Neural Architecture Search in A Proxy Validation Loss Landscape

Yanxi Li [1] **Minjing Dong** [1] **Yunhe Wang** [2] **Chang Xu** [1]

Abstract

This paper searches for the optimal neural architecture by minimizing a proxy of validation loss. Existing neural architecture search (NAS) methods used to discover the optimal neural architecture that best fits the validation examples given the up-to-date network weights. However, back propagation with a number of validation examples could be time consuming, especially when it needs to be repeated many times in NAS. Though these intermediate validation results are invaluable, they would be wasted if we cannot use them to predict the future from the past. In this paper, we propose to approximate the validation loss landscape by learning a mapping from neural architectures to their corresponding validate losses. The optimal neural architecture thus can be easily identified as the minimum of this proxy validation loss landscape. A novel sampling strategy is further developed for an efficient approximation of the loss landscape. Theoretical analysis indicates that the validation loss estimator learnt with our sampling strategy can reach a lower error rate and a lower label complexity compared with a uniform sampling. Experimental results on benchmarks demonstrate that the architecture searched by the proposed algorithm can achieve a satisfactory accuracy with less time cost.

1. Introduction

Designing neural network architecture by neural architecture search (NAS) methods without human participation has attracted the interest of researchers since decades ago (Fahlman & Lebiere, 1990; Kitano, 1990). Recently, the explosive growth in hardware performance has boosted many excellent studies in NAS (Cai et al., 2017; Liu et al., 2017; Real et al., 2017; Zoph & Le, 2016; Zoph et al., 2018), which show a promising future of this field. Although hardware performance has improved, NAS is still computational prohibitive and expensive for relatively small institutions, organizations and companies due to its nature. Typically, NAS involves solving a bi-level optimization problem with model training as the lower optimization problem and architecture searching as the upper one. Previous study has shown that such problems are computationally difficult to be optimized, due to the fact that the solutions of optimizations in both levels dynamically affect each other (Dempe, 2018).

To solve the bi-level optimization problem, extensive researches have been conducted for searching neural architectures by reinforcement learning (RL) (Baker et al., 2016; Bello et al., 2017; Cai et al., 2017; Zoph & Le, 2016; Zoph et al., 2018) or evolutionary algorithms (EA) (Liu et al., 2017; Real et al., 2017; Miikkulainen et al., 2019). These methods often need to evaluate numerous different architectures and therefore require thousands of GPU days to find an architecture on CIFAR-10. Recent works represented by DARTS (Liu et al., 2018b) have established a new family of approaches using gradient-based method to optimize neural architectures, which reduce the searching cost dramatically from thousands of GPU days to several GPU hours. However, in NAS iterations, knowledge from prior steps are far from fully explored and exploited to instruct the subsequent searching by existing methods. Intermediate validation results are often discarded without a thorough investigation, which causes a huge waste of computation resources.

In this paper, we propose to learn the proxy validation loss landscape for an efficient neural architecture search. Instead of discarding the historical validation losses in iterations of NAS, we reuse them to train a validation loss estimator. Based on the gradients of this validation loss estimator, we can then derive neural architectures that correspond to the smaller validation loss. The validation loss estimator can be further upgraded with these discovered architectures and their corresponding actual validation losses. We conduct theoretical analysis on the convergence of the validation loss estimator and suggest that it needs fewer labelling costs. Experimental results on benchmark datasets demonstrate that searching in the proposed proxy validation loss landscape, we can outperform comparison methods by achieving up to 2.70% test error rate on CIFAR-10 and 25.6% top-1 error rate on ImageNet 2012 within 0.20 GPU days.

[1]School of Computer Science, University of Sydney [2]Noah's Ark Lab, Huawei Technoligies. Correspondence to: Yanxi Li <yali0722@uni.sydney.edu.au>, Chang Xu <c.xu@sydney.edu.au>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

2. Related Works

Recently, the automatic design of neural architectures without human experts involvement, which is generally referred as *Neural Architecture Search* (NAS), has been significantly developed (Baker et al., 2016; Cai et al., 2017; Zoph & Le, 2016; Zoph et al., 2018; Bello et al., 2017; Liu et al., 2017; Real et al., 2017; Baker et al., 2017; Deng et al., 2017). NAS methods are commonly classified from three perspectives: search space, search strategy, and performance estimation strategy (Elsken et al., 2018).

The majority of earlier NAS approaches typically choose the *macro search space* (Brock et al., 2017; Cai et al., 2018; Veniat & Denoyer, 2018; Zoph & Le, 2016), where the entire network is discovered by algorithms. As its name indicated, macro search space only allows coarse-grained optimizations, in which architectures are parameterized by: the number of convolutional layers, the operations applied by each layer, the hyper-parameters of each operation (e.g. size of a filter, number of parallel filters or strides), and the number of units in fully connected layers (Elsken et al., 2018). To make the optimization more fine-grained, inspired by repeated motifs (a.k.a. cells or blocks) in hand-crafted architectures (He et al., 2016; Szegedy et al., 2015), later works usually apply the *micro search space* (Pham et al., 2018; Real et al., 2019; 2017; Zoph et al., 2018). The scale of motifs is relatively small, which allows efficient searching for more specific and complex architectures, while powerful large scale networks can still be built by stacking multiple motifs together.

While search spaces define what architectures to search, search strategies determine how to search in such spaces. In earlier NAS approaches, architectures are discretely sampled and evaluated (Baker et al., 2016; Bello et al., 2017; Cai et al., 2017; Zoph & Le, 2016; Zoph et al., 2018; Liu et al., 2017; Real et al., 2017; Miikkulainen et al., 2019). For each sampled architecture, a new network is constructed, trained and evaluated independently, which increases the inference cost dramatically. Recent differentiable search approaches represented by DARTS (Liu et al., 2018b) propose to consider target architectures as sub-graphs of a large computation graph and use continuous architecture weights to parameterize them, which allows joint optimization of model weights and architecture weights using gradient descent. GDAS (Dong & Yang, 2019) further proposes to sample one operation each step rather than use the weighted sum of all operations, which significantly reduces inference cost and improves search speed. A similar work aiming to boost search speed is PC-DARTS (Xu et al., 2019), which performs operation search in a subset of channels instead of operations. Besides search speed, Amended-DARTS (Bi et al., 2019) uses mathematical methods to stabilize the search progress of DARTS.

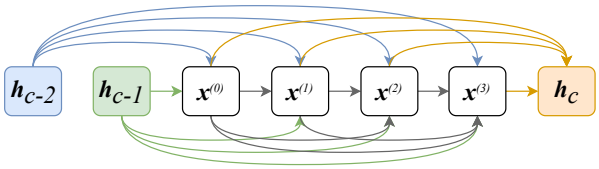

Figure 1. A cell in the form of computation graph. The blue and green nodes ($\boldsymbol{h}_{c-2}, \boldsymbol{h}_{c-1}$) are outputs of previous cells. The orange node ($\boldsymbol{h}_c$) is the output of current cell. The white nodes ($\boldsymbol{x}^{(i)}, i = 0, 1, 2, 3$) are latent representations (i.e. feature maps) in the current cell.

Training a network for several epochs and then evaluating it is the most straightforward method to estimate the performance of a network. However, if we can predict the performance of a network directly from its architecture, things will be much easier. Predicting neural network performance is an emerging topic, which is initially proposed to speed up hyper-parametric search (Baker et al., 2017; Domhan et al., 2015; Klein et al., 2016; Swersky et al., 2014). In Peephole (Deng et al., 2017), individual layers of a network are encoded into vectors and compressed as a single integrated description via LSTM. Performance of the network is then predicted given the integrated description of architecture. Predicting performance of networks with repeated motifs is much simpler, Progressive NAS (Liu et al., 2018a) proposes to use a surrogate model to predict architecture performance.

3. Methodology

In this section, we introduce the search space and the differentiable architecture sampling, and then propose a new NAS algorithm based on proxy validation loss landscape.

3.1. Search Space

We adopt the recently popular micro search space (Zoph et al., 2018). A cell c is represented by a DAG in Fig. 1, whose nodes are latent representations and edges are operations. We limit each cell to contain 7 nodes, i.e. $nodes = \{\boldsymbol{I}^{(i)} | 0 \leq i \leq 6\}$. The first two nodes $\boldsymbol{I}^{(0)}$ and $\boldsymbol{I}^{(1)}$ are outputs from previous two cells $\boldsymbol{h}_{c-2}$ and $\boldsymbol{h}_{c-1}$. From node $\boldsymbol{I}^{(2)}$ to node $\boldsymbol{I}^{(5)}$ are 4 computation nodes calculated by

$$\boldsymbol{I}^{(j)} = \sum_{i<j} o_{i,j}(\boldsymbol{I}^{(i)}), \tag{1}$$

where $o_{i,j}(\cdot)$ represents the operation on the edge from $\boldsymbol{I}^{(i)}$ to $\boldsymbol{I}^{(j)}$. The last node $\boldsymbol{I}^{(6)} = \bigcup_{i=2}^{5} \boldsymbol{I}^{(i)}$ is the output of current cell $\boldsymbol{h}_c$, which is the channel-wise concatenation of the four computation node. The difference between the normal cell and the reduction cell is that operations in the former has a stride of one while operations in the latter has a stride of two to half the height and width of latent representations.

3.2. Differentiable Architecture Sampling

The aforementioned operation $o_{i,j}$ is chosen from a set of K operation candidates $\mathcal{O}$ according to specific rules. The most straightforward sampling method is to use the argmax of a K-dimensional architecture weight $\boldsymbol{a}_{i,j} \in \mathbb{R}^K$ to select the operation. An architecture can be represented by a one-hot vector and Eq. 1 can be rewritten with it as:

$$\begin{aligned} \boldsymbol{I}^{(j)} &= \sum_{i<j}^{K} \sum_{k=1} \boldsymbol{h}_{i,j}^{(k)} \cdot \mathcal{O}^{(k)}(\boldsymbol{I}^{(i)}), \\ \textbf{s.t. } \boldsymbol{h}_{i,j} &= \mathrm{one\_hot}(\mathrm{argmax}_k(\boldsymbol{a}_{i,j}^{(k)}), \end{aligned} \tag{2}$$

where $\boldsymbol{a}_{i,j}^{(k)}$ is the k-th element of the architecture weight. The problem of Eq. 2 is that the discrete architecture representation $\boldsymbol{H} = \{\boldsymbol{h}_{i,j} | 2 \leq j \leq 5, i < j\}$ is not differentiable. To relax the discrete candidate sampling for gradient-based optimization, a differentiable formula of architecture parameter is required. Additionally, the Gumbel distribution is introduced to incorporate randomness. Thus, a continuous Gumbel-Softmax distribution can be used to approximate the one-hot distribution in a differentiable way as:

$$\tilde{\boldsymbol{h}}_{i,j}^{(k)} = \frac{\exp\left((\boldsymbol{a}_{i,j}^{(k)} + \boldsymbol{\xi}_{i,j}^{(k)})/\tau\right)}{\sum_{k'=1}^{K} \exp\left((\boldsymbol{a}_{i,j}^{(k')} + \boldsymbol{\xi}_{i,j}^{(k')})/\tau\right)}, \tag{3}$$

where τ is the softmax temperature and $\boldsymbol{\xi}_{i,j}^{(k)}$ is an i.i.d sample from $\mathrm{Gumbel}(0,1)$ [1]. The balance between random distribution and architecture weights allows the sampler explores the search space at the beginning of search and gradually converges to a relatively stable state with the increasing of architecture weights. For simplicity, the sampler is denoted as $\tilde{\boldsymbol{H}} = \mathrm{GumbelSoftmax}(\boldsymbol{A}; \boldsymbol{\xi}, \tau)$.

By applying Eq. 3, we can reformulate Eq. 2 into an approximated differentiable form. We replace the one-hot vector $\boldsymbol{h}_{i,j}$ with the Gumbel-Softmax distribution: $\boldsymbol{I}^{(j)} \approx \sum_{i<j} \sum_{k=1}^{K} \tilde{\boldsymbol{h}}_{i,j}^{(k)} \cdot \mathcal{O}^{(k)}(\boldsymbol{I}^{(i)})$. During model training, terms in the approximated architecture parameter $\tilde{\boldsymbol{h}}_{i,j}$ with coefficient approaching zero can be omitted. As $\boldsymbol{h}_{i,j}$ is very close to an one-hot vector, only the k-th term where $k = \mathrm{argmax}_k \tilde{\boldsymbol{h}}_{i,j}^{(k)}$ need to be calculated:

$$\begin{aligned} \boldsymbol{I}^{(j)} &\approx \sum_{i<j} \tilde{\boldsymbol{h}}_{i,j}^{(k)} \cdot \mathcal{O}^{(k)}(\boldsymbol{I}^{(i)}) \\ \textbf{s.t. } k &= \mathrm{argmax}_k \, \tilde{\boldsymbol{h}}_{i,j}^{(k)}. \end{aligned} \tag{4}$$

With Eq. 4, only one operation candidate on each edge needs to be trained at a time, which significantly reduces the memory and computation overhead comparing to (Liu et al., 2018b).

The optimization of an architecture is typically based on its validation loss. To evaluate the performance of a sampled architecture $\tilde{\boldsymbol{H}}$, we build a target super-network $\hat{y} = f(\boldsymbol{x}; \boldsymbol{w}, \tilde{\boldsymbol{H}})$ by stacking several copies of it together, where $\boldsymbol{x}$ is the network input, $\hat{y}$ is the prediction output, and $\boldsymbol{w}$ is the model weight. Loss of the target network is measured by the commonly used negative log likelihood for classification tasks:

$$\mathcal{L}(\mathbb{D}; \boldsymbol{w}, \tilde{\boldsymbol{H}}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathbb{D}} \left[-\log\left(f(\boldsymbol{x}; \boldsymbol{w}, \tilde{\boldsymbol{H}}) \right) \right], \tag{5}$$

where $\mathbb{D}$ is the corresponding training, validation or testing set depending on the context. The validation loss of an architecture is therefore calculated by $\mathcal{L}(\mathbb{D}_{val}; \boldsymbol{w}, \tilde{\boldsymbol{H}})$. With this validation loss, the architecture weights can be jointly optimized together with the target network weights as a bi-level optimization problem:

$$\begin{aligned} \min_{\boldsymbol{A}} \quad & \mathcal{L}(\mathbb{D}_{valid}; \boldsymbol{w}^*(\tilde{\boldsymbol{H}}), \tilde{\boldsymbol{H}}) \\ \textbf{s.t.} \quad & \boldsymbol{w}^*(\tilde{\boldsymbol{H}}) = \arg\max_{\boldsymbol{w}} \mathcal{L}(\mathbb{D}_{train}; \boldsymbol{w}, \tilde{\boldsymbol{H}}) \\ & \tilde{\boldsymbol{H}} = \mathrm{GumbelSoftmax}(\boldsymbol{A}; \boldsymbol{\xi}, \tau). \end{aligned} \tag{6}$$

3.3. Proxy Validation Loss Landscape

For efficient evaluation of architectures and approximation of architecture gradient, we propose to use a validation loss estimator to learn a proxy validation loss landscape in the search space. For simplicity, the real validation loss is normalized to a range of $[0, 1]$. The estimator is thus a mapping $\psi : \tilde{\boldsymbol{H}} \mapsto \hat{\mathcal{L}}$ from any given architecture weight $\tilde{\boldsymbol{H}} \in \mathbb{H}$ to a real value estimated loss $\hat{\mathcal{L}} \in [0, 1]$. In practice, we use a network ψ to fit the mapping:

$$\hat{\mathcal{L}} = \psi(\tilde{\boldsymbol{H}}). \tag{7}$$

The network ψ consists of two parts, an RNN E as architecture encoder and a dense layer D outputting a real value estimated loss. The encoder $E : \mathbb{R}^{K \times N} \to \mathbb{R}^M$ takes a $K \times N$ dimension architecture parameter as its input, where K is the candidates number and N is the edge number in a cell, and outputs a M dimension vector as embedded representation. Architecture parameters are fed to the encoder as a sequence with length N, in which each step is a K dimension vector representing a selection of candidate operations on the corresponding edge. Based on the embedding vector, the dense layer $D : \mathbb{R}^M \to [0, 1]$ can estimate the validation loss. Since the estimated validation loss has been normalized to the range of $[0, 1]$, instead of using the linear activation on the output layer as the majority of regression models usually do, we use sigmoid activation function to limit the output to fall within the above range.

[1] The Gumbel distribution is generated via a commonly used approach: $\boldsymbol{\xi}_{i,j}^{(k)} = -\log(-\log(u))$ with $u \sim \mathrm{unif}[0, 1]$

The validation loss estimator is optimized with a weighted mean squared error (MSE) loss function:

$$\min_{\psi} \quad L_T(\psi) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{p_t}\left(\psi(\tilde{\boldsymbol{H}}_t) - \mathcal{L}_t\right)^2. \tag{8}$$

where T is the total time step and $\mathcal{L}_t$ is the real validation loss of $\tilde{\boldsymbol{H}}_t$ evaluated in the super-network. We introduce p_t as a weight of H_t to stabilize the training of ψ. p_t is defined as $p_t = \psi_t^*(\tilde{\boldsymbol{H}}_t) - \psi_t^*(\tilde{\boldsymbol{H}}_{t-1})$, where ψ_t^* is the optimal validation loss estimator at time step t. Note that if $\tilde{\boldsymbol{H}}_t$ is far from $\tilde{\boldsymbol{H}}_{t-1}$, we assign a smaller weight $1/p_t$ to $\tilde{\boldsymbol{H}}_t$, so that the significant perturbation of the architecture will not influence the optimization of the validation loss estimator too much. All the pairs of sampled architecture and its real validation loss evaluated in the super-network are stored in a memory $\boldsymbol{M}$, i.e. $\boldsymbol{M} = \{(\tilde{\boldsymbol{H}}_t, \mathcal{L}_t), \forall 1 \leq t \leq T\}$. After each sampling, memory $\boldsymbol{M}$ is updated by $\boldsymbol{M} = \boldsymbol{M} \cup \{(\tilde{\boldsymbol{H}}_t, \mathcal{L}_t)\}$.

Via minimizing Eq. 8, we can obtain the optimal validation loss estimator ψ_T^* at time step T. By replacing the validation loss in Eq. 6 with the optimal validation loss estimator ψ_T^*, we can reformulate the bi-level optimization problem into a simpler formula:

$$\begin{aligned} \min_{\boldsymbol{A}} \quad & \psi_T^*(\tilde{\boldsymbol{H}}) \\ \textbf{s.t.} \quad & \tilde{\boldsymbol{H}} = \text{GumbelSoftmax}(\boldsymbol{A}; \boldsymbol{\xi}, \tau), \end{aligned} \tag{9}$$

where the lower level optimization problem is bypassed with the estimator. The major difference between Eq. 9 and Eq. 6 is that $\boldsymbol{w}^*$ is no longer involved in the searching process. In this way, we do not need to solve the bi-level optimization anymore, but only need to learn an estimator and use it to search for architectures.

We use the gradient of the proxy loss $\nabla_{\boldsymbol{A}}\psi(\tilde{\boldsymbol{H}})$ to update the architecture parameter for both searching and sampling. At a time step t, starting from an architecture parameter $\boldsymbol{A}$ with corresponding $\tilde{\boldsymbol{H}}$, updating is conducted along the gradient direction induced by estimator ψ_t^* with η as the step size:

$$\boldsymbol{A}' \leftarrow \boldsymbol{A} - \eta \cdot \nabla_{\boldsymbol{A}}\psi_t^*(\tilde{\boldsymbol{H}}), \tag{10}$$

where $\boldsymbol{A}'$ is the new architecture parameter. With appropriate step size, $\boldsymbol{A}'$ should satisfy that the estimated validation loss of it $\psi_t^*(\boldsymbol{A}') \leq \psi_t^*(\boldsymbol{A})$. The next architecture to evaluate is therefore obtained according to $\boldsymbol{A}'$ by $\tilde{\boldsymbol{H}}' = \text{GumbelSoftmax}(\boldsymbol{A}'; \boldsymbol{\xi}', \tau)$. The memory $\boldsymbol{M}$ is then updated to $\boldsymbol{M} = \boldsymbol{M} \cup \{(\tilde{\boldsymbol{H}}', \mathcal{L}')\}$.

3.4. Search Procedure

The overall procedure of our framework is shown in Algorithm 1. Firstly, we randomly initialize a warm-up population $\mathbb{H}$ of size N and use architecture $\boldsymbol{H}_i \in \mathbb{H}$ to warm up the target super-network f. The validation loss of $\boldsymbol{H}_i$ after network warm-up along with the architecture itself is then saved to a memory $\boldsymbol{M}$ as architecture-performance pair, which is used to warm up and train the validation loss estimator ψ. The purpose of warm-up is to let the validation loss estimator equipped with prior knowledge about the validation loss landscape before the searching conducts.

When the target network f and validation loss estimator ψ are both warmed up, the search commences. In each epoch of the search stage, we sample an architecture weight $\tilde{\boldsymbol{H}}_t$ according to the current architecture parameter $\boldsymbol{A}_t$ as in Eq. 4, and then update the corresponding sub-graph in the super network. The sub-graph is then evaluated to obtain a validation loss for validation loss estimator training. The architecture parameter $\boldsymbol{A}_t$ is then updated with the estimated loss with Eq. 10 for the subsequent sampling.

Algorithm 1 Loss Space Regression

Initialize a warm-up population:
$\mathbf{P} = \{\tilde{\boldsymbol{H}}_i | i = 1, ..., N\}$
for each $\tilde{\boldsymbol{H}}_i \in \mathbf{P}$ **do**
  Warm-up architecture $\tilde{\boldsymbol{H}}_i$ for 1 epoch
end for
Initialize a performance memory $\boldsymbol{M} = \emptyset$
for each $\tilde{\boldsymbol{H}}_i \in \mathbf{P}$ **do**
  Train architecture $\tilde{\boldsymbol{H}}_i$ for 1 epoch
  Evaluate architecture $\tilde{\boldsymbol{H}}_i$'s loss $\mathcal{L}_i$
  Set $\boldsymbol{M} = \boldsymbol{M} \cup \{(\tilde{\boldsymbol{H}}_i, \mathcal{L}_i)\}$
end for
Warm-up ψ with $\boldsymbol{M}$
for $t = 1 \rightarrow T$ **do**
  Sample an architecture as in Eq. 4 with $\tilde{\boldsymbol{H}}_t$:
  $\tilde{\boldsymbol{H}}_t = \text{GumbelSoftmax}(\boldsymbol{A}_t; \boldsymbol{\xi}_t, \tau)$
  Optimize network with loss in Eq. 5
  Evaluate architecture to obtain loss $\mathcal{L}_t$
  Set $\boldsymbol{M} = \boldsymbol{M} \cup \{(\tilde{\boldsymbol{H}}_t, \mathcal{L}_t)\}$
  Update ψ with Eq. 8
  Update $\boldsymbol{A}_t$ to $\boldsymbol{A}_{t+1}$ with Eq. 10
end for

4. Theoretical Analysis

In this section, we apply theoretical analysis to demonstrate our method that trains validation loss estimator with architectures sampled by Eq. 10 is consistent and has a lower label complexity compared to a baseline method that samples architectures following a uniform distribution.

4.1. The Algorithm Consistency

A desirable feature of our method is that with large enough total step T, Eq. 9 can finally output a validation loss estimator ψ_T whose loss is at most $L^* + \epsilon$, where L^* is the optimal loss can be reached and ϵ is a constant small enough.

This feature is typically referred as the consistency.

We define a hypothesis class Ψ, which is a parameter space such that $\psi \in \Psi$. The training of validation loss estimator ψ is to find the optimal parameter $\psi^* = \text{argmin}_{\psi \in \Psi} L(\psi)$ within the hypothesis class Ψ. The architecture search space $\mathbb{H}$ is the input space of the validation loss estimator, and the $[0, 1]$ normalized architecture loss space is the corresponding output space. The loss function defined in Eq. 9 is usually referred as the empirical loss. Replacing MSE with a more general formula $\ell(\cdot)$, we can rewrite the empirical loss as:

$$L_T(\psi) = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{p_t} \ell(\psi(\boldsymbol{H}_t), \mathcal{L}_t), \tag{11}$$

where T is the total time step and p_t is the probability that $\boldsymbol{H}_t$ is sampled. $\mathcal{L}_t$ is the validation loss of sampled architecture $\boldsymbol{H}_t$, which is evaluated in the super-network f. It is the ground truth for validation loss estimator training. The loss function $\ell(\cdot)$ in Eq. 11 measures the error of estimator output $\psi(\boldsymbol{H}_t)$ w.r.t. the ground truth $\mathcal{L}_t$.

To compare with the empirical loss calculated on samples selected given Eq. 10, we also define an *expected loss* (a.k.a. *true loss*). The expected loss is the loss of a given validation loss estimator ψ calculated on the entire data space $D = \{(\boldsymbol{H}, \mathcal{L}_{\boldsymbol{H}}) | \boldsymbol{H} \in \mathbb{H}\}$, which is defined as

$$L(\psi) = \mathbb{E}_{(\boldsymbol{H}, \mathcal{L}) \sim D} \left[l(\psi(\boldsymbol{H}), \mathcal{L}) \right]. \tag{12}$$

If our Eq. 8 is consistent, the difference between the empirical and expected loss should be within an arbitrarily small value, with probability arbitrarily to 1:

$$\mathbf{P}[|L_T(\psi) - L(\psi)| < \epsilon] > 1 - \delta, \tag{13}$$

where δ is a constant close to 0. We gives the error bound ϵ in this inequality in Theorem 1. The detailed proof is provided in the supplementary material.

Theorem 1. *Let Ψ be a hypothesis class, L_T be the empirical loss, and L be the expected loss. For any $\delta > 0$, with probability at lest $1 - \delta$, $\forall \psi \in \Psi$:*

$$|L_T(\psi) - L(\psi)| < \sqrt{\frac{2\left(d + \ln \frac{2}{\delta}\right)}{T}}, \tag{14}$$

where d is the Pollard's pseudo-dimension of Ψ.

Theorem 1 shows that the gap between the empirical and expected loss of is at most of order $\mathcal{O}(\sqrt{d/T})$. The gap is only related to Pollard's pseudo-dimension d and total time step T. With a fixed d, the gap decreases with T increases. The consistency indicates that the validation loss estimator trained with architectures sampled by our sampling strategy is competitive with an estimator trained on the entire search space.

4.2. The Label Complexity

In NAS problem, we typically does not have all the validation losses of architectures in the search space and the cost of evaluating an architecture is expensive. Therefore we focus the complexity analysis on how architectures to be evaluated are sampled. We use the *label complexity*, which measures how many architectures we need to evaluate to learn a validation loss estimator by Eq. 8.

According to our sampling strategy, the architecture to be sampled $\boldsymbol{H}_{t+1}$ at time step $t+1$ satisfies that the estimated validation loss of it is smaller than the estimated validation loss of architecture previously sampled at time step t, i.e. $\psi_t(\boldsymbol{H}_{t+1}) \leq \psi_t(\boldsymbol{H}_t)$, as long as the step size η is within a reasonable range (e.g. small enough). According to the standard label complexity, the validation loss estimator ψ_t has an maximum allowed slack $\Delta_t = \sqrt{(8/t)(d + \ln(2/\delta))}$, which means the estimated validation loss at most larger than the ground truth than Δ_t, i.e. $\psi_t(\boldsymbol{H}_t) \leq \mathcal{L}_t + \Delta_t$. This is equivalent to shrink the current search space into a smaller subset $\mathbb{H}_{t+1}$:

$$\mathbb{H}_{t+1} = \{\boldsymbol{H} \in \mathbb{H}_t : \psi_t(\boldsymbol{H}) \leq \psi_t(\boldsymbol{H}_t) \leq \mathcal{L}(\boldsymbol{H}_t) + \Delta_t\}. \tag{15}$$

The new sample $\mathbb{H}_{t+1}$ should locate in the subset $\boldsymbol{H}_{t+1}$. The initial search space is set to be the entire search space: $\mathbb{H}_0 = \mathbb{H}$.

In Theorem 2, we demonstrate that by updating the architecture sampler with Eq. 10, we can reach a sub-linear label complexity, which makes the training of our validation loss estimator more efficient than sampling architectures uniformly or following any other fixed distribution. The detailed proof is provided in the supplementary material.

Theorem 2. *Let Ψ be a hypothesis class containing all the possible hypothesises of estimator ψ, and N be the size of a input space, with probability at least $1 - \delta$, the number of labels required to optimize Eq. 8 is at most the order of*

$$\mathcal{O}\left(\sqrt{N(d + \ln(2/\delta))}\right), \tag{16}$$

where d is the Pollard's pseudo-dimension of Ψ.

Theorem 2 defines the bound of label complexity of Eq. 8. The label complexity is proportional to the size N of the search space in a sub-linear form, which outperforms linear complexity.

5. Experiments

In experiments, we evaluated the performance of architectures searched by the proposed algorithm on the CIFAR and ImageNet datasets. Besides, we specifically design some further experiments to demonstrate the effectiveness of the proposed estimator and sampling strategy.

5.1. Search and Evaluation on CIFAR-10

Following previous works (Liu et al., 2018b; Dong & Yang, 2019), we use the CIFAR-10 dataset (Krizhevsky et al., 2009) for architecture searching and results evaluation. The CIFAR-10 dataset contains 50,000 training images together with 10,000 testing images from 10 classes. During the searching phase, we shuffle the training set and divide it into two parts with equal size for model weights training and validation performance inference respectively.

In our framework, operation candidates are formed in a ReLU-Conv-BN pattern. We set candidates number $K = 8$, including 4 convolutional operations: 3×3 separable convolutions, 5×5 separable convolutions, 3×3 dilated separable convolutions and 5×5 dilated separable convolutions, 2 pooling operations: 3×3 average pooling and 3×3 max pooling, and two special operations: an $\mathrm{identity}$ operation representing skip-connection and a zero operation representing two nodes are not connected.

The super-network for searching is constructed by stacking 8 cells. Cells locate at the $1/3$ and $2/3$ of the network (in this case, the 3rd and 6th cells) are set as reduction cells, while others are normal cells. The network has 16 initial channels, which are doubled after each reduction cell. After the searching, a large-scale network is constructed following the pattern of obtained architecture and retrained with the whole training set.

We follow the search procedure in Section 3.4. The warm-up population is initialized with 100 random sampled architectures. The performance memory is a queue with a maximum length of 100, which is just equal to the size of the warm-up population. In this way, the first sampled architecture can survive until the warm-up stage finish, and as the search progresses, the warm-up population will be forgotten gradually. This setting also ensures that the estimator is always trained with an equal number of samples.

We trained models in the warm-up population with mini-batch gradient descent, whose batch size is set to 64 and the base learning rate is set to 0.025. The learning rate is gradually reduced with cosine annealing. The architecture weights and validation loss estimator are both optimized by Adam with a constant learning rate of 0.1. The Softmax temperature τ in Gumbel-Softmax is set to 0.1. To evaluate the performance of the obtained architecture, a larger network is constructed with 20 stacked cells and 36 initial channels. The network is trained with the same training setting as in the searching phase for 600 epochs on the complete CIFAR-10 training set.

The corresponding test results on CIFAR-10 is shown in Table 1 to compare with both hand-crafted and NAS searched state-of-the-art architectures. As for our method, PVLL-NAS outperforms all previous methods with similar parameter numbers and achieves the fastest searching speed, as shown in Table 1. After searching for 0.2 GPU day, our algorithm achieves better trade-offs between error rate and parameters, which reaches 2.70% error rate on CIFAR-10 with 3.3M parameters.

Model	GPUs	Time (Days)	Params (M)	Test Error (%)
ResNet-110	-	-	1.7	6.61
DenseNet-BC	-	-	25.6	3.46
MetaQNN	10	8-10	11.2	6.92
NAS	800	21-28	7.1	4.47
NAS+more filters	800	21-28	37.4	3.65
ENAS	1	0.32	21.3	4.23
ENAS+more channels	1	0.32	38.0	3.87
NASNet-A	450	3-4	3.3	3.41
NASNet-A+cutout	450	3-4	3.3	2.65
ENAS	1	0.45	4.6	3.54
ENAS+cutout	1	0.45	4.6	2.89
DARTS(1st)+cutout	1	1.50	3.3	3.00
DARTS(2nd)+cutout	1	4	3.3	2.76
NAONet+cutout	200	1	128	2.11
NAONet+WS	1	0.30	2.5	3.53
GDAS	1	0.21	3.4	3.87
GDAS+cutout	1	0.21	3.4	2.93
PVLL-NAS	1	0.20	3.3	2.70

Table 1. Comparison of PVLL-NAS with different state-of-the-art CNN models on CIFAR-10 dataset.

Comparing to NAO (Luo et al., 2018), the gradient of the validation loss estimator is employed to determine architectures to be evaluated. Such a sampling strategy is more specific and architecture been sampled can contribute more to the training of validation loss estimator. We can thus learn a competitive proxy validation loss landscape with fewer samples and hence fewer search time. Comparing to GDAS (Dong & Yang, 2019), we search architectures with the proxy validation loss landscape, which includes knowledge from all previous evaluations, instead of only using the current gradient information to update the architecture parameter. Therefore we can achieve better results with equivalent searching time.

5.2. Generality on ImageNet

The generality of architecture we obtained is tested on ImageNet 2012 (Russakovsky et al., 2015). ImageNet 2012 is a large-scale dataset containing 1.3 million training images and 50,000 testing images in 10,000 categories. The training images in it is 26 times that of CIFAR-10, and its classes are 100 times that of CIFAR-10, which makes it a common dataset for testing the generality of architectures in NAS studies. We follow the mobile setting (Sandler et al., 2018) for ImageNet, where the input size is fixed to 224×224 with 3 channels and the number of multiply-add operations is restricted to be less than 600M. We set other hyper-parameters following (Dong & Yang, 2019; Liu et al., 2018b; Xu et al., 2019). The constructed model contains 14 layers with 48 initial channels. Our best architecture on

Model	GPUs	Time (Days)	Params (M)	+× (M)	Test Error (%) Top-1	Top-5
Inception-V1	-	-	6.6	1448	30.2	10.1
MobileNet-V2	-	-	3.4	300	28.0	-
ShuffleNet	-	-	~ 5	524	26.3	-
MnasNet-A3	8	472*	4.4	403	23.3	6.7
AmoebaNet-A	450	7	5.1	555	25.5	8.0
AmoebaNet-B	450	7	5.3	555	26.0	8.5
AmoebaNet-C	450	7	6.4	570	24.3	7.6
NASNet-A	450	3-4	5.3	564	26.0	8.4
NASNet-B	450	3-4	5.3	488	27.2	8.7
NASNet-C	450	3-4	4.9	558	27.5	9.0
Progressive NAS	100	1.5	5.1	588	25.8	8.1
DARTS	1	4	4.9	595	26.7	8.7
FBNet-C	8	1.13	5.5	375	25.1	-
GDAS	1	0.21	5.3	581	26.0	8.5
PVLL-NAS	1	0.20	5.0	556	25.6	8.1

* MnasNet takes 4.5 days on 64 TPUv2 for one search. The GPU days is estimated by Wu et al. (2019).

Table 2. Top-1 and top-5 error rates of PVLL-NAS and other state-of-the-art cnn models on ImageNet dataset.

ImageNet is illustrated in Fig. 2.

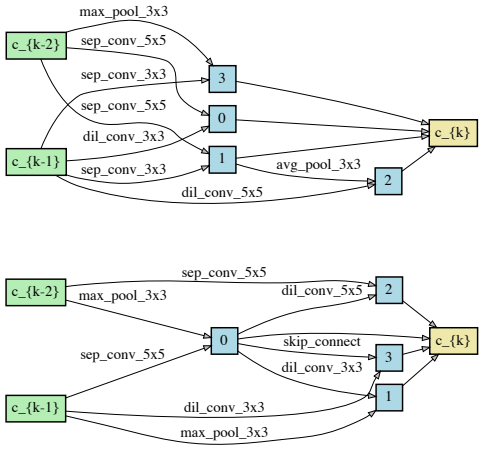

Figure 2. Normal (the upper one) and reduction (the lower one) cells that generalize best to ImageNet.

We compare our results with other NAS algorithms in Table 2. PVLL-NAS outperforms most gradient-based NAS algorithms, including DARTS and GDAS, which reaches 25.6% error rate on ImageNet with 4.8M parameters. Compared with other baselines, our algorithm achieves competitive performance with much lower search cost, which demonstrates the efficiency of PVLL-NAS. Consistently, PVLL-NAS achieves better trade-offs on both CIFAR-10 and ImageNet, which highlights the generality of our algorithm.

5.3. Warm-up with Different Population Sizes

To reach a better trade-off between performance and efficiency, we evaluate various learned PVLLs with different population sizes on a separated validation set. Table 5.3 shows the estimation loss of those PVLLs. The result shows that a population size of 100 can ensure a competitive validation performance with reasonable cost (about 48% of the overall search time), and further enlarging the population size cannot guarantee a consistent increasing of performance. Besides, network weights are updated gradually, and a large population contains samples from a wide period which might confuse the estimator. This can explain why the loss with a population size of 500 is even worse than the loss with a population size of 200.

Population Size	50	100	200	500
GPU hours	1.2	2.3	4.6	11.6
Estimation Loss	0.0023	0.0009	0.0005	0.0007

Table 3. Estimation loss of various PVLLs with different population sizes

5.4. The First and Second Order Estimation

We argue that optimize the architecture parameter with the architecture gradient estimated with the proxy validation loss landscape is reasonable, and the effect of this estimation is competitive comparing to other estimation methods. To demonstrate that, we design an experiment following the concept of the 2nd order approximation in DARTS (Liu et al., 2018b). The 2nd order approximation of architecture gradient in DARTS is to approximate the optimal model weight w^* by a single training step and compute architecture gradient based on the approximated model weight w'. A counterpart of the 2nd order approximation in our Algorithm 1 is that after sampling each architecture, we further train it in the super-network for 1 epoch and evaluate its validation loss. We refer this validation loss as the 2nd order validation loss. By contrast, we evaluate the validation losses of architectures in the super-network without further training, which is referred as the 1st order validation loss. Note that super-networks in both cases are warmed up, otherwise the validation loss will make no sense. Estimations depending on the 1st order and 2nd order validation losses are referred as 1s order and 2nd order estimation respectively. The results of searching with 1st order and 2nd order estimations are listed in Table 4.

Method	Order	Time (Days)	Test Error (%)
DARTS	1st	1.5	3.00 ± 0.14
	2nd	4.0	2.76 ± 0.09
Amended-DARTS	1st	-	-
	2nd	1.0	2.81 ± 0.21
PVLL-NAS	1st	0.10	3.48
	2nd	0.20	2.72 ± 0.02

Table 4. Performances of architectures found on CIFAR-10 with different order of approximation.

Not surprisingly, the performance of architecture obtained by searching with the 1st order estimation is worse than that obtained by the regular search scheme using 2nd order estimation, which is even worse than the 1st order approximation in the original DARTS. However, after the 2nd order estimation is introduced, the improvement is significant. Besides, our 2nd order estimation method is about two times slower than its 1st order counterpart, which is mainly caused by the sampled architecture training. We also compared our results with a DARTS approach amended by a mathematical method to make the 2nd order approximation, namely Amended-DARTS (Bi et al., 2019). Amended-DARTS is experimented on two different search spaces. We only compared with their results on S_1 search space, which is the same to the original DARTS. The comparison with other baselines demonstrates the effectiveness and stability of proposed validation loss estimator.

5.5. Comparison of Different Sampling Strategies

We furthermore conduct in-depth analysis towards the effect of our sampling strategies. In Fig. 3 (a), the learning curve of estimator trained with our sampling strategy is compared against learning curve with uniformly sampling as a baseline. We sampled 100 architectures with our sampling strategy. Another 100 architectures are uniformly sampled. The test set is 100 independently and randomly (also uniformly) sampled architectures from the search space. We also ensure that training sets are mutually exclusive from the test set. The same elements are allowed to exist in both training sets, but due to the fact that the search space is large enough, this is very unlikely to happen. For fairness, we train two estimators with the same setting. Training data is passed to the estimators one by one at each epoch during the training progress to simulate the behaviour in architecture searching. The estimator performance is tested on the remaining test set after each epoch.

As shown in Fig. 3 (a), our sampling strategy shows superior prediction performance when sampling number is limited compared with uniform sampling. The sampled architectures fed to estimator are determined by the gradient of proposed validation loss estimator for architecture updating, which makes full use of sampled architectures for estimator updating and thus accelerates the training as well as reduces the requirement for sample number. As sample number increases, our method still shows stable performance and continuously outperforms uniformly sampling.

Furthermore, we compare the performance of architectures found with different sampling strategies in Table 5. Besides, the involvement of warm-up and the weigh term of the loss function in Eq. 8 is also justified. For searching with warm-up, we use 100 randomly sampled architectures to warm up the model and then search for 100 steps in which

With Sampler	Warm-up	Weighted Loss	Test Error (%)
Y	Y	Y	2.72 ± 0.02
Y	Y	N	2.81 ± 0.08
Y	N	Y	3.10 ± 0.22
Y	N	N	3.03 ± 0.30
N	Y	N/A	3.08 ± 0.24
N	N	N/A	3.20 ± 0.32

Table 5. Ablation studies on the performances of architectures searched on CIFAR-10 with different strategies.

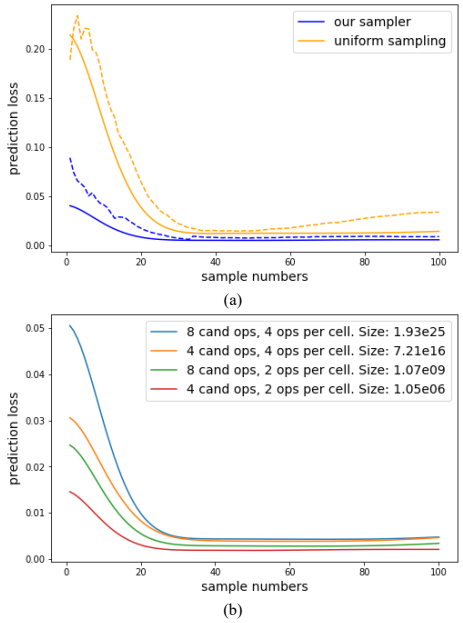

Figure 3. (a) presents the learning curves with different sampling strategies. The dashed line indicates training loss, and the solid line indicates test loss. (b) shows the learning curves on search space with different sizes.

100 new architectures are extended to the training set. It is worth noting that when there is no warm-up, we will search for 200 steps to maintain the same search budget. The purpose of using warm-up is to enable our sampler to acquire prior knowledge about loss space at the beginning of searching. Without a warm-up, the initial behaviour of our sampler will be close to uniform sampling, but as the search proceeds, it still has the potential to outperform uniform sampling. Comparing all the variants in Table. 5, the usage of sampler, warm-up and weighted loss all contribute to the performance and warm-up boosts the stability of our algorithm. This observation empirically verifies that the prior knowledge of loss space is necessary for the sampler to start with and the weighted loss contributes to the algorithm consistency. With the combination of these components, the discovered architecture achieves the best and the most stable performance among all the variants, which illustrates the complementarity of each component.

5.6. Estimators on Different Search Spaces

In Theorem 2, we have shown that the label complexity of our algorithm is related to the size of the search space. In this section, we furthermore justify this conclusion with experiments. We define 4 search spaces with different size. The size of the search space is controlled by both the number of operation candidates and the number of operations allowed in each cell. Detailed settings for each search space is shown in the legend of Fig. 3 (b). The total number of architectures in a search space can be easily computed given those settings. For example, with 8 operation candidates and 4 operations per cell, the number of cells can be constructed is $8^{(2+3+4+5)} = 4.39 \times 10^{12}$. As there are two kinds of cells (normal and reduction), the total number of architectures in the search space is $(4.39 \times 10^{12})^2 = 1.93 \times 10^{25}$. Test loss curves of estimators trained on those search spaces are compared. Theorem 2 suggests that the speed of convergence is proportional to the size of data space in sub-linear relation. Thus, the increment of size of search space impacts slightly on convergence speed of estimator training. As shown in Fig. 3 (b), 4 search spaces with different size have similar converge speed and requirement of sample number, which illustrates the efficiency of our algorithm.

6. Conclusion

In this paper, we propose to search for neural architectures with a proxy validation loss landscape. We introduce a novel method to dynamically sample architecture to be evaluated for the efficient validation loss estimator training. Both theoretical analysis and experiments show that this approach can establish a satisfactory proxy validation loss landscape with less computational resource. Experimental results demonstrate that the proposed NAS algorithm can efficiently design networks of the competitive performance compared to state-of-the-art methods.

Acknowledgement

The authors would like to thank the Area Chair and the reviewers for their constructive comments. This work was supported by the Australian Research Council under Project DE180101438.

References

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Baker, B., Gupta, O., Raskar, R., and Naik, N. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

Bello, I., Zoph, B., Vasudevan, V., and Le, Q. V. Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 459–468. JMLR. org, 2017.

Bi, K., Hu, C., Xie, L., Chen, X., Wei, L., and Tian, Q. Stabilizing darts with amended gradient estimation on architectural parameters. *arXiv preprint arXiv:1910.11831*, 2019.

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. Reinforcement learning for architecture search by network transformation. *arXiv preprint arXiv:1707.04873*, 2017.

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Dempe, S. *Bilevel optimization: theory, algorithms and applications*. TU Bergakademie Freiberg, Fakultät für Mathematik und Informatik, 2018.

Deng, B., Yan, J., and Lin, D. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017.

Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Dong, X. and Yang, Y. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1761–1770, 2019.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

Fahlman, S. E. and Lebiere, C. The cascade-correlation learning architecture. In *Advances in neural information processing systems*, pp. 524–532, 1990.

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

Kitano, H. Designing neural networks using genetic algorithms with graph generation system. *Complex systems*, 4(4):461–476, 1990.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. Learning curve prediction with bayesian neural networks. *ICLR 2017*, 2016.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018a.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in neural information processing systems*, pp. 7816–7827, 2018.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pp. 293–312. Elsevier, 2019.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

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

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

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Swersky, K., Snoek, J., and Adams, R. P. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

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

Veniat, T. and Denoyer, L. Learning time/memory-efficient deep architectures with budgeted super networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3492–3500, 2018.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., and Xiong, H. Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. *arXiv preprint arXiv:1907.05737*, 2019.

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

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.