Pruning Meta-Trained Networks for On-Device Adaptation

Dawei Gao*
SKLSDE & BDBC, Beihang University
Beijing, China
david_gao@buaa.edu.cn

Xiaoxi He*
ETH Zürich
Zürich, Switzerland
hex@ethz.ch

Zimu Zhou
Singapore Management University
Singapore, Singapore
zimuzhou@smu.edu.sg

Yongxin Tong
SKLSDE & BDBC, Beihang University
Beijing, China
yxtong@buaa.edu.cn

Lothar Thiele
ETH Zürich
Zürich, Switzerland
thiele@ethz.ch

ABSTRACT

Adapting neural networks to unseen tasks with few training samples on resource-constrained devices benefits various Internet-of-Things applications. Such neural networks should learn the new tasks in few shots and be compact in size. Meta-learning enables few-shot learning, yet the meta-trained networks can be over-parameterised. However, naive combination of standard compression techniques like network pruning with meta-learning jeopardises the ability for fast adaptation. In this work, we propose adaptation-aware network pruning (ANP), a novel pruning scheme that works with existing meta-learning methods for a compact network capable of fast adaptation. ANP uses weight importance metric that is based on the sensitivity of the meta-objective rather than the conventional loss function, and adopts approximation of derivatives and layer-wise pruning techniques to reduce the overhead of computing the new importance metric. Evaluations on few-shot classification benchmarks show that ANP can prune meta-trained convolutional and residual networks by 85% without affecting their fast adaptation.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks.*

KEYWORDS

deep neural networks; meta learning; network pruning

ACM Reference Format:
Dawei Gao, Xiaoxi He, Zimu Zhou, Yongxin Tong, and Lothar Thiele. 2021. Pruning Meta-Trained Networks for On-Device Adaptation. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482378

*Both authors contributed equally to the paper

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

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482378

1 INTRODUCTION

On-device adaptation refers to learning previously unseen tasks by updating an initial model on-board. This is desired in internet of things (IoT) applications including personal drones, home robots and self-driving vehicles, since uploading newly collected data for model updating can be infeasible due to unstable wireless connections, limited bandwidth or privacy concerns. An initial model for on-device adaptation should allow *fast adaptation* and should be *compact in size* due to the following reasons: *(i)* Only a limited amount of training data is available locally, which requires the model to adapt to new tasks with few samples. This requirement is reasonable because users are often asked to provide supervision with a few private data samples so that the general initial model can be rapidly customized, personalized, or calibrated to new tasks, users or environments without affecting user experiences. *(ii)* IoT applications often run on resource-constrained devices, where a large model easily overwhelms the computation and memory resources. In contrast to the cloud, IoT platforms powered by systems on chips (SoCs) or micro-controllers are particularly limited by the small memory system (*KB* to *MB*) to store model parameters [2].

An effective solution to enable fast adaptation is meta-learning, where the initial model for deployment is meta-trained, and adaptation is implemented and assessed by few-shot learning [8, 25]. Of particular interest is Model-Agnostic Meta-Learning (MAML), a general gradient-based algorithm that learns the weights of an given initial architecture, such that the meta-trained model excels at few-shot learning [8]. Gradient-based algorithms [8, 24, 30] are suited for on-device adaptation since recent research due to the feasibility of gradient-based training on low-resource devices [12, 22]. Despite allowing fast adaptation, MAML fails to generate a compact model as it only optimised the network parameters, but does not alter the initial architecture [7]. The initial architecture, however, has to be over-parameterised for effective meta-training [1]. Consequently, the meta-trained model is also over-parameterised.

An intuitive remedy for compact models is network pruning, which has the potential to radically remove unimportant parameters in a neural network without deterioration in inference accuracy [5]. However, existing pruning methods [6, 13, 15, 16, 21, 29] are incompatible with meta-learning and may jeopardise the ability of fast adaptation, as they are designed to retain the *inference accuracy on a known single task*. In contrast, the goal of meta-learning (in the following called *meta-objective*) is to optimise the *ability to adapt to new tasks following certain distribution*, which differs from the

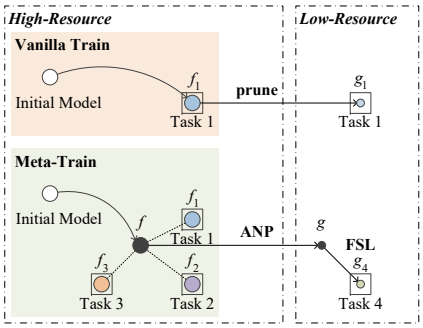


Figure 1: Comparison between pruning a vanilla-trained model and a meta-trained model. Existing pruning methods are designed for vanilla-trained networks, where the task is the same before and after pruning. For example, the vanilla-trained model f_1 is pruned into a compact model g_1 to deploy on low-resource platforms, where both f_1 and g_1 are optimised for Task 1. Our focus is to prune a meta-trained network, where the model is optimised on batches of tasks. The pruned meta-trained model should fast adapt to new tasks after deployment. For example, the weights for the initial architecture is meta-trained to f, which is a good initialisation for Task 1, 2, and 3. Then f is pruned into a compact model g, where g is expected to adapt into g_4 after few-shot learning (FSL) and yield high inference accuracy on Task 4.

objective of single task pruning. Therefore, to construct a compact network capable of fast adaptation, a new network pruning scheme , which can optimise the topology in accordance with the meta-objective, is required, as illustrated in Fig. 1.

In this paper, we propose Adaptation-aware Network Pruning (ANP), a novel network pruning scheme that works in synergy with meta-learning. While meta-learning optimises the weights, ANP compresses and optimises the topology. Together we are able to construct compact neural networks capable of fast adaptation.

At a high level, ANP extends the analysis of second order derivatives for pruning vanilla-trained networks [15, 19] to meta-learning scenario. That is, ANP calculates weight importance values from the training data for *tasks sampled from a certain distribution* and removes those weights that induces minimal changes on the meta-objective. However, pruning based on such second order derivatives of the meta-objective is computation-intensive, as it requires the global third order derivatives and generalised inverses of Hessian matrices. ANP avoids calculations of third order derivatives via a novel approximation approach. It further reduces the computation overhead by layer-wise pruning such that the generalised inverse Hessian matrices are obtained efficiently and stably.

Evaluations on Mini-ImageNet [25] and Caltech-UCSD Birds-200-2011 (CUB) [27] show that ANP can prune common used initial architectures by 85% with less than 1% accuracy loss in few-shot classification and it works with different gradient-based meta-learning methods (*e.g.*, MAML [8, 24, 30]). In contrast, pruning the initial architecture to the same ratio with existing methods [6, 13] will lead to a loss of 7.01% to 26.70% in few-shot classification accuracy.

Our main contributions and results are as follows.

- To the best of our knowledge, this is the first investigation of pruning *meta-trained* neural networks for model compression. Due to the inconsistency between the meta-objective and the weight importance metrics in network pruning, naive combination of pruning and meta-learning deteriorates the model adaptability in few-shot learning.
- We design ANP, a novel meta-learning-compatible network pruning scheme. ANP can prune over-parameterized meta-trained networks without sacrificing their ability for fast adaptation. It applies approximation of derivatives and layer-wise pruning to reduce the computation overhead in pruning meta-trained deep models.
- Evaluations on few-shot classification benchmarks show that ANP can prune the initial architectures for meta-learning by 85% while retaining the few-shot classification accuracy.

In the rest of this paper, we review related work in Sec. 2, introduce our ANP method in Sec. 3, present its evaluations in Sec. 4 and conclude in Sec. 5.

2 RELATED WORK

Our work is relevant to the following threads of research.

Meta-Learning for Few-Shot Learning. Training a deep neural network upon limited samples *i.e.*, in few-shots, tends to overfit [10]. Meta-learning has been a successful solution to few-shot learning [17, 28], where the meta-trained model is able to learn a new task from a few training samples. In this work, we focus on gradient-based meta-learning methods [8, 24, 30] for their applicability in various learning tasks and the potential to enable on-device adaptation. Specifically, we aim to generate an initial model that can fast adapt to new tasks and is compact in size.

MAML-like algorithms [8, 24, 30] only adapt *weights* of the initial architecture without alerting its topology. A few studies [7, 20] propose to integrate MAML with neural architecture search to optimise the initial architecture. Since their primary goal is higher few-shot inference accuracy, the resulting model can even have more parameters than the initial architecture in MAML [7]. Our work also adapts the initial architecture, yet with a complementary objective. Particularly, we sparsify it without sacrificing its ability of fast adaptation, which results in a much smaller model.

Network Pruning. Given an over-parameterised network well-trained for a given task, network pruning eliminates unimportant parameters without major accuracy loss on the inference task [5]. Fine-grained pruning (*e.g.*, weights) [6, 13] results in a higher compression rate whereas coarse-grained pruning (*e.g.*, filters) [4, 23] is a better fit for acceleration on commodity hardware. Various importance criteria have been proposed, such as magnitude [13], second order derivatives [6, 15, 16], and information bottleneck [4]. However, all existing parameter importance metrics are derived for vanilla-trained networks, *i.e.*, the pruned network targets at the same task as before pruning. In contrast, we propose a new weight importance metric for meta-trained networks, where the pruned network should fast adapt to new tasks unseen before pruning.

A very recent study [26] explored improving the meta-training procedure via pruning. Specifically, in [26], pruning is applied as a model capacity constraint to avoid meta-overfitting, where the pruned parameters are re-activated during the retraining phase.

Its final output is still a *large dense* network which is unfit for deployment on resource-constrained devices. As will be shown in our evaluations (see Sec. 4.2), directly combining iterative hard thresholding (equivalent to the "Magnitude" baseline in our work) and meta-learning method like Reptile [24] as in [26] leads to significant drop in few-shot classification accuracy at high pruning ratios.

3 METHOD

In this section, we first provides a primer on meta learning (Sec. 3.1) and then explain our ANP in detail. Specifically, we introduce our new weight importance metric for pruning meta-trained networks (Sec. 3.2), followed by derivative approximations (Sec. 3.3) and layer-wise pruning (Sec. 3.4) for efficient calculating of the weight importance metric, and finally present the complete algorithm (Sec. 3.5).

3.1 Primer on MAML

Model-Agnostic Meta-Learning (MAML) [8] learns an initial model f such that given a new task, f can learn it with a few training samples. MAML is a two-tier gradient decent based optimisation process. In each iteration of the optimisation, M tasks with corresponding training datasets $\{\mathcal{D}_i\}, i \in \{1, \cdots, M\}$ are sampled from a certain distribution. We use $\boldsymbol{\theta}$ to represent the current parameters of f. In each iteration, $\boldsymbol{\theta}$ is updated to $\boldsymbol{\theta''}$ as follows.

$$\boldsymbol{\theta'}^i = \boldsymbol{\theta} - \alpha \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}, \mathcal{D}_i\right) \tag{1}$$

$$\boldsymbol{\theta''} = \boldsymbol{\theta} - \beta \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}_m \tag{2}$$

where

$$\mathcal{L}_m = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}\left(\boldsymbol{\theta'}^i, \mathcal{D}_i\right) \tag{3}$$

is called the meta-objective, $\mathcal{L}(\cdot, \cdot)$ is a loss function, and α and β are learning rates. The inner-loop gradient decent (1) updates parameters $\{\boldsymbol{\theta'}^i\}$ for task-specific objectives. Note that $\boldsymbol{\theta'}^i$ is the vector for parameters trained on task i. The outer-loop gradient descent (2) then updates the parameters for the meta-objective. Since the meta-objective $\mathcal{L}_m$ contains first derivatives of $\boldsymbol{\theta}$ in (1), the gradient in (2) is in effect a second derivative of $\boldsymbol{\theta}$.

3.2 Weight Importance in Meta-Training

Pruning eliminates unimportant weights in the network, where the weight importance is assessed by the impact of its removal on the inference accuracy. A classic and effective approach to quantify weight importance is an analysis based on second order derivatives, which measures the change in the objective caused by a weight change [6, 15, 16, 19]. Weight importance of vanilla-trained neural networks is defined using the traditional loss functions as the objective. We now define weight importance for meta-trained neural networks via an analysis based on the second order derivatives of the meta-objective as (3).

Defining Weight Importance. To quantify the weight importance of meta-trained networks, the conventional loss function is replaced by the meta-objective. Specifically, the Taylor series of the change in the meta-objective due to a parameter change is

$$\delta \mathcal{L}_m = \left(\frac{\partial \mathcal{L}_m}{\partial \boldsymbol{\theta}}\right)^{\top} \delta\boldsymbol{\theta} + \frac{1}{2} \delta\boldsymbol{\theta}^{\top} \mathbf{H} \delta\boldsymbol{\theta} + O(||\delta\boldsymbol{\theta}||^3) \tag{4}$$

where $\mathbf{H} = \partial^2 \mathcal{L}_m / \partial \boldsymbol{\theta}^2$ is the Hessian matrix of the meta-objective with respect to $\boldsymbol{\theta}$.

Similar to the analysis of second derivatives for vanilla-trained networks [15, 19], the first term vanishes for a well meta-trained network, and the higher order derivatives can be ignored. Therefore,

$$\delta \mathcal{L}_m \approx \frac{1}{2} \delta\boldsymbol{\theta}^{\top} \mathbf{H} \delta\boldsymbol{\theta} \tag{5}$$

Then, identifying the q-th weight in parameter $\boldsymbol{\theta}$ that minimizes the impact on the meta-objective can be formulated as the following optimisation problem:

$$\min_{q} \quad \frac{1}{2} \delta\boldsymbol{\theta}^{\top} \mathbf{H} \delta\boldsymbol{\theta} \quad \text{s.t.} \quad \mathbf{e}_q^{\top} \delta\boldsymbol{\theta} + \boldsymbol{\theta}_q = 0 \tag{6}$$

where $\mathbf{e}_q$ is the unit vector whose q-th element is 1 and otherwise 0, and $\boldsymbol{\theta}_q$ is the same as $\boldsymbol{\theta}$ except that the q-th element is set to 0. Forming a Lagrange from (6) as in [15], we find a closed-form solution for $\delta\boldsymbol{\theta}$

$$\delta\boldsymbol{\theta} = -\frac{\boldsymbol{\theta}_q}{[\mathbf{H}^{-1}]_{qq}} \mathbf{H}^{-1} \mathbf{e}_q \tag{7}$$

and the corresponding minimal change in the meta-objective

$$\Delta \mathcal{L}_m = \frac{1}{2} \frac{\boldsymbol{\theta}_q^2}{[\mathbf{H}^{-1}]_{qq}} \tag{8}$$

This term is also considered as the importance of the q-th element (weight) in the parameter vector $\boldsymbol{\theta}$.

Challenges to Compute Hessian. From (8), it seems that the weight importance metric for pruning vanilla-trained and meta-trained networks share the same form, except that the Hessian matrix is defined on the meta-objective rather than the traditional loss function. We show that naively calculating the Hessian involves third order derivatives with respect to $\boldsymbol{\theta}$, which is computation-intensive for deep neural networks.

Assume $\boldsymbol{\theta} \in \mathbb{R}^d$. Then $\mathbf{H} \in \mathbb{R}^{d \times d}$ and each element $H_{m,n}$ in $\mathbf{H}$ is computed as

$$H_{m,n} = \frac{\partial^2 \mathcal{L}_m}{\partial \theta_m \, \partial \theta_n} \tag{9}$$

$$= \frac{1}{M} \sum_{i=1}^{M} \frac{\partial^2 \mathcal{L}(\boldsymbol{\theta'}^i, \mathcal{D}_i)}{\partial \theta_m \, \partial \theta_n} \tag{10}$$

$$= \frac{1}{M} \sum_{i=1}^{M} \frac{\partial^2 \mathcal{L}\left(\boldsymbol{\theta} - \alpha \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}, \mathcal{D}_i\right), \mathcal{D}_i\right)}{\partial \theta_m \, \partial \theta_n} \tag{11}$$

where (10) and (11) simply substitute $\mathcal{L}_m$ and $\boldsymbol{\theta'}^i$ with (2) and (1).

From (11), each element in $\mathbf{H}$ requires computing third order derivatives with respect to $\boldsymbol{\theta}$.

3.3 Approximation of Derivatives

We avoid computing the third order derivatives in (11) by approximating them as follows.

Since $\boldsymbol{\theta'}^i$ is a function of θ_m and θ_n, we apply the Faà di Bruno's formula [14] to (10):

$$H_{m,n} = \frac{1}{M} \sum_{i=1}^{M} \frac{\partial^2 \mathcal{L}(\boldsymbol{\theta'}^i, \mathcal{D}_i)}{\partial \theta_m \, \partial \theta_n}$$

$$= \frac{1}{M} \sum_{i=1}^{M} \left(\sum_{k} \frac{\partial \mathcal{L}(\boldsymbol{\theta'}^i, \mathcal{D}_i)}{\partial \theta'^i_k} \frac{\partial^2 \theta'^i_k}{\partial \theta_m \, \partial \theta_n} + \sum_{k,l} \frac{\partial^2 \mathcal{L}(\boldsymbol{\theta'}^i, \mathcal{D}_i)}{\partial \theta'^i_k \, \partial \theta'^i_l} \frac{\partial \theta'^i_k}{\partial \theta_m} \frac{\partial \theta'^i_l}{\partial \theta_n} \right) \tag{12}$$

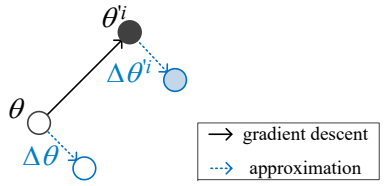


Figure 2: Derivative approximation in (17). A small change $\Delta\theta$ in θ induces approximately the same change $\Delta\theta'^i$ as in θ'^i. It especially well for a well-meta-trained network, as $\nabla_\theta \mathcal{L}_m$ is small while $\nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i)$ remains large.

From (1), we know

$$\frac{\partial \theta'^i_k}{\partial \theta_m} = \begin{cases} 1 - \alpha \frac{\partial^2 \mathcal{L}(\theta, \mathcal{D}_i)}{\partial \theta_k \, \partial \theta_m}, & \text{if } k = m, \\ -\alpha \frac{\partial^2 \mathcal{L}(\theta, \mathcal{D}_i)}{\partial \theta_k \, \partial \theta_m}, & \text{if } k \neq m \end{cases} \tag{13}$$

Omitting the second derivative $\partial^2 \mathcal{L}(\theta, \mathcal{D}_i) \big/ \partial \theta_k \, \partial \theta_m$, we obtain the first order approximation for $\partial \theta'^i_k \big/ \partial \theta_m$

$$\frac{\partial \theta'^i_k}{\partial \theta_m} \approx \begin{cases} 1, & \text{if } k = m, \\ 0, & \text{if } k \neq m \end{cases} \tag{14}$$

And therefore,

$$\frac{\partial^2 \theta'^i_m}{\partial \theta_m \, \partial \theta_n} = 0 \tag{15}$$

The first term in (12) is therefore vanished, and according to (14), we can also simplify the second term with

$$\sum_{k,l} \frac{\partial^2 \mathcal{L}(\theta'^i, \mathcal{D}_i)}{\partial \theta'^i_k \, \partial \theta'^i_l} \frac{\partial \theta'^i_k}{\partial \theta_m} \frac{\partial \theta'^i_l}{\partial \theta_n} = \frac{\partial^2 \mathcal{L}(\theta'^i, \mathcal{D}_i)}{\partial \theta'^i_m \, \partial \theta'^i_n} \frac{\partial \theta'^i_m}{\partial \theta_m} \frac{\partial \theta'^i_n}{\partial \theta_n} = \frac{\partial^2 \mathcal{L}(\theta'^i, \mathcal{D}_i)}{\partial \theta'^i_m \, \partial \theta'^i_n} . \tag{16}$$

which finally leads to

$$H_{m,n} \approx \frac{1}{M} \sum_{i=1}^{M} \frac{\partial^2 \mathcal{L}(\theta'^i \mathcal{D}_i)}{\partial \theta'^i_m \, \partial \theta'^i_n} . \tag{17}$$

In summary, (17) approximates the computation of the third order derivatives w.r.t. θ by calculating the second derivatives w.r.t. θ'^i.

Understanding the Derivatives Approximation. The approximation used for reducing the computation of third derivatives to second derivatives relies on (14), which suggests that a small change in the *pre-adaptation* parameters θ leads to approximately the same small change in the *post-adaptation* parameters θ'^i (see Fig. 2). Such an approximation is reasonable because for a well-meta-trained network, $\nabla_\theta \mathcal{L}_m$ is small as the network nearly converged. Meanwhile, $\nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i)$ is large, as the tasks can substantially differ from each other. In essence, (14) performs a first order approximation by omitting $\partial^2 \mathcal{L}(\theta, \mathcal{D}_i) \big/ \partial \theta_k \, \partial \theta_m$. Such an approach is supported by observations in other studies that the second derivatives are usually close to nought [11]. Prior studies [8, 24] have also shown that first order approximations can be as effective as full second derivatives in meta-learning.

3.4 Layer-Wise Pruning

To further reduce the computation, we adapt the layer-wise approach for pruning vanilla-trained networks [6] and expand it to the pruning of meta-trained networks.

Layer-Wise Meta-Objective. The pre-activation output vector of the l-th layer is denoted as $\mathbf{y}_l$, and the post-activation output vector is denoted as $\mathbf{z}_l = \sigma(\mathbf{y}_l)$, where $\sigma\cdot$ is the activation function. We use a layer-wise loss function

$$\mathcal{L}^l(\theta, \mathcal{D}_i) = \frac{1}{K} \sum^{K} ||\hat{\mathbf{y}}_l - \mathbf{y}_l||^2 \tag{18}$$

where $\hat{\mathbf{y}}_l$ is the pre-activation output after the pruning, K is the number of training samples, and $||\cdot||$ is the l^2-norm. Note that the summation is over all training samples from the same task. The layer-wise meta-objective is then

$$\mathcal{L}^l_m = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}^l(\theta'^i, \mathcal{D}_i) \tag{19}$$

Layer-Wise Hessian. As the network is pruned layer by layer, we need only the Hessian of $\mathcal{L}^l_m$ w.r.t. θ_l, which is the equivalent of θ for the l-th layer. From (17), we find

$$\mathbf{H}_l \approx \frac{\partial^2 \mathcal{L}^l_m}{\partial (\theta'^i_l)^2} = \frac{1}{M} \sum_{i=1}^{M} \frac{\partial^2 \mathcal{L}^l(\theta'^i, \mathcal{D}_i)}{\partial (\theta'^i_l)^2} . \tag{20}$$

where θ'^i_l is the equivalent of θ'^i (see (1)) for the l-th layer.

Similar to [6], $\mathbf{H}_l$ is a block diagonal square matrix with each diagonal blocks being $\mathbf{H}_{l_{kk}} = \partial^2 \mathcal{L}^l_m \big/ \partial (\theta'^i_{l_k})^2$, where $\theta'^i_{l_k}$ are the vectorised incoming weights of the k-th neuron in the l-th layer. All blocks $\mathbf{H}_{l_{kk}}$ are identical and can be calculated as

$$\mathbf{H}_{l_{kk}} = \frac{1}{M} \frac{1}{K} \sum_{i=1}^{M} \sum^{K} \mathbf{z}_{l-1} \cdot (\mathbf{z}_{l-1})^\top \tag{21}$$

Efficient Computing of Inverse of the Hessian. As shown in (7) and (8), we need the inverse of the Hessian $\mathbf{H}_l^{-1}$, a block diagonal square matrix with its diagonal blocks being $\mathbf{H}_{l_{kk}}^{-1}$. In ANP, we calculate $\mathbf{H}_l^{-1}$ recursively over the training samples of all M tasks using the Sherman–Morrison–Woodbury formula [18]

$$\begin{aligned} \mathbf{H}^{-1}_{l_{kk},j} &= \mathbf{H}^{-1}_{l_{kk},j-1} - \frac{\mathbf{H}^{-1}_{l_{kk},j-1} \cdot \mathbf{z}_{l-1,j} \cdot \mathbf{z}^\top_{l-1,j} \cdot \mathbf{H}^{-1}_{l_{kk},j-1}}{M \cdot K + \mathbf{z}^\top_{l-1,j} \cdot \mathbf{H}^{-1}_{l_{kk},j-1} \cdot \mathbf{z}_{l-1,j}} \\ \text{with} \quad & \mathbf{H}^{-1}_{l_{kk},0} = \alpha^{-1}\mathbf{I} \text{ and } \mathbf{H}^{-1}_{l_{kk},M\cdot K} = \mathbf{H}^{-1}_{l_{kk}} \end{aligned} \tag{22}$$

where $\alpha \in [10^{-8}, 10^{-4}]$ is a small constant to make $\mathbf{H}^{-1}_{l_{kk},0}$ meaningful and to which the method is insensitive [15]. Note that the two summations in (21) are integrated in (22), as the iteration goes through all $M \cdot K$ samples.

3.5 Putting It Together

Algorithm 1 outlines the process of ANP for K-shot learning. The pruning first begins after the network is well-meta-trained (Line 2). As in MAML, a batch of M tasks are sampled from a given distribution $p(\mathcal{T})$ (Line 4), and K data-points are sampled from each task (Line 6). Then the post-adaptation θ'^i is calculated with gradient descent (1) for each task. As the training samples and post-adaptation weights are ready, the inverse Hessian can be recursively calculated with (22) (Line 9). The pruning is done iteratively (Line 3). We use a tuning parameter β_l to control the proportion of weights to be pruned in each iteration. The importance $\Delta\mathcal{L}_m$ of each weight is assessed with (8) (Line 10), and β_l of the least important weights in each layer are removed, while the remaining weights are updated

Algorithm 1: Adaptation-aware Network Pruning

Input: $p(\mathcal{T})$: distribution over tasks
α: a small constant ($10^{-8} \leq \alpha \leq 10^{-4}$)
β_l: pruning step size hyper-parameter ($0 < \beta_l < 1$)
γ_{pr}: pruning ratio ($0 < \gamma_{pr} < 1$)
Output: Sparse network

```
randomly initialise weights θ
meta-train the network until the meta-objective converges
while required γ_pr not achieved do
    sample a batch of M tasks T_i ~ p(L)
    for each task T_i do
        sample K data-points from T_i and form datasets D_i
        compute post-adaptation parameters θ'^i with Eq. (1)
    end
    for all layers do
        calculate H^{-1}_{l_kk} recursively using Eq. (22)
        calculate δθ and ΔL_m for each weight using Eq. (7)
          and Eq. (8), respectively
        prune β_l of the weights with the least ΔL_m, and
          update the rest with δθ
    end
    meta-train the network again until the meta-objective
      converges, such that the performance is re-boosted
end
return the sparse network
```

using those $\delta\boldsymbol{\theta}$ calculated with (7). Combining derivatives approximation (Sec. 3.3) and layer-wise pruning approach (Sec. 3.4), ANP is able to prune meta-trained neural network effectively.

Extensions beyond MAML. It is worth mentioning that ANP is not restricted to MAML. Here we briefly explain how to extend Algorithm 1 to two popular variants of MAML: CAVIA [30], an improvement of MAML with context parameters, and Reptile [24], the first order simplification of MAML. On CAVIA, as the number of the context parameters is limited, we can apply Algorithm 1 only to the weights *i.e.*, non-context parameters in the initial architecture. On Reptile, the meta-objective (3) is already omitted. We compute an averaged importance of weights in the inner loop, then prune the least important weights as in Algorithm 1.

4 EVALUATION

This section presents the evaluations of our method.

4.1 Experimental Settings

Metrics. Since we aim at pruning meta-trained networks without sacrificing their ability of fast adaptation, we compare different methods with the following metrics:

- *Pruning Ratio (PR)*: the ratio of pruned parameters to the original parameters of the initial architecture.
- *Few-Shot Accuracy*: the few-shot classification accuracy.

In experiments where many testing tasks are available, we conduct multi-run testing by repeatedly selecting 5 random tasks for the few-shot learning test. In Fig. 3, Fig. 4, Fig. 5 and Fig. 6, we use error bars to represent the standard deviation over multiple runs.

Datasets. We use two standard few-shot classification benchmarks.

- Mini-ImageNet [25]: it is a dataset for image classification, which contains 60,000 colour images with 100 classes, each having 600 images of size 84×84. The dataset is split into 64 training classes, 12 validation classes, and 24 test classes as [8, 30]. We use 5-way 1-shot and 5-way 5-shot settings.
- Caltech-UCSD Birds-200-2011 (CUB) [27]: it is a dataset for fine-grained classification, which contains 11,788 images of 200 bird species, each having about 60 images. The dataset is split into 100 training classes, 50 validation classes, and 50 test classes and the images are resized to 84×84 as [3]. We use the 5-way 1-shot setting.

Initial Architectures. We use two common initial architectures from gradient-based meta-learning literature [8, 9, 24, 30].

- ConvNet-4: it consists of 4 layers with 3×3 convolutions followed by batch normalisation, ReLU, and 2×2 max-pooling. We use 32-filter convolutions for evaluations as in [3, 8, 30].
- ResNet-12: it consists of 4 residual blocks, each containing three 3×3 convolutional layers [3, 9]. In each residual block, the first two convolution layers are followed by batch normalisation and ReLu, and the last convolution layer is followed by batch normalisation and a skip connection. A 2×2 max-pooling is used after each residual block. The number of filters in each residual block is 64, 128, 256, and 512.

Methods for Meta-Training. By default, the weights is meta-trained by MAML [8]. We also compare ANP with baselines using the more advanced CAVIA method [30] and the popular first-order method Reptile [24]. The context parameters in CAVIA follow the default settings, which is a 100 dimensional vector initialized as 0 before each adaptation step and update during the inner training loops. For Reptile, we follow the hyper-parameter settings in [24]. Table 1 summarises the hyperparameters to meta-train the initial architecture via MAML, CAVIA and Reptile.

All the experiments use Adam optimizer for meta training and SGD optimizer for inner training loops with default hyperparameters. When optimizing the initial architectures, the pruned weights are set as zero and their gradients are masked by point-wise production with a zero-one matrix.

Baselines. Since ANP is a pruning scheme for meta-trained networks, we compare it with two existing pruning methods.

- Magnitude [13]: a classic pruning method that removes network weights based on their magnitude.
- L-OBS [6]: a representative Hessian-based pruning strategy that removes weights by a layer-wise Hessian-based metric.

In contrast to ANP, Magnitude and L-OBS are originally designed for single-task pruning. Therefore, one *pruning target task* has to be given for pruning, then the algorithms assess and prune the weights based on their importance to this pruning target task. In our experiments, we randomly select a pruning target tasks. During each pruning iteration, about 10% weights in each layer are pruned, and the pruned model is retrained for 40 epochs. We repeat this step-wise pruning iteration until a desired pruning ratio is reached.

Table 1: Hyperparameter setup for meta-training.

Method	Backbone	Dataset	Setup	Inner/Outer LR	Meta Epoch	LR Decay	Meta Batch Size	# Update Step
MAML [8]	ConvNet-4	Mini-ImageNet	5-way, 1-shot	$1.0 \times 10^{-2}/1.0 \times 10^{-3}$	80,000	-	4	5
			5-way, 5-shot	$1.0 \times 10^{-2}/1.0 \times 10^{-3}$	80,000	-	4	5
		CUB	5-way, 1-shot	$1.0 \times 10^{-2}/1.0 \times 10^{-3}$	80,000	-	4	5
			5-way, 5-shot	$1.0 \times 10^{-2}/1.0 \times 10^{-3}$	80,000	-	4	5
	ResNet-12	Mini-ImageNet	5-way, 1-shot	$1.0 \times 10^{-2}/1.0 \times 10^{-3}$	100,000	-	4	5
		CUB	5-way, 1-shot	$1.0 \times 10^{-2}/1.0 \times 10^{-3}$	100,000	-	4	5
CAVIA [30]	ConvNet-4	Mini-ImageNet	5-way, 1-shot	$1.0 \times 10^{-2}/1.0$	200,000	0.9	16	2
		CUB	5-way, 1-shot	$1.0 \times 10^{-3}/1.0$	200,000	0.9	16	2
Reptile [24]	ConvNet-4	Mini-ImageNet	5-way, 1-shot	$1.0 \times 10^{-2}/1.0$	100,000	1.0×10^{-5}	5	50
		CUB	5-way, 1-shot	$1.0 \times 10^{-2}/1.0$	100,000	1.0×10^{-5}	5	50

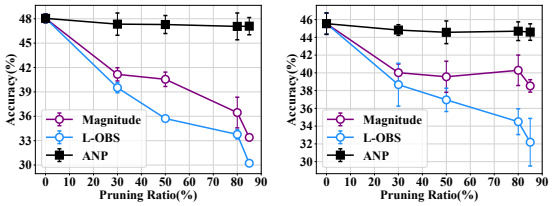
4.2 Main Experimental Results

Results Organisation. We have conducted extensive experiments and compared the performance of ANP with the two baselines on varieties of initial architectures (ConvNet-4 and ResNet-12) , meta-training methods (MAML, CAVIA and Reptile), datasets (Mini-ImageNet and CUB) and FSL setups (5-way 1-shot and 5-way 5-shot). In general, we conducted five experiments and their results are organised as follows:

- **Exp. 1:** We first use ConvNet-4 as initial architecture, MAML as meta-training methods, and 5-way 1-shot as the FSL setup. The comparison of ANP with the baselines is shown in Fig. 3, and the detailed results are listed in Table 2 & Table 3.
- **Exp. 2:** On the basis of Exp. 1, we change the few-shot learning setup to 5-way 5-shot. Results are shown in Fig. 4 and details in Table 4 & Table 5.
- **Exp. 3:** On the basis of Exp. 1, we change the initial architecture to the more complicated ResNet-12. Results are shown in Fig. 5 and details in Table 6 & Table 7.
- **Exp. 4:** On the basis of Exp. 1, we change the meta-training method to the more advanced CAVIA. Results are shown in Fig. 6 and details in Table 8 & Table 9.
- **Exp. 5:** Finally, similar to Exp. 4, we change the meta-training method to the popular first-order method Reptile. All other settings are the same as Exp. 1. Results are shown in Fig. 7 and details in Table 10 & Table 11.

All the aforementioned experiments are conducted on both the Mini-ImageNet and CUB dataset. The error bars in the figures represent the standard error over different tasks chosen for few-shot learning.

Overall Performance. Inducing a decrease in few-shot accuracy no more than 1%, our ANP achieves a pruning ratio of at least 85% across all the initial architectures, datasets and meta-training methods. In comparison, given a pruning ratio of 85%, Magnitude induces a drop of 7.01% to 19.80% in few-shot accuracy, and L-OBS introduces a drop of 10.05% to 26.70% in few-shot accuracy. In fact, the baselines already induce over 5.16% loss in few-shot accuracy at a pruning ratio of 30% even in the best case (5-way 1-shot on CUB, ResNet-12 as initial architecture, pruned by Magnitude and meta-trained by MAML).



(a) Mini-ImageNet **(b) CUB**

Figure 3: 5-way 1-shot accuracy vs. PR of ConvNet-4 meta-trained by MAML and pruned by Magnitude, L-OBS, or ANP on (a) Mini-ImageNet and (b) CUB.

Table 2: 5-way, 1-shot accuracy vs. PR of ConvNet-4 meta-trained by MAML and pruned on Mini-ImageNet.

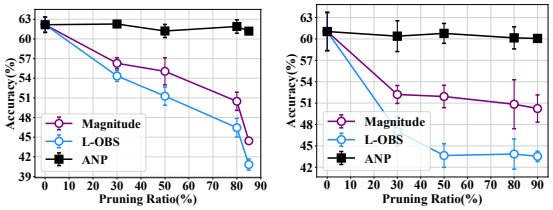
PR (%)	Mini-ImageNet Accuracy (%)		
	Magnitude	L-OBS	ANP
0	48.05 ± 0.0053		
30	41.16 6.89↓ ± 0.0081	39.52 8.53↓ ± 0.0063	47.34 0.71↓ ± 0.0136
50	40.55 7.50↓ ± 0.0088	35.72 12.33↓ ± 0.0027	47.29 0.76↓ ± 0.0111
80	36.46 11.59↓ ± 0.0189	33.77 14.28↓ ± 0.0057	47.06 0.99↓ ± 0.0165
85	33.40 14.65↓ ± 0.0036	30.22 17.83↓ ± 0.0034	47.09 0.96↓ ± 0.0106

Takeaways. ANP significantly and consistently outperforms both baselines across different scenarios (1-shot or 5-shot), regardless of initial architectures (ConvNet-4 or ResNet-12) , meta-training methods (MAML, CAVIA or Reptile) and datasets (Mini-ImageNet or CUB). It turns out that ANP can find a *topology* that is meta-optimised for potential new tasks and, combined with existing meta-learning methods which find the meta-optimised *weights*, provides in the end a compact network for fast adaptation.

Observations and Comments. From the aforementioned experiment results, we made the following observations. *(i)* L-OBS generally performs worse on meta-learning than Magnitude. The reason may be that L-OBS is able to find a topology more specialised for

Table 3: 5-way, 1-shot accuracy vs. PR of ConvNet-4 meta-trained by MAML and pruned on CUB.

PR (%)	CUB Accuracy (%)		
	Magnitude	L-OBS	ANP
0	45.54 ± 0.0120		
30	40.02 5.52↓ ± 0.0098	38.67 6.87↓ ± 0.0241	44.82 0.72↓ ± 0.0060
50	39.56 5.98↓ ± 0.0175	36.96 8.58↓ ± 0.0131	44.57 0.97↓ ± 0.0128
80	40.29 5.26↓ ± 0.0171	34.50 11.04↓ ± 0.0145	44.69 0.85↓ ± 0.0105
85	38.53 7.01↓ ± 0.0070	32.19 13.35↓ ± 0.0268	44.59 0.95↓ ± 0.0093



(a) Mini-ImageNet (b) CUB

Figure 4: 5-way 5-shot accuracy vs. PR of ConvNet-4 meta-trained by MAML and pruned by Magnitude, L-OBS, or ANP on (a) Mini-ImageNet and (b) CUB.

Table 4: 5-way, 5-shot accuracy vs.PR of ConvNet-4 meta-trained by MAML and pruned on Mini-ImageNet.

PR (%)	Accuracy (%)		
	Magnitude	L-OBS	ANP
0	62.17 ± 0.0117		
30	56.28 5.89↓ ± 0.0085	54.35 7.82↓ ± 0.0083	62.28 0.11↑ ± 0.0037
50	55.05 7.12↓ ± 0.0207	51.27 10.90↓ ± 0.0139	61.21 0.96↓ ± 0.0100
80	50.48 11.69↓ ± 0.0138	46.45 15.72↓ ± 0.0143	61.90 0.27↓ ± 0.0103
85	44.44 17.73↓ ± 0.0045	40.82 21.35↓ ± 0.0085	61.19 0.98↓ ± 0.0020

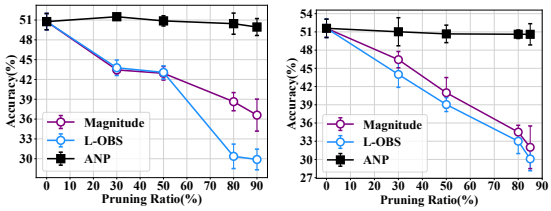
the "target task" than Magnitude, which leads to actually worse FSL performance. This is further discussed in Sec. 4.3. *(ii)* When used with Reptile, the first-order approximation of ANP is not required. The significant advantage against the baselines still persists, which is the consequence of the better optimisation target of ANP: finding the topology optimised for all potential new tasks instead of the single "target task". *(iii)* In some cases with low pruning ratio (30% to 50%), ANP is even capable of slightly improving the FSL accuracy. This is due to the generalisation effect known for network pruning.

4.3 Ablation Study

Why Baseline Pruning Methods Fail. As mentioned in Sec. 4.1, the baseline methods require one pruning target task during the pruning. Although the pruned networks are meta-trained and therefore their weights are considered to be optimised for all the potential new tasks, their topology, constructed via Magnitude or L-OBS, is biased to the pruning target task, which leads to sub-optimal performance of fast adaptation. To illustrate this bias, we randomly selected a 5-way "target task" from the Mini-ImageNet dataset, then train & prune via both baseline methods (Magnitude and L-OBS) ConvNet-4 backbones up to a compression ratio of 85%. These compressed models are then tested for few-shot accuracy on not only the aforementioned "target task", but also the so-called "other tasks", which are tasks again randomly selected from the dataset. Note that none of the "other tasks" has been used during the pruning. Fig. 8 shows the training accuracy curves on the "target task" and "other tasks" when performing 5-way, 1-shot learning tests. The few-shot accuracy results on the multiple "other tasks" are aggregated into

Table 5: 5-way, 5-shot accuracy vs.PR of ConvNet-4 meta-trained by MAML and pruned on CUB.

PR (%)	Accuracy (%)		
	Magnitude	L-OBS	ANP
0	61.04 ± 0.0268		
30	52.21 8.83↓ ± 0.0208	47.07 13.97↓ ± 0.0125	60.39 0.65↓ ± 0.0216
50	51.92 9.12↓ ± 0.0166	43.64 17.40↓ ± 0.0157	60.78 0.26↓ ± 0.0140
80	50.84 10.20↓ ± 0.0212	43.85 17.19↓ ± 0.0344	60.15 0.89↓ ± 0.0155
90	50.24 10.80↓ ± 0.0073	43.52 17.52↓ ± 0.0191	60.07 0.97↓ ± 0.0036



(a) Mini-ImageNet (b) CUB

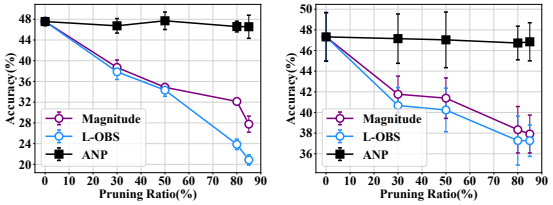
Figure 5: 5-way 1-shot accuracy vs. PR of ResNet-12 meta-trained by MAML and pruned by Magnitude, L-OBS, or ANP on (a) Mini-ImageNet and (b) CUB.

Table 6: 5-way, 1-shot accuracy vs. PR of ResNet-12 meta-trained by MAML and pruned on Mini-ImageNet.

PR (%)	Accuracy (%)		
	Magnitude	L-OBS	ANP
0	50.76 ± 0.0123		
30	43.45 7.31↓ ± 0.0058	43.77 6.99↓ ± 0.0117	51.50 0.74↑ ± 0.0028
50	42.93 7.83↓ ± 0.0102	43.07 7.69↓ ± 0.0098	50.87 0.11↑ ± 0.0077
80	38.63 12.13↓ ± 0.0138	30.36 20.40↓ ± 0.0185	50.45 0.31↓ ± 0.0160
90	36.60 14.16↓ ± 0.0242	29.89 20.87↓ ± 0.0156	49.94 0.82↓ ± 0.0128

Table 7: 5-way, 1-shot accuracy vs. PR of ResNet-12 meta-trained by MAML and pruned on CUB.

PR (%)	Accuracy (%)		
	Magnitude	L-OBS	ANP
0	51.59 ± 0.0151		
30	46.43 5.16↓ ± 0.0134	44.00 7.59↓ ± 0.0213	51.02 0.57↓ ± 0.0231
50	40.98 10.61↓ ± 0.0250	39.04 12.55↓ ± 0.0115	50.67 0.92↓ ± 0.0144
80	34.50 17.09↓ ± 0.0112	33.00 18.59↓ ± 0.0203	50.61 0.98↓ ± 0.0070
85	32.00 19.59↓ ± 0.0351	30.09 21.50↓ ± 0.0195	50.60 0.99↓ ± 0.0175



(a) Mini-ImageNet (b) CUB

Figure 6: 5-way 1-shot accuracy vs. PR of ConvNet-4 meta-trained by CAVIA and pruned by Magnitude, L-OBS, or ANP on (a) Mini-ImageNet and (b) CUB.

Table 8: 5-way, 1-shot accuracy vs. PR of ConvNet-4 meta-trained by CAVIA and pruned on Mini-ImageNet.

PR (%)	Accuracy (%)		
	Magnitude	L-OBS	ANP
0	47.56 ± 0.0076		
30	38.71 8.85↓ ± 0.0145	37.86 9.70↓ ± 0.0148	46.74 0.82↓ ± 0.0140
50	34.87 12.69↓ ± 0.0072	34.32 13.24↓ ± 0.0118	47.71 0.15↑ ± 0.0169
80	32.14 15.42↓ ± 0.0060	23.86 23.70↓ ± 0.0100	46.59 0.97↓ ± 0.0103
85	27.76 19.80↓ ± 0.0156	20.86 26.70↓ ± 0.0099	46.57 0.99↓ ± 0.0223

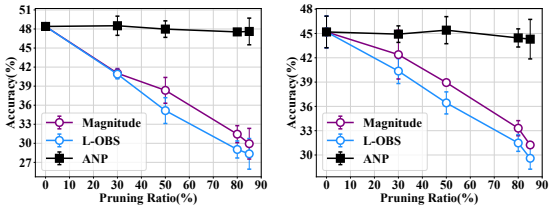
one line with error bars. As we can see, the accuracy on the pruning "target task" is high, but on "other tasks", which may notably differ from the "target task", the accuracy is low.

Few-Shot Learning: Convergence. Fig. 9 plots the few-shot training loss and few-shot accuracy curves of a meta-trained & pruned initial architecture to perform few-shot learning on test tasks. ANP provides not only a better accuracy, but also a faster convergence. This is a crucial benefit for resource-constrained devices, as with fewer training steps the computation cost and power consumption can be reduced during adaptation. Moreover, in some experiments (*e.g.*, the loss curve for Magnitude in Fig. 9(a)), the training loss curve of the baseline methods may even not converge. In contrast, ANP provides consistent and stable convergence.

Fig. 10, Fig. 11 and Fig. 12 show the loss and accuracy curves of a meta-trained & pruned ConvNet-4 to perform 5-way, 1-shot

Table 9: 5-way, 1-shot accuracy vs. PR of ConvNet-4 meta-trained by CAVIA and pruned on CUB.

PR (%)	CUB Accuracy (%)		
	Magnitude	L-OBS	ANP
0	47.32 ± 0.0235		
30	41.77 5.55↓ ± 0.0176	40.67 6.65↓ ± 0.0173	47.15 0.17↓ ± 0.0239
50	41.39 5.93↓ ± 0.0196	40.24 7.08↓ ± 0.0211	47.03 0.29↓ ± 0.0269
80	38.33 8.99↓ ± 0.0225	37.28 10.04↓ ± 0.0237	46.73 0.59↓ ± 0.0163
85	37.91 9.41↓ ± 0.0183	37.26 10.05↓ ± 0.0152	46.85 0.47↓ ± 0.0185



(a) Mini-ImageNet (b) CUB

Figure 7: 5-way 1-shot accuracy vs. PR of ConvNet-4 meta-trained by Reptile and pruned by Magnitude, L-OBS, or ANP on (a) Mini-ImageNet and (b) CUB.

Table 10: 5-way, 1-shot accuracy vs. PR of ConvNet-4 meta-trained by Reptile and pruned on Mini-ImageNet.

PR (%)	Accuracy (%)		
	Magnitude	L-OBS	ANP
0	48.40 ± 0.0051		
30	41.03 7.37↓ ± 0.0074	40.89 7.51↓ ± 0.0213	48.51 0.11↑ ± 0.0151
50	38.34 10.06↓ ± 0.0203	35.15 13.25↓ ± 0.0084	47.98 0.42↓ ± 0.0130
80	31.55 16.85↓ ± 0.0134	29.03 19.37↓ ± 0.0093	47.54 0.86↓ ± 0.0034
85	29.94 18.46↓ ± 0.0241	28.34 20.06↓ ± 0.0124	47.61 0.79↓ ± 0.0210

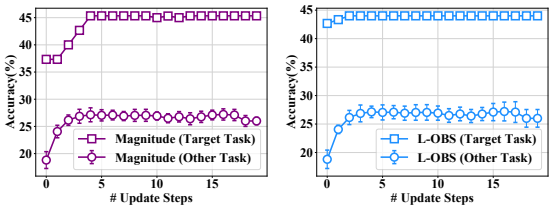
learning on Mini-ImageNet. The initial architecture is compressed to a pruning ratio of 30%, 50%, and 80%, respectively. The accuracy advantage of ANP against baselines increases as the PR increases as expected. Furthermore, the faster and more stable convergence with ANP compared to the baselines, which is observed on PR = 85%, can also be observed with PR = 30%, 50%, and 80%.

5 CONCLUSION

In this work, we explored pruning *meta-trained* deep neural networks for few-shot learning on resource-constrained platforms. Prior pruning methods deteriorate the fast adaptability of meta-trained networks due to inconsistent weight importance metrics with the meta-objective. In response, we propose ANP, the first meta-training-compatible network pruning scheme. ANP defines a weight importance metric for the meta-objective to find a topology

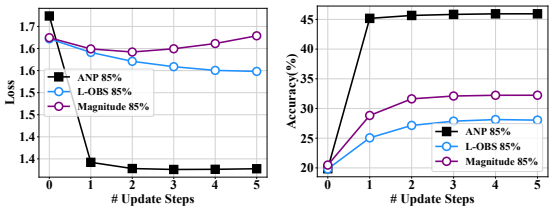
Table 11: 5-way, 1-shot accuracy vs. PR of ConvNet-4 meta-trained by Reptile and pruned on CUB.

PR (%)	Accuracy (%)		
	Magnitude	L-OBS	ANP
0	45.19 ± 0.0195		
30	42.39 2.8↓ ± 0.0301	40.36 4.83↓ ± 0.0154	44.92 0.27↓ ± 0.0102
50	38.94 6.25↓ ± 0.0013	36.43 8.76↓ ± 0.0136	45.41 0.22↑ ± 0.0167
80	33.31 11.88↓ ± 0.0093	31.48 13.71↓ ± 0.0103	44.45 0.74↓ ± 0.0112
85	31.24 13.95↓ ± 0.0035	29.59 15.60↓ ± 0.0134	44.30 0.89↓ ± 0.0244



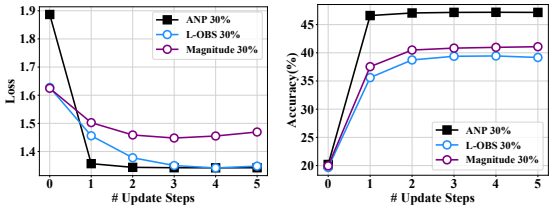
(a) Magnitude (b) L-OBS

Figure 8: 5-way 1-shot training accuracy curves on the task selected for pruning and other tasks of ConvNet-4 meta-trained by MAML and pruned with (a) Magnitude and (b) L-OBS on Mini-ImageNet.



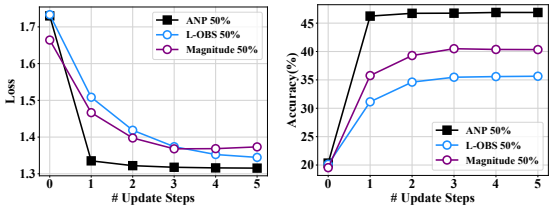
(a) Loss Curves (b) Accuracy Curves

Figure 9: 5-way 1-shot (a) training loss and (b) testing accuracy curve of ConvNet-4 meta-trained by MAML and pruned by Magnitude, L-OBS, or ANP on Mini-ImageNet (PR = 85%).



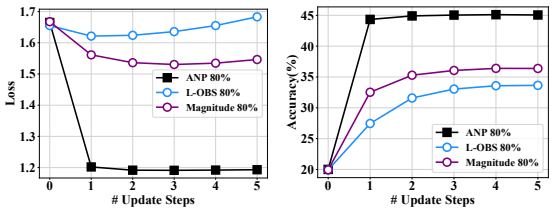
(a) Loss Curves (b) Accuracy Curves

Figure 10: 5-way 1-shot (a) training loss and (b) testing accuracy curve of ConvNet-4 meta-trained by MAML and pruned by Magnitude, L-OBS, or ANP on Mini-ImageNet (PR = 30%).



(a) Loss Curves (b) Accuracy Curves

Figure 11: 5-way 1-shot (a) training loss and (b) testing accuracy curve of ConvNet-4 meta-trained by MAML and pruned by Magnitude, L-OBS, or ANP on Mini-ImageNet (PR = 50%).



(a) Loss Curves (b) Accuracy Curves

Figure 12: 5-way 1-shot (a) training loss and (b) testing accuracy curve of ConvNet-4 meta-trained by MAML and pruned by Magnitude, L-OBS, or ANP on Mini-ImageNet (PR = 80%).

meta-optimised for learning new tasks in few shots. Evaluations on few-shot classification benchmarks show that ANP can prune MAML-like meta-trained convolutional and residual backbones by 85% with a minimal drop in few-shot classification accuracy. We envision our work will offer guidelines to enable fast model adaptation on low-resource platforms.

ACKNOWLEDGEMENT

We are grateful to anonymous reviewers for their constructive comments. Dawei Gao and Yongxin Tong's work are partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100, the National Science Foundation of China (NSFC) under Grant Nos. 61822201, U1811463 and 62076017, the CCF-Huawei Database System Innovation Research Plan No. CCF-HuaweiDBIR2020008B, and the State Key Laboratory of Software Development Environment Open Funding No. SKLSDE-2020ZX-07. Xiaoxi He and Lothar Thiele's work was supported by the Swiss National Science Foundation in the context of the NCCR Automation. Zimu Zhou's research was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant. Zimu Zhou is the corresponding author.

REFERENCES

[1] Sébastien Arnold, Shariq Iqbal, and Fei Sha. 2021. When MAML Can Adapt Fast and How to Assist When It Cannot. In *Proceedings of International Conference on Artificial Intelligence and Statistics*. PMLR, 244–252.

[2] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. 2021. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems* 3 (2021).

[3] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. 2019. A closer look at few-shot classification. In *Proceedings of International Conference on Learning Representations*.

[4] Bin Dai, Chen Zhu, Baining Guo, and David Wipf. 2018. Compressing neural networks using the variational information bottleneck. In *Proceedings of International Conference on Machine Learning*. ACM, New York, NY, USA, 1135–1144.

[5] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: a comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532.

[6] Xin Dong, Shangyu Chen, and Sinno Pan. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 4860–4874.

[7] Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. 2020. Meta-learning of neural architectures for few-shot learning. In *Proceedings of Conference on Computer Vision and Pattern Recognition*. IEEE Press, Piscataway, NJ, USA, 12365–12375.

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of International Conference on Machine Learning*. ACM, New York, NY, USA, 1126–1135.

[9] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. 2018. Bilevel programming for hyperparameter optimization and meta-learning. In *Proceedings of International Conference on Machine Learning*. ACM, New York, NY, USA, 1568–1577.

[10] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. arXiv:1412.6572

[12] Mary Gooneratne, Khe Chai Sim, Petr Zadrazil, Andreas Kabel, Françoise Beaufays, and Giovanni Motta. 2020. Low-rank gradient approximation for memory-efficient on-device training of deep neural network. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*. IEEE Press, Piscataway, NJ, USA, 3017–3021.

[13] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of International Conference on Learning Representations*.

[14] Michael Hardy. 2006. Combinatorics of partial derivatives. *The Electronic Journal of Combinatorics* 13, R1 (2006), 1.

[15] Babak Hassibi and David G. Stork. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 164–171.

[16] Xiaoxi He, Zimu Zhou, and Lothar Thiele. 2018. Multi-task zipping via layer-wise neuron sharing. In *Proceedings of Advances in Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 6016–6026.

[17] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. 2020. Meta-learning in neural networks: A survey. arXiv:2004.05439

[18] Thomas Kailath. 1980. *Linear Systems*. Vol. 156. Prentice-Hall Englewood Cliffs, NJ.

[19] Yann Le Cun, John S. Denker, and Sara A. Solla. 1990. Optimal brain damage. In *Advances in Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 598–605.

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

[21] Tao Lin, Sebastian U Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. 2020. Dynamic model pruning with feedback. In *Proceedings of International Conference on Learning Representations*.

[22] Jie Liu, Jiawen Liu, Wan Du, and Dong Li. 2019. Performance analysis and characterization of training deep learning models on mobile device. In *Proceedings of International Conference on Parallel and Distributed Systems*. IEEE Press, Piscataway, NJ, USA, 506–515.

[23] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. 2019. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of International Conference on Computer Vision*. IEEE Press, Piscataway, NJ, USA, 3296–3305.

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

[25] Sachin Ravi and Hugo Larochelle. 2017. Optimization as a model for few-shot learning. In *Proceedings of International Conference on Learning Representations*.

[26] Hongduan Tian, Bo Liu, Xiao-Tong Yuan, and Qingshan Liu. 2020. Meta-learning with network pruning. In *Proceedings of European Conference on Computer Vision*. Springer, Berlin, Germany, 675–700.

[27] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. 2011. *The caltech-ucsd birds-200-2011 dataset*. Technical Report. California Institute of Technology.

[28] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *Comput. Surveys* 53, 3 (2020), 1–34.

[29] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv:1710.01878

[30] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. 2019. Fast context adaptation via meta-learning. In *Proceedings of International Conference on Machine Learning*. ACM, New York, NY, USA, 7693–7702.