] over 1,000 epochs. Batch size is 96 and drop path probability is 0.2.

ImageNet evaluation on DARTS is performed using the same hyperparameters as PC-DARTS [38]. The network consists of 14 cells (12 normal, 2 reduction) and is trained for 250 epochs. An initial learning rate of 0.5 is used and is annealed down to zero after an initial 5 epochs of warmup.

3.3 Once-for-All

OFA, as originally proposed, uses MobileNetV3 (MBv3) [15] as a backbone structure and provides pre-trained supernets with elastic depth and width, allowing architectures to be searched for in the MBv3 design space and evaluated on ImageNet [9].

[2]First 120 classes of ImageNet downsampled to 32x32 images.

Table 1: Accuracies obtained on NAS-Bench-201 datasets compared to other methods. The horizontal line demarcates weight-sharing algorithms from those that directly query oracle information. We run L^{2}NAS for 500 and 1,000 steps per experiment across 10 different random seeds, and report the mean and standard deviation.

	CIFAR-10		CIFAR-100		ImageNet-16-120	
Method	**Valid [%]**	**Test [%]**	**Valid [%]**	**Test [%]**	**Valid [%]**	**Test [%]**
DARTS [22]	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
ENAS [26]	37.51 ± 3.19	53.89 ± 0.58	13.37 ± 2.35	13.96 ± 2.33	15.06 ± 1.95	14.84 ± 2.10
GDAS [11]	89.89 ± 0.08	93.61 ± 0.09	71.34 ± 0.04	70.70 ± 0.30	41.59 ± 1.33	41.71 ± 0.98
GAEA [19]	–	94.10 ± 0.29	–	**73.43 ± 0.13**	–	46.36 ± 0.00
RS [12]	90.93 ± 0.36	93.70 ± 0.36	70.93 ± 1.09	71.04 ± 1.07	44.45 ± 1.10	44.57 ± 1.25
REA [12]	91.19 ± 0.31	93.92 ± 0.30	71.81 ± 1.12	71.84 ± 0.99	45.15 ± 0.89	45.54 ± 1.03
REINFORCE [12]	91.09 ± 0.37	93.85 ± 0.37	71.61 ± 1.12	71.71 ± 1.09	45.05 ± 1.02	45.24 ± 1.18
BOHB [12]	90.82 ± 0.53	93.61 ± 0.52	70.74 ± 1.29	70.85 ± 1.28	44.26 ± 1.36	44.42 ± 1.49
arch2vec-RL [39]	91.32 ± 0.42	94.12 ± 0.42	73.12 ± 0.72	73.15 ± 0.78	46.22 ± 0.30	46.16 ± 0.38
arch2vec-BO	91.41 ± 0.22	94.18 ± 0.24	**73.35 ± 0.32**	73.37 ± 0.30	46.34 ± 0.18	46.27 ± 0.37
L^{2}NAS-500	91.36 ± 0.19	94.11 ± 0.16	72.47 ± 0.74	72.69 ± 0.58	46.23 ± 0.28	46.74 ± 0.39
L^{2}NAS-1k	**91.47 ± 0.15**	**94.28 ± 0.08**	73.02 ± 0.52	73.09 ± 0.35	**46.58 ± 0.08**	**47.03 ± 0.27**
True Optimal	*91.61*	*94.37*	*73.49*	*73.51*	*46.77*	*47.31*

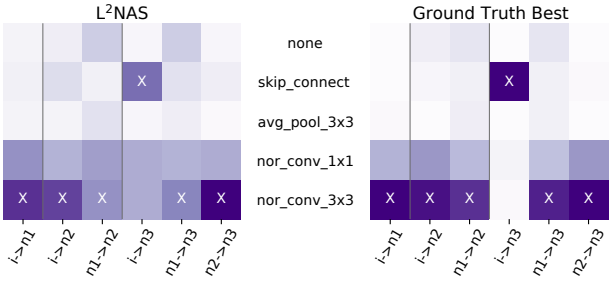


Figure 2: Comparison of the final state representation at the end of an L^{2}NAS search on NAS-Bench-201 CIFAR-100 test accuracy (left) with the average of α_d of the true top-K architectures (right). $K = 64$. Rows indicate operations, columns indicate edges. 'i' means input, 'n' denotes an intermediate node. Darker color indicates higher values. Vertical bars demarcate destination nodes. 'X' marks the operation-edge pairs corresponding to the best architecture.

The search space contains approximately 10^{19} architectures. The supernet contains 10–20 layers, divided into 5 units, each of which contains 2–4 layers to give $|E|_{\text{OFA}} = 20$. Each layer contains a block operation. The operation space consists of *MBConv* [15] blocks of the form `MBConv-e-k` where 'e' refers to a channel expansion factor in $\{3, 4, 6\}$, and 'k' refers to a square kernel size in $\{3, 5, 7\}$, giving rise to $|O|_{\text{OFA}} = 9$ different blocks.

Therefore, for OFA, α_{OFA} consists of two matrices: $\alpha_{\text{OFA},O} \in \mathbb{R}^{20\times9}$, which controls the operations selected for each layer and $\alpha_{\text{OFA},D} \in \mathbb{R}^{5\times3}$, which controls the depth of the network. Discretization is performed using Algorithm 1 separately on each matrix.

The publicly available OFA code repository[3] provides pre-trained supernets and an API for evaluating individual architectures on the ImageNet validation set, which contains 50k 224×224 color images spread across 1000 classes. We make use of these resources. In particular, we use both versions of the MBv3 supernets, which we denote as *OFA*[4] and OFA_{Large}[5].

The only difference between these two supernets is the base channel width multiplier, which is not searchable. By using both of these supernets we are able to test and compare L^{2}NAS across more than 1 version of the search space.

3.4 DDPG Agent

Our DDPG agent consists of 2 neural networks: an actor and a critic. Both are multi-layer perceptrons with 3 hidden layers. The actor produces vectorized actions a_t which are then split if α consists of more than one matrix (e.g., DARTS and OFA) and then re-shaped into matrices. The number of neurons in each hidden layer is 128 for NAS-Bench-201 and 256 for DARTS and OFA as their respective α matrices are larger. ReLU [25] serves as the activation function in the hidden layers while the final layer of the actor network features a sigmoid activation. The critic uses an identity function as it produces a scalar. Both networks are optimized using Adam [17] with $\vec{\beta} = (0.9, 0.99)$. Learning rates for the actor and critic networks are $1e^{-8}$ and $1e^{-4}$, respectively.

4 RESULTS

In this section, we first evaluate L^{2}NAS on NAS-Bench-201 [12]. We then perform search experiments with L^{2}NAS in the DARTS search space based on CIFAR-10 and compare against other state-of-the-art methods that also operate in the same search space. We also present search results in the OFA search space evaluated on ImageNet. Finally, we show the feasibility of transferring a search policy found by L^{2}NAS on a smaller CIFAR-10 dataset to architecture searches performed on CIFAR-100 and ImageNet with superior efficiency.

[3] https://github.com/mit-han-lab/once-for-all

[4] `ofa_mbv3_d234_e346_k357_w1.0`

[5] `ofa_mbv3_d234_e346_k357_w1.2`

4.1 NAS Benchmark Performance

NAS-Bench-201 architectures are evaluated on three datasets: CIFAR-10, CIFAR-100 and ImageNet-16-120[6] [8]. Therefore, in this context, our goal is to use the accuracy information provided by NAS-Bench-201 to find the best performing architecture within a small number of queries (steps) to the benchmark, since each query for accuracy converts into an architecture evaluation in reality.

We set $K = 64$, $\tau = 0.9$, $|B_R| = 8$, $\xi = 1e^{-4}$ and $C_{max} = 10$. On NAS-Bench-201, L^{2}NAS performs exploration using ϵ-greedy. The initial value of ϵ is 1.0 and it is annealed via a cosine schedule to a minimum of 0.05 by step $t = 175$. We use Equation 3 to calculate NAS-Bench-201 rewards. 1000 steps of L^{2}NAS on NAS-Bench-201 executes in less than 4 minutes when only using a CPU.

Table 1 shows the results in comparison with a range of popular search methods with NAS-Bench-201 results reported. The methods in the first category, i.e., DARTS, ENAS, GDAS, GAEA, are gradient-based and must operate on a weight-sharing supernet, while the other methods like L^{2}NAS can simply perform search by querying the ground-truth oracle performance. We observe that with only 500 steps (queries), L^{2}NAS achieves high performance and with 1000 steps, achieves state-of-the-art performance on CIFAR-10 and ImageNet-16-120. Indeed, it is worth noting that both L^{2}NAS-1k and L^{2}NAS-500 clearly outperform all other methods in the search on test set of ImageNet-16-120. The ability of L^{2}NAS to find top architectures within a small number of queries stems from the effective strategy to balance exploration and exploitation.

Note that *arch2vec*, achieving better results on CIFAR-100 validation set, pre-trains architecture embeddings using unsupervised learning before the search. Therefore, in fact, L^{2}NAS can be complemented by the architecture representation learning in *arch2vec* to further enhance search efficiency. GAEA results are from searches performed on a weight-sharing supernet and is thus not comparable to results in the second category.

Figure 2 shows the final state representation $s_{t=500}$ of a 500-step L^{2}NAS experiment as compared to the ground-truth best architectures in NAS-Bench-201. We observe that the average α^d of the top-K ($K = 64$) architectures found by L^{2}NAS after querying only a fraction of NAS-Bench-201 is very close to that of the true top-K architectures. This implies that L^{2}NAS can effectively and efficiently charter a search space.

4.2 CIFAR-10 Performance on DARTS

Next, we evaluate the performance of L^{2}NAS searching in the DARTS search space based on CIFAR-10 using weight-sharing and compare with other state-of-the-art results. Given that DARTS dwarfs NAS-Bench-201 by many magnitudes [29] and the ground truth accuracy of every architecture is not known, we do not focus on finding the best architecture in the least number of queries. Instead, we adjust our search to perform additional emphasis on exploration so a more thorough sweep of the search space is performed while the critic focuses on learning a narrower accuracy quantile.

We set $K = 500$, $\tau = 0.95$, $|B_R| = 64$, $\xi = 5e^{-5}$ and $C_{max} = 1$. We run the agent for 20k steps and limit the replay buffer to contain only the last 5k experiences. Exploration is achieved using random warm-up with $W = 3000$. Like NAS-Bench-201, we use Equation 3 to calculate the reward. Architecture search takes roughly 1 GPU day on a single RTX 2080 Ti.

[6]First 120 classes of ImageNet downsampled to 16×16 images.

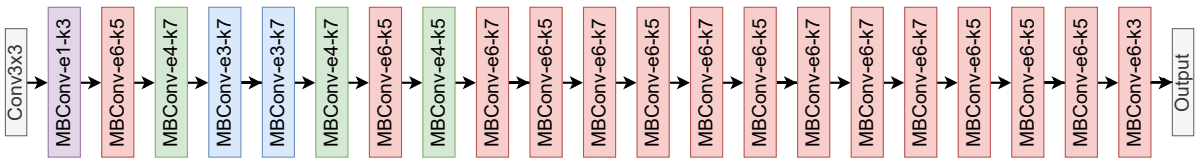
Table 2: CIFAR-10 Results. Methods in the second and third categories search on P-DARTS and DARTS search spaces, respectively.

Architecture	Test Acc. [%]	Params [M]
ENAS [26]	97.11	4.6
GDAS [11]	97.18	2.5
AlphaX [32]	97.46 ± 0.06	2.8
P-DARTS [5]	97.50	3.4
P-SDARTS [4]	97.52 ± 0.02	3.4
DARTS 1st [22]	97.00 ± 0.14	3.3
DARTS 2nd	97.24 ± 0.09	3.3
SNAS [37]	97.15 ± 0.02	2.8
EcoNAS [44]	97.38 ± 0.02	2.9
ISTA-NAS 2S [40]	97.46 ± 0.05	3.3
arch2vec-RL [39]	97.35 ± 0.05	3.3
arch2vec-BO	97.44 ± 0.05	3.6
MdeNAS [43]	97.45	3.6
MiLeNAS [14]	97.49 ± 0.04	3.9
PC-DARTS [38]	97.43 ± 0.07	3.6
PC-SDARTS	97.51 ± 0.04	3.5
PC-GAEA [19]	97.50 ± 0.06	3.7
L^{2}NAS	**97.51 ± 0.12**	**3.8**

Table 2 lists the results as compared to a wide range of other algorithms. For fair comparisons, we specifically compare to NAS methods that also reported performance exactly on the DARTS search space, as shown in the third category of Table 2, while the first category illustrates cell search methods using other search spaces. We measure test accuracies over 5 evaluation runs for the best architecture found by L^{2}NAS. Other results are the top results taken from their respective publications.

DARTS achieved 97.24%, while PC-DARTS increased the accuracy to 97.43% before GAEA (PC-GAEA) further increased it to 97.50% on average with a maximum accuracy of 97.61% among independent runs. The architecture found by L^{2}NAS attained an average accuracy of 97.51%, with a maximum of **97.64%**, which exceeds the performance of other state-of-the-art results achieved by GAEA and S-DARTS. At 3.8M parameters, our architecture is also more efficient than other larger ones like MiLeNAS.

It is worth noting that schemes based on Progressive DARTS (P-DARTS) [5] are listed separately from other methods that exactly operate on DARTS search space. P-DARTS uses multiple supernets and gradually prune the number of operations are while the network depth is increased during search. Additionally, restrictions are placed on the number of times specific operations can be selected. Since we do not employ these features, they are not a fair comparison. Furthermore, it is also worth noting that the 'One-stage' (1S) variant of ISTA-NAS achieved an even higher average accuracy. However, it employs unique and novel modifications to the original operation set of DARTS which boosts the performance. Thus, it is not listed for fair comparison. In contrast, the 'Two-stage' (2S) version of ISTA-NAS operates in the same DARTS search space and is thus listed here. Finally, PC-DARTS enables partial channel



(a) L²NAS OFA Architecture

(b) L²NAS OFA$_{Large}$ Architecture

Figure 3: Architectures found on the OFA (top) and OFA$_{Large}$ (bottom) supernets; initial `MBConv-e1-k3` block is not searchable.

connections to enhance the memory efficiency of DARTS during search. It still uses the DARTS supernet without altering operations, and therefore is a fair comparison.

4.3 ImageNet Performance

We evaluate the ImageNet performance of L²NAS by applying it to the search space of Once-for-All (OFA) [2] on MobileNetV3.

We use the same values for K, τ, $|B_R|$, C_{max} and W as DARTS. However, since ImageNet inference is costlier than CIFAR, we only train the agent for 10k steps with $\xi = 1e^{-4}$ for additional exploration. We use the rescaled reward Equation 7 with $Acc(Env)$ set to 76% for OFA and 79% for OFA$_{Large}$. We use L²NAS to search for the architecture with the best accuracy on OFA supernet and report test accuracy by directly using model weights inherited from the supernet without further fine-tuning the weights. Search takes about 150 GPU hours or under 1 week on a single RTX 2080 Ti.

Table 3: Comparison of L²NAS with other state-of-the-art architectures on ImageNet.

Architecture	Top-1 Acc. [%]	Top-5 Acc. [%]	MACs [M]
PC-SDARTS	75.7	92.6	–
P-SDARTS	75.8	92.8	–
PC-GAEA	76.0	92.7	–
MBv2	74.7	–	585
ProxylessNAS	75.1	92.5	320
MBv3-L 0.75	73.3	–	155
MBv3-L 1.0	75.2	–	219
EfficientNet-B0	77.1	93.9	390
EfficientNet-B1	79.1	94.4	700
EfficientNet-B2	80.1	94.9	1000
Cream-S	77.6	93.3	287
Cream-M	79.2	94.2	481
Cream-L	80.0	94.7	604
OFA	76.0	–	230
OFA$_{Large}$	79.0	94.5	595
L²NAS	**77.4**	**93.4**	**467**
L²NAS$_{Large}$	**79.3**	**94.6**	**618**

Figure 3 illustrates the OFA architectures found. Like NAS-Bench-201, L²NAS learns to select large architectures in order to achieve high accuracy on OFA. Specifically, L²NAS focuses on selecting the largest expansion ratio (6) in the latter half of the network.

ImageNet results are provided in Table 3, in which the last category lists methods that also search in OFA, while the other 4 categories list results of other methods obtained in other related search spaces involving different backbone setups and operation sets. These other related search spaces include the predecessor to MBv3, MobileNetV2 (MBv2) [27], which ProxylessNAS [3] is based on, using different set of operations. Variants of MBv3 include EfficientNet [30], which focuses on striking a balance between network depth and resolution. Cream of the Crop also uses MBv3 as the backbone structure, yet with a modified set of searchable operations introduced to boost performance.

From a comparison against the models OFA and OFA$_{Large}$, which are obtained through an evolutionary algorithm applied on the supernet, we observe that L²NAS outperforms both of these in terms of both top-1 accuracy, while both are directly using weights inherited from the OFA supernet for evaluation. Note that we do not compare to the result of OFA$_{Large}$ after fine-tuning it, which is also reported in [2]. These findings further demonstrate the effectiveness of L²NAS as a search method, excluding the factors of search space and evaluation methodology.

In addition, we are able to achieve a much higher top-1 and top-5 accuracy than any state-of-the-art DARTS architecture. Additionally, we outperform EfficientNet-B0 and B1, while EfficientNet-B2 is a large architecture with more than 1 billion MACs. Cream of the Crop, although a state-of-the-art method in the mobile regime, is not directly comparable to our result, due to the different search space and evaluation method used. It searches in a variant space of MBv3 with different operations than OFA, and trains its supernet for 120 epochs and then retrains specific architectures for 500 epochs.

4.4 Transferability of Search Policies

Finally, we perform transferability tests by training a policy on a simpler dataset like CIFAR-10, and then perform policy fine-tuning on more complex datasets, including CIFAR-100 and ImageNet. This is in contrast to previous NAS methods like DARTS that search for

Table 4: Highest validation accuracies measured during PC-DARTS supernet training. These values become $Acc(Env)$ for Equation 7 for transferability experiments.

Dataset	Max Accuracy [%]	GPU Days
CIFAR-10	82.316	3
CIFAR-100	56.380	3
ImageNet-32-12	48.765	3.5

an architecture on CIFAR-10, then directly transfer the architecture to ImageNet for evaluation. Other methods like PC-DARTS and GAEA perform direct searches on ImageNet, i.e., a separate search for each individual dataset must be done.

We use the rescaled reward in Equation 7 with $\tau = 0.95$ to pre-train a transferable policy on CIFAR-10. $Acc(Env)$ is set to the maximum accuracy observed during supernet training. Table 4 lists the highest accuracy for each supernet. We train the agent for 10k steps. The first $W = 3000$ steps are used for exploration and we set $\xi = 1e^{-4}$ to further increase exploration. The pretrained actor and critic networks are then fine-tuned on CIFAR-100 and ImageNet-32-120 supernets trained according to the procedure in Section 3.2. We set $K = 100$ and fine-tune for a total of 1k steps, using the first $W = 500$ steps for exploration. Initial policy pretraining takes 12 GPU hours. We perform the fine-tuning task 5 times with different random seeds. Each run of fine-tuning takes about 1.5 and 2 GPU hours for CIFAR-100 and ImageNet, respectively.

Table 5 provides evaluation results. For comparison, we also evaluate the original architectures published by DARTS on CIFAR-100. For both CIFAR-100 and ImageNet, transferred policies can find architectures that actually outperform the ones found via direct search, as the search policies instead of architectures are migrated from CIFAR-10 to operate with more complex datasets. Moreover, the cost of search on CIFAR-100 and ImageNet is significantly reduced from a few GPU days to a few GPU hours of fine-tuning if a search policy is pretrained on CIFAR-10.

As the core motivation of NAS is automating the search, we offer a search procedure that can better generalize and transfer across different datasets instead of manually tuning the customized hyperparameters and optimizers to search on each individual dataset.

5 RELATED WORK

DARTS [22] proposes differentiable architecture search and has given rise to many follow-up works including P-DARTS [5] which breaks the search procedure into different stages and [42] which propose early stopping. PC-DARTS [38] uses partial channel connections to decrease the memory cost in backpropagation and reduce bias toward parameterless operations.

However, recent works suggest that gradient-based NAS methods suffer from weaknesses such as the discretization error [36, 41], low reproducibility [41] and bias towards shallow architectures [28]. As a result, many attempts have focused on correcting these flaws. [4] reduces the discretization error by forcing the supernet weights to generalize to a broader range of architecture parameters. [11] and [37] help bridge the optimization gap by using Gumbel-Softmax to sample a single operation per edge. ISTA-NAS [40] recast NAS as a sparse coding problem in order to perform search and evaluation

Table 5: Transferability results for L^{2}NAS for CIFAR-100 and ImageNet. 'Direct' means directly searching on CIFAR-100 and ImageNet using the procedure in Section 4.2.

CIFAR-100	Test Acc. [%]	Params [M]	GPU days
DARTS 1st	82.37	3.3	1.5
DARTS 2nd	82.65	3.3	4.0
L^{2}NAS-Direct	82.24 ± 0.19	3.5	1.0
L^{2}NAS-Transfer	82.97 ± 0.29	4.0	0.1
ImageNet	**Top-1/Top-5 [%]**	**Params [M]**	**Search Cost**
DARTS 2nd	73.1/91.3	4.7	4.0
L^{2}NAS-Direct	74.8/92.2	4.9	1.0
L^{2}NAS-Transfer	75.4/92.5	5.4	0.1

in the same setting. To improve generalizability, [14] reformulate DARTS as a mixed-level optimization problem. GAEA [19] uses a simplex projection to ensure better convergence.

In comparison, we learn an agent in an actor-critic framework, which can be generalized and transferred across different datasets as a search policy. The RL-based training setup also allows for improved reproducibility and a natural incorporation of various exploration strategies. Our supernet training uses random sampling inspired by [20] to reduce the discretization error and bias on architecture depth. Our experiments on transferability are partially inspired by [31], who use an RL agent to search for scalable analog circuit designs.

Finally, a number of RL-based NAS methods have been proposed. [45] use a controller trained by REINFORCE [33] to select architecture hyperparameters. ENAS [26] is the first reinforcement learning scheme in weight-sharing NAS. [1] show that guided policies exceed the performance of random search on vast search spaces. [32] uses Monte Carlo Tree Search to balance exploration with exploitation. These methods operate in a discrete space where the RL state keeps track of the partial architecture that is built over many steps. By contrast, L^{2}NAS selects an entire architecture per step and uses the RL state to keep track of the distribution of top performing architectures.

6 CONCLUSION

In this paper, we propose L^{2}NAS to learn an optimizer for architecture hyperparameters in neural architecture search. Through an actor network taking feedback from the search history, L^{2}NAS produces continuous architecture hyperparameters α, that are mapped into discrete architectures to obtain rewards. We propose a learning procedure for L^{2}NAS based on the continuous RL framework and a quantile-driven loss. L^{2}NAS learns to explore a large search space efficiently and achieves fast convergence to high-performing architectures. Experiments show that L^{2}NAS achieves state-of-the-art results on NAS-Bench-201 after querying only 1000 architectures. When working in the DARTS or OFA search space, L^{2}NAS produces architectures that achieve state-of-the-art test accuracies on CIFAR-10 and ImageNet compared to a wide range of algorithms that exactly operate on the same search spaces. With transferability tests, we also demonstrate that an L^{2}NAS search policy pre-trained on CIFAR-10, can be used for effective search on CIFAR-100 and ImageNet with a low fine-tuning cost.

REFERENCES

[1] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. 2020. Can Weight Sharing Outperform Random Architecture Search? An Investigation With TuNAS. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14323–14332.

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HylxE1HKwS

[3] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *International Conference on Learning Representations*. https://arxiv.org/pdf/1812.00332.pdf

[4] Xiangning Chen and Cho-Jui Hsieh. 2020. Stabilizing Differentiable Architecture Search via Perturbation-based Regularization. *arXiv preprint arXiv:2002.05283* (2020).

[5] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. 2019. Progressive Differentiable Architecture Search: Bridging the Depth Gap between Search and Evaluation. arXiv:1904.12760 [cs.CV]

[6] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. 2017. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*. PMLR, 748–756.

[7] Yutian Chen, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Timothy P Lillicrap, and Nando de Freitas. 2016. Learning to learn for global optimization of black box functions. *arXiv preprint arXiv:1611.03824* (2016).

[8] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2017. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819* (2017).

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

[10] Terrance DeVries and Graham W Taylor. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. *arXiv preprint arXiv:1708.04552* (2017).

[11] Xuanyi Dong and Yi Yang. 2019. Searching for a Robust Neural Architecture in Four GPU Hours. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[12] Xuanyi Dong and Yi Yang. 2020. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*.

[13] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2019. Single Path One-Shot Neural Architecture Search with Uniform Sampling. *CoRR* abs/1904.00420 (2019). http://arxiv.org/abs/1904.00420

[14] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. 2020. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11993–12002.

[15] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1314–1324.

[16] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. 2018. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in neural information processing systems*. 2016–2025.

[17] Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (12 2014).

[18] Roger Koenker. 2005. *Quantile Regression*. Number no. 38 in Econometric Society Monographs. Cambridge University Press.

[19] Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. 2020. Geometry-Aware Gradient Algorithms for Neural Architecture Search. *arXiv preprint arXiv:2004.07802* (2020).

[20] Liam Li and Ameet Talwalkar. 2019. Random Search and Reproducibility for Neural Architecture Search. *CoRR* abs/1902.07638 (2019). http://arxiv.org/abs/1902.07638

[21] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. http://arxiv.org/abs/1509.02971

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. http://arxiv.org/abs/1806.09055

[23] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR*.

[24] Keith G. Mills, Mohammad Salameh, Di Niu, Fred X. Han, Seyed Saeed Changiz Rezaei, Hengshuai Yao, Wei Lu, Shuo Lian, and Shangling Jui. 2021. Exploring Neural Architecture Search Space via Deep Deterministic Sampling. *IEEE Access* 9 (2021), 110962–110974. https://doi.org/10.1109/ACCESS.2021.3101975

[25] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Haifa, Israel) *(ICML'10)*. Omnipress, Madison, WI, USA, 807–814.

[26] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. *CoRR* abs/1802.03268 (2018). http://arxiv.org/abs/1802.03268

[27] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[28] Yao Shu, Wei Wang, and Shaofeng Cai. 2019. Understanding Architectures Learnt by Cell-based Neural Architecture Search. *arXiv preprint arXiv:1909.09569* (2019).

[29] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. 2020. NAS-Bench-301 and the Case for Surrogate Benchmarks for Neural Architecture Search. *arXiv preprint arXiv:2008.09777* (2020).

[30] Mingxing Tan and Quoc V Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946* (2019).

[31] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. 2020. GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[32] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. 2018. Neural Architecture Search using Deep Neural Networks and Monte Carlo Tree Search. *arXiv preprint arXiv:1805.07440* (2018).

[33] R. J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8 (1992), 229–256.

[34] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. 2017. The marginal value of adaptive gradient methods in machine learning. In *Advances in neural information processing systems*. 4148–4158.

[35] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 10734–10742.

[36] Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Zhengsu Chen, Lanfei Wang, An Xiao, Jianlong Chang, Xiaopeng Zhang, et al. 2020. Weight-Sharing Neural Architecture Search: A Battle to Shrink the Optimization Gap. *arXiv preprint arXiv:2008.01475* (2020).

[37] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. SNAS: Stochastic Neural Architecture Search. *CoRR* abs/1812.09926 (2018). http://arxiv.org/abs/1812.09926

[38] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. 2020. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJlS634tPr

[39] Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. 2020. Does unsupervised architecture representation learning help neural architecture search? *Advances in Neural Information Processing Systems* 33 (2020).

[40] Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. 2020. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *Advances in Neural Information Processing Systems* 33 (2020).

[41] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. 2019. Evaluating the Search Phase of Neural Architecture Search. *arXiv preprint arXiv:1902.08142* (2019).

[42] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. 2019. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656* (2019).

[43] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. 2019. Multinomial distribution learning for effective neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1304–1313.

[44] Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. 2020. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11396–11404.

[45] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. https://arxiv.org/abs/1611.01578

[46] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.