CONNECTION-ADAPTIVE META-LEARNING

Anonymous authors
Paper under double-blind review

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 initialization (meta-weights) for learners, but ignore the impact of neural architectures. In this paper, we aim to obtain better meta-learners by co-optimizing the architecture and meta-weights simultaneously. Existing NAS-based methods apply a two-stage strategy, *i.e.*, first searching architectures and then re-training meta-weights for the searched architecture. However, this two-stage strategy would lead to a suboptimal meta-learner, since the meta-weights are overlooked during searching architectures for meta-learning. Differently, we propose a more efficient and effective method for meta-learning, namely Connection-Adaptive Meta-learning (CAML), which jointly searches architectures and trains the meta-weights on consolidated connections. During searching, we consolidate the architecture connections layer by layer, in which the layer with the largest weight value would be fixed first. With searching only once, our CAML is able to obtain both adaptive architecture and meta-weights for meta-learning. Extensive experiments show that CAML achieves state-of-the-art performance with 130x 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 (Finn et al., 2017; Zhang et al., 2018; Sun et al., 2019). Particularly, the gradient-based meta-learning methods like MAML (Finn et al., 2017) attempt to find a set of initialization of model weights (*meta-weights*), which is capable of adapting to new tasks quickly with only a few update steps. In addition, to obtaining optimized meta-weights, it is also vital to find better architectures that are good at meta-learning. Different from 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, we propose Connection-Adaptive Meta-Learning (CAML), as demonstrated in Fig.1. CAML desires to find both optimal architectures and meta-weights for a meta-learner, which adapts to new tasks better. 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. 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.

Our CAML optimizes connection parameters and network weights simultaneously. During architecture searching, we train a supernet[2] to induce architecture flexibility, while our final goal is an optimal sub-network pruned from the supernet. To narrow the optimization gap between the supernet and the sub-network, we propose *progressive connection consolidation*. During searching, we prune the supernet layer by layer, in which the layer with the largest connection weight value will be pruned first. As the connections get fixed gradually, we can also train the related meta-weights on these consolidated connections. In return, the meta-weights would further affect the update of the

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

[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, left one specific operation at most.

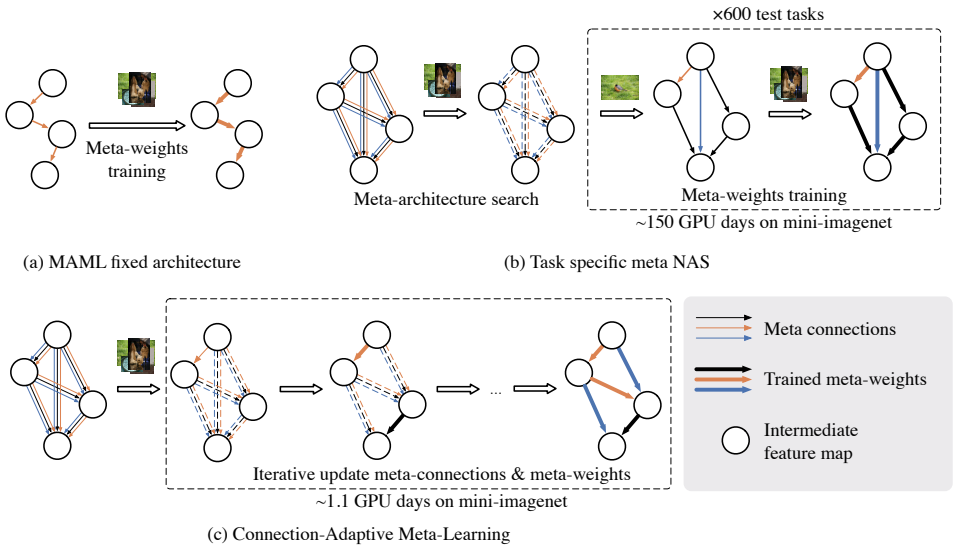


Figure 1: (a) MAML focus on the meta-weights, but ignores the architecture impact. (b) The current nas-based meta-learning methods consists of two stages. The task-specific architectures and their meta-weights are obtained separately. It overlooks meta-weights during searching architectures for meta-learning. In addition, it is computationally intensive to retrain each architecture. (c) By co-optimizing the architecture and the network weights, CAML can obtain the adaptive architecture and the meta-weights simultaneously for all unseen tasks, requiring 130x less computational cost.

other unfixed connections. In this way, we strengthen the co-optimization of the meta-connections and the meta-weights in the sub-network.

There have been some recent works focusing on the exploration of architecture impact in meta-learning (Kim et al., 2018; Shaw et al., 2019; Lian et al., 2020; Elsken et al., 2020). They either apply NAS methods to sample fixed architectures for meta-training (Kim et al., 2018), or perform a task-specific search on each meta-test task (Elsken et al., 2020; Shaw et al., 2019; Lian et al., 2020). However, these methods need to perform the whole meta-training process repeatedly during searching or train every task-specific architecture from scratch during the evaluation, which requires hundreds of GPU days. Moreover, these methods use a two-stage training strategy to obtain architectures and their meta-weights separately, *i.e.*, first searching architectures and then re-training meta-weights for the searched architecture. However, this would lead to a suboptimal meta-learner, since the meta-weights are totally overlooked during searching architectures for meta-learning. As shown in *the lottery ticket hypothesis* (Frankle & Carbin, 2019), sub-networks pruned from the supernet cannot get optimized effectively unless they are initialized with the supernet's network weights. It reveals that architectures and network weights have a mutual impact on each other. Thus we need to co-optimize them together in building a successful meta-learner.

Our contributions are summarized as follows:

- We propose an effective and efficient method, namely Connection-Adaptive Meta-Learning (CAML), which co-optimizes architecture and network weights simultaneously for meta-learning.
- To perform a smooth transition from the supernet to our target architecture, we propose the progressive connection consolidation to prune the supernet gradually during searching, which also strengthen the mutual interaction of meta-connections and meta-weights.
- Extensive experiments show that CAML achieves state-of-the-art performance on both FC100 and Mini-Imagenet datasets under various settings with 130x less computational cost, which reveals the effectiveness and efficiency of our method.

2 RELATED WORK

2.1 META-LEARNING

Meta-learning (learning to learn) (Finn et al., 2017; Ravi & Larochelle, 2017) 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 been proved 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 (Santoro et al., 2016), metric learning (Vinyals et al., 2016) and gradient-based approaches (Finn et al., 2017).

In gradient-based approaches, an optimizer called meta-learner is learned to perform fast adaption on new tasks (Hochreiter et al., 2001). Instead of using the learned optimizer, model-agnostic meta-learning (MAML) (Finn et al., 2017) 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, in these methods, the impact of the architecture is overlooked.

2.2 META-LEARNING WITH NEURAL ARCHITECTURE SEARCH

Neural architecture search (NAS) 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 (Zoph & Le, 2017; Liu et al., 2019b), semantic segmentation (Liu et al., 2019a), and object detection (Xu et al., 2019).

In gradient-based NAS methods like DARTS (Liu et al., 2019b), 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 on searching architectures for a single specific task.

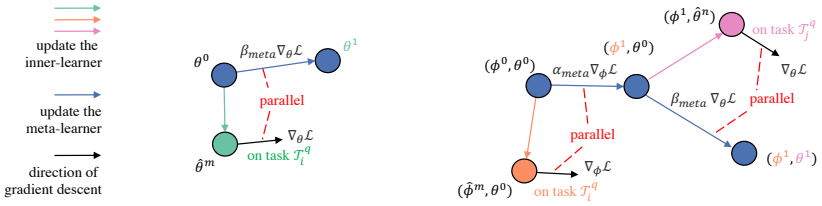
Recently, there have been some works combining NAS and meta-learning. In Auto-meta (Kim et al., 2018), progressive neural architecture search (Liu et al., 2018) is applied to meta-learning to search optimal architectures. However, in every iteration of the NAS process, the entire meta-training process must be carried out, which takes more than one hundred GPU days to converge. BASE (Shaw et al., 2019), Meta-NAS (Elsken et al., 2020) and T-NAS (Lian et al., 2020) are proposed to design task-dependent architectures for new tasks. Meta-NAS (Elsken et al., 2020) employs Reptile (Nichol et al., 2018) as its backbone and utilizes a soft pruning strategy over all layers with the search progressing. T-NAS (Lian et al., 2020) attempts to learn a general meta-architecture through MAML (Finn et al., 2017). Then both Meta-NAS and T-NAS perform fast adaptation for a new test task. Soft pruning (Elsken et al., 2020) does not really prunes the operations of slight importance. Thus Meta-NAS still need to do one shot pruning for the final architectures like T-NAS. However, all these methods need to train every task-specific architecture from scratch, which is computationally expensive. Nevertheless, current methods separate the architecture searching and meta-weights training. They search architectures first and then train meta-weights based on searched architectures. In this two-stage strategy, the meta-weights are overlooked, leading to a suboptimal meta-learner.

3 APPROACH

Before introducing our approach, we make a short review of Model-Agnostic Meta-Learning (MAML) (Finn et al., 2017) and Differentiable Architecture Search (DARTS) (Liu et al., 2019b), which will help us make a better understanding of our method. Then we introduce our CAML in Section 3.3 and the progressive connection consolidation in Section 3.4.

3.1 MAML

In MAML (Finn et al., 2017), 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 Fig.9. 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, while tasks sampled from $\{\mathcal{T}^{\text{q}}\}$ are uses



(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 (Finn et al., 2017) optimizes meta-weights using the same update direction as the inner-learner's. (b) Our CAML optimizes both the connection parameters ϕ and network weights θ using the same update direction as the inner-learners', respectively.

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 on 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 (Liu et al., 2019b) 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 relatively largest attention values are preserved, while the others are pruned. There is a bi-level optimization problem which the connection parameters and the network weights need 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, which is set to 0 in our work.

3.3 Connection-Adaptive Meta-Learning

The main goal of our CAML is to find meta-learners with both adaptive architectures and meta-weights. Note that in our method, we represent the candidate operations of each layer as connections.

Thus the architecture search is to learn the adaptive connection of each layer, which we call *meta-connections*. However, as described in Liu et al. (2019b), there lies a bi-level optimization problem. We cannot optimize connection parameters ϕ solely without regard to the network weights θ.

As demonstrated in Fig.2, in MAML (Finn et al., 2017), they pick the direction of the last step gradient descent in the inner-learner, which is employed for updating the meta-learner. 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.*, the inner-learner and the meta-learner for ϕ and θ, respectively. During the inner updates for connection parameters ϕ, the network weights θ is fixed. Following the common settings in NAS methods Li et al. (2020); Liu et al. (2019b), 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. 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,

$$\phi_i^{m+1} = \phi_i^m - \alpha_{\text{inner}} \nabla_{\phi_i^m} \mathcal{L}(f_{\phi_i^m, \tilde{\theta}}; \mathcal{T}_i^{\text{split-arch,s}}), \tag{5}$$

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

$$\tilde{\phi} = \tilde{\phi} - \alpha_{\text{meta}} \nabla_{\tilde{\phi}} \mathcal{L}(f_{\phi_i^M, \tilde{\theta}}; \mathcal{T}_i^{\text{split-arch,q}}), \tag{6}$$

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:

$$\theta_j^{m+1} = \theta_j^m - \beta_{\text{inner}} \nabla_{\theta_j^m} \mathcal{L}(f_{\theta_j^m, \tilde{\phi}}; \mathcal{T}_j^{\text{split-weights,s}}), \tag{7}$$

$$\tilde{\theta} = \tilde{\theta} - \beta_{\text{meta}} \nabla_{\tilde{\theta}} \mathcal{L}(f_{\theta_j^M, \tilde{\phi}}; \mathcal{T}_j^{\text{split-weights,q}}), \tag{8}$$

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^*. The complete algorithm of our CAML is shown in Algorithm 1

In MAML++ (Antoniou et al., 2019), several techniques are proposed to improve the performance of MAML (Finn et al., 2017), including cosine annealing of the meta learning rate, multi-step loss, etc. Since our CAML is based on MAML, these techniques could be directly employed to our method, which can further promote the performance. We represent the promoted CAML as CAML++.

3.4 PROGRESSIVE CONNECTION CONSOLIDATION

To enrich architecture flexibility, CAML employs a supernet during the architecture searching, while our final meta-learner is a sub-network pruned from the supernet. There lies an optimization gap between the supernet and the sub-network. In previous work like T-NAS (Lian et al., 2020), they apply a hard-pruning strategy that all layers of the supernet are pruned once at the end of searching. Then they retrain the meta-weights based on searched architectures. This two-stage training would lead to a suboptimal meta-learner since meta-weights are overlooked during architecture searching.

We propose to provide a smooth transition from the supernet to the sub-network and then co-optimize the meta-connections and the meta-weights in the sub-network simultaneously. We propose progressive connection consolidation that prunes the supernet layer by layer during searching. To determine the pruning order of layers, we define the *layer confidence* as follows:

Layer confidence. A layer (i, j) consists of all operations from the candidate operation set $\mathcal{O}$. Following DARTS (Liu et al., 2019b), we use a *zero* operation in the candidate set to represent a lack of connection. $\phi_o^{(i,j)}$ are the related connection parameters for layer (i, j). Thus, the layer

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 α_{inner}, α_{meta}, β_{inner}, β_{meta}.

Randomly initialize network weights θ and connection parameters ϕ.
while *not terminated* **do**
  Sample batch of tasks $\{ \mathcal{T}^{\text{split-arch}} \}$ from $\mathcal{D}_{\text{meta-train-split-arch}}$;
  for $\mathcal{T}_i^{\textit{split-arch}} \in \{\mathcal{T}^{\textit{split-arch}}\}$ **do**
    Get datapoints $\mathcal{T}_i^{\text{split-arch,s}}$ from support set.
    Update architecture parameters ϕ_i^m with Equation 5 for M steps.
    Get datapoints $\mathcal{T}_i^{\text{split-arch,q}}$ from query set for the meta-learner of ϕ.
  end
  Update $\tilde{\phi}$ with Equation 6 for one step.
  Sample batch of tasks $\{ \mathcal{T}^{\text{split-weights}} \}$ from $\mathcal{D}_{\text{meta-train-split-weights}}$;
  for $\mathcal{T}_j^{\textit{split-weights}} \in \{\mathcal{T}^{\textit{split-weights}}\}$ **do**
    Get datapoints $\mathcal{T}_j^{\text{split-weights,s}}$ from support set.
    Update network weights θ_i^m with Equation 7 for M steps.
    Get datapoints $\mathcal{T}_j^{\text{split-weights,q}}$ from query set for the meta-learner of θ.
  end
  Update $\tilde{\theta}$ with Equation 8 for one step.
  if *pruning required in this iteration* **then**
    Prune the network architecture and weights.
  end
end

confidence of layer (i, j) is defined as the maximum attention value on non-zero operations:

$$S_{\text{LC}}^{(i,j)} = \max_{o \in \mathcal{O}, o \neq zero} \frac{\exp\left(\phi_o^{(i,j)}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\phi_{o'}^{(i,j)}\right)} \tag{9}$$

In our experiments, we apply S_{LC} to determine the importance of each layer. The process of fixing one connection can be disassembled into two steps. First, at every five epochs during searching, 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 connections searching. On the convergences of the meta-learner, we obtain an adaptive neural 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.

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 (Lake et al., 2011), FC100 (Oreshkin et al., 2018) and Mini-Imagenet (Ravi & Larochelle, 2017). Our experiments consist of architecture search and evaluation. During the training stage, we search for a meta-learner which has both the adaptive architecture and the meta-weights. Then we do an evaluation on the searched meta-learner. For better comparison, we also evaluate the adaptive architecture by training it from scratch. At last, we do some ablation studies, including the contribution of CAML and progressive connection consolidation, the impact of searched meta-weights and the effectiveness of pruning strategies.

Table 1: 5-way accuracy results on Mini-Imagenet.

Method	Arch.	Params (K)	Cost (GPU days)	Accuracy (%)	
				1-shot	5-shot
MAML ((Finn et al., 2017))	4CONV	32.9	N/A	48.7 ± 1.8	63.1 ± 0.9
MAML (first-order) (Finn et al., 2017)	4CONV	32.9	N/A	48.1 ± 1.8	63.2 ± 0.9
MAML++ ((Antoniou et al., 2019))	4CONV	32.9	N/A	52.2 ± 0.3	$\mathbf{68.3 \pm 0.4}$
BASE (Softmax) (Shaw et al., 2019)	Cell	1200	-	-	65.4 ± 0.7
BASE (Gumbel) (Shaw et al., 2019)	Cell	1200	-	-	66.2 ± 0.7
Auto-Meta (small) (Kim et al., 2018)	Cell	28.0	112	49.6 ± 0.2	65.1 ± 0.2
T-NAS (Lian et al., 2020)	Cell	26.5	152	52.8 ± 1.4	67.9 ± 0.9
Meta-NAS (small) (Elsken et al., 2020)	Cell	30.0	7	49.7 ± 0.4	62.1 ± 0.9
Meta-NAS (big) (Elsken et al., 2020)	Cell	100.0	7	$\mathbf{53.2 \pm 0.4}$	67.8 ± 0.7
CAML (train from scratch)	Cell	**25.0**	**1.1**	51.0 ± 0.1	65.1 ± 0.2
CAML	Cell	**25.0**	**1.1**	51.5 ± 0.1	66.3 ± 0.2
CAML++	Cell	**25.0**	1.3	$\mathbf{53.4 \pm 0.1}$	$\mathbf{68.5 \pm 0.1}$

4.1 ARCHITECTURE SEARCH

We apply the basic searching settings in Liu et al. (2019b) to CAML. A cell (Zoph et al., 2018) represented as a directed acyclic graph consists of an ordered sequence of computational nodes. For generalization and efficiency, we only search for two cells composed of a normal cell and a reduction cell. Then we stack two cells to build the whole network architecture. Therefore, the adaptive architecture ϕ is determined by $\{ \phi_{normal}, \phi_{reduce} \}$.

Candidate operation set. As for the candidate operation set, we use the same set as Liu et al. (2019b), 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 detail searching settings and searched architectures are summarized in A.

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. During the evaluation, we train the searched meta-learner for 100 epochs with 1200 independent tasks for each epoch. Note that different from Liu et al. (2019b) and Lian et al. (2020), 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 a randomly initialized weights.

The experiments results on Mini-Imagenet, FC100 and Omniglot are represented in Table 1, Table 2 and Table 5, respectively. Our method CAML outperforms the baseline MAML by 2.8% (51.5 % versus 48.7 %) with fewer parameters (25.0K versus 32.9K), verifying the advantages of our method. Moreover, we can also observe that our CAML++, an upgraded version of CAML described in 3.3, achieves the best performance among the baselines above.

Besides, we make a comparison of the computational cost of search and evaluation with other state-of-the-art methods. Compared to other architecture search methods (*e.g.*, T-NAS (Lian et al., 2020)), our CAML++ achieves a better performance with 130x less total computational cost.

4.3 ABLATION STUDIES

Contribution of CAML and progressive connection consolidation. We evaluate the contribution made by two components of our methods, namely CAML and progressive connection consolidation. Results are shown in Table 3. In existing works (*e.g.*, T-NAS (Lian et al., 2020)), the connection parameters and the network weights are treated equally. Thus ϕ and θ are optimized by backpropogating once. However, since learning rates of θ and ϕ are usually unequal, the update direction of the meta-learner is not parallel to the inner-learner's, which is against MAML (Finn et al., 2017) as

Table 2: 5-way accuracy results on FC100.

Method	Accuracy (%)		
	1-shot	5-shot	10-shot
MAML (first-order)(Finn et al., 2017)	35.6 ± 0.1	49.1 ± 0.1	54.1 ± 0.9
MAML((Finn et al., 2017))	38.1 ± 1.7	50.4 ± 1.0	56.2 ± 0.8
MAML++((Antoniou et al., 2019))	38.7 ± 0.4	52.9 ± 0.4	58.8 ± 0.4
T-NAS (Lian et al., 2020)	$\mathbf{39.7 \pm 1.4}$	53.1 ± 1.0	58.9 ± 0.7
CAML (train from scratch)	37.7 ± 0.4	51.1 ± 0.6	56.1 ± 0.3
CAML	38.3 ± 0.3	51.4 ± 0.4	56.4 ± 0.3
CAML++	$\mathbf{39.3 \pm 0.2}$	$\mathbf{54.3 \pm 0.7}$	$\mathbf{59.8 \pm 0.6}$

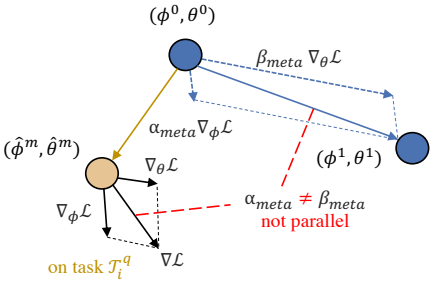


Figure 3: In previous works (*e.g.*, T-NAS and Meta-NAS), connection parameters and network weights are treated equally in optimization. Due to the unequal learning rates, the update direction of the meta-learner is not parallel to the inner-learner's, which is against MAML.

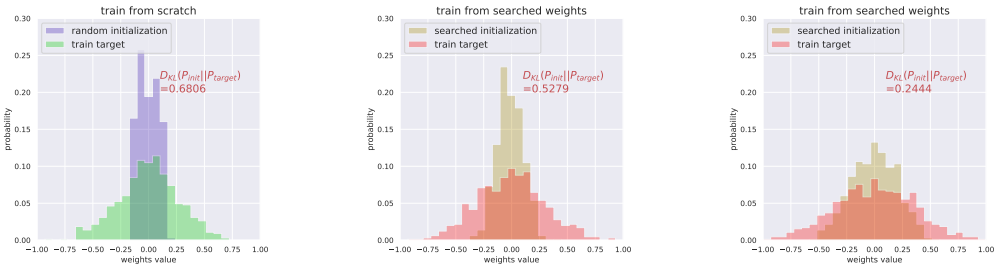
Table 3: 5-shot, 5-way accuracy results of different methods on Mini-Imagenet. Architectures searched without CAML are derived by updating ϕ and θ by one backpropagation. PCC represents progressive connection consolidation. Architectures derived without PCC means that we prune the supernet at the end of searching once.

CAML	PCC	Params (K)	Accuracy (%)
✗	✗	51.3	59.0 ± 0.3
✗	✓	44.9	61.0 ± 0.4
✓	✗	20.0	62.6 ± 0.1
✓	✓	25.0	$\mathbf{65.1 \pm 0.2}$

shown in Fig.3. We believe that our method would lead to the parallel direction of gradient descent, which helps to find better meta-learners. Progressive connection consolidation strengthens the co-optimization between the architecture and the network weights. Besides, CAML can cooperate well with progressive connection consolidation to provide further improvement.

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 contribution of the kept network weights, we compare our model with the meta-learner trained from scratch. We also compare to the model with the hard-pruning criterion Liu et al. (2019b) instead of our PCC. We train the hard-pruned architectures from a random initialization and from the kept network weights for evaluation. Results are summarized in Fig. 5. Obviously, our CAML learns knowledge from task distribution $p_{\mathcal{T}}$ more efficiently and effectively from the kept initialization compared to the random initialization. In addition, without our PCC, keeping weights does not perform better compared to the one with 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 Fig.4. Clearly, the distribution of our searched meta-weights is closer to the optimization target, showing the effectiveness of the co-optimization. Fig.4 (b) also shows the previous two-stage training strategy leads to a sub-optimal meta-learner, whose distribution is far from the training target.

Comparison of different pruning strategies. To prove the effectiveness of our layer confidence based pruning strategy, we also prune the supernet with fixed orders like forwarding sequence or backward. The results are summarized in Table 4. Clearly, layer confidence based pruning strategy in our PCC could help us find better adaptive architectures, which achieves higher performance with fewer parameters. Besides, we could also observe performance improvement by taking searched network weights as initialization, which proves the effectiveness of our methods.



(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.

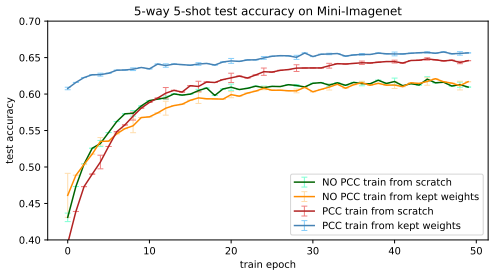


Figure 5: 5-shot, 5-way meta-test accuracy on Mini-Imagenet during the evaluation.

Table 4: Comparison of 5-way 5-shot accuracy by three pruning strategies of CAML. S_{LC} means the layer confidence. In progressive connection consolidation, we prune the layers of the supernet in descending order of S_{LC}.

Sequence	Params (K)	Train from scratch	Train from searched-weigths
Forward	34.0	61.0 ± 1.0	66.0 ± 0.1
Backward	27.7	63.6 ± 0.6	64.5 ± 0.2
S_{LC} based	25.0	65.1 ± 0.2	66.3 ± 0.2

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 solutions are inefficient and ignore the co-optimization of the architecture and the network weights. To tackle the existing problems, we propose a novel CAML. CAML updates the architecture parameters and the network weights simultaneously by two different backpropagations in one iteration. By fixing the architecture layer by layer during searching, CAML strengthens the co-optimization between the architecture and the meta-weights. Extensive experiments show that our CAML and progressive connection consolidation are both helpful to the success of a meta-learner. Our method achieves state-of-the-art performance on all three few-shot datasets with 130x less computational cost.

References

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

Harrison Edwards and Amos Storkey. Towards a neural statistician. In *International Conference on Learning Representations*, 2017.

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*, pp. 12365–12375, 2020.

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*, pp. 1126–1135, 2017.

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

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

Łukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. *arXiv preprint arXiv:1703.03129*, 2017.

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.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

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.

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*, pp. 1620–1630, 2020.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

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.

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 *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018.

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*, pp. 82–92, 2019a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019b.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

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*, pp. 721–731, 2018.

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

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, pp. 4780–4789, 2019.

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

Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *Advances in Neural Information Processing Systems*, pp. 11227–11237, 2019.

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*, pp. 403–412, 2019.

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

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*, pp. 6649–6658, 2019.

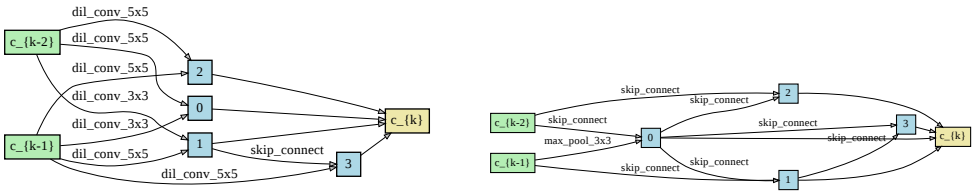
Ruixiang Zhang, Tong Che, Zoubin Ghahramani, Yoshua Bengio, and Yangqiu Song. Metagan: An adversarial approach to few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 2365–2374, 2018.

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

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*, pp. 8697–8710, 2018.

A DETAIL SEARCHING SETTINGS

To avoid the computational cost of Hessian matrix, the first-order DARTS (Liu et al., 2019b) and the first-order approximation of MAML (Finn et al., 2017) are employed for searching meta-learners. As for the inner-learners of ϕ and θ, we use the vanilla SGD with inner learning rate $\alpha_{inner} = 1 \times e^{-2}$ for optimizing ϕ, while a inner learning rate $\beta_{inner} = 0.1$ for training θ. In the meta-learner of ϕ, an Adam (Kingma & Ba, 2015) optimizer is employed for updating, with an initial learning rate $\alpha_{meta} = 1 \times e^{-3}$ and a weight decay of 3×10^{-4}. A similar Adam without weight decay is applied to training the meta-learner of θ. We choose $M = 5$ as the inner update step. The searching is executed on both Omniglot and Mini-Imagenet under the setting of 5-way, 5-shot. For each dataset, we sample 1200 tasks from $\mathcal{D}_{meta-train}$ for training and 600 tasks from $\mathcal{D}_{meta-test}$ for evaluation. On Omniglot, we prune the architecture every three epochs from the fifth epoch, while we do it every five epoch from ninth epoch in Mini-Imagenet. All search and adaptation experiments are carried out on NVIDIA RTX 2080TI GPUs. The whole search process requires about 0.6 GPU days on Mini-Imagenet. The searched architectures is visualized in Fig.6 and Fig.7.

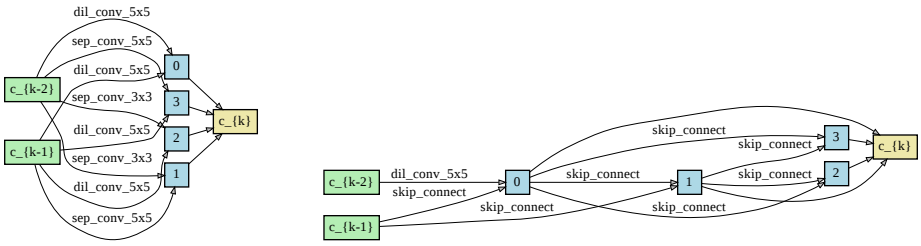


(a) Normal cell (b) Reduction cell

Figure 6: Architecture searched in 5-way 5-shot setting of Mini-imagenet.

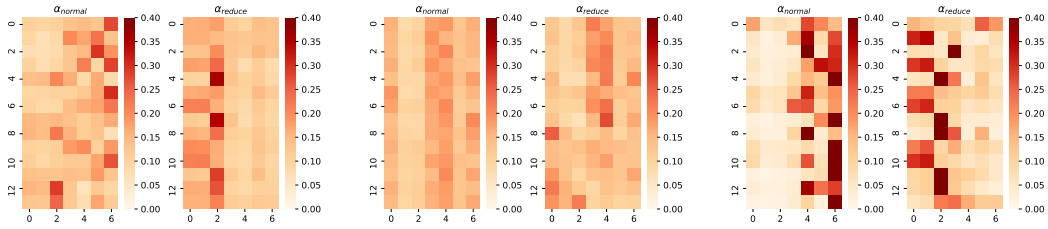
B HEATMAP OF THE CONECTION PARAMETERS

We illustrate the heatmap of connection parameters when we do pruning in Fig.8. It is evident that without CAML (treat connection parameters and network weights as the same kind of parameters), we will find a suboptimal architecture, which contains more convolution layers. Without progressive connection consolidation, the searched architecture cannot cooperate better with the kept weights in the supernet than random initialization.



(a) Normal cell (b) Reduction cell

Figure 7: Architecture searched in 5-way 5-shot setting of FC100.



(a) Our method (b) Without CAML (c) Without PCC

Figure 8: Heatmap while we do pruning. (a). We use standard CAML with progressive connection consolidation (PCC). (b). We treat connection parameters and network weights equally. (c). We only prune the supernet at the end of searching.

C 5-WAY ACCURACY RESULTS ON OMNIGLOT DATASET

We illustrate the results of 5-way 1-shot and 5-way 5-shot on Omniglot dataset in Tab.5. We can observe that CAML++ achieves state-of-the-art performance among existing NAS-based methods.

Table 5: 5-way accuracy results on Omniglot dataset.

Method	Accuracy (%)	
	1-shot	5-shot
Siamese nets ((Koch et al., 2015))	97.3	98.4
Matching nets ((Vinyals et al., 2016))	98.1	98.9
Neural statistician ((Edwards & Storkey, 2017))	98.1	99.5
Memory mod. (Kaiser et al., 2017)	98.4	99.6
Meta-sgd (Li et al., 2017)	99.53 ± 0.26	99.93 ± 0.09
MAML ((Finn et al., 2017))	98.7 ± 0.4	99.9 ± 0.1
MAML++ ((Antoniou et al., 2019))	99.47	99.93
Auto-Meta ((Kim et al., 2018))	97.44 ± 0.07	-
T-NAS ((Lian et al., 2020))	99.16 ± 0.34	99.93 ± 0.07
CAML(train from scratch)	96.11 ± 0.14	99.18 ± 0.06
CAML	96.89 ± 0.09	99.57 ± 0.07
CAML++	$\mathbf{99.69 \pm 0.14}$	$\mathbf{99.95 \pm 0.03}$

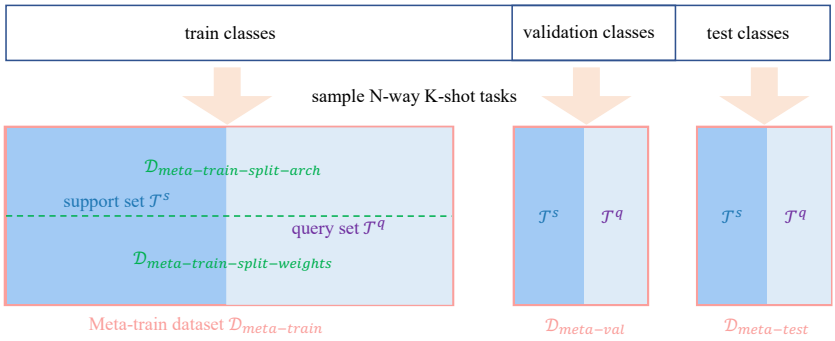


Figure 9: Dataset splits

D DATASET SPLITS

In few-shot learning, the dataset is composed by train, validation and test classes. Under N-way K-shot setting, we sample N classes, of which each contains K examples as one task. Tasks sampled from train classes is denoted $D_{\text{meta-train}}$. So as $D_{\text{meta-val}}$ and $D_{\text{meta-test}}$. Each of them is divided into two subset: support set $\mathcal{T}^{\text{s}}$ and query set $\mathcal{T}^{\text{q}}$. The former is used for updating the inner-learners, while the later is for the meta-learners. In CAML, during one update step of each component of the architecture and weights, both the support set and query set data are needed. In NAS methods (Liu et al., 2019b), the architecture is considered a hyper-parameter selected to maximize the performance on the validation set, based on the network weights trained with the data of the training set in the inner loop. So we split the $D_{\text{meta-train}}$ into $D_{\text{meta-train-split-arch}}$ and $D_{\text{meta-train-split-weights}}$ as the validation set and training set, respectively. As visualized in Figure 9, both splits are composed of the support set and the query set. One iteration of CAML consists of two backpropagations, as shown in Figure 2. During the backpropagation one, the architecture is optimized with the data of $D_{\text{meta-train-split-arch}}$ (including the support set and the query set), while the network weights are fixed and trained with the data of $D_{\text{meta-train-split-weights}}$ in the previous iteration. During backpropagation two, the weights are trained on the fixed architecture.

E RESULTS ON CIFAR-10 AND IMAGENET

We also perform evaluation of the searched architecture on Mini-Imagenet on standard NAS benchmarks. The results are demonstrated in Tab.6 and Tab.7.

Table 6: Comparison with state-of-the-art methods on CIFAR-10.

Method	Test Error (%)	Params (M)	Search Cost (GPU days)
Random search baseline + cutout	3.29 ± 0.15	3.2	-
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	180
AmoebaNet-A + cutout (Real et al., 2019)	3.34	3.2	3150
PNAS (Liu et al., 2018)	3.41 ± 0.09	3.2	225
DARTS (first order) (Liu et al., 2019b)	3.00 ± 0.14	3.3	1.5
DARTS (second order) (Liu et al., 2019b)	2.76 ± 0.09	3.37	4
Ours + cutout	3.03 ± 0.14	2.83	0.5

Table 7: Comparison with state-of-the-art methods on ImageNet.

Method	Test Error(%)		Params	Search Cost
	top-1	top-5	(M)	(GPU days)
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	1800
NASNet-B (Zoph et al., 2018)	27.2	8.7	5.3	1800
NASNet-C (Zoph et al., 2018)	27.5	9.0	4.9	1800
AmoebaNet-A (Real et al., 2019)	25.5	8.0	5.1	3150
AmoebaNet-B (Real et al., 2019)	27.2	8.7	5.3	3150
AmoebaNet-C (Real et al., 2019)	27.5	9.0	4.9	3150
PNAS (Liu et al., 2018)	25.8	8.1	5.1	~ 255
DARTS (Liu et al., 2019b)	26.9	9.0	4.9	4
Ours	27.3	9.5	4.1	0.5