Learning to Learn by Jointly Optimizing Neural Architecture and Weights

Yadong Ding[1] Yu Wu[2] Chengyue Huang[1] Siliang Tang[1*]
Yi Yang[1] Longhui Wei[3] Yueting Zhuang[1] Qi Tian[4]
[1]Zhejiang University [2]Princeton University
[3]University of Science and Technology of China [4]Huawei Cloud & AI
{dyadongcs, hcyue, siliang, yangyics, yzhuang}@zju.edu.cn , yuwu@princeton.edu
longhuiwei@pku.edu.cn, tian.qi1@huawei.com

Abstract

Meta-learning enables models to adapt to new environments rapidly with a few training examples. Current gradient-based meta-learning methods concentrate on finding good model-agnostic initialization (meta-weights) for learners. In this paper, we aim to obtain better meta-learners by co-optimizing the architecture and meta-weights simultaneously. Existing NAS-based meta-learning methods apply a two-stage strategy, i.e., first searching architectures and then re-training meta-weights on the searched architecture. However, this two-stage strategy would break the mutual impact of the architecture and meta-weights since they are optimized separately. Differently, we propose progressive connection consolidation, fixing the architecture layer by layer, in which the layer with the largest weight value would be fixed first. In this way, we can jointly search architectures and train the meta-weights on fixed layers. Besides, to improve the generalization performance of the searched meta-learner on all tasks, we propose a more effective rule for co-optimization, namely Connection-Adaptive Meta-learning (CAML). By searching only once, we can obtain both adaptive architecture and meta-weights for meta-learning. Extensive experiments show that our method achieves state-of-the-art performance with 3x less computational cost, revealing our method's effectiveness and efficiency.

1. Introduction

As a popular solution for the few-shot learning problem[1], meta-learning develops deep learning models with the ability to fit unseen tasks using only a few training examples [6, 30, 37]. Particularly, the gradient-based meta-learning methods like MAML [6] attempt to find a set of initialization of models' weights (*meta-weights*). The model with meta-weights can produce good generalization performance on an unseen task quickly with only a few gradient steps. In addition, to obtain the optimized meta-weights, it's also vital to find better architectures good at meta-learning. Unlike previous methods built on hand-crafted architectures, we aim to obtain better meta-learners by enriching architecture flexibility via Neural Architecture Search (NAS).

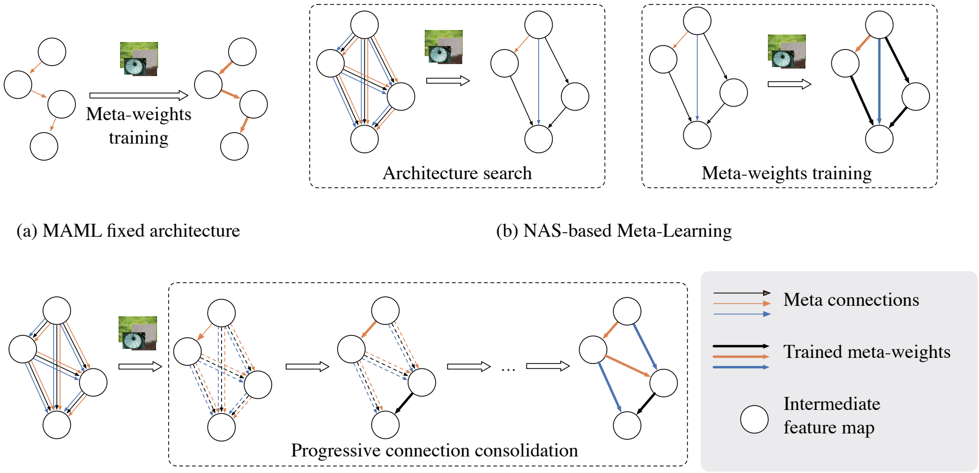
In this work, our target is to find optimal architecture and meta-weights for a meta-learner which can quickly adapt to new tasks with a few training samples. We represent the candidate operations (*e.g.*, *conv* and *pooling*) in each layer as *connections*. Each of them is weighted by an attention value over all candidate operations in the same layer, which is called *connection parameters*. Larger values mean more important operations/connections and we call each layer's adaptive connection as *meta-connections*. Thus the adaptive architecture is composed of meta-connections, and the training process can be regarded as a co-optimization problem of the connection parameters and the network weights.

There have been some recent works focusing on the exploration of architecture impact in meta-learning [10, 15]. However, most of these works either fall into the dilemma of breaking the *mutual impact* between the architecture and meta-weights or optimize the learner with a biased updating rule. First, both Auto-Meta [10] and Auto-MAML [15] apply a two-stage training strategy that obtains architectures and meta-weights separately, *i.e.*, first searching architectures and then retraining meta-weights using the searched architecture, as illustrated in Figure 1 (b). As mentioned in *the lottery ticket hypothesis* [7], sub-networks pruned from the supernet[2] cannot get optimized effectively unless they are initialized with the supernet's network weights. It inspires us that architectures and network weights have a mutual impact on each other. Therefore, in NAS-based meta-learning, we need to preserve the mutual impact between the architecture

[*] Siliang Tang is the corresponding author.

[1] A N-way, K-shot task denotes K samples from each class and N classes in few-shot learning.

[2] A supernet is a neural network whose layers consist of more than one candidate operation (*e.g.*, *convolution, pooling*). When searching finished, each layer is pruned, leaving one specific operation at most.



(a) MAML fixed architecture

(b) NAS-based Meta-Learning

(c) Our method

Figure 1. (a) MAML focuses on the **model-agnostic** meta-weights. (b) The current NAS-based meta-learning methods consist of two stages. The fixed architectures and their meta-weights are obtained separately, overlooking the mutual impact between them. (c) By co-optimizing the architecture and meta-weights, our method can simultaneously obtain adaptive architecture and meta-weights for all unseen tasks, requiring 3x less computational cost.

and meta-weights. Nevertheless, during the searching phase, Auto-Meta [10] and Auto-MAML [15] could only obtain the architecture with non-matched meta-weights. The matched meta-weights are acquired based on the searched architecture in the second stage (the re-training stage). Thus, the architecture and meta-weights of existing works are trained one by one (separately) instead of jointly optimized, breaking the mutual impact between the architecture and meta-weights. Second, to co-optimize the architecture parameters and network weights, Auto-MAML [15] proposed one simple solution called One-Propagation NAS-Based Meta-Learning (OPML) [5, 15], as illustrated in Figure 3. For simplicity, OPML treats the connection parameters and network weights equally and updates them by one backpropagation in every iteration. Nevertheless, due to the unequal learning rates, the meta-learner's real update direction is not parallel to the calculated meta-gradient, which may harm the generalization performance of meta-learner on all tasks.

To preserve the mutual impact between the architecture and meta-weights, we propose *progressive connection consolidation*, as shown in Figure 1 (c). During searching, we prune the supernet layer by layer, in which the layer with the largest connection weight value will be consolidated first. As the connections get fixed gradually, we can train the matched meta-weights on these consolidated connections. In return, the meta-weights would further affect the update of the other unfixed connections. In this way, we continuously preserve the mutual impact of the meta-connections and meta-weights during the entire training phase. Meanwhile, we remove the update of the pruned connections and weights and avoid retraining the derived architecture from scratch, saving 66% computational cost. To update the meta-learner in an un-biased way, we propose Connection-Adaptive Meta-Learning (CAML), as demonstrated in Figure 2.(b). By backpropagating twice and updating alternately, CAML can optimize both the connection parameters and network weights using the same update direction as the meta-gradients', respectively. Thus, CAML improves the searched meta-learner's generalization performance on all tasks, which is essential in meta-learning. Our contributions are summarized as follows:

- To address the two-stage strategy's separate optimization problem, we propose progressive connection consolidation to gradually prune the supernet during searching, preserving the mutual impact of the meta-connections and meta-weights.
- We propose a more effective method, namely Connection-Adaptive Meta-Learning (CAML), which can improve the generalization performance of the optimal architecture and meta-weights on all tasks.
- Extensive experiments show that our method achieves state-of-the-art performance on both FC100 and Mini-Imagenet datasets under various settings with 3x less computational cost, revealing the effectiveness and efficiency of our method.

2. Related work

2.1. Meta-learning

Meta-learning (learning to learn) [2, 6, 8, 13, 20, 22] methods learn from a series of learning tasks, enabling neural networks to adapt to new data and new tasks quickly. In recent years, meta-learning has proven effective in the few-shot classification task, which requires neural networks to solve new tasks given only a few training examples. Meta-learning approaches can be classified into three major categories: memory network [3, 25], metric learning [28, 31, 31] and gradient-based approaches [1, 6, 20].

In gradient-based approaches, an optimizer called meta-learner is learned to perform fast adaption on new tasks [9]. Instead of using the learned optimizer, model-agnostic meta-learning (MAML) [6] tries to find a set of parameters (meta-weights) for initializing the meta-learner. With a few steps of gradient descent, the meta learner can fast adapt to new tasks. However, previous methods focus on finding good model-agnostic initialization.

2.2. Neural architecture search

Neural architecture search (NAS) [4, 14, 16, 19, 29, 35, 38] aims to automatically design neural network architecture to reduce human experts' manual labour. The architectures searched by NAS approaches have surpassed hand-designed ones in many diverse tasks, such as image classification [19, 35, 39], semantic segmentation [17, 26], and object detection [32, 34]. Most NAS methods can be classified into three categories: based on evolutionary algorithms [18, 23, 24], based on reinforcement learning [39, 40] and gradient-based methods [19, 33, 36].

In gradient-based NAS methods like DARTS [19], the connection parameters and network weights can be optimized jointly based on gradient descent. Therefore, gradient-based NAS methods are capable of finishing searching within one GPU day. However, the existing NAS approaches merely target searching architectures for a single specific task. But while turning to multiple tasks or multiple datasets, they encounter troubles.

2.3. Meta-learning with neural architecture search

Recently, there have been some works combining NAS and meta-learning to obtain a better meta-learner [10, 15]. However, in every iteration of searching, Auto-Meta [10] needs to perform the entire meta-training process, while we only train the meta-learners once. As a result, Auto-Meta takes 112 GPU Days to converge, while our method only requires 0.7 GPU Day. More importantly, current methods separate the architecture searching and meta-weights training. They search architectures first and then re-train meta-weights based on searched architectures. Unfortunately, in this two-stage strategy, the meta-weights are overlooked during architecture searching, breaking the mutual impact of the architecture and meta-weights. In our method, both architecture and meta-weights can benefit each other and lead to better overall optimization.

Besides, some works concentrate on designing task-specific architectures. Based on Bayesian inference, BASE [27] is proposed to design task-dependent architectures for each meta-test task. MetaNAS [5] employs Reptile [20] as its backbone and utilizes a soft pruning strategy over all layers with the search progressing. T-NAS [15] attempts to learn a general meta-architecture through MAML [6]. Then both MetaNAS and T-NAS perform architecture adaptation for a new test task. Soft pruning does not prune the operations of slight importance. Thus MetaNAS still need to do one-shot pruning for the final architectures like T-NAS. However, these methods need to train every task-specific architecture from scratch, which is computationally expensive. Moreover, these task-specific methods also utilize the two-stage strategy, overlooking the mutual impact of the connections and meta-weights.

3. Approach

Before introducing our approach, we make a review of Model-Agnostic Meta-Learning (MAML) [6] and Differentiable Architecture Search (DARTS) [19], which will help us make a better understanding of our method. Then we introduce our the progressive connection consolidation in Section 3.3. and CAML in Section 3.4.

3.1. MAML

In MAML [6], the whole task dataset $\mathcal{D}$ is divided into three subsets, *i.e.*, meta-train $\mathcal{D}_{\text{meta-train}}$, meta-val $\mathcal{D}_{\text{meta-val}}$ and meta-test dataset $\mathcal{D}_{\text{meta-test}}$, respectively, as visualized in the supplementary material. Each of them consists of two tasks set, the support set $\{\mathcal{T}^{\text{s}}\}$ and the query set $\{\mathcal{T}^{\text{q}}\}$. In meta-train phase, MAML samples a set of tasks $\{\mathcal{T}\}$ from the task distribution $p_{\mathcal{T}}$ in $\mathcal{D}_{\text{meta-train}}$. Tasks sampled from $\{\mathcal{T}^{\text{s}}\}$ are employed for optimizing the inner-learner [19], while tasks sampled from $\{\mathcal{T}^{\text{q}}\}$ are used to optimize the meta-learner. The main goal of MAML is to find good initialized weights $\tilde{\theta}$ for the meta-learner, which can quickly adapt to new tasks drawn from $p_{\mathcal{T}}$. In the i-th meta-train task, the gradient-based learning rule for updating the inner-learner can be formulated as:

$$\theta_i^{m+1} = \theta_i^m - \beta_{\text{inner}} \nabla_{\theta_i^m} \mathcal{L}(f_{\theta_i^m}; \mathcal{T}_i^{\text{s}}), \tag{1}$$

where m represents the inner update step, and $\mathcal{T}_i^{\text{s}}$ is the i-th task sampled from $\{\mathcal{T}^{\text{s}}\}$. β_{inner} is the inner learning rate of weights. θ_i^0 is a copy of $\tilde{\theta}$. $f_{\theta_i^m}$ is the parameterized function with parameters θ_i^m, while $\mathcal{L}$ means the loss function. After M steps of gradient descent, tasks $\mathcal{T}_i^{\text{q}}$ sampled from $\{\mathcal{T}^{\text{q}}\}$ are used for updating the meta-learner by the following rule:

$$\tilde{\theta} = \tilde{\theta} - \beta_{\text{meta}} \nabla_{\tilde{\theta}} \sum_{\mathcal{T}_i^{\text{q}} \sim p(\mathcal{T})} \mathcal{L}(f_{\theta_i^M}; \mathcal{T}_i^{\text{q}}), \tag{2}$$

where β_{meta} is denoted as the outer (meta) learning rate of weights. After the meta-train phase, the model learns well-initialized weights, which help the meta-learner adapt to any specific task in $\mathcal{D}_{\text{meta-test}}$ within only a few steps of gradient descent optimization.

3.2. DARTS

To obtain a continuous architecture search space, DARTS [19] apply a softmax over all possible operation candidates. The softmax relaxes the categorical choice of one specific operation to a soft one. The output of each layer is the expectation of all the outputs of operations,

$$\bar{o}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\phi_o)}{\sum_{o' \in \mathcal{O}} \exp(\phi_{o'})} o(x), \tag{3}$$

where x is the input, $\mathcal{O}$ is the candidate operation set, and ϕ_o is the softmax attention on operation o. On the convergence of DARTS, only operations with the relatively largest attention values are preserved, while the others are pruned. There is a bi-level optimization problem where the connection parameters and the network weights need to be optimized jointly. DARTS solves the conflict by updating the connection parameters ϕ and weights θ alternately:

$$\begin{cases} \phi &= \phi - \alpha \nabla_\phi \mathcal{L}_{\text{val}}(\theta - \xi \nabla_\theta \mathcal{L}_{\text{train}}(\theta, \phi), \phi), \\ \theta &= \theta - \beta \nabla_\theta \mathcal{L}_{\text{train}}(\theta, \phi), \end{cases} \tag{4}$$

where $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$ are the loss function on training dataset and validation dataset. α and β are the learning rates of the connection parameters and the network weights, respectively. ξ is the inner optimization learning rate and is a proxy for obtaining ϕ^*, which is set to 0 in our work.

3.3. Progressive connection consolidation

To enrich architecture flexibility, we employ a supernet during the architecture searching, while our final meta-learner is a sub-network pruned from the supernet. Note that in our method, we represent the candidate operations of each layer as connections. Thus, the architecture searching is to learn each layer's adaptive connection, which we call *meta-connections*.

The lottery ticket hypothesis [7] reveals the mutual impact between the architectures and network weights. However, previous work like T-NAS [15] utilizes a two-stage strategy, *i.e.*, first search architectures and then retrain meta-weights based on the searched architectures. This two-stage training would break the interaction since the two targets are optimized separately. To preserve the mutual impact and build a better co-optimization, we propose progressive connection consolidation (PCC), pruning the supernet layer by layer during searching. We define *layer confidence* as follows:

Layer confidence. A layer e consists of all operations from the candidate operation set $\mathcal{O}$. Following DARTS [19], we use a *zero* operation in the candidate set to represent a lack of connection. ϕ_o^e are the related connection parameters for layer e. Thus, the layer confidence of layer e is defined as the maximum attention value on non-zero operations:

$$S_{\text{LC}}^e = \max_{o \in \mathcal{O}, o \neq zero} \frac{\exp(\phi_o^e)}{\sum_{o' \in \mathcal{O}} \exp(\phi_{o'}^e)}, \tag{5}$$

In our experiments, we apply layer confidence to determine each layer's importance. The process of fixing one connection can be disassembled into two steps. First, we compute the layer confidence S_{LC} for all layers. The layer with largest S_{LC} is selected. Second, for the selected layer, we only keep the operation with the largest weight value and remove others. The kept operation is called meta-connection. As the connections get pruned gradually, the meta-weights in the fixed connections would further affect the update of the other unfixed connections' searching. On the convergences of the meta-learner, we obtain an adaptive architecture and the corresponding meta-weights simultaneously. We argue that such a learner can learn knowledge from task distribution $p_{\mathcal{T}}$ more efficiently and effectively.

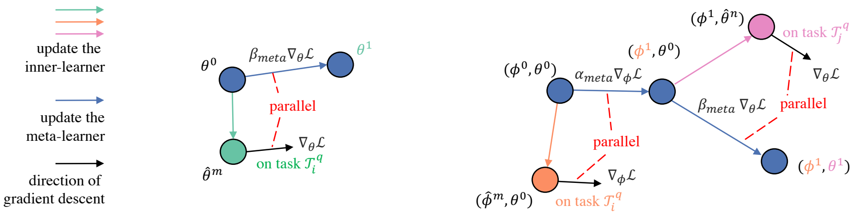
3.4. Connection-adaptive meta-learning

The main goal of our method is to find meta-learners with both adaptive architecture and meta-weights. However, as described in DARTS [19], there lies a bi-level optimization problem. We cannot optimize connection parameters ϕ solely without regard to the network weights θ.

As demonstrated in Figure 2, in MAML [6], they need to solve another bi-level optimization problem [19] over initial network weights and tasks. Therefore, we need to tackle a 4-level optimization problem in NAS-based meta-learning. Following MAML and DARTS, in each iteration, we use two different backpropagations for optimizing ϕ and θ, respectively. In other words, our CAML updates the meta-learners of ϕ and θ alternately. Since we jointly optimize the connection parameters and the weights, we have four learners, *i.e.*, inner-learner for ϕ, meta-learner for ϕ, inner-learner for θ, and meta-learner for θ. During the inner updates for connection parameters ϕ, the network weights θ is fixed. Following the common settings in NAS methods [12, 19], we split $\mathcal{D}_{\text{meta-train}}$ into $\mathcal{D}_{\text{meta-train-split-arch}}$ and $\mathcal{D}_{\text{meta-train-split-weights}}$ (as shown in the supplementary material), where $\mathcal{D}_{\text{meta-train-split-arch}}$ is used for updating the connection parameters ϕ, while the other is used for optimizing the network weights θ. Note that every split has both the support set and query set [31]. Given the i-th task $\mathcal{T}_i^{\text{split-arch,s}}$ sampled from the support set of $\mathcal{D}_{\text{meta-train-split-arch}}$, we optimize ϕ by,

$$\phi_i^{m+1} = \phi_i^m - \alpha_{\text{inner}} \nabla_{\phi_i^m} \mathcal{L}(f_{\phi_i^m, \bar{\theta}}; \mathcal{T}_i^{\text{split-arch,s}}), \tag{6}$$

where α_{inner} is the inner learning rate of the meta-connections and m is the inner update step. $f_{\phi,\theta}$ means the parameterized


update the inner-learner
update the meta-learner
direction of gradient descent
θ^0
$\beta_{meta}\nabla_\theta\mathcal{L}$
θ^1
parallel
$\nabla_\theta\mathcal{L}$
$\hat{\theta}^m$
on task $\mathcal{T}_i^q$
$(\phi^1, \hat{\theta}^n)$
on task $\mathcal{T}_j^q$
(ϕ^1, θ^0)
$\nabla_\theta\mathcal{L}$
(ϕ^0, θ^0)
$\alpha_{meta}\nabla_\phi\mathcal{L}$
$\beta_{meta}\ \nabla_\theta\mathcal{L}$
parallel
parallel
$\nabla_\phi\mathcal{L}$
(ϕ^1, θ^1)
$(\hat{\phi}^m, \theta^0)$
on task $\mathcal{T}_i^q$

(a) Model-Agnostic Meta-Learning

(b) Connection-Adaptive Meta-Learning

Figure 2. $\mathcal{L}$ is the loss function. $\hat{\theta}^m$, $\hat{\theta}^n$ and $\hat{\phi}^m$ are updated by the inner-learners, while θ^0, θ^1 and ϕ^1 are optimized by the meta-learners. (a).MAML [6] optimizes meta-weights using the same update direction as the meta-gradient's. (b).Our CAML optimizes both the connection parameters ϕ and network weights θ using the same update direction as the meta-gradients', respectively.

function with connections ϕ ($\phi_i^0 = \tilde{\phi}$) and network weights θ. After M inner update steps, the connections ϕ are updated to be well-adapted to the specific task. We optimize the meta-learner of ϕ according to the following formulation,

$$\tilde{\phi} = \tilde{\phi} - \alpha_{\text{meta}} \nabla_{\tilde{\phi}} \mathcal{L}(f_{\phi_i^M, \tilde{\theta}}; \mathcal{T}_i^{\text{split-arch,q}}), \tag{7}$$

where α_{meta} is the meta (outer) learning rate of ϕ. We use similar rules to optimize the inner-learner and the meta-learner of θ, as follows:

$$\theta_j^{m+1} = \theta_j^m - \beta_{\text{inner}} \nabla_{\theta_j^m} \mathcal{L}(f_{\tilde{\phi}, \theta_j^m}; \mathcal{T}_j^{\text{split-weights,s}}), \tag{8}$$

$$\tilde{\theta} = \tilde{\theta} - \beta_{\text{meta}} \nabla_{\tilde{\theta}} \mathcal{L}(f_{\tilde{\phi}, \theta_j^M}; \mathcal{T}_j^{\text{split-weights,q}}), \tag{9}$$

where β_{inner} and β_{meta} are the inner and meta learning rate of network weights θ ($\theta_j^0 = \tilde{\theta}$). $\mathcal{T}_j^{\text{split-weights,q}}$ and $\mathcal{T}_j^{\text{split-weights,s}}$ are tasks from $\mathcal{D}_{\text{meta-train-split-weights}}$. On the convergence of the meta learners of ϕ and θ, we obtain an adaptive architecture ϕ^* and the meta-weights θ^*. We simplify our method by two groups of bi-level optimization as approximation. The complete algorithm of our CAML is described in Alg.1. Recently, T-NAS [15] and MetaNAS [5] have proposed a group updating rules named One-Propagation NAS-Based Meta-Learning (OPML),

$$\begin{cases} [\phi_i^{m+1}; \theta_i^{m+1}] & = [\phi_i^m; \theta_i^m] - \eta_{\text{inner}} \nabla_{[\phi_i^m, \theta_i^m]} \mathcal{L}(f; \mathcal{T}_i^{\text{s}}), \\ \left[\tilde{\phi}; \tilde{\theta}_i\right] & = \left[\tilde{\phi}; \tilde{\theta}_i\right] - \eta_{\text{meta}} \nabla_{[\phi_i^M, \theta_i^M]} \mathcal{L}(f; \mathcal{T}_i^{\text{q}}), \end{cases} \tag{10}$$

where $\eta_{\text{inner}} = [\alpha_{\text{inner}}; \beta_{\text{inner}}]$ and $\eta_{\text{meta}} = [\alpha_{\text{meta}}; \beta_{\text{meta}}]$. f denotes the parameterized function, and $\mathcal{T}_i^{\text{s}}$ and $\mathcal{T}_i^{\text{q}}$ are sampled from $\mathcal{D}_{\text{meta-train}}$. In other words, they treat connection parameters and network weights equally and update them in one propagation, as shown in Figure 3. We also conducted our experiments based on OPML, and the quantitative comparison can be found in Table 3.

Algorithm 1: CAML

```
Input: Meta-train dataset split-arch $\mathcal{D}_{\text{meta-train-split-arch}}$
Input: Meta-train dataset split-weights
       $\mathcal{D}_{\text{meta-train-split-weights}}$.
Input: learning rate $\alpha_{\text{inner}}$, $\alpha_{\text{meta}}$, $\beta_{\text{inner}}$, $\beta_{\text{meta}}$.
Randomly initialize network weights $\theta$ and
  connection parameters $\phi$.
while not terminated do
    Sample batch of tasks { $\mathcal{T}^{\text{split-arch}}$ } from
      $\mathcal{D}_{\text{meta-train-split-arch}}$;
    for $\mathcal{T}_i^{split\text{-}arch} \in \{\mathcal{T}^{split\text{-}arch}\}$ do
        Get datapoints $\mathcal{T}_i^{\text{split-arch,s}}$ from support set.
        Update architecture parameters $\phi_i^m$ with
          Equation 6 for M steps.
        Get datapoints $\mathcal{T}_i^{\text{split-arch,q}}$ from query set for
          the meta-learner of $\phi$.
    end
    Update $\tilde{\phi}$ with Equation 7 for one step.
    Sample batch of tasks { $\mathcal{T}^{\text{split-weights}}$ } from
      $\mathcal{D}_{\text{meta-train-split-weights}}$;
    for $\mathcal{T}_j^{split\text{-}weights} \in \{\mathcal{T}^{split\text{-}weights}\}$ do
        Get datapoints $\mathcal{T}_j^{\text{split-weights,s}}$ from support set.
        Update network weights $\theta_i^m$ with Equation 8
          for M steps.
        Get datapoints $\mathcal{T}_j^{\text{split-weights,q}}$ from query set
          for the meta-learner of $\theta$.
    end
    Update $\tilde{\theta}$ with Equation 9 for one step.
    if pruning required in this iteration then
        Prune the network architecture and weights.
    end
end
```

4. Experiments

To verify the effectiveness of our approach, we conduct the experiments under the settings of few-shot learning on some popular datasets, *e.g.*, Omniglot [11], FC100 [21] and Mini-Imagenet [22]. Our experiments consist of architecture search and evaluation. We search for a meta-learner that has both the adaptive architecture and the meta-weights during the training stage. Then we evaluate the searched meta-

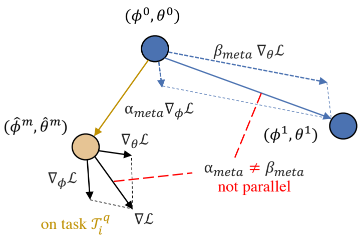

(ϕ^0, θ^0)
$\beta_{meta} \nabla_\theta \mathcal{L}$
$\alpha_{meta} \nabla_\phi \mathcal{L}$
$(\hat{\phi}^m, \hat{\theta}^m)$
(ϕ^1, θ^1)
$\nabla_\theta \mathcal{L}$
$\nabla_\phi \mathcal{L}$
$\alpha_{meta} \neq \beta_{meta}$
not parallel
on task $\mathcal{T}_i^q$
$\nabla \mathcal{L}$

Figure 3. In previous works (*e.g.*, T-NAS [15] and MetaNAS [5]), connection parameters and network weights are treated equally and updated by one backpropagation. Due to the unequal learning rates, the meta-learner's update direction is not parallel to the meta-gradient.

learner. At last, we do some ablation studies to demonstrate the effectiveness of our CAML and PCC.

4.1. Architecture search

We apply the basic searching settings in DARTS [19] to CAML. A cell [40] represented as a directed acyclic graph consists of an ordered sequence of computational nodes. We search for two cells composed of normal and reduction cells for generalization and efficiency. Then we stack two cells to build the whole network architecture. Therefore, the architecture ϕ is determined by $\{ \phi_{normal}, \phi_{reduce} \}$.

Candidate operation set. As for the candidate operation set, we use the same set as DARTS [19], which contains 8 kinds of operations: (1) *zero*, (2) *identity*, (3) *3*3 max pooling*, (4) *3*3 average pooling*, (5) *3*3 depth-wise separate conv*, (6) *3*3 dilated depth-wise separate conv*, (7) *5*5 depth-wise separate conv*, (8) *5*5 dilated depth-wise separate conv*. Other detailed searching settings and searched architectures are summarized in the supplementary material.

4.2. Evaluation on few-shot learning datasets

After the searching phase, a meta-learner with both adaptive architecture and corresponding meta-weights is obtained. We train the searched meta-learner for 100 epochs with 1200 independent tasks for each epoch during the evaluation. Note that different from DARTS [19], we train the searched architecture without any modification (*e.g.*, channels and architecture). We employ the Adam optimizer (cosine decay) with meta learning rate $\beta_{meta} = 0.001$ for the meta update. A vanilla SGD with inner learning rate $\beta_{inner} = 0.01$ is used for optimizing the inner-learner. We also report the performance of models by training the adaptive architecture from randomly initialized weights. All results comes from three different experiments with ± 1 std as error bars.

The experiments results on Mini-Imagenet and FC100 are represented in Table 1 The experiments results on Omniglot can be found in the supplementary material. On all datasets, our method achieves the best performance with less computational cost. CAML outperforms the baseline Auto-MAML by 4.0% (68.1 % versus 64.1 %) with fewer parameters (24.2K versus 26.1K), verifying the advantages of our method. Moreover, our method can save at least 66 % search cost compared to other state-of-the-art NAS-based methods. Thus, we finally obtain a meta-learner with adaptive architecture and meta-weights by co-optimizing connection parameters and network weights simultaneously.

We also make a comparison with other task-specific methods (like BASE [27], T-NAS [15], and MetaNAS [5]), as shown in Table 2. Compared to those task-dependent methods, our CAML can achieve comparable performance with fewer parameters. T-NAS [15] utilizes the two stage-strategy for every meta-test task to obtain a higher accuracy of 52.8%, and also reports a 215x more search cost.

4.3. Ablation studies

Contribution of CAML and PCC. We evaluate the contribution made by two components of our methods, namely CAML and PCC. Results are shown in Table 3. Progressive connection consolidation (PCC) plays a vital role in both one-propagation NAS-based meta-learning and CAML, which helps to find meta-learners with higher performance. PCC strengthens the co-optimization and mutual interaction between the architecture and the network weights. Thus, the searched weights show more significant potential than a random initialization on derived architectures with PCC. In CAML without PCC, we perform one-shot pruning at the end of searching. Also, CAML achieves better performance than OPML from two initialization conditions, demonstrating the effectiveness of our methods. Besides, CAML can cooperate well with progressive connection consolidation to provide further improvement.

CAML *versus* OPML. In existing works (*e.g.*, T-NAS [15]), the connection parameters and network weights are treated equally. Thus ϕ and θ are optimized by backpropagating once. We call the updating rules as one propagation NAS-based meta-learning (OPML), as described in Section 3.4. As shown in Table 3, though OPML can cooperate well with PCC, it obtains a lower accuracy compared to CAML in experiments. A potential reason for the performance improvement of our CAML might be the parallel optimization direction of learners. In MAML [6], they design the update direction of meta-gradient to update the meta-learner, as shown in Figure 2.(a). The parallel update direction produces the meta-learner's good generalization performance on all new tasks. But in OPML, since learning rates of θ and ϕ are usually unequal, the meta-learner's composite update direction is not parallel to the meta-gradient, as illustrated in Figure 3. Analogous to MAML, our method would lead to the same update direction as the meta-gradient, which helps find better meta-learners.

Comparison of different search space. MetaNAS [5] considers a different set of operations in its search space (which is named as **S1**) so the results are not directly comparable. To

Dataset	Method	Params (K)	FLOPS (M)	Search cost (GPU days)	Accuracy (%)		
					1-shot	5-shot	10-shot
Mini-Imagenet	Auto-Meta [10]	28.0	-	112	49.6 ± 0.2	65.1 ± 0.2	-
	Auto-MAML [15]	26.1	27.2	2	51.2 ± 1.8	64.1 ± 1.1	-
	Ours	**24.2**	**15.0**	**0.7**	**52.2 ± 0.4**	**68.1 ± 0.3**	-
FC100	Auto-MAML [15]	26.1	3.9	2	38.8 ± 1.8	52.2 ± 1.2	57.5 ± 0.8
	Ours	**18.4**	3.9	**0.7**	**39.2 ± 0.4**	**53.6 ± 0.2**	**57.7 ± 0.4**

Table 1. Comparison with NAS-based methods on Mini-Imagenet and FC100 for 5-way classification accuracy.

Method	Params (K)	Search Cost (GPU Days)	Accuracy (%)	
			1-shot	5-shot
BASE (Softmax) [27]	1200	-	-	65.4 ± 0.7
BASE (Gumbel) [27]	1200	-	-	66.2 ± 0.7
MetaNAS [5]	30.0	7	49.7 ± 0.4	62.1 ± 0.9
T-NAS [15]	26.5	150	**52.8 ± 1.4**	67.9 ± 0.9
Ours	**24.2**	**0.7**	52.2 ± 0.4	**68.1 ± 0.3**

Table 2. Comparison with task-specific NAS-based methods on Mini-Imagenet for 5-way accuracy.

Updating rules	PCC	Params	Train from scratch	Train from kept-weights
OPML	✗	51.3K	59.0 ± 0.3	59.1 ± 0.6
OPML	✓	44.9K	61.0 ± 0.4	64.2 ± 0.2
CAML	✗	20.0K	62.6 ± 0.1	62.8 ± 0.4
CAML	✓	24.2K	**67.4 ± 0.5**	**68.1 ± 0.3**

Table 3. Average 5-way, 5-shot accuracy on Mini-Imagenet. OPML means One-Propagation NAS-Based Meta-Learning, which is mentioned in Section 3.4 and updates ϕ and θ in one backpropagation. Architectures derived without PCC means that we only do one-shot pruning at the end of searching like previous works [5, 15]. Note that the supernet without pruning achieves an accuracy of 57.6 ± 1.2% with 220.0K parameters.

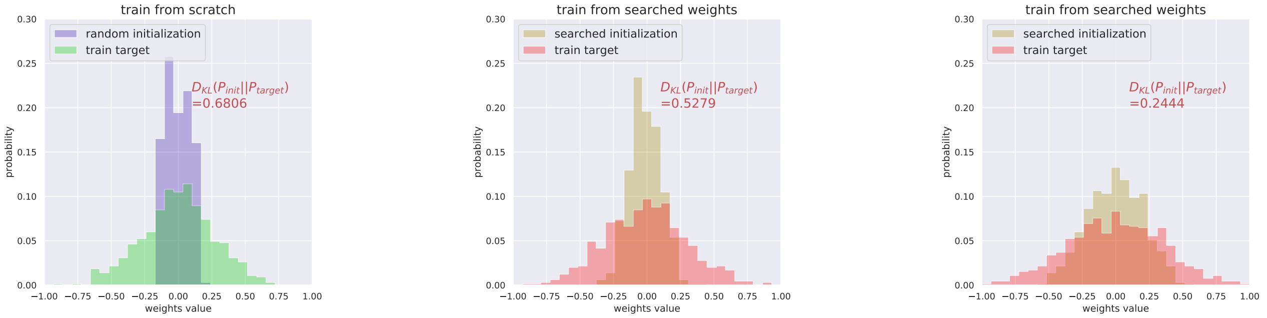
Search Space	Method	Params (K)	GPU Days	Accuracy (%)	
				1-shot	5-shot
S1	MetaNAS [5]	≈ 30	7	49.7 ± 0.4	62.1± 0.9
S1	Ours	16.8	0.7	50.4 ± 0.4	65.4± 0.1

Table 4. Comparison of average 5-way accuracy on Mini-Imagenet. In the search space of S1, our method can also achieve better performance with fewer parameters and less search cost.

better illustrate the effectiveness of our method, we conduct experiments that evaluate our method using the search space of **S1** on Mini-Imagenet. The experimental results are shown in Table 4. Clearly, our method can also achieve better performance with fewer parameters and less search cost using the search space of **S1**, demonstrating the effectiveness of our approach.

Train from kept meta-weights *versus* Train from scratch. We propose progressive connection consolidation (PCC) to fix the architecture gradually during the searching phase. To validate the kept network weights' contribution, we compare our model with the meta-learner trained from scratch (a random initialization). We also compare the model with the hard-pruning criterion [19] instead of our PCC. We train the hard-pruned architectures from a random initialization and the kept network weights for evaluation. Results are summarized in Figure 5. Our method learns knowledge from task distribution $p_{\mathcal{T}}$ more efficiently and effectively from the kept initialization compared to the random initialization. Besides, without our PCC, keeping weights does not perform better than random weights. It indicates that our PCC helps to enhance the mutual interaction between the architectures and meta-weights. To better validate our motivation, we sample the first layer of the models and show the distribution in Figure 4. The distribution of our searched meta-weights is closer to the optimization target, demonstrating the effectiveness of the co-optimization. Figure 4 (b) also shows the previous two-stage training strategy leads to a weak meta-learner, whose weights distribution is far from the training target.

Comparison of different pruning strategies. To prove our layer confidence-based pruning strategy's effectiveness, we also prune the supernet with fixed orders like forwarding sequence or backward. In addition, we also experimented with two different pruning strategies, named variance-based strategy and entropy-based strategy. The variance-based strategy picks the layer with the largest variance of its operations' architecture parameters to prune the supernet gradually, while the entropy-based strategy chooses the layer with the smallest entropy of its architecture parameters. The results are summarized in Table 5. Clearly, layer confidence-based pruning strategy in our PCC could help us find better adaptive architectures, achieving higher performance with fewer parameters. Besides, we could also observe perfor-



(a) Without PCC, train from scratch.

(b) Without PCC, train from searched weights.

(c) With PCC, Train from searched weights.

Figure 4. The network weights distribution of the first convolution layer during the evaluation. Without PCC, the searched weights are not significantly closer to the training target than random initialization. PCC narrows the distribution gap between the searched weights and the training target. Note that the train targets differ since they are derived from well-trained meta-learners with different architecture and initialization.

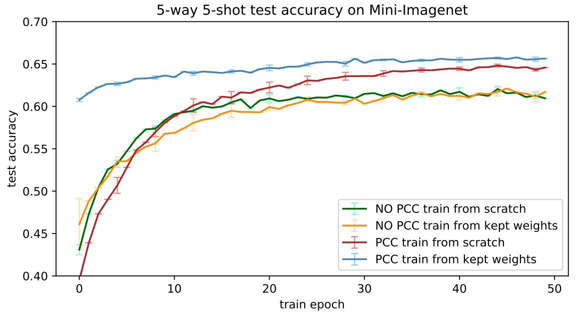


Figure 5. 5-shot, 5-way meta-test accuracy on Mini-Imagenet during the evaluation. The searched weights can cooperate better with the final architecture than a random initialization with PCC.

Method	Accuracy (%)	
	1-shot	5-shot
Ours + MAML [6]	52.2 ± 0.4	68.1± 0.3
Ours + Reptile [20]	51.6 ± 0.3	68.5± 0.3
Ours + MAML++ [1]	53.4 ± 0.3	69.1± 0.5

Table 6. Comparison of average 5-way accuracy on Mini-Imagenet by different meta-learning methods.

mance improvement by taking searched network weights as initialization, which proves our methods' effectiveness.

Comparison of different meta-learning methods. To further evaluate the impact of NAS for meta-learning, we apply CAML with other meta-learning methods like Reptile [20] and MAML++ [1]. The results are presented in Table 6. As shown in the table, our method can cooperate well with other meta-learning methods (e.g., Reptile [20]). Several techniques in MAML++ [1] can be directly employed in our method, further promoting the performance.

Sequence	Params (K)	Train from scratch	Train from searched-weights
Forward	34.0	61.0 ± 1.0	66.0 ± 0.1
Backward	27.7	63.6 ± 0.6	64.5 ± 0.2
Entroy-based	25.1	66.7 ± 1.0	67.7 ± 0.1
Variance-based	26.8	66.9 ± 0.2	67.8 ± 0.1
S_{LC} based	24.2	67.4 ± 0.1	68.1 ± 0.2

Table 5. Average 5-way, 5-shot accuracy on Mini-Imagenet by five pruning strategies of CAML. S_{LC} means the layer confidence. Clearly, among all five orders, S_{LC}-based strategy outperforms the other pruning strategies. Thus, in PCC, we prune the layers of the supernet in descending order of S_{LC}.

5. Conclusion

In this work, we focus on the exploration of the architecture impact in meta-learning. We target to find a meta learner with both the adaptive architecture and the meta-weights that can perform well on multiple similar tasks. The current two-stage solutions are inefficient and ignore the co-optimization of the architecture and meta-weights. To tackle the existing problems, we propose a novel *Progressive Connection Consolidation* (PCC). By fixing the architecture layer by layer during searching, PCC preserves the mutual impact between the architecture and meta-weights, leading to better overall optimization. Besides, we propose CAML to update the architecture parameters and network weights simultaneously by two different backpropagations in one iteration, improving the generalization performance of the searched meta-learner on all tasks. Extensive experiments show that our CAML and progressive connection consolidation are both helpful to a meta-learner's success. Our method achieves state-of-the-art performance on all few-shot datasets with 3x less computational cost.

Acknowledgment

This work has been supported in part by National Key Research and Development Program of China (2018AAA0101900), Zhejiang NSF (LR21F020004), Key Research and Development Program of Zhejiang Province, China (No. 2021C01013), Chinese Knowledge Center of Engineering Science and Technology (CKCEST).

References

[1] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your MAML. In *International Conference on Learning Representations*, 2019. 3, 8

[2] David Brüggemann, Menelaos Kanakis, Anton Obukhov, Stamatios Georgoulis, and Luc Van Gool. Exploring relational context for multi-task dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15869–15878, 2021. 2

[3] Qi Cai, Yingwei Pan, Ting Yao, Chenggang Yan, and Tao Mei. Memory matching networks for one-shot image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4080–4088, 2018. 3

[4] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Dr{nas}: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021. 3

[5] Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12365–12375, 2020. 2, 3, 5, 6, 7

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1126–1135, 2017. 1, 2, 3, 4, 5, 6, 8

[7] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 1, 4

[8] Fred X Han, Di Niu, Haolan Chen, Weidong Guo, Shengli Yan, and Bowei Long. Meta-learning for query conceptualization at web scale. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3064–3073, 2020. 2

[9] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001. 3

[10] Jaehong Kim, Sangyeul Lee, Sungwan Kim, Moonsu Cha, Jung Kwon Lee, Youngduck Choi, Yongseok Choi, Dong-Yeon Cho, and Jiwon Kim. Auto-meta: Automated gradient based meta learner search. *arXiv preprint arXiv:1806.06927*, 2018. 1, 2, 3, 7

[11] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the annual meeting of the cognitive science society*, volume 33, 2011. 5

[12] Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1620–1630, 2020. 4

[13] Juncheng Li, Xin Wang, Siliang Tang, Haizhou Shi, Fei Wu, Yueting Zhuang, and William Yang Wang. Unsupervised reinforcement learning of transferable meta-skills for embodied navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12123–12132, 2020. 2

[14] Ting Li, Junbo Zhang, Kainan Bao, Yuxuan Liang, Yexin Li, and Yu Zheng. Autost: Efficient neural architecture search for spatio-temporal prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 794–802, 2020. 3

[15] 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. 1, 2, 3, 4, 5, 6, 7

[16] Bill Yuchen Lin, Ying Sheng, Nguyen Vo, and Sandeep Tata. Freedom: A transferable neural architecture for structured information extraction on web documents. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1092–1102, 2020. 3

[17] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019. 3

[18] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017. 3

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

[20] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018. 2, 3, 8

[21] Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pages 721–731, 2018. 5

[22] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017. 2, 5

[23] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 3

[24] 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 *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017. 3

[25] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016. 3

[26] Albert Shaw, Daniel Hunter, Forrest Landola, and Sammy Sidhu. Squeezenas: Fast neural architecture search for faster

semantic segmentation. In *Proceedings of the IEEE international conference on computer vision workshops*, 2019. 3

[27] Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *Advances in Neural Information Processing Systems*, pages 11227–11237, 2019. 3, 6, 7

[28] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017. 3

[29] Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian, and Xia Hu. Towards automated neural interaction discovery for click-through rate prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 945–955, 2020. 3

[30] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 403–412, 2019. 1

[31] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016. 3, 4

[32] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11943–11951, 2020. 3

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

[34] Hang Xu, Lewei Yao, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6649–6658, 2019. 3

[35] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. 3

[36] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. 3

[37] Ruixiang Zhang, Tong Che, Zoubin Ghahramani, Yoshua Bengio, and Yangqiu Song. Metagan: An adversarial approach to few-shot learning. In *Advances in Neural Information Processing Systems*, pages 2365–2374, 2018. 1

[38] Xuanyang Zhang, Pengfei Hou, Xiangyu Zhang, and Jian Sun. Neural architecture search with random labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10907–10916, 2021. 3

[39] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 3

[40] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 3, 6