Meta-Learning of Neural Architectures for Few-Shot Learning

Thomas Elsken[1,2], Benedikt Staffler[1], Jan Hendrik Metzen[1] and Frank Hutter[2,1]

[1]Bosch Center for Artificial Intelligence, [2]University of Freiburg

{thomas.elsken, benediktsebastian.staffler, janhendrik.metzen}@de.bosch.com,
fh@cs.uni-freiburg.de

Abstract

The recent progress in neural architecture search (NAS) has allowed scaling the automated design of neural architectures to real-world domains, such as object detection and semantic segmentation. However, one prerequisite for the application of NAS are large amounts of labeled data and compute resources. This renders its application challenging in few-shot learning scenarios, where many related tasks need to be learned, each with limited amounts of data and compute time. Thus, few-shot learning is typically done with a fixed neural architecture. To improve upon this, we propose METANAS, *the first method which fully integrates NAS with gradient-based meta-learning.* METANAS *optimizes a meta-architecture along with the meta-weights during meta-training. During meta-testing, architectures can be adapted to a novel task with a few steps of the task optimizer, that is: task adaptation becomes computationally cheap and requires only little data per task. Moreover,* METANAS *is agnostic in that it can be used with arbitrary model-agnostic meta-learning algorithms and arbitrary gradient-based NAS methods. Empirical results on standard few-shot classification benchmarks show that* METANAS *with a combination of DARTS and REPTILE yields state-of-the-art results.*

1. Introduction

Neural architecture search (NAS) [14] has seen remarkable progress on various computer vision tasks, such as image classification [58, 40, 7], object detection [20], semantic segmentation [8, 30, 35], and disparity estimation [43]. One key prerequisite for this success is the availability of large and diverse (labeled) data sets for the respective task. Furthermore, NAS requires considerable compute resources for optimizing the neural architecture for the target task.

This makes it difficult to apply NAS in use-cases where one does not focus on a single task but is interested in a large set (distribution) of tasks. To be effective in this setting, learning must not require large amounts of data and
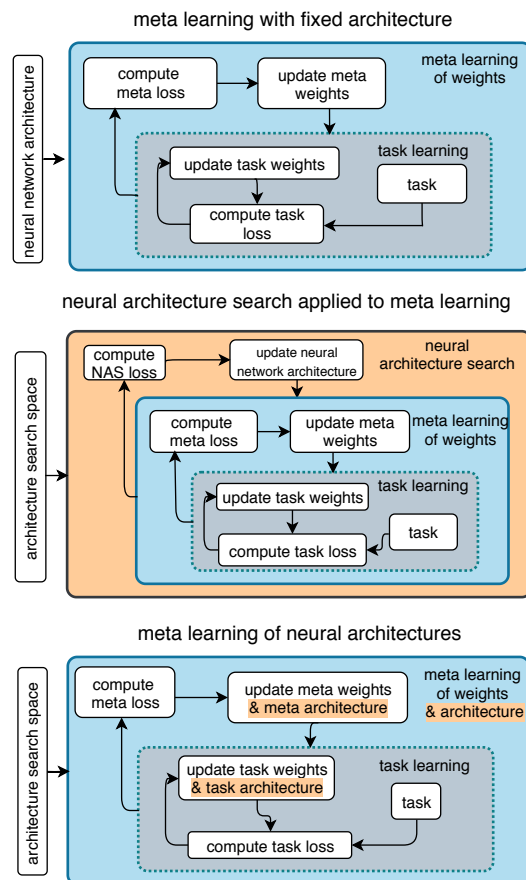

Figure 1: Illustration of our proposed method METANAS and related work. Gray highlights task learning, blue meta-learning, and orange NAS components. Top: gradient-based meta-learning with fixed architecture such as MAML [16] or REPTILE [36]. Middle: applying NAS to meta-learning such as AutoMeta [24]. Bottom: Proposed joint meta-learning of architecture and weights with METANAS. Since architectures are adapted during task learning, the proposed method can learn *task-specific* architectures.

compute for every task but should, like humans, be able to rapidly adapt to novel tasks by building upon experience

from related tasks [26]. This concept of learning from experience and related tasks is known as *meta-learning* or *learning to learn* [45, 49, 21]. Here, we consider the problem of few-shot learning, i.e., learning new tasks from just a few examples. Prior work has proposed meta-learning methods for this problem that are model-agnostic [16, 36] and allow meta-learning weights of fixed neural architectures (see Figure 1, top).

In this work, we fully integrate meta-learning with NAS, by proposing METANAS. METANAS allows adapting architectures to novel tasks based on few datapoints with just a few steps of a gradient-based task optimizer. This allows METANAS to generate *task-specific architectures* that are adapted to every task separately (but from a joint meta-learned meta-architecture). This is in contrast to prior work that applied NAS to multi-task or few-shot learning, where a single neural architecture is optimized to work well on average across all tasks [24, 38] (see Figure 1, middle). Moreover, our method directly provides trained weights for these task-specific architectures, without requiring meta retraining them as in concurrent work [29]. Conceptual illustrations of our method are shown in Figure 1, bottom, and in Figure 3. The key contributions of this work are as follows:

1. We show that model-agnostic, gradient-based meta-learning methods (such as [16]) can very naturally be combined with recently proposed gradient-based NAS methods, such as DARTS [32]. This allows for joint meta-learning of not only the weights (for a given, fixed architecture) but also *meta-learning the architecture itself* (Section 3, see Figure 1 for an illustration).

2. We propose METANAS, a meta-learning algorithm that can quickly adapt the meta-architecture to a *task-dependent architecture*. This optimization of the architecture for the task can be conducted with few labeled datapoints and only a few steps of the task optimizer (see Figure 3 for an illustration).

3. We extend DARTS such that task-dependent architectures need not be (meta) re-trained, which would be infeasible in the few-shot learning setting with task-dependent architectures for hundreds of tasks (requiring hundreds re-trainings). We achieve this by introducing a novel *soft-pruning* mechanism based on a temperature annealing into DARTS (see Figure 2). This mechanism lets architecture parameters converge to the architectures obtained by the hard-pruning at the end of DARTS, while giving the weights time to adapt to this pruning. Because of this, pruning no longer results in significant drops in accuracy, which might also be of interest for the standard single-task setting. We give more details in Section 4.

METANAS is agnostic in the sense that it is compat-

ible with arbitrary gradient-based model-agnostic meta-learning algorithms and arbitrary gradient-based NAS methods employing a continuous relaxation of the architecture search space. Already in combination with the simple meta-learning algorithm REPTILE[36] and NAS algorithm DARTS[32], METANAS yields state-of-the-art results on the standard few-shot classification benchmarks Omniglot and MiniImagenet.

This paper is structured as follows: in Section 2, we review related work on few-shot learning and neural architecture search. In Section 3, we show that model agnostic, gradient-based meta-learning can naturally be combined with gradient-based NAS. The soft-pruning strategy to obtain task-dependent architectures without the need for retraining is introduced in Section 4. We conduct experiments on standard few-shot learning data sets in Section 5 and conclude in Section 6.

2. Related Work

Few-Shot Learning via Meta-Learning Few-Shot Learning refers to the problem of learning to solve a task (e.g., a classification problem) from just a few training examples. This problem is challenging in combination with deep learning as neural networks tend to be highly over-parameterized and therefore prone to overfitting when only very little data is available. Prior work [39, 16, 18, 19] often approaches few-shot learning via *meta-learning* or *learning to learn* [45, 49, 21], where one aims at learning from a variety of learning tasks in order to learn new tasks much faster than otherwise possible [50]. There are various approaches to few-shot learning, e.g., learning to compare new samples to previously seen ones [47, 51] or meta-learning a subset of weights that is shared across tasks but fixed during task learning [57, 19].

In this work, we focus on a particular class of approaches denoted as *model-agnostic meta-learning* [16, 17, 36, 1, 18]. These methods meta-learn an initial set of weights for neural networks that can be quickly adapted to a new task with just a few steps of gradient descent. For this, the meta-learning objective is designed to explicitly reward quick adaptability by incorporating the task training process into the meta-objective. This meta-objective is then usually optimized via gradient-based methods. Our method extends these approaches to not only meta-learn an initial set of weights for a given, fixed architecture but to also meta-learn the architecture itself. As our method can be combined with any model-agnostic meta-learning method, future improvements in these methods can be directly utilized in our framework.

Neural Architecture Search Neural Architecture Search (NAS), the process of automatically designing neural network architectures [14], has recently become a popular

approach in deep learning as it can replace the cumbersome manual design of architectures while at the same time achieving state-of-the-art performance on a variety of tasks [58, 40, 9, 43]. We briefly review the main approaches and refer to the recent survey by Elsken *et al.* [14] for a more thorough literature overview. Researchers often frame NAS as a reinforcement learning problem [2, 58, 56, 59] or employ evolutionary algorithms [48, 34, 41, 40]. Unfortunately, most of these methods are very expensive, as they require training hundreds or even thousands of architectures from scratch. Therefore, most recent work focuses on developing more efficient methods, e.g., via network morphisms [12, 6, 13, 46], weight sharing [44, 5, 4], or multi-fidelity optimization [3, 15, 28, 55]; however, they are often still restricted to relatively small problems.

To overcome this problem, Liu *et al.* [32] proposed a continuous relaxation of the architecture search space that allows optimizing the architecture via gradient-based methods. This is achieved by using a weighted sum of possible candidate operations for each layer, where the real-valued weights then effectively parametrize the network's architecture as follows:

$$
\begin{aligned}
x^{(j)} &= \sum_{i<j} \sum_{o \in \mathcal{O}} \hat{\alpha}_o^{i,j} o\left(x^{(i)}, w_o^{i,j}\right) \\
&=: \sum_{i<j} MixedOp\left(x^{(i)}, w^{i,j}\right),
\end{aligned}
\tag{1}
$$

where $\hat{\alpha}_o^{i,j} = \frac{\exp(\alpha_o^{i,j})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{i,j})}$ are normalized mixture weights that sum to 1, $x^{(j)}$ and $x^{(i)}$ represent feature maps in the network, $\mathcal{O}$ denotes a set of candidate operations (e.g., 3×3 convolution, 5×5 convolution, 3×3 average pooling, ...) for transforming previous feature maps to new ones, $w = (w_o^{i,j})_{i,j,o}$ denotes the regular weights of the operations, and $\alpha = (\alpha_o^{i,j})_{i,j,o}$ serves as a real valued, unconstrained parameterization of the architecture. The mixture of candidate operations is denoted as *mixed operation* and the model containing all the mixed operations is often referred to as the *one-shot model*.

DARTS then optimizes both the weights of the one-shot model w and architectural parameters α by alternating gradient descent on the training and validation loss, respectively. After the search phase, a discrete architecture is obtained by choosing a predefined number (usually two) of most important incoming operations (those with the highest operation weighting factor $\hat{\alpha}_o^{i,j}$) for each intermediate node j while all others are pruned. This hard-pruning deteriorates performance [53, 54]: e.g., Xie *et al.* [53] report a performance drop from 88% (one-shot model's accuracy) to 56% (pruned model's accuracy). Thus, the pruned model requires retraining w. We address this shortcoming in Section 4.

In our work, we choose DARTS for neural architecture

search because of conceptual similarities to gradient-based meta-learning, such as MAML [16], which will allow us to combine the two kinds of methods.

Neural Architecture Search for Meta-Learning There has been some recent work on combining NAS and meta-learning. Wong *et al.* [52] train an automated machine learning (AutoML [22]) system via reinforcement learning on multiple tasks and then use transfer learning to speed up the search for hyperparameters and architecture via the learned system on new tasks. Their work is more focused on hyperparameters rather than the architecture; the considered architecture search space is limited to choosing one of few pretrained architectures.

Closest to our work are [24, 29]. Kim *et al.* [24] wrap neural architecture search around meta-learning as illustrated in Figure 1 (middle). They apply progressive neural architecture search [31] to few shot learning, but this requires running the entire meta-training from scratch in every iteration of the NAS algorithm; therefore, their approach requires large computational costs of more than a hundred GPU days. The approach is also limited to searching for a single architecture suitable for few-shot learning rather than learning task-dependent architectures, which our methods supports. In a concurrent work [29], the authors proposed to combine gradient-based NAS and meta-learning for finding task-dependent architectures, similar to our work. However, as they employ the hard-pruning strategy from DARTS with significant drops in performance, they require re-running meta-training for *every* task-dependent architecture (potentially hundreds), rendering the evaluation of novel tasks expensive. In contrast, our method does not require re-training task-dependent architectures and thus a single run of meta-training suffices.

3. Marrying Gradient-based Meta-Learning and Gradient-based NAS

Our goal is to build a meta-learning algorithm that yields a meta-learned architecture α_{meta} with corresponding meta-learned weights w_{meta}. Then, given a new task $\mathcal{T}_i$, both α_{meta} and w_{meta} shall quickly adapt to $\mathcal{T}_i$ based on few labeled samples. To solve this problem, we now derive METANAS, a method that naturally combines gradient-based meta-learning methods with gradient-based NAS and allows meta-learning α_{meta} along with w_{meta}. In Section 4, we will then describe how the meta-architectures encoded by α_{meta} can be quickly specialized to a new task without requiring re-training of w_{meta}.

3.1. Problem Setup for Few-Shot Classification

In the classic supervised deep learning setting, the goal is to find optimal weights of a neural network by minimiz-
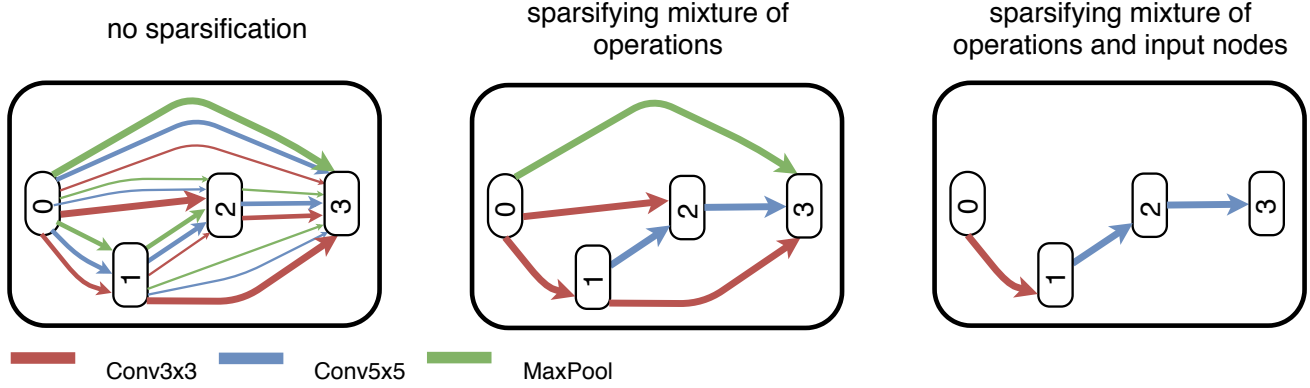
Figure 2: Illustration of sparsity of the one-shot model after search. Left: vanilla DARTS (no sparsity enforced at all). Middle: enforcing sparsity over mixture of operations. Right: additionally enforcing sparsity over input nodes (here only a single input per node).

ing a loss function $\mathcal{L}_{\mathcal{T}_{train}}(w)$ given a single, large task $\mathcal{T} = (D_{train}, D_{test})$ with corresponding training and test data. In contrast, in few-shot learning, we are given a distribution over comparably small training tasks $\mathcal{T}^{train} \sim p^{train}(\mathcal{T})$ and test tasks $\mathcal{T}^{test} \sim p^{test}(\mathcal{T})$. We usually consider n-way, k-shot tasks, meaning each task is a classification problem with n classes (typically $n \in \{5, 20\}$) and k (typically $k \in \{1, 5\}$) training examples per class. In combination with meta-learning, the training tasks are used to meta-learn how to improve learning of new tasks from the test task distribution.

3.2. Gradient-based Meta-Learning of Neural Architectures

Similar to MAML's meta-learning strategy in the weight space [16], our goal is to meta-learn an architecture with corresponding weights which is able to quickly adapt to new tasks. In accordance with MAML we do so by minimizing a meta-objective

$$
\begin{aligned}
&\mathcal{L}_{meta}(w, \alpha, p^{train}, \Phi^k) \\
&= \sum_{\mathcal{T}_i \sim p^{train}} \mathcal{L}_{\mathcal{T}_i}\big(\Phi^k(w, \alpha, D^{\mathcal{T}_i}_{train}), D^{\mathcal{T}_i}_{test}\big) \\
&= \sum_{\mathcal{T}_i \sim p^{train}} \mathcal{L}_{\mathcal{T}_i}\big((w^*_{\mathcal{T}_i}, \alpha^*_{\mathcal{T}_i}), D^{\mathcal{T}_i}_{test}\big)
\end{aligned}
$$

(2)

with respect to a real-valued parametrization α of the neural network architecture and corresponding weights w. $\mathcal{T}_i = (D^{\mathcal{T}_i}_{train}, D^{\mathcal{T}_i}_{test})$ denotes a training task sampled from the training task distribution $p^{train}(\mathcal{T})$, $\mathcal{L}_{\mathcal{T}_i}$ the corresponding task loss, and $\Phi^k(w, \alpha, D^{\mathcal{T}_i}_{train})$ the task learning algorithm or simply *task-learner*, where k refers to k iterations of learning/ weight updates (e.g., by SGD). Prior work [16, 36, 24] considered a *fixed, predefined* architecture α_{fixed} and chose Φ^k to be an optimizer like SGD for the weights:

$$
w^* = w^k = \Phi^k(w, \alpha_{\text{fixed}}, D^{\mathcal{T}_i}_{train})
$$

with the one-step updates

$$
w^{j+1} = \Phi(w^j, D^{\mathcal{T}_i}_{train}) := w^j - \lambda_{task}\nabla_w\mathcal{L}_{\mathcal{T}}(w^j, D^{\mathcal{T}_i}_{train})
$$

and $w_0 = w$. In contrast, we choose Φ^k to be k steps of *gradient-based neural architecture search* inspired by DARTS [32] with weight learning rate λ_{task} and architecture learning rate ξ_{task}:

$$
\begin{aligned}
\begin{pmatrix} w^{j+1} \\ \alpha^{j+1} \end{pmatrix} &= \Phi(w^j, \alpha^j, D^{\mathcal{T}_i}_{train}) \\
&= \begin{pmatrix} w^j - \lambda_{task}\nabla_w\mathcal{L}_{\mathcal{T}}(w^j, \alpha^j, D^{\mathcal{T}_i}_{train}) \\ \alpha^j - \xi_{task}\nabla_\alpha\mathcal{L}_{\mathcal{T}}(w^j, \alpha^j, D^{\mathcal{T}_i}_{train}) \end{pmatrix}.
\end{aligned}
$$

(3)

Therefore, Φ^k does not only optimize task weights $w^*_{\mathcal{T}_i}$ but also optimizes task architecture $\alpha^*_{\mathcal{T}_i}$. Note that we use the same data set to update w^j and α^j (Equation 3) in contrast to Liu *et al.* [32] due to the limited amount of data in the few-shot setting not allowing to split into training and validation per task. Moreover, using the same data set also allows updating both sets of parameters with a single forward and backward pass, see Lian *et al.* [29]. As we use a real-valued parametrization of α and a gradient-based task optimizer, the meta-objective $\mathcal{L}_{meta}$ (Equation 2) is differentiable with respect to w and α. This means we can use any gradient-based meta-learning algorithm Ψ not only for w but also for the architecture α. As an example, one could use MAML [16] as a meta-learning algorithm, which runs SGD on the meta-objective, yielding meta-updates

$$
\begin{aligned}
\begin{pmatrix} w^{i+1}_{meta} \\ \alpha^{i+1}_{meta} \end{pmatrix} &= \Psi^{MAML}(\alpha^i_{meta}, w^i_{meta}, p^{train}, \Phi^k) \\
&= \begin{pmatrix} w^i_{meta} - \lambda_{meta}\nabla_w\mathcal{L}_{meta}(w^i_{meta}, \alpha^i_{meta}, p^{train}, \Phi^k) \\ \alpha^i_{meta} - \xi_{meta}\nabla_\alpha\mathcal{L}_{meta}(w^i_{meta}, \alpha^i_{meta}, p^{train}, \Phi^k) \end{pmatrix}
\end{aligned}
$$

Algorithm 1 METANAS: Meta-Learning of Neural Architectures

1: Input:
   distribution over tasks $p(\mathcal{T})$,
   task-learner $\Phi^k(w, a, D_{train}^{\mathcal{T}_i})$ # e.g. DARTS [32]
   meta-learner Ψ_w, Ψ_α # e.g. REPTILE [36]
2: Initialize w_{meta}, α_{meta}
3: **while** not converged **do**
4:   Sample tasks $\mathcal{T}_1, \dots, \mathcal{T}_n$ from $p(\mathcal{T})$
5:   **for all** $\mathcal{T}_i$ **do**
6:     $w_{\mathcal{T}_i}^*, \alpha_{\mathcal{T}_i}^* \leftarrow \Phi^k(w_{meta}, \alpha_{meta}, D_{train}^{\mathcal{T}_i})$
7:   **end for**
8:   $w_{meta} \leftarrow \Psi_w\big(w_{meta}, \{w_{\mathcal{T}_i}^*, \alpha_{\mathcal{T}_i}^*, \mathcal{T}_i\}_{i=1}^n\big)$
9:   $\alpha_{meta} \leftarrow \Psi_\alpha\big(\alpha_{meta}, \{w_{\mathcal{T}_i}^*, \alpha_{\mathcal{T}_i}^*, \mathcal{T}_i\}_{i=1}^n\big)$
10: **end while**
11: **return** w_{meta}, α_{meta}

Algorithm 2 Meta-Learning of Neural Architectures with DARTS and REPTILE

1: Input:
   distribution over tasks $p(\mathcal{T})$,
   task loss function $\mathcal{L}_{\mathcal{T}}$
2: Initialize w_{meta}, α_{meta}
3: **while** not converged **do**
4:   Sample tasks $\mathcal{T}_1, \dots, \mathcal{T}_n$ from $p(\mathcal{T})$
5:   **for all** $\mathcal{T}_i$ **do**
6:     $w_{\mathcal{T}_i} \leftarrow w_{meta}$
7:     $\alpha_{\mathcal{T}_i} \leftarrow \alpha_{meta}$
8:     **for** $j \leftarrow 1, \dots, k$ **do**
9:       $w_{\mathcal{T}_i} \leftarrow w_{\mathcal{T}_i} - \lambda_{task}\nabla_w\mathcal{L}_{\mathcal{T}}(w_{\mathcal{T}_i}, \alpha_{\mathcal{T}_i}, D_{train}^{\mathcal{T}_i})$
10:      $\alpha_{\mathcal{T}_i} \leftarrow \alpha_{\mathcal{T}_i} - \xi_{task}\nabla_\alpha\mathcal{L}_{\mathcal{T}}(w_{\mathcal{T}_i}, \alpha_{\mathcal{T}_i}, D_{train}^{\mathcal{T}_i})$
11:    **end for**
12: **end for**
13: $w_{meta} \leftarrow w_{meta} + \lambda_{meta} \sum_{\mathcal{T}_i}(w_{\mathcal{T}_i}^* - w_{meta}^i)$
14: $\alpha_{meta} \leftarrow \alpha_{meta} + \xi_{meta} \sum_{\mathcal{T}_i}(a_{\mathcal{T}_i}^* - \alpha_{meta}^i)$
15: **end while**
16: **return** w_{meta}, α_{meta}

---

or, as an alternative, one could use REPTILE [36], which simply computes the updates as

$$
\begin{pmatrix} w_{meta}^{i+1} \\ \alpha_{meta}^{i+1} \end{pmatrix} = \Psi^{REPTILE}(\alpha_{meta}^i, w_{meta}^i, p^{train}, \Phi^k)
$$
$$
= \begin{pmatrix} w_{meta}^i + \lambda_{meta} \sum_{\mathcal{T}_i}(w_{\mathcal{T}_i}^* - w_{meta}^i) \\ \alpha_{meta}^i + \xi_{meta} \sum_{\mathcal{T}_i}(a_{\mathcal{T}_i}^* - \alpha_{meta}^i) \end{pmatrix}. \tag{4}
$$

We chose REPTILE for all our experiments due to its conceptual simplicity and computational efficiency compared to MAML. Note that one could also use different meta-learning algorithms for w_{meta} and α_{meta}. However, we restrict ourselves to the same meta-learning algorithm for both for simplicity. We refer to Algorithm 1 for a generic template of our proposed framework for meta-learning neural architectures and to Algorithm 2 for a concrete implementation using DARTS as task learning and REPTILE as a meta-learning algorithm.

By incorporating a NAS algorithm directly into the meta-learning algorithm, we can search for architectures with a single run of the meta-learning algorithm, while prior work [24] required full meta-learning of hundreds of proposed architectures during the architecture search process. We highlight that Algorithm 1 is agnostic in that it can be combined with any gradient-based NAS method and any gradient-based meta-learning method.

4. Task-dependent Architecture Adaptation

Using a NAS algorithm as a task optimizer does not only allow directly incorporating architecture search into the meta-training loop, but it also allows an adaptation of the found meta-architecture after meta-learning to new tasks (i.,e., during meta-testing). That is, it allows in principle finding a *task-dependent architecture*, compare Algorithm

3. This is in contrast to prior work where the architecture is always fixed during meta-testing [16, 36]. Also prior work using NAS for meta-learning [24] searches for a single architecture that is then shared across all tasks.

Unfortunately, the task-dependent architecture obtained by the DARTS task optimizer is non-sparse, that is, the $\alpha_{\mathcal{T}_i}$ do not lead to mixture weights being strictly 0 or 1, compare Figure 2 (left) for an illustration. As discussed in Section 2, DARTS addresses this issue with a hard-pruning strategy at the end of the architecture search to obtain the final architecture from the one-shot model (line 8 in Algorithm 3). Since this hard-pruning deteriorates performance heavily (see Appendix A.1), the pruned architectures require retraining. This is particularly problematic in a few-shot learning setting as it requires meta re-training all the task-dependent architectures. This is the approach followed by [29], but it unfortunately increases the cost of a single task training during meta-testing from a few steps of the task optimizer to essentially a full meta-training run of MAML/REPTILE with a fixed architecture.

We now propose a method to remove the need for retraining by proposing two modification to DARTS that essentially re-parameterize the search space and substantially alleviate the drop in performance resulting from hard-pruning. This is achieved by enforcing the mixture weights $\hat{\alpha}$ of the mixed operations to slowly converge to 0 or 1 during task training while giving the operation weights time to adapt to this soft pruning.
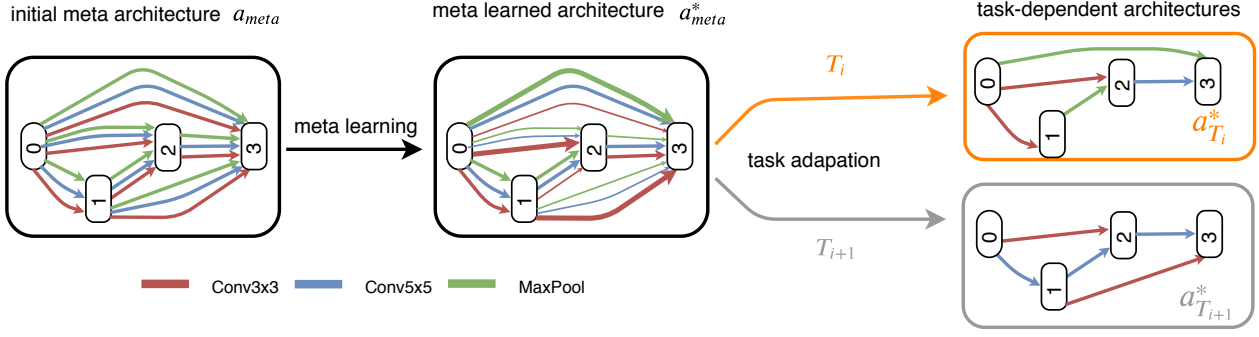
Figure 3: Conceptual illustration of the architectures at different stages of METANAS. Left: after initializing the one-shot model. Middle: meta-learned architecture. Right: architecture adapted to respective tasks based on meta-architecture. Colours of the edges (red, blue, green) denote different operations (Conv3x3, Conv5x5 and MaxPooling, respectively). Line-width of edges visualizes size of architectural weights α (i.e., large line-width correspond to large α values).

Algorithm 3 Learning of new task after meta-learning (i.e., meta-testing) with DARTS.

1: Input: new task $\mathcal{T} = (D_{train}, D_{test})$
   meta-learned architecture and weights α_{meta}, w_{meta}
2: $w_{\mathcal{T}} \leftarrow w_{meta}$
3: $\alpha_{\mathcal{T}} \leftarrow \alpha_{meta}$
4: **for** $j \leftarrow 1, \ldots, k$ **do**
5:     $w_{\mathcal{T}} \leftarrow w_{\mathcal{T}} - \lambda_{task} \nabla_w \mathcal{L}_{\mathcal{T}}(w_{\mathcal{T}}, \alpha_{\mathcal{T}}, D_{train})$
6:     $\alpha_{\mathcal{T}} \leftarrow \alpha_{\mathcal{T}} - \xi_{task} \nabla_\alpha \mathcal{L}_{\mathcal{T}}(w_{\mathcal{T}}, \alpha_{\mathcal{T}}, D_{train})$
7: **end for**
8: $\bar{\alpha}_{\mathcal{T}} \leftarrow \text{PRUNE}(\alpha_{\mathcal{T}})$
9: Evaluate D_{test} with $\bar{\alpha}_{\mathcal{T}}, w_{\mathcal{T}}$

4.1. Soft-Pruning of Mixture over Operations

The first modification sparsifies the mixture weights of the operations forming a mixed operation that transforms node i to node j. We achieve this by changing the normalization of the mixture weights $\hat{\alpha}^{i,j}$ from Equation 1 to become increasingly sparse, that is: more similar to a one-hot encoding for every i, j. To achieve this, we add a temperature τ_α that is annealed to 0 over the course of a task training:

$$\hat{\alpha}^{i,j}_{\tau_\alpha}(o) = \frac{\exp(\alpha^{i,j}_o / \tau_\alpha)}{\sum_{o' \in \mathcal{O}} \exp(\alpha^{i,j}_{o'} / \tau_\alpha)}. \quad (5)$$

A similar approach was proposed by [53] and [11] in the context of relaxing discrete distributions via the Gumbel distribution [23, 33]. Note that this results in (approximate) one-hot mixture weights in a single mixed operation (compare Figure 2 (middle) for an illustration); however, the sum over all $j - 1$ possible input nodes $0, \ldots, j - 1$ in Equation 1 is still non-sparse, meaning node j is still connected to all prior nodes. As DARTS selects only the top k ($k = 2$ in the default) input nodes, we need additionally also to spar-

sify across the possible input nodes (that is a soft-pruning of them) rather than just summing them up (as done in Equation 1), see Figure 2 (right) for an illustration.

4.2. Soft-Pruning of Mixture over Input Nodes

A natural choice to also sparsify the inputs would be to also introduce weights $\beta^{i,j}$ of the inputs and sparsify them the same way as the operations' weights by annealing a temperature τ_β to 0 over the course of a task training:

$$x^{(j)} = \sum_{i < j} \frac{\exp(\beta^{i,j} / \tau_\beta)}{\sum_{k < j} \exp(\beta^{k,j} / \tau_\beta)} MixedOp(x^{(i)}). \quad (6)$$

Unfortunately, this would results in selecting exactly one input rather than a predefined number of inputs (e.g., the literature default 2 [59, 53, 32]). Instead, we weight *every combination* of k inputs to allow an arbitrary number of inputs k:

$$x^{(j)} = \sum_{\mathbf{i} = \{i_1, \ldots, i_k\} \in \mathcal{I}} \frac{\exp(\beta^{\mathbf{i},j} / \tau_\beta)}{\sum_{\mathbf{k} \in \mathcal{I}} \exp(\beta^{\mathbf{k},j} / \tau_\beta)} \cdot \left(MixedOp(x^{(i_1)}) + \cdots + MixedOp(x^{(i_k)}) \right), \quad (7)$$

where $\mathcal{I} = \{\{i_1, \ldots, i_k\} | \{i_1, \ldots, i_k\} \subseteq \{0, \ldots, j - 1\}\}$ denotes the set of all combinations of inputs of size k. This introduces $\binom{j}{k}$ additional parameters per node, which is negligible for practical settings with $j \leq 5$. The input weights $\beta^{\mathbf{k},j}$ are optimized along with the operation's weights α. Note that we simply subsume α and β into α in Algorithm 3.

With these two modifications, we can now not only find task-dependent optimal weights (given meta-weights) but also find task-dependent architectures (given a meta-architecture) that can be hard pruned without notable drop in performance, and thus without retraining. We refer to Appendix A.1 for a comparison of the three different

		Omniglot		MiniImagenet	
Parameters	Architecture	1-shot, 20-way	5-shot, 20-way	1-shot, 5-way	5-shot, 5-way
	REPTILE	86.7 ± 0.38	97.4 ± 0.04	46.5 ± 0.33	63.3 ± 0.3
≈30k	AutoMeta	89.0 ± 0.52	96.7 ± 0.13	$\mathbf{49.8 \pm 0.55}$	$\mathbf{64.5 \pm 1.36}$
	METANAS	$\mathbf{92.2 \pm 0.04}$	$\mathbf{98.8 \pm 0.05}$	49.7 ± 0.4	62.1 ± 0.9
	REPTILE	90.0 ± 0.14	98.0 ± 0.06	48.0 ± 0.44	65.4 ± 0.31
≈100k	AutoMeta	$\mathbf{96.2 \pm 0.22}$	$\mathbf{99.2 \pm 0.08}$	50.1 ± 0.16	66.3 ± 0.4
	METANAS	$\mathbf{96.2 \pm 0.16}$	$\mathbf{99.2 \pm 0.07}$	$\mathbf{53.2 \pm 0.4}$	$\mathbf{67.8 \pm 0.7}$

Table 1: Results (mean $\pm$ standard deviation for 3 independent runs) on different data sets and different few-shot tasks. For all architecture, REPTILE was used as a meta-learning algorithm and all results were obtained using the same training pipeline to ensure a fair comparison. Accuracy in %.

pruning strategies discussed on a standard NAS setting on CIFAR-10 (single task). While in theory we can now enforce a one-hot encoding of the mixture operation as well as over the input nodes, we empirically found that it is sometimes helpful to not choose the minimal temperature too small but rather allow a few (usually not more than two) weights larger than 0 instead of hard forcing an one-hot encoding. At the end of each task learning, we then simply keep all operations and input nodes with corresponding weight $\hat{\alpha}$ larger than some threshold (e.g., $\hat{\alpha} \geq 0.01$), while all others are pruned.

5. Experiments

We evaluate our proposed method on the standard few-shot image recognition benchmarks Omniglot [25] and MiniImagenet (as proposed by [39]) in the n-way, k-shot setting (as proposed by [51]), meaning that a few-shot learning task is generated by random sampling n classes from either Omniglot or MiniImagenet and k examples for each of the n classes. We refer to [51] for more details.

5.1. Comparison under the same meta-learning algorithm.

We first compare against the architectures from the original REPTILE [36] paper and from AutoMeta [24] when training all models with the same meta-learning algorithm, namely REPTILE. This ensures a fair comparison and differences in performance can be clearly attributed to differences in the architecture. We re-train the architectures from REPTILE and AutoMeta with our own training pipeline for 30,000 meta epochs (which we found to be sufficient to approximately reproduce results from the REPTILE paper) to further ensure that all architectures are trained under identical conditions. A detailed description of the experimental setup including all hyperparameters can be found in Appendix A.2 in the supplementary material.

For our method, we consider the following search space based on DARTS and AutoMeta: we search for a normal and reduction cell (which is common practice in the NAS literature [59, 32, 53, 7, 54]). Both cells are com-

posed of three intermediate nodes (i.e., hidden states). The set of candidate operations is *MaxPool3x3, AvgPool3x3, SkipConnect, Conv1x5-5x1, Conv3x3, SepConv3x3, DilatedConv3x3*. Our models are composed of 4 cells, with the first and third cells being normal cells and the second and forth cell being reduction cells. The number of filters is constant throughout the whole network (rather than doubling the filters whenever the spatial resolution decreases); it is set so that the pruned models match the size (in terms of number of parameters) of the REPTILE and AutoMeta models to ensure a fair comparison. We consider models in the regime of 30,000 parameters as well as 100,000 parameters. Note that, in contrast to DARTS, we optimize the architectural parameters also on training data (rather than validation data) due to very limited amount of data in the few-shot setting not allowing a validation split per task.

The results are summarized in Table 1. In the 1-shot, 20-way Omniglot experiment, METANAS achieves superior performance in the case of small models, while in the case of large models METANAS is on-par with AutoMeta while both outperform REPTILE. On Omniglot 5-shot, 20-way, all methods perform similarly well with METANAS and AutoMeta having slight advantages over REPTILE. On MiniImagenet 1-shot, 5way, both AutoMeta and METANAS outperform REPTILE. In the 5-shot, 5-way setting, METANAS does also outperform AutoMeta in the case of larger models while being slightly worse for small models. In summary, METANAS nearly always outperforms the original REPTILE model while it is on-par or does slightly outperform AutoMeta in almost all cases. We highlight that METANAS achieves this while being 10x more efficient than AutoMeta; the AutoMeta authors report computational costs in the order of *100 GPU days* while METANAS was run for approximately *one week on a single GPU*. Moreover, METANAS finds task-specific architectures; see Figure 4 for the most common cells and Appendix 5 for two other commonly used (reduction) cells, with considerable differences in the operations and connectivity.
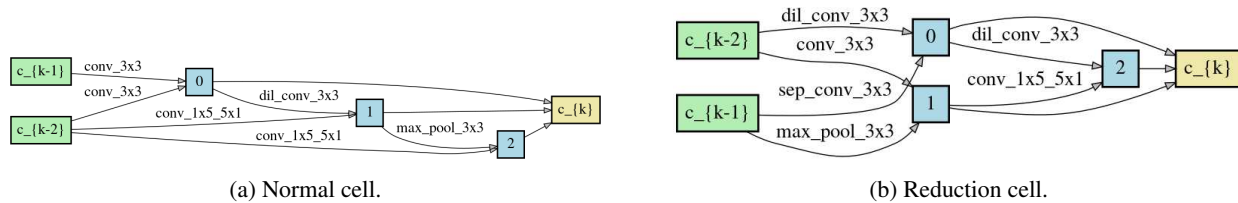
(a) Normal cell.　　　　　　　　(b) Reduction cell.

Figure 4: The most common normal and reduction cell found by METANAS that are used for the evaluation in Section 5.2.

5.2. Scaling up architectures and comparison to other meta-learning algorithms.

We now compare to other meta-learning algorithms in the fixed architecture setting; that is: we use METANAS not with task-dependent architectures but with a single fixed architecture extracted after running METANAS. For this, we extract the most common used task-dependent architecture (see Figure 4) and scale it up by using more channels and cells. We retrain the resulting architecture, which has approximately 1 million parameters, for more meta-epochs with stronger regularization (weight decay and DropPath[59]), which is common practice in the NAS literature [59, 40, 32, 53, 7]. Please refer to Appendix A.2 for details. Note that naively enlarging models for few-shot learning without regularization does not improve performance due to overfitting as reported by [39, 16].

Results are presented in Table 2. Again, METANAS improves over the standard REPTILE architecture and AutoMeta. Compared to other methods that meta-learn an initial set of parameters (first block), METANAS significantly outperforms all other methods on MiniImagenet and achieves new state-of-the-art performance. On Omniglot, METANAS is on-par with MAML++. As MAML++ outperforms REPTILE as a meta-learning algorithm, it is likely that using MAML++ in combination with METANAS would further improve our results. Also compared to other meta-learning approaches (second block), METANAS is on par or outperforms them while employing a significantly smaller architecture (1 million parameters for METANAS compared to more than 10 million for TADAM [37], LEO [42] and MetaOptNet [27]).

6. Conclusion

We have proposed METANAS, the first method which fully integrates gradient-based meta-learning with neural architecture search. METANAS allows meta-learning a neural architecture along with the weights and adapting it to *task-dependent architectures* based on few labeled datapoints and with only a few steps of gradient descent. We have also proposed an extension of DARTS [32] that reduces the performance drop incurred during hard-pruning,

		MiniImagenet		Omniglot
Method	# params	1-shot, 5-way	5-shot, 5-way	1-shot, 20 way
MAML[16]	30k	48.7 ± 1.8	63.1 ± 0.9	95.8 ± 0.03
MAML++[1]	-	52.2 ± 0.3	68.3 ± 0.4	$\mathbf{97.65 \pm 0.05}$
T-NAS++[29]	27k	54.1 ± 1.4	69.6 ± 0.9	-
REPTILE[36]	30k	50.00 ± 0.3	66.0 ± 0.6	89.43 ± 0.14
AutoMeta[24]	100k	57.6 ± 0.2	74.7 ± 0.2	-
METANAS *	1M	$\mathbf{61.7 \pm 0.3}$	$\mathbf{78.8 \pm 0.2}$	$\mathbf{97.74 \pm 0.08}$
TADAM [37]	12M	58.5 ± 0.3	76.7 ± 0.3	-
MetaOptNet [27][1]	12M	61.1 ± 0.6	77.4 ± 0.5	-
LEO [42]	36.5M	$\mathbf{61.8 \pm 0.1}$	77.6 ± 0.1	-

Table 2: Comparison to other meta-learning algorithm. The first block contains methods that, similar to ours, learn an initial set of parameters that is quickly adapted to new tasks. The second block contains other meta-learning methods. Here we list the numbers stated in other papers. METANAS * denotes the results of our proposed method after increasing model size, regularization and a longer meta-training period. Accuracy in %.

which might be of independent interest. Empirical results on standard few-shot learning benchmarks show the superiority with respect to simple CNNs mostly used in few-shot learning so far. METANAS is on-par or better than other methods applying NAS to few-shot learning while being significantly more efficient. After scaling the found architectures up, METANAS significantly improves the state-of-the-art on MiniImagenet, achieving 61.7% accuracy in the 1-shot, 5-way setting and 78.8% in the 5-shot, 5 way setting.

As our framework is agnostic with respect to the meta-learning algorithm as well as to the differentiable architecture search method, our empirical results can likely be improved by using more sophisticated meta-learning methods such as MAML++ [1] and more sophisticated differentiable architecture search methods such as ProxylessNAS [7]. In the future, we plan to extend our framework beyond few shot classification to other multi-task problems.

---

[1]We report results without label smoothing and without training on the validation set, as this is also not used in our work.

References

[1] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your MAML. In *International Conference on Learning Representations*, 2019. 2, 8

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017. 3

[3] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating Neural Architecture Search using Performance Prediction. In *NIPS Workshop on Meta-Learning*, 2017. 3

[4] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, 2018. 3

[5] Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018. 3

[6] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018. 3

[7] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. 1, 7, 8, 12

[8] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8713–8724. Curran Associates, Inc., 2018. 1

[9] Liu Chenxi, Chen Liang Chieh, Schroff Florian, Adam Hartwig, Hua Wei, Yuille Alan L., and Fei Fei Li. Autodeeplab: Hierarchical neural architecture search for semantic image segmentation. In *Conference on Computer Vision and Pattern Recognition*, 2019. 3

[10] Tristan Deleu, Tobias Würfl, Mandana Samiei, Joseph Paul Cohen, and Yoshua Bengio. Torchmeta: A Meta-Learning library for PyTorch, 2019. Available at: https://github.com/tristandeleu/pytorch-meta. 12

[11] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770, 2019. 6

[12] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Simple And Efficient Architecture Search for Convolutional Neural Networks. In *NIPS Workshop on Meta-Learning*, 2017. 3

[13] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019. 3

[14] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. 1, 2, 3

[15] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 3

[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. 1, 2, 3, 4, 5, 8

[17] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9516–9527. Curran Associates, Inc., 2018. 2

[18] Sebastian Flennerhag, Pablo Garcia Moreno, Neil Lawrence, and Andreas Damianou. Transferring knowledge across learning processes. In *International Conference on Learning Representations*, 2019. 2

[19] Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent. In *International Conference on Learning Representations*, 2020. 2

[20] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[21] Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to learn using gradient descent. In Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *Artificial Neural Networks — ICANN 2001*, pages 87–94, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. 2

[22] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book. 3

[23] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. 2017. 6

[24] Jaehong Kim, Youngduck Choi, Moonsu Cha, Jung Kwon Lee, Sangyeul Lee, Sungwan Kim, Yongseok Choi, and Jiwon Kim. Auto-meta: Automated gradient based meta learner search. *arXiv preprint*, 2018. 1, 2, 3, 4, 5, 7, 8

[25] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 7

[26] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017. 2

[27] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *CVPR*, 2019. 8

[28] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`. 3

[29] Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta learning. In *International Conference on Learning Representations*, 2020. 2, 3, 4, 5, 8

[30] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[31] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive Neural Architecture Search. In *European Conference on Computer Vision*, 2018. 3

[32] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 2, 3, 4, 5, 6, 7, 8, 12

[33] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017. 6

[34] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving Deep Neural Networks. In *arXiv:1703.00548*, Mar. 2017. 3

[35] Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[36] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint*, 2018. 1, 2, 4, 5, 7, 8, 12, 13

[37] Boris N. Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, 2018. 8

[38] Ramakanth Pasunuru and Mohit Bansal. Continual and multi-task architecture search. *CoRR*, abs/1906.05226, 2019. 2

[39] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *In International Conference on Learning Representations (ICLR)*, 2017. 2, 7, 8

[40] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Aging Evolution for Image Classifier Architecture Search. In *AAAI*, 2019. 1, 3, 8

[41] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2902–2911, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. 3

[42] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *ICLR*, 2019. 8

[43] Tonmoy Saikia, Yassine Marrakchi, Arber Zela, Frank Hutter, and Thomas Brox. Autodispnet: Improving disparity estimation with automl. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 1, 3, 12

[44] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4053–4061. Curran Associates, Inc., 2016. 3

[45] Jurgen Schmidhuber. Evolutionary principles in self-referential learning. on learning how to learn: The meta-meta-meta...-hook. Master's thesis, Technische Universitat Munchen, Germany, 1987. 2

[46] Christoph Schorn, Thomas Elsken, Sebastian Vogel, Armin Runge, Andre Guntoro, and Gerd Ascheid. Automated design of error-resilient and hardware-efficient deep neural networks. In *arXiv preprint*, 2019. 3

[47] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. *arXiv preprint*, 2017. 2

[48] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002. 3

[49] Sebastian Thrun and Lorien Pratt. Learning to learn. In *Springer Science+Business Media*, 1998. 2

[50] Joaquin Vanschoren. *Meta-Learning*, pages 35–61. Springer International Publishing, 2019. 2

[51] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3630–3638. Curran Associates, Inc., 2016. 2, 7

[52] Catherine Wong, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo. Transfer learning with neural automl. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8356–8365. Curran Associates, Inc., 2018. 3

[53] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. 3, 6, 7, 8, 12

[54] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020. 3, 7

[55] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. In *ICML 2018 Workshop on AutoML (AutoML 2018)*, 2018. 3

[56] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2423–2432, 2018. 3

[57] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7693–7702, Long Beach, California, USA, 2019. PMLR. 2

[58] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR) 2017 Conference Track*, 2017. 1, 3, 12

[59] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2018. 3, 6, 7, 8