Task-Specific Meta Adaptive Learning

ABSTRACT

The gradient-based meta-learning is popular for few-shot learning problems. However, in the few-data scenario, the fast adaptation for meta-learning to unseen data still remains a challenge. The neurobiology research reveals that the capability of modifying neural functions according to specific tasks is essential for task adaptation, which has been overlooked in most of the meta-learning approaches. Motivated by this, we design a task-specific block to re-calibrate the model parameter with the context of tasks explicitly. Notably, we introduce a novel task-specific meta adaptive optimization where an adaptive function is deployed into backward optimization of neural networks. Generally, the proposed method consists of two components: the *backward* refers to conduct an adaptive back-propagation according to the context, and *meta* refers to evolve the structure of neural network so that the predictive function can be changed dynamically with respect to individual tasks. Moreover, the proposed meta-block is lightweight and applicable to various neural architectures. Extensive empirical experiments on the challenging few-shot learning benchmarks validate that our approach trained with task-aware meta adaptive learning scheme achieves promising performance.

ACM Reference Format:
. 2021. Task-Specific Meta Adaptive Learning. In *WWW '21: The Web Conference 2021, April 19 – 23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Learning to fast adapt from few-shot instances is an essential ability to artificial intelligence. Such a task is a challenge to deep neural networks, as deep learning relies heavily on large-scale labeled data. In contrast, there is a gap with the human's ability to learn, where the human can draw inference about new cases from prior experience. Such ability in fast adaptation has been realized in existing work. Transfer learning [23] exploits source tasks to train a base-learner and then fine-tune the model to learn the target tasks via the knowledge transferred from related tasks. Multi-task learning (MTL) [3] is regarded as a form of inductive transfer. The inductive bias is distilled by the auxiliary tasks and derives hypotheses across tasks to the targets. However, these techniques require strong assumptions over the learned model. For instance, a common form of inductive bias in MTL is ℓ_1 regularization, which leads to a preference for a sparse solution. While these efforts could partially remedy model adaptation, the learner, based on prior knowledge, is biased to the specific hypothesis and lack of generalization. We thus motivate the study of meta-learning, which aims to learn an effective strategy for fast adaptation.

Recent work has been studying how meta-learning algorithms [6, 7] distills the knowledge of related tasks and then adapt to new tasks with only a few examples [5, 32]. Optimization-based meta-learning algorithms aim to learn a learning strategy to guide the optimization procedure of the weight updates. The approaches in this domain can be categorized in terms of structure-based [15, 30] or context-based [17, 25] models. The structure-based models generally learn a task-specific update direction and step-size by operating the neural network structure. In contrast, the context-based models usually guide the update of model parameters by exploiting the context of individual tasks. Despite the promising properties, existing meta-learning algorithms suffer from two drawbacks: a) The learning framework requires a bunch of similar tasks for meta-learner to maximize the generalization of multiple tasks; and b) Each task is typically optimized by a fix-structured neural network, which restrains the model capability to capture personalized characteristics of individual tasks.

Rather than having a fixed function over all tasks, neurons should be formulated as adaptive processors [8], modifying the function according to the tasks' context. The context of tasks should be exploited to explicitly recalibrate the neural network structure so that the predictive function can be changed dynamically concerning individual tasks. However, to the best of our knowledge, such an adapting mechanism has rarely been exploited in meta-learning, even though it is an efficient and intuitive way. Motivated by this, we will model the meta-learner as an adaptive function and explore how to leverage the context of tasks to guide task-specific models' back-prorogation.

In this paper, we propose the **task-specific meta adaptive learning**, a new perspective of recalibrated learning with the task context's awareness. Our contribution lies in three-fold:

- To the best of our knowledge, this is the first attempt to explore the task context to implement an adaptive backward optimization algorithm in meta-learning scenarios, in which *backward* refers to a task-specific back-propagation process. In contrast, *meta* refers to modify the predictive function dynamically with respect to individual tasks.
- We propose a learn-to-adapt framework where we introduce a "Compress-Recalibrate" (CR) block to explicitly model the task dependency through scaling the filters of transformation weights in convolution layers. The proposed CR block is lightweight and applicable to various neural architectures. The proposed algorithm can better compose discriminative features with the context of individual tasks and maximize the generalization with the calibrated embedding of tasks.
- Extensive empirical experiments on the challenging few-shot learning benchmarks demonstrate the effectiveness of the proposed algorithm. The model trained with task-aware adaptive

WWW '21, April 19 – 23, 2021, Ljubljana, Slovenia
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

learning achieves promising performance and shows to be incredibly helpful for boosting the prediction accuracy.

2 RELATED WORK

It is a challenge to apply a gradient-based optimization into few-shot learning [17], due to two factors: (1) Multiple tasks with unknown distribution; (2) Few-shot examples in each task. We raise a question: is there a way to adjust the optimization algorithm so that the model can learn unobserved tasks with only a few labeled examples? That is the ultimate goal of the meta-learning. Existing meta-learning algorithms could be divided into three groups: metric-based approaches that aim to learn a suitable embedding space with a neural network so that nearest neighbor classification works well given a metric in this space [22, 29, 32], context-based approaches that train a recurrent neural network to take the context of cells as input and adjust parameter optimization [1, 16, 25], and structure-based approaches [15, 30] that introduce the operations onto the parameters to indicate how to transfer knowledge. Besides, some optimization-based approaches [5, 17, 24, 35], e.g., MAML, search for the optimal initialization across tasks such that fine-tuning from the base learner leads to fast adaptation to new tasks. Despite their appealing properties, most of the context-based approaches lack the adaptation in the structure space. Simultaneously, many structure-based algorithms try to generalize well across the tasks without capturing the specific structure of individual tasks. Unlike previous work, we study task-aware adaptive learning in the back-propagation of neural network optimization. In particular, our method can perform structure adaptation in meta-learning, and the context of individual tasks determines these parameters. Silver et al. [27] proposed context-sensitive multi-task learning that use additional context as input to neutral network. Similarly, Zintgraf et al. [35] introduced task-specific context into meta-learning that learns the contextual feature via backprogation. They show that context-based neural network outperform the shared-feature learners. Other related work [11] proposed a context-aware block to learn the channel-wise correlation of feature embedding and generalize well across large dataset. Unlike these work, we implement an adaptive backward optimization with the awareness of task context. The proposed task-aware block aims to capture the relationship of tasks such that the knowledge can be transferable among the related tasks. Table 1 presents a technical comparison with other related algorithms.

One related work is MTL [30], which learns the scaling and shifting operations in MAML to realize an adaptive network structure. Another related work is Meta-SGD [17], which train an adaptive operator in the back-propagation step to determine the direction to update in MAML. Note that these operations are learned across all tasks, thus unable to capture the specific relations between tasks. Besides, this work allows the adaptive learning to be conducted on new tasks at the meta-test phase, where recalibrate learning is achieved via exploiting the knowledge from the related tasks.

3 ELEMENTS OF META LEARNING

In this section, we formalize the supervised classification setup of meta-learning [5, 22, 25, 32], and propose a task-specific and learn-to-adapt framework.

3.1 Few-Shot Meta-Learning

Similar to the general supervised classification problem, we usually split the whole dataset into three parts: the training, the validation and the testing dataset. In the few-shot meta-learning scenario, however, the three splits will have disjoint label spaces. We aim to adapt the learned information *e.g.* parameters and update rule, from training dataset in the *meta-train* phase to the testing dataset in the *meta-test* phase [25].

Meta-Train: Given the training dataset, this phase aims to learn how to leverage the knowledge from a batch of tasks for unseen instances of the corresponding task . Follow the convention terminology in MAML [5], we define a task distribution $\mathcal{T} \sim p(\mathcal{T})$ from each of which, *e.g.* $\mathcal{T}_t$, we acquire two separate data splits $\mathcal{D}_t^{tr}$ and $\mathcal{D}_t^{ts}$ for the purpose of an *inner-loop* update and an *outer-loop* update respectively. The goal of inner-loop update is to optimize multiple *base-learners* using the data split $\mathcal{D}^{tr}$, and subsequently, the data split $\mathcal{D}^{ts}$ is used to optimize the base-learner to acquire the *meta-learner* in the outer-loop update. Specifically, given any task $\mathcal{T}_t \sim p(\mathcal{T})$, the base-learner is learned from the task-specific data split $\mathcal{D}_t^{tr}$ and its corresponding loss $\mathcal{L}(\theta, \mathcal{D}_t^{tr})$ is computed by the gradient descent in Eq. (1a). Therefor, each base-learner of the task $\mathcal{T}_i$ has intermediate parameters $\widehat{\theta}_t$. Then a meta-learner is optimized with $\mathcal{D}_t^{ts}$ based on the loss $\mathcal{L}(\widehat{\theta}_t, \mathcal{D}_t^{ts})$ as in Eq. (1b). Note that in the optimization of meta-training, the gradient is computed to update base-learner θ, but the meta outer-loop loss is computed with respect to intermediate parameter $\widehat{\theta}_t$ fined-tuned by specific task $\mathcal{T}_t$. Assume $\{\mathcal{T}_t\}_{t=1}^{B}$ is the task batch for training, the two-loop update in meta-train phase is formally given by

$$\begin{aligned} &(a) \quad \widehat{\theta}_t \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_t^{tr}), \\ &(b) \quad \theta \leftarrow \theta - \beta \nabla_\theta \sum_{t=1}^{B} \mathcal{L}(\widehat{\theta}_t, \mathcal{D}_t^{ts}) \end{aligned} \tag{1}$$

where α and β are learning rates. Note that, Eq. (1a) defines the task-specific parameter update for all $\mathcal{T}_t$ and Eq. (b) leverages the overall information learned from multiple tasks to unseen $\mathcal{D}^{ts}$. In the N-way K-shot meta-learning, each task contains N classes each of which has K inputs. Therefore, each $\mathcal{D}_t^{tr}$ will have K inputs and one iteration of meta-train uses NK labeled instances to do the classification. The empirical cross-entropy loss $\mathcal{L}$ of Eq. (1a) is expressed as,

$$\mathcal{L}(\theta, \mathcal{D}_t) = \frac{1}{|\mathcal{D}_t|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_t} \ell(f_\theta(\mathbf{x}), y) \tag{2}$$

where $f_\theta(\mathbf{x})$ is the prediction of the input $\mathbf{x}$ with model θ and y is the corresponding ground-truth label. Intuitively, a well-learned model parameter θ is close to local optimums for the multiple tasks $\mathcal{T}_t \sim p(\mathcal{T})$. Apparently, the update in Eq. (1a) is sensitive to task specificity. For example, $\widehat{\theta}_t$ of one task $\mathcal{T}_t$ and $\widehat{\theta}_{t'}$ of the other task $\mathcal{T}_{t'}$ have specific behaviors such that the evaluation of the outer-loop will be essentially affected. **We motivate the proposed *task-specific meta adaptive* method in this regard.**

Meta-Test: Given any batch of tasks sampled from the unseen testing dataset, this phase aims to test the model's fast adaptation ability via a limited number of inner-loop update as in Eq. (1a). So, we can take the meta-test phase as a realization of the fine-tuning on the model learned from meta-train.

Table 1: Comparison with gradient-based related work in terms of optimization objective, variables and advantage

	Context-based Models [17, 25]	Structure-based Models [15, 30]	This work
Learning Objective	Back-propagation	Forward Prediction	Back-propagation + Structure Learning
Learning Variable	Momentum, Learning rate	Scaling and Shifting Operators	Calibrate Function on Gradient
Advantage	Temporary Learning	Structure Adaptation	Task-adaptive Structure Learning

In this work, the learning setting is inspired by fine-tuning, and it learns initial parameters of a network such that the loss over a particular task after a few gradient steps is minimized [5]. It alternates between the two updates in Eq. (1) to determine initial parameter θ for task-specific learners, such that model optimization can be warm-started and new tasks can be optimized using the few-shot examples.

3.2 Model Adaptation

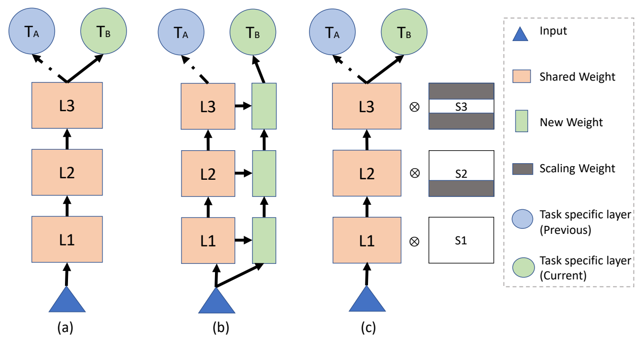

Figure 1: Illustration of different meta-learning approaches: (a) All tasks share a single set of model parameters and are optimized with single update rules. (b) Each task will add additional task-specific parameters. All layers' original weights are not tuned. (c) Our approach, where the network is adaptively trained with a subset of neural networks. The figure shows that the meta-learner decides to "re-use" the first layer and "re-scale" some parts of the next two layers.

As illustrated in Figure 1(a), one straightforward solution of meta-learning is to utilize a single set of model parameter θ for all tasks and optimize the shared θ using gradient descent over a collection of tasks. Following this direction, the performance of new tasks is affected by prior knowledge across all tasks. Well-learned initial model parameters are essential for ensuring reasonably good performance for new tasks [5]. Since such methods use fixed neural structure over different tasks, they usually require many tasks that are positively related to meta-training, impractical in real-world applications. Furthermore, there is no guarantee for initial model optimization effectiveness since some tasks may have different behaviors in the model training phase.

Figure 1(b) presents another way of addressing few-shot learning is to augment neural networks with external memory [32]. Generally, it keeps previous shared model θ fixed when building new model $\theta_t = \theta \cup \psi_t$, which memorizes the new knowledge when ψ_t is the new parameter introduced for a new task $\mathcal{T}_t$. Since introducing ψ_t for each task is usually hand-crafted, we often adopt simple heuristics, e.g., by adding extra channels on top of the current network structure or adding extra support vectors into the dictionary. By doing this, the model will become more complicated with the increased number of tasks, and θ_t is enforced to be reused for a new task without accounting for their potential dissimilarities. Nonetheless, such methods usually achieve a trade-off between computational efficiency and predictive accuracy for new tasks in few-shot learning.

Our work belongs to optimization-based meta-learning [5] in that the proposed algorithm optimizes performance via gradient update step after fine-tuning. Unlike [5] that constrains a fixed model across all tasks, Figure 1(c) shows that this work actively searches for a subset of its weights for adaptive learning for different tasks. In other words, it learns a partial "super" network, which is most suitable for a specific task at hand. Furthermore, whereas [5] learns with standard gradient descent, a subset of parameters effectively "sketch" the parameter space of the deep model to be learned during the meta-train phase to enable faster adaptation.

3.3 Learn-to-Adapt Framework

As shown in Figure 1(c), the learn-to-adapt framework aims to achieve the best model adaptation among multiple tasks. For the base-learner θ, we can either keep them fixed or allow them to change subject to some scaling functions. In this way, the framework can preserve prior knowledge of the tasks without sacrificing efficiency when new tasks are different. Now the structure is explicitly taken into consideration when learning different tasks. We denote $\Omega_{scale}(\cdot)$ as a scaling function for parameter adaptation. When optimizing the loss function in Eq. (2), one needs to calibrate the optimal parameter using the function $\widetilde{\theta_t} \leftarrow \Omega_{scale}(\theta, \mathbf{B}_t)$ where $\mathbf{B}_t$ is the scaling score for parameters weights learned on the task $\mathcal{T}_t$, and $\widetilde{\theta_t}$ is the recalibrated parameter. Formally, the objective of our work is to minimize

$$\mathcal{L}_{\mathcal{D}_t} = \frac{1}{|\mathcal{D}_t|} \sum_{(\mathbf{x},y)\in\mathcal{D}_t} \ell(f_{\widetilde{\theta_t}}(\mathbf{x}), \mathrm{y}), \quad \widetilde{\theta_t} \leftarrow \Omega_{scale}(\theta, \mathbf{B}_t) \tag{3}$$

We can interpret the loss from two perspectives: (a) One can interpret it as selecting a task-specific network from a "super network" that has parameter $\widetilde{\theta_t} = \mathbf{B}_t \odot \theta_t$; (b) For each task, one can train or update a task-specific model with $\widetilde{\theta_t} = \theta - \mathbf{B}_t \odot \nabla_\theta \mathcal{L}_t$. There is a subtle difference between these two views. The former selects a partial neural network for forwarding prediction, while the other updates a subset of model parameters with back-propagation. In this way, we can train different models for different tasks. Our goal is to jointly learn the scaling score $\mathbf{B}_t$ and model parameter θ for different tasks in a few-shot learning setting. To achieve this, we propose a task-aware meta adaptive learning framework in the rest of this paper.

4 TASK-SPECIFIC META ADAPTIVE LEARNING FRAMEWORK

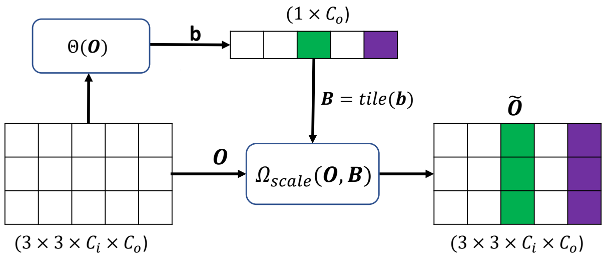
$\Theta(\boldsymbol{O})$
$\mathbf{b}$
$(1 \times C_o)$
$\boldsymbol{B} = tile(\boldsymbol{b})$
$\tilde{\boldsymbol{O}}$
$\boldsymbol{O}$
$\Omega_{scale}(\boldsymbol{O}, \boldsymbol{B})$
$(3 \times 3 \times C_i \times C_o)$
$(3 \times 3 \times C_i \times C_o)$

Figure 2: Illustration of task-specific adaptive learning: With the input context, the meta-learner decides to *re-use* or *adapt* parameter weight in each layer of neural network. The *re-use* choice will make use of the same parameters as the previous tasks, while the *adapt* choice will recalibrate the parameter weights subject to the context of individual tasks. (Each square represents a 3×3 kernel.)

It is a challenging problem to optimize the loss in Eq. (3) since it involves explicit optimization of both model parameters and the recalibrate function. We introduce a feasible solution on adaptive meta-learning with an awareness of the specific task context. The solution involves on two essential components: (*i*) **Structure Activation** calculates the scaling matrix $\mathbf{B}_t$ subject to task context; (*ii*) **Structure Adaptation** realizes meta adaptive learning through weight recalibration. More specifically, the structure activation searches for a subset of neural structure for task-specific models. It learns whether to reuse or adapt functional blocks in the current neural network. Simultaneously, the structure adaptation recalibrates model parameters subject to the activation decision. It dynamically fine-tunes a partial of neural structures if preferred.

Figure 2 illustrates a computation flow of the proposed framework. It provides a computational unit over the input context $\mathbf{O}^l \in \mathbb{R}^{k \times k \times C_i^l \times C_o^l}$ at the layer l, by using a channel-wise activation operation $\Theta : \mathbf{O}^l \rightarrow \mathbf{b}^l$, where outputs $\mathbf{b}^l \in \mathbb{R}^{1 \times C_o^l}$ denotes the vector with the scaling score on each channel. We denote k as kernel size, C_i^l, and C_o^l as the numbers of input and output channels. Assuming that the convolutional neural network consists of L layers, the filter activation operation on each layer l follows,

$$\mathbf{b}^l = \Theta(\mathbf{O}^l), \quad \widetilde{\mathbf{O}}^l = \mathsf{tile}(\mathbf{b}^l) \odot \theta^l, \quad \forall l \in [1, L], \tag{4}$$

where $\odot$ is the element-wise product between matrices of the same dimension, and the tensor $\mathsf{tile}(\mathbf{b})$ is constructed by tiling a given vector $\mathbf{b}$ with the same dimension as the tensor $\mathbf{O}$. As an aggregation produces the output across the filters of parameter weight in $\mathbf{O}$, the dependencies between tasks and model structure could be captured by the activation function Θ. In contrast, these dependencies are implicitly embedded in corresponding output $\mathbf{b}$. Our goal is to ensure the activation function to identify informative filters to specific tasks and suppress less useful ones.

Specifically, the proposed framework introduces an architectural block for filter activation, namely the "Compress-Recalibrate" (CR) block, which explicitly models task dependency by activating the transformation weights in convolutional layers. On the one hand, the CR block enables the network to recalibrate the filter response adaptively. On the other hand, it exploits inter-task dependencies to emphasize informative filters and suppress unrelated ones. For any layers of neural transformation, the CR block determines the layers' choices in two steps: *compressing* and *recalibrate*, before it optimizes the parameters. When the context $\mathbf{O}$ arrives, it first passes through a *compress* operation, which clusters (summarizes) the context features across spatial dimensions to produce a channel-wise descriptor. This descriptor summarizes the spatial features, sketching essential signals across the receptive fields. Then we use an *recalibrate* operation to learn the activation function for each filter with a gate mechanism, enables adaptive neural structures on specific tasks. Then the context after recalibration can be exploited to optimize the current layers.

4.1 Compress: Context Summarization

In the convolutional layer, each unit of the output can only receive local contextual signals within the receptive field. This issue becomes more severe in the lower layers of the network whose receptive field sizes are small. Due to the sparse and very high-dimensional space in the filters of transformation weight, directly learning activation function over this space would result in high computational complexity and over parameterizations.

We propose to sketch the embedding over the spatial dimension and outline essential signals across receptive fields to mitigate this problem. This is achieved by using an average pooling to capture informative signals across the channels. Formally, a compressed matrix $\mathbf{Z} \in \mathbb{R}^{C_i \times C_o}$ is derived by shrinking the context $\mathbf{O}$ through spatial dimensions $k \times k$, where the (i, j)-th element of the matrix $\mathbf{Z}$ is formulated as

$$Z_{i,j} = \frac{1}{k \times k} \sum_{h=1}^{k} \sum_{w=1}^{k} |O_{[h,w,i,j]}|. \tag{5}$$

In convolutional neural network, such context can be interpreted as a collection of the local signal whose contextual information are explicitly expressive for the whole image. Exploiting such information is prevalent in feature engineering work. Noting that other aggregation strategies (e.g., max pooling) could be employed here as well.

4.2 Recalibrate: Filter Activation

After receiving the *compressed* information, we require the *activate* function to further capture inter-task dependencies via activating the filter response in convolutional layers. First, the filter activation needs to learn a nonlinear transformation function. Second, it must be able to exploit inter-task dependencies to activate informative filters for related tasks. To fulfill this, we employ a simple gating mechanism:

$$\mathbf{b} = \sigma(\mathbf{A}\text{vec}(\mathbf{Z}) + \mathbf{a}),$$

where σ refers to a non-linear activation function, $\text{vec}(\cdot)$ is a flatten operator that converts a matrix into a vector, while $\mathbf{A} \in \mathbb{R}^{C_o \times C_i C_o}$ and $\mathbf{a} \in \mathbb{R}^{C_o}$ are parameters weights in a fully-connected (FC) layer. To reduce the computational complexity, we parameterise the FC layer by forming a low-rank weight matrix $\mathbf{A}$ with a matrix decomposition,

$$\mathbf{A} = \mathbf{P}\mathbf{Q}^\top, \quad \text{s.t., } \mathbf{P} \in \mathbb{R}^{C_o \times d}, \mathbf{Q} \in \mathbb{R}^{C_i C_o \times d} \tag{6}$$

where the number of latent factor d has to satisfy $d \ll C_i$ or C_o, which not only guarantees a lightweight building of the block, but captures the low-rank dependencies among the tasks. Motivated by [10], we can further design a "bottleneck" building block to forward the output $\mathbf{m}$

$$\mathbf{m} = \mathbf{P}\delta\left(\mathbf{Q}^\top \text{vec}(\mathbf{Z})\right) + \mathbf{a}, \tag{7}$$

through the activation function σ to obtain the activation $\mathbf{b} = \sigma(\mathbf{m})$, where δ is denoted as the ReLU [20] function. Different from the residual architectures [10], the proposed structure is implemented with two FC layers with smaller output/input dimension d as the bottleneck (d will be discussed in Experiments).

The vector $\mathbf{b} \in \mathbb{R}^{C_o \times 1}$ is a binary activation which is sampled each time the learner encounters various tasks. Since each element of $\mathbf{b}$ is either 1 or 0, we parameterize the probability of each element in $\mathbf{b}$ being 1 based on the corresponding scalar in $\mathbf{m}$:

$$\mathbf{b} = [b_1, \ldots, b_{C_o}], \quad \text{where} \quad b_i \sim \sigma(m_i) = \mathcal{B}\left(\frac{\exp(m_i)}{\sum_i \exp(m_i)}\right), \tag{8}$$

where $\mathcal{B}$ denotes the Bernoulli sampling. Nonetheless, Eq. (8) cannot be optimized with gradient descent. Fortunately, we can approximately differentiate through the Bernoulli sampling of activation using the Gumbel-Softmax estimator [12]:

$$b_i \leftarrow \left(\frac{\exp(\frac{m_i+g_i}{\tau})}{\sum_i \exp(\frac{m_i+g_i}{\tau})}\right), \quad g_i \sim \text{Gumbel}(0,1), \tag{9}$$

where $\tau > 0$ is a temperature hyperparameter. This reparameterization allows us to backpropagate through binary activation directly. At the limit of $\tau \approx 0$, Eq. (9) follows the behavior of Eq. (8) and become one-hot. Each logit acts on a channel of context representation $\mathbf{O}$, so that model parameter that contributes to the same channels are activated or not.

$$\widehat{\mathbf{O}}_{[.,.,.,i]} = b_i \cdot \mathbf{O}_{[.,.,.,i]}, \tag{10}$$

where the operator $\cdot$ is channel-wise multiplication between the channel $\mathbf{O}_{[.,.,.,i]} \in \mathbb{R}^{k \times k \times O_i \times 1}$ and the scalar b_i. The activations act as channel weights adapted to task-specific context $\mathbf{O}$. In this regard, CR blocks intrinsically introduce adaptation conditioned on the input, helping to boost filter activation for specific tasks.

Discussion: Without losing generalization, the CR block could work for all convolutional neural networks of interest, such as LeNet [14] and VGG [28]. Consider a network with L sharable layers; the goal of CR block is to seek the candidate choice for each of the L layers, given the current task data $\mathcal{D}_t$ and all the model layers' weights θ. The candidate choices for each layers are defined by "*reuse*" and "*adapt*". The *reuse* choice will make the current task use the same parameter as the previous task. The *adapt* operation scales the filter with a real number that is subject to the context. Here, we denote the channel number of l-th layer as C^l. The total number of choices in the l-th layer is $2|C^l|$, because we will have $|C^l|$ *reuse* and $|C^l|$ *adapt*. Thus, the total selection space is $\prod_l^L |C^l|$. One potential issue here is that, in the worst case, the search space may grow exponentially concerning the number of parameters. One way to address this is to limit the total number of the *adapt* choices via tuning the temperature hyperparameter τ in the Gumbel-Softmax estimator. In this regard, the Gumbel-Softmax estimator could have the same expressive power as a sparse regularization.

4.3 Recalibrated Backward Optimization

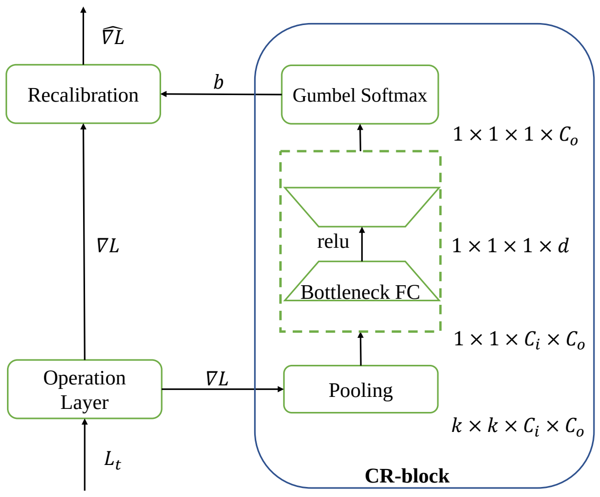
$\widehat{\nabla L}$
Recalibration
b
Gumbel Softmax
$1 \times 1 \times 1 \times C_o$
∇L
relu
$1 \times 1 \times 1 \times d$
Bottleneck FC
$1 \times 1 \times C_i \times C_o$
Operation Layer
∇L
Pooling
$k \times k \times C_i \times C_o$
L_t
CR-block

Figure 3: The schema of task-specific backward network (TBNet).

We adapt the CR block onto a convolutional neural network to build a task-specific backward network (TBNet) in meta learning. The detailed architecture of TBNet is present in Figure 3. The TBNet proposes to carry out adaptive backward optimization through scaling gradient descent over specific tasks. In the TBNet, CR blocks are deployed into each layer of the network by taking the gradient descent $\mathbf{O} = \nabla_\theta \mathcal{L}_\mathcal{D}$ as the context input, then recalibrating the gradient descent before updating it to the model. By introducing CR blocks into the backward learning, one can adaptively update a part of model parameters subject to individual tasks.

Formally, we consider the input context $\mathbf{O} = \nabla_\theta \mathcal{L}_\mathcal{D}$ and the general activation function $\phi(\cdot)$ in CR block. Assume that the structure of TBNet consists of L layers, parameter recalibration in each layer $l \in [L]$ is given by

$$\mathbf{O}^l = \nabla_{\theta^l} \mathcal{L}_{\mathcal{D}_t}, \ \mathbf{b}^l = \phi^l(\mathbf{O}^l), \ \theta^l \leftarrow \theta^l - \alpha \cdot \text{tile}(\mathbf{b}^l) \odot \mathbf{O}^l, \tag{11}$$

where $\alpha > 0$ is the learning rate, $\odot$ is the element-wise product between matrices of the same shape, while the tensor $\mathbf{B}^l = \text{tile}(\mathbf{b}^l)$ is constructed by tiling a given vector $\mathbf{b}^l$ with the same dimension as $\mathbf{O}^l$. In addition, TBNet plays a crucial role in forward prediction since the CR block governs the excitation scale of the filters in each layer. An illustration of how TBNet recalibrates the filters of convolutional neural network is present in Figure 3.

Discussion: The CR block can be regarded as a plug-in replacement for the transformation function block at any depth in the architecture. As in Figure 1(c), the role it performs at different depths adapts to the needs of the network. In the early layers, it learns to activate informative features in a class agnostic manner, strengthening the shared basic-level representation quality. The CR block becomes increasingly specialized in later layers and responds to different input in a task-specific manner. In this way, the benefits of filter recalibration conducted by CR block can be accumulated through the multi-round task epochs.

5 ALGORITHMS

For a neural network with L transformation layers, the goal of adaptation is to seek the optimal calibrated learning for each of the L layers. Given the task $\mathcal{T}_t$, we adopt a meta-learning strategy to split data into two subsets: a training subset D^{tr} for parameter learning over specific-task input and a validation subset D^{ts} for meta calibrated learning and parameter estimation. We use training loss $\mathcal{L}_{train}$ to update the task-specific parameter $\widehat{\theta}$ for all the model layers, while the recalibrate function ϕ and model parameters θ are estimated by the validation loss $\mathcal{L}_{val}$ with the intermediate parameters $\widehat{\theta}$. The recalibrate weights and parameters are updated alternately by

$$(\theta, \phi) = \arg\min_{\theta,\phi} \mathcal{L}_{val}(\widehat{\theta}, \phi) \quad \text{s.t.} \quad \widehat{\theta} = \arg\min_{\theta} \mathcal{L}_{train}(\theta, \phi). \tag{12}$$

The iterative procedure is outlined in Algorithm 1, which summarizes the training process of two main stages: context-aware meta-adaptive learning (line 7-10) and base model and meta-learner update under specific tasks (line 12). Note that MAML [5] is the special cases of our proposed technique, where all tasks optimize the same neural architecture with a single update rule. In the proposed framework, a subset of neural structure could be focussed and learned in the context of task-specific information. While we are not currently aware of the convergence guarantee for the optimization algorithm, it can reach a fixed point with a suitable choice of learning rate in practice.

Discussion: Because it is a nested bi-level optimization problem, we could use a second-order approximation for more accurate optimization [18]. In this work, we find it is sufficient to use simple alternately updating approaches as in Eq. (12), which was a first-order approximation. Such nested formulation exists in gradient-based hyperparameter optimization. In TBNet, we can also view the structure adaptor ϕ as a particular type of hyperparameter. However, its dimension is substantially higher than scalar-valued hyperparameters, such as the learning rate, and it is harder to optimize.

Algorithm 1: Task-Specific Meta Adaptive Learning (TBNet)

```
1  Input: task distribution $p(\mathcal{T})$, learning rate $\alpha$, and temperature $\tau$;
2  Output: base learner $\theta$, CR-block $\phi$;
3  Initialize $\phi$ and $\theta$;
4  while not done do
5      Sample a batch of tasks $\mathcal{T}_t$ from $p(\mathcal{T})$;
6      for all $\mathcal{T}_t$ do
7          $\mathcal{L}_{train(\mathcal{T}_t)}(\theta) \leftarrow \frac{1}{|\mathcal{D}_t^{tr}|} \sum_{(x,y)\in\mathcal{D}_t^{tr}} \ell(f_\theta(x), y)$;
8          $b = \phi(O)$ where $O = \nabla_\theta \mathcal{L}_{train(\mathcal{T}_t)}(\theta)$;
9          $\widehat{\theta}_t \leftarrow \theta - \alpha \cdot \mathrm{tile}(b) \odot O$;
10         $\mathcal{L}_{val(\mathcal{T}_t)}(\widehat{\theta}_t) \leftarrow \frac{1}{|\mathcal{D}_t^{ts}|} \sum_{(x,y)\in\mathcal{D}_t^{ts}} \ell(f_{\widehat{\theta}_t}(x), y)$;
11     end
12     $(\theta, \phi) \leftarrow (\theta, \phi) - \alpha \nabla_{(\theta,\phi)} \mathcal{L}_{val(\mathcal{T}_t)}(\widehat{\theta}_t)$;
13 end
```

5.1 Complexity and Discussion

As the proposed CR block is deployed into each layer of the neural network, it introduces additional parameters and increases computational complexity. To solve this issue, we have to design a scalable model that achieves a tradeoff between computational complexity and accuracy. Specifically, each CR block exploits the average pooling operations in the *Compressing* phase and a bottle-neck FC layer in the *Recalibrate* phase by a channel-wise activation operation. In CR block, the hyperparameter that mostly increases complexity is the latent factor d in the FC layer. For all layers, we let the number of latent dimensions to be $d \ll C_i, C_o$. Such a parameter setting brings about an inexpensive overhead, especially since neither average pooling nor Gumbel-Softmax estimator requires variable learning in the training phase. Moreover, each layer's CR block could be updated in parallel with the model parameter in meta-learning. Thus, additional time overhead could be compensated for the performance improved by CR block.

Besides the bottleneck parameters $\mathbf{P} \in \mathbb{R}^{C_o \times d}$ and $\mathbf{Q} \in \mathbb{R}^{C_i C_o \times d}$, remaining parameters in CR block are bias parameter $\mathbf{a} \in \mathbb{R}^{C_o}$ and the temperature $\tau \in \mathbb{R}$ in Gumbel-softmax estimator, which span a small faction of the total network. Formally, the total number of parameters in CR block is given by

$$\sum_{l=1}^{L} dC_o^l(1 + C_i^l) + C_o^l + 1,$$

where L refers to the number of the layers. If the latent factor is set to $d = 1$ and the filter is set to 64 for all layers, CR block would introduce $\sim 13,000$ additional parameters beyond the $\sim 116,000$ parameters required by conventional 4 convolutional neural network, corresponding to a $\sim 11\%$ increase. However, we found that the CR block in the first layers could be removed at a marginal cost in performance ($< 0.1\%$ error rate on Omniglot) to reduce the related parameter increase to $\sim 10\%$.

To clarify how *reuse* and *adapt* operations work, we walk through a concrete example in the following. Let us take a convolutional neural network with all the layers using a 3×3 kernel size as an example. The choice of *reuse* is just using the current weights and keep them fixed during learning. For *adapt* choice, it uses a 1×1 convolution layer scaled to the original convolution layer, similar to the adapter used in [30]. In the meta-train phase, the weight of the original filter is fine-tuned by specific inputs. Simultaneously, the activation choice of the 1×1 adapter is generated dynamically with the context of individual tasks. For *reuse* choice, it simply uses zero weight to mask the filters. We make use of the Gumbel-Softmax estimator to realize the regularizer $\mathcal{R}(\theta)$. The value of the regularizer is set proportional to the ratio of the parameter size $Z^l = |\theta^l|$ and its activation number $\mathrm{tile}(\mathbf{b}^l)$ (i.e., $\mathcal{R}(\theta) \leftarrow \sum_l |\mathrm{tile}(\mathbf{b}^l)|/Z^l$). The model activation decision $\mathbf{b}$ can be optimized in terms of both accuracy and model efficiency simultaneously.

6 EXPERIMENTS

We evaluate the proposed algorithm in terms of few-shot classification accuracy and model robustness. Below we describe the datasets and detailed settings, followed by a sensitivity study and a comparison to state-of-the-art methods.

6.1 Datasets and Setting

We implement the proposed algorithms in the few-shot supervised learning, evaluating the meta-learning strategy on two different benchmark datasets.

Omniglot is a handwritten character recognition dataset [13], which spans 1623 characters from 50 different alphabets with 20 examples for each class. Following the setting in [26], the data is randomly split into two parts: 1200 characters for training and the remaining for testing, and augment the dataset by randomly rotating multiple times of 90 degrees.

Mini-Imagenet dataset [32] is a subset of data sampled from the original Imagenet [4], which includes 100 classes in total with 600 examples for each class. All image examples are cropped to 84×84 pixels and the whole dataset is randomly split into 64 training classes, 16 validation classes and 20 test classes.

The CR block could be adaptable to the convolutional neural network (CNN) and its sophisticated variants, e.g., LeNet [14] or VGG [28]. Motivated by the related work [5, 30], we utilize the four-layers convolutional neural network with 3×3 convolution filters, followed by one fully connected layer at the output. As in [5], we used 32 filters per layer in miniImagenet, and 64 filters in Omniglot. Convolutions have stride 2×2 on Ominiglot, while 2×2 max-pooling is used after batch normalization instead of stride convolutions on MiniImagenet. In both the meta-train and meta-test phases, the objective function, and learning strategies (i.e., learning rate, updating rule) follow the original setting in MAML [5]. The source code will be available soon.

We utilize the training and validation data of the dataset for structure adaptation. In N-way K-shot setting, we first randomly sample N classes from the training classes, and then randomly sample K instances for each class to build a task. That is, each task contains $N \times K$ instances. In the 5-way 1-shot meta-train phase, the model is trained with 4 independent tasks for each epoch on MiniImagenet and 32 on Omniglot. At the same time, the inner update step is set to be $\{3, 5, 10\}$ on both MiniImagenet and Omniglot for keeping the balance between accuracy and efficiency. We evaluate with $\{0.001, 0.005, 0.01, 0.05, 0.1\}$ inner learning rates for both the Omniglot and miniImagenet. All training and evaluation experiments are performed using TITAN X GPUs. The whole learning process takes around 1 GPU days. For the details of hyperparameter grid-search on two datasets, please refer to Table 5 in Appendix.

6.2 Model Evaluation

In the CR block, we employ the similar operation as [11]: average-pooling (or sum-pooling) operations on filter summarization, and the FC layers to produce the weight, followed by the Gumbel-Softmax estimator [12] for activation. Such CR block is applied to back-propagation for task-adaptive parameter learning. Note that CR block is determined by compression and recalibration operation. We utilize average-pooling operation for all datasets and exploit a bottleneck FC layer weight $\mathbf{A} = \mathbf{P}\mathbf{Q}^\top$. To improving efficiency and preventing overfitting, we set the latent dimension of FC weight to be $d = \{1, 3, 5\}$ over all layers. We use vanilla SGD to optimize the network parameter θ and CR block parameter ϕ with the same learning rate 0.001 for outer optimization. Once the adaptor ϕ in CR block could be optimized in the proposed framework, we can exploit CR block to adaptively optimize the model parameter in the context of individual tasks.

We evaluate the effectiveness of the algorithms in few-shot and supervised learning settings over those datasets. For each dataset, we conduct experiments containing model adaptive learning and model evaluation. In the model adaptation phase, we exploit the CR block for adaptive structure learning. In the model evaluation stage, we evaluate the task-specific structure by learning it and compare its performance with baselines. The experimental results on Omniglot are shown in Table 2, where the proposed technique achieves state-of-the-art performance. Specifically, TBNet outperforms most of baselines on all evaluation metrics, which validates the efficiency of the proposed CR block on meta adaptive learning. In particular, TBNet achieves a better performance than the baselines with more expressive neural structures (e.g., 99.76% vs. 99.07%, 98.6% compared with SNAIL [19] and Hyper-representation [7] in 5-way 1-shot setting), which further demonstrate that TBNet can perform effective structure learning on the shallow networks.

The experimental results on MiniImagenet are shown in Table 3, in which the proposed technique achieves better or comparable performance among the baselines with the similar neural structures (e.g., Matching Networks [32] and Prototypical Networks [29]). Our goal is to improve the gradient-based optimization techniques in meta-learning. TBNet outperforms most of gradient-based techniques in Conv-4 Backbone, demonstrating the effectiveness of our task-specific adaptive learning in MniImagenet classification problem. To fair comparison, we re-run the Meta-SGD algorithm with our training/test splitting since its code is not open-sourced, and training/test split is not available. Note that our method is incomparable with Antoniou et al. [2], since various modifications over objective function, optimization strategy, and normalization would lead to unfair comparison with the algorithms with original settings in [5]. Besides, the performance can be fairly compared only when methods have equal or comparable expressiveness as in [32]. Several works (e.g., SNAIL [19], MTL [30], MT-net [15]) have reported improved performance on MiniImagenet using a significantly more expressive architecture. Not controlling for network expressivity, the reported accuracy so far on 5-way 1-shot miniImagenet classification could reach to 62.1 [33]. Finally, we conduct the running-time comparison with MAML on two datasets. The results in Table 4 show that TBNet spends slightly more time than MAML on two datasets. However, additional time overhead could be compensated for the performance improved by TBNet.

6.3 Sensitivity Study on Activation Sparseness

In the Gumbel-softmax estimator, the temperature parameter τ determines the sparseness of activation. When $\tau \to 0$, the Gumbel-softmax computation smoothly approaches the operator $\arg\max$, and the activation vector $\mathbf{b}$ approach one-hot; when $\tau \to \infty$, the Gumbel-softmax distribution becomes identical to the categorical input distribution $\mathbf{m}$. Specifically, we study the impact of temperature parameter and set τ to be $\{0.1, 0.5, 1, 5, 10\}$, and run the algorithm under each value of τ. We show the comparison result in Figure 4.

We observe that TBNet achieves the better or comparable performance consistently under different values of τ. This validates the

Table 2: Results on Omniglot data set on 5-way and 20-way with 1-shot and 5-shot accuracy (%).

Few-shot Learning Model	Feature extractor	5-way		20-way	
		1-shot (%)	5-shot (%)	1-shot (%)	5-shot (%)
Hyper-representation [7]	Conv-5	98.6	99.5	95.5	98.4
SNAIL [19]	ResNet-12	99.07 ± 0.16	99.78 ± 0.09	**97.64 ± 0.30**	**99.36 ± 0.18**
MT-net [15]	Conv-5	99.10 ± 0.30	-	96.20 ± 0.40	-
Matching Networks [32]	Conv-4	98.1	99.7	93.8	98.5
Prototypical Networks [29]	Conv-4	97.4	99.7	96.0	98.9
mAP-SSVM [31]	Conv-4	98.6	99.6	95.2	98.6
MAML [5]	Conv-4	98.70 ± 0.40	99.90 ± 0.10	95.80 ±0.30	98.90 ± 0.20
MetaGAN + MAML [34]	Conv-4	99.10 ± 0.3	99.70 ± 0.21	96.40 ± 0.27	98.90 ± 0.18
Meta-SGD [17]	Conv-4	99.53 ± 0.26	99.92 ± 0.09	95.93 ± 0.38	98.97 ± 0.20
TBNet	Conv-4	**99.76 ± 0.25**	**99.93 ± 0.05**	**96.45 ± 0.36**	**98.96 ± 0.14**

Table 3: Results on MiniImagenet data set on 5-way 1-shot and 5-shot accuracy (%). $*$ denotes re-run results on the same training/test split used in TBNet.

Category	Conv-4 Backbone	5-way 1-shot (%)	5-way 5-shot (%)
Metric Learning	Matching Networks [32]	43.56 ± 0.84	55.31 ± 0.73
	Prototypical Networks [29]	46.61 ± 0.78	**68.20 ± 0.66**
	mAP-SSVM [31]	50.32 ± 0.80	63.94 ± 0.72
Gradient-based	Meta-LSTM[25]	43.44 ± 0.77	60.60 ± 0.71
	MAML [5]	48.70 ± 1.84	63.11 ± 0.92
	REPTILE [21]	49.97 ± 0.32	65.99 ± 0.58
	FLATIPUS [6]	50.13 ± 1.86	–
	Hierarchical-MAML [9]	49.40 ± 1.83	–
	MetaGAN + MAML [34]	46.13 ± 1.78	60.71 ± 0.89
	Meta-SGD* [17]	49.10 ± 1.87	64.05 ± 0.84
Ours	TBNet	**50.35 ± 1.81**	64.11 ± 0.35

Table 4: Running time (GPU hours) comparison on two experimental datasets

Model	MiniImagenet	Omniglot
MAML	7.31	29.33
TBNet	7.50	31.67

effectiveness of CR block that can prioritize adaptive select informative parameters to optimize the task-specific model. We also observe that the performance of TBNet has a slight drop with τ increased. The reason is that the algorithm with increased τ tends to treat model parameters equally regardless of input context. This case reduces to a task-agnostic learning method that tries to generalize well on all tasks. The performance in Figure 4 indicates that the sparse activation on the neural structure is useful to compose discriminative models concerning individual tasks and maximize the generalization with the calibrated embedding of tasks.

6.4 Sensitivity Study on Bottleneck Structure

In the CR block, the FC layer's bottleneck structure is essential to build the activation function. To analyze the bottleneck structure's impact, we set the dimension d using the grid search $\{1, 3, 5, 10\}$. We use Omniglot 5-way and MiniImagenet 5-way as the case study as

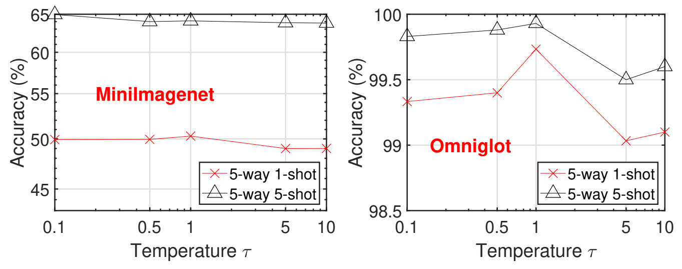

Figure 4: Sensitivity study on activation sparseness

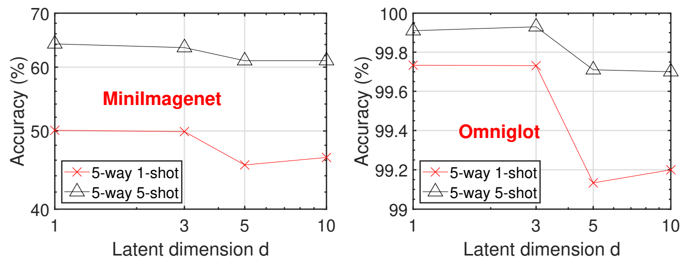

Figure 5: Sensitivity study on bottleneck structure

similar observations are obtained in other experimental settings. The results in Figure 5 show that the proposed framework achieves better or comparable performance consistently in the low dimension of bottlenecks. Obviously, with an increased dimension from 1 to 3, the performance becomes slightly better in terms of accuracy. However, further increasing d brings more parameters, which actually degrades the generalization due to overfitting issue. It motivates us to select a proper value of d to achieve a balance. Therefore, we chose $d = 3$ in the rest of the experiments, since in this setting, the algorithm achieves high accuracy while computational overhead is light.

7 CONCLUSION

This paper shows that the proposed framework trained with task-adaptive meta optimization achieves decent performance for tackling few-shot learning problems. The CR block's critical operations on pre-trained DNN neurons prove highly efficient for adapting the new task's learning experience. The superiority was mainly achieved in the extreme 1-shot cases on two challenging benchmarks - MiniImageNet and Omniglot. In terms of learning schemes, the meta-learning showed consistently good performance in baseline comparison and sensitive study. On the more challenging benchmark, it leads to being incredibly helpful for boosting adaptive capability. This design is independent of various neural architectures and could be generalized well whenever the task's hardness is easy to evaluate in online iterations.

APPENDIX

Table 5: Results of hyperparameter grid-search

	MiniImagenet 5-way		Omniglot 5-way	
Hyperparameters	1-shot	5-shot	1-shot	5-shot
inner-updates	5	10	5	3
channel-sizes	32	32	64	64
Max-pooling	True	True	False	False
Bottleneck	1	2	1	1
Initial state	Random	Random	Random	Random
Outer learning rate	1e-3	1e-3	1e-3	1e-3
Inner learning rate	0.01	0.005	0.05	0.05
Max Iterations	70000	70000	80000	80000
Temperature	1.0	1.0	1.0	1.0
Batch size	4	4	32	32

REFERENCES

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*. 3981–3989.

[2] Antreas Antoniou, Harrison Edwards, and Amos Storkey. 2018. How to train your MAML. In *International Conference on Learning Representations*.

[3] Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1126–1135.

[6] Chelsea Finn, Kelvin Xu, and Sergey Levine. 2018. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*. 9516–9527.

[7] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. 2018. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *International Conference on Machine Learning*. 1568–1577.

[8] Charles D Gilbert and Wu Li. 2013. Top-down influences on visual processing. *Nature Reviews Neuroscience* 14, 5 (2013), 350–363.

[9] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. 2018. Recasting gradient-based meta-learning as hierarchical bayes. In *ICLR*.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[11] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7132–7141.

[12] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparametrization with Gumble-Softmax. In *International Conference on Learning Representations (ICLR 2017)*. OpenReview. net.

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

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[15] Yoonho Lee and Seungjin Choi. 2018. Gradient-based meta-learning with learned layerwise metric and subspace. *arXiv preprint arXiv:1801.05558* (2018).

[16] Ke Li and Jitendra Malik. 2016. Learning to optimize. *arXiv preprint arXiv:1606.01885* (2016).

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

[18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*.

[19] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. 2018. A simple neural attentive meta-learner. In *ICLR*.

[20] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.

[21] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On First-Order Meta-Learning Algorithms. *arXiv:1803.02999v3* (2018).

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

[23] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

[24] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. 2019. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*. 113–124.

[25] Sachin Ravi and Hugo Larochelle. 2016. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*.

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

[27] Daniel L Silver, Ryan Poirier, and Duane Currie. 2008. Inductive transfer with context-sensitive neural networks. *Machine Learning* 73, 3 (2008), 313.

[28] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

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

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

[31] Eleni Triantafillou, Richard Zemel, and Raquel Urtasun. 2017. Few-shot learning through an information retrieval lens. In *In Advances in Neural Information Processing Systems*. 2255–2265.

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

[33] Jin Xu, Jean-Francois Ton, Hyunjik Kim, Adam R Kosiorek, and Yee Whye Teh. 2019. MetaFun: Meta-Learning with Iterative Functional Updates. *arXiv preprint arXiv:1912.02738* (2019).

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

[35] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. 2019. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*. 7693–7702.