Learning to Learn Task Transformations for Improved Few-Shot Classification

Guangtao Zheng* Qiuling Suo† Mengdi Huai‡ Aidong Zhang*

Abstract

Meta-learning has shown great promise in few-shot image classification where only a small amount of labeled data is available in each classification task. Many training tasks are provided to train a meta-model that can quickly learn new and similar concepts with few labeled samples. Data augmentation is often used to augment training tasks to avoid overfitting. However, existing data augmentation methods are often manually designed and fixed during training, ignoring training dynamics and the difference between various meta-learning settings specified by meta-model architectures and meta-learning algorithms. To address this problem, we add a task transformation layer between a training task and a meta-model such that the right amount of perturbation is added to training tasks for a certain meta-learning setting at a certain training stage. By jointly optimizing the task transformation layer and the meta-model, we avoid the risk of providing tasks that are either too easy or too difficult during training. We design the task transformation layer as a stochastic transformation function, adding the flexibility in how a training task can be transformed. We leverage differentiable data augmentations as the building blocks of the task transformation function for efficient optimization. Extensive experiments show that our method can consistently improve the few-shot generalization performance of various meta-models trained with different meta-learning algorithms, meta-model architectures, and datasets.

1 Introduction

Learning new concepts with a small amount of data is recognized as a hallmark of human intelligence [8]. In contrast, modern deep neural networks typically are trained with a large amount of labeled data. Meta-learning, which learns a meta-model that can quickly generalize to new concepts with a few labeled examples and adaptation steps, has recently attracted tremendous interest [25, 2, 13, 31, 33]. A widely used test bed for meta-learning algorithms is few-shot image classification where classifications are performed on new image categories after learning a few labeled training examples for each category.

In few-shot image classification, existing meta-learning algorithms [25, 2, 13, 31, 33] often adopt data augmentation methods in their implementations for performance improvement. These methods are often manually designed as a sequence of fixed image transformation functions, ignoring the training dynamics of meta-learning. As the training progresses, the meta-model could gradually memorize the seen tasks. Despite that a fixed augmentation strategy is applied, the augmented tasks could be memorized by the meta-model at a certain training stage, and thus the meta-model lacks the ability to generalize to new tasks. To tackle this, we need to provide harder tasks with more perturbations added to the images than previous ones. However, this is not possible with fixed augmentation strategies.

Moreover, existing data augmentation methods are often designed to be agnostic to various meta-learning settings specified by meta-model architectures and meta-learning algorithms. Hence, the difference between various meta-learning settings is ignored, resulting in tasks that are suboptimal for certain meta-learning settings. For example, if the same augmented tasks are provided to a deep and shallow meta-models, the deep one may simply remember the provided tasks, leading to severe overfitting. Similarly, as will be shown later, each algorithm also has its own level of task difficulty at which the algorithm is most effective in training a meta-model that generalizes well to unseen tasks.

To address the above challenges, we aim to construct tasks with task difficulty levels optimized for a certain meta-learning setting and at each training stage. Direct optimization of the task construction is infeasible in practice since we need to search all possible combinations of examples in a training set to obtain optimal tasks. To circumvent this, instead of constructing tasks from scratch, we propose to learn to transform given training tasks to get transformed ones with optimized task difficulties. From the information theoretic perspective [4], the information shared between the meta-model input and the corresponding output is reduced when the input goes through additional transformations. By transforming an input task, we control the amount of information flowing from the input

*Department of Computer Science, University of Virginia, USA, {gz5hp,aidong}@virginia.edu

†Department of Computer Science and Engineering, University at Buffalo, USA, qiulings@buffalo.edu

‡Department of Computer Science, Iowa State University, USA, mdhuai@iastate.edu

to the output. With less information provided to the meta-model, it becomes more challenging to learn new concepts from the input task. Therefore, learning to transform tasks provides a feasible way to provide tasks with optimized task difficulties during meta-training.

Inspired by the above idea, we propose to add a task transformation layer between a training task and a meta-model. The layer transforms a training task by applying learnable transformation functions to all the examples in the task. We design a task transformation function as a sequence of differentiable image operations with learnable transformation magnitudes. This design has two benefits: 1) the image operations, such as changing brightness and rotating an image, are label-preserving, and can avoid the change of labels and unwanted biases in the transformed task caused by an arbitrary task transformation function which may distort the semantics of images in the task; 2) differentiable image operations allow us to back propagate through meta-learning problems, enabling efficient optimization of the task transformation functions. To add the flexibility in how a training task can be transformed, the task transformation layer is designed as a stochastic function which follows a learnable distribution containing a set of transformation functions with learnable probabilities. During meta-training, the layer is jointly optimized with the meta-model, allowing the transformed tasks to co-adapt with the meta-model.

We summarize our contributions as follows:

- We propose a new meta-learning framework with a task transformation layer that mediates the discrepancy between training tasks and meta-learning settings specified by meta-model architecture and meta-learning algorithms, and controls task difficulty in accommodation to training dynamics.
- We design the task transformation layer as a differentiable and stochastic function for efficient optimization. As a benefit of such design choice, we get a new metric indicating the overall task difficulty required for training on a specific dataset in a certain meta-learning setting.
- We show that our method can consistently improve the few-shot generalization performance of various meta-models trained with different meta-learning algorithms, meta-model architectures on two benchmark datasets.

2 Related Work

Meta-learning focuses on adapting a model to an unseen task with limited data by learning from many training tasks. Each task is constructed to have a support set and a query set. The support set is used to simulate the procedure of adapting a model, and the query set is used to evaluate the adapted model. The two procedures are also known as the inner and outer loop of meta-learning. Existing meta-learning algorithms can be generally divided into optimization-based and metric-based algorithms. Optimization-based algorithms [8, 15, 14] use gradient-descent to fine-tune all or part of model parameters in the inner loop using the support set data. Metric-based algorithms [28, 26, 23, 13, 19, 2, 31] usually freeze feature extractor layers and only update the last carefully designed classification head in the inner loop. Different from existing algorithms, we propose a new meta-learning framework with a task transformation layer that mediates the discrepancy between training tasks and a certain meta-learning setting during meta-training so that the performance of existing meta-learning algorithms can be improved.

Many meta-learning algorithms adopt simple data augmentation on images in meta-training. A recent work [18] analyzed how support, query, task, and shot augmentations affect the performance of various meta-learning algorithms. Then, they proposed a set of manually designed augmentation policies for meta-learning. However, these policies are manually designed and not optimized for meta-learning. Instead of focusing on the input space, some recent works [29, 30] propose feature mixing and task interpolation to increase the diversity of tasks and mitigate overfitting for gradient-based meta-learning. Different from the above methods, we focus on the mismatch between training tasks and various meta-learning settings, and address it by introducing a learnable task transformation layer in existing meta-learning frameworks. Hence, our method can be used in various meta-learning settings with different model architectures and meta-learning algorithms.

Differentiable data augmentation [9, 34, 17] is a promising approach to automatic data augmentation [5, 16, 6] which finds augmentation policies without resorting to expert knowledge and has become a promising paradigm for data augmentation. Differentiable data augmentation greatly reduces the cost of searching for optimal data augmentation policies by constructing data augmentation policies with differentiable image operations, making the search differentiable. Our method uses differentiable image operations as the building blocks of the task transformation layer. Instead of creating diverse data samples, our goal of using differentiable image operations is to efficiently optimize task difficulty without distorting the semantics of images in a task. Moreover, automatic data augmentation methods often rely on constructing an adversarial loss [34] or an extra reward signal [5] to learn augmentation policies, while

our method does not have this limitation thanks to the bi-level learning structure of meta-learning.

3 Preliminaries

The goal of meta-learning is to learn a meta-model that can quickly generalize to unseen but related tasks with only a handful of labeled examples per task. The meta-model is meta-trained on a sequence of training tasks under a meta-learning algorithm. A meta-learning algorithm $\mathcal{E}$ can be specified by the inner loop learning algorithm $\mathcal{A}$ and the outer loop learning algorithm $\mathcal{B}$, i.e., $\mathcal{E} = \{\mathcal{A}, \mathcal{B}\}$. A typical meta-learning setting can be specified with the meta-learning algorithm $\mathcal{E}$ and the meta-model f_θ parameterized by θ. The meta-learning framework under this setting is formulated as:

$$f_{\theta^*} = \mathcal{B}\Big(\mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}\mathcal{L}(\mathcal{T})\Big) \tag{3.1}$$

$$\text{s.t.} \quad \mathcal{L}(\mathcal{T}) = \frac{1}{n_Q}\sum_{(x,y)\in\mathcal{Q}} \ell(f_{\hat{\theta}}(x), y), \tag{3.2}$$

$$f_{\hat{\theta}} = \mathcal{A}(f_\theta, \mathcal{S}), \tag{3.3}$$

where $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\}$ is a training task and consists of a support set $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{n_S}$ and a query set $\mathcal{Q} = \{(x_j, y_j)\}_{j=1}^{n_Q}$ containing n_S and n_Q sample-label pairs, respectively; $\mathcal{A}$ is the inner loop algorithm, which fine-tunes the meta-model f_θ with the training data in $\mathcal{S}$ from task $\mathcal{T}$, and outputs the adapted model $f_{\hat{\theta}} = \mathcal{A}(f_\theta, \mathcal{S})$; and $\mathcal{B}$ is the outer loop algorithm that outputs an optimized meta-model f_{θ^*} considering all the training tasks from the task distribution $p(\mathcal{T})$. Typically, $p(\mathcal{T})$ describes the distribution of training tasks that are constructed randomly from a training dataset. To construct an N-way K-shot task, we first randomly sample N image categories from the training dataset, and then randomly sample K images for each of the N categories for the support set $\mathcal{S}$. For the query set $\mathcal{Q}$, we sample K_q images for each category so that $n_Q = N \cdot K_q$. The task loss $\mathcal{L}(\mathcal{T})$ is calculated on the query set $\mathcal{Q}$ with the adapted model $f_{\hat{\theta}}$ and the cross-entropy loss function $\ell(\cdot, \cdot)$.

The above formulation subsumes a wide range of meta-learning algorithms. For gradient-based algorithms, both $\mathcal{A}$ and $\mathcal{B}$ are certain kinds of optimizers. For example, in MAML [8], $\mathcal{A}$ and $\mathcal{B}$ are designed as a stochastic gradient descent optimizer and an Adam optimizer [11], respectively. For metric-based algorithms, $\mathcal{B}$ is commonly designed as an optimizer, such as SGD with momentum [27], and $\mathcal{A}$ is designed as a classification method with a certain metric, such as ridge regression in R2D2 [2], and an SVM classifier in MetaOptNet [13]. Hence, our analysis based on the general meta-learning framework can be applied to many meta-learning algorithms that follow this framework.

Problem definition. From Eq. (3.1) to Eq. (3.3), we observe that under a given meta-learning setting specified by the meta-learning algorithm $\mathcal{E}$ and the meta-model f_θ with a certain architecture, the only factor that affects the performance of the optimized meta-model f_{θ^*} is the task distribution $p(\mathcal{T})$. In practice, tasks are randomly sampled from the training set for different meta-learning settings, and $p(\mathcal{T})$ represents the random task distribution. However, this ignores the difference between various meta-learning settings as they require different levels of task difficulties. Moreover, the training dynamics is also not considered when the meta-model parameter θ is continuously updated during training with a fixed $p(\mathcal{T})$. To achieve improved few-shot performance under a specific meta-learning setting, it is critical to construct tasks with optimized task difficulties that fit specific meta-learning algorithms and meta-models during training.

4 Methodology

To address the mismatch between $p(\mathcal{T})$, $\mathcal{E}$, and f_θ, an ideal approach is to learn to construct tasks with optimized task difficulties for the considered meta-learning setting. However, this approach is challenging in practice since we need to search combinatorially in the whole training dataset which is typically large. To circumvent this, we propose to dynamically control task difficulty by introducing a learnable task transformation layer which can be jointly optimized with the meta-model during training.

4.1 Task Transformation Layer We add a task transformation layer between an input task and a meta-model to control the task difficulty. The layer perturbs the task with learnable transformation functions to optimize the task difficulty at a certain training stage in a certain meta-learning setting. Based on the data processing theorem from information theory [4], additional transformations to the input reduce the information shared between the meta-model input and the corresponding output. Therefore, we optimize a task difficulty by controlling the amount of information flowing from the input to the output via transforming an input task. With less information provided to the meta-model, it becomes more challenging to learn new concepts from the input task.

To add the flexibility in how a task can be transformed by the task transformation layer, we design the layer as a stochastic function which samples a task transformation function τ for each task from the distribution $p_\tau(\tau; \omega)$ parameterized by ω. Intuitively, using multiple task transformations at a time can cre-

ate many diverse distributions to increase the chance of producing $\tilde{p}(\mathcal{T}')$ optimal for a given meta-learning setting. The goal of the layer is to transform a population of training tasks such that they follow a task distribution $\tilde{p}(\mathcal{T}')$ optimized at a certain training stage in a specific meta-learning setting. In practice, since $\tilde{p}(\mathcal{T}') = \int_{\tau,\mathcal{T}} p(\mathcal{T}'|\mathcal{T},\tau)p_\tau(\tau;\omega)p(\mathcal{T})$, the task transformation layer first samples a τ from $p_\tau(\tau;\omega)$ and then transforms a training task $\mathcal{T}$ from $p(\mathcal{T})$ via $p(\mathcal{T}'|\mathcal{T},\tau)$. We design $p(\mathcal{T}'|\mathcal{T},\tau)$ to be a distribution over $\mathcal{T}'$ calculated as follows,

$$\mathcal{T}' = \tau(\mathcal{T}) = \{x'|x' = \tau(x), x \in \mathcal{T}\}, \tag{4.4}$$

where x' and x have the same dimension, and the sampled task transformation τ is either a deterministic mapping or a stochastic one. In other words, $\mathcal{T}'$ is obtained by transforming all the samples in task $\mathcal{T}$ with the same function τ. Although τ is applied sample-wise, we still recognize it as a task transformation function since it is learned from a population of training tasks. In practice, our design enjoys fast convergence and small memory consumption. In addition to the above method, we could transform all the samples in a task with a single transformation, transform each sample in a task with an individual transformation, or other methods in between. We leave more sophisticated designs of transforming a task as our future work.

4.2 Task Transformation Functions The task transformation function τ used by the task transformation layer needs to satisfy certain constraints. With an arbitrary τ, the semantics of images in a task may be distorted, leading to change of labels and unwanted biases in the transformed task. We also require τ sampled from $p_\tau(\tau;\omega)$ to be differentiable so that ω can be directly optimized. Inspired from differentiable data augmentation [9], we design a task transformation function as a sequence of differentiable and label-preserving image operations, such as changing brightness and rotating an image, with learnable transformation magnitudes.

Specifically, given L as the length of a task transformation τ, we have $\tau(\cdot) = O_L \circ \cdots \circ O_1(\cdot)$, where $\circ$ denotes function composition, and $O_1, \ldots, O_L$ are differentiable image operations. We denote each image operation as $O(\cdot) = g(\cdot; m)$, where $g \in \mathcal{G}$ is an image operation in the set of candidate image operations $\mathcal{G}$, $m \in \mathcal{M}$ is a learnable transformation magnitude for g, and $\mathcal{M}$ is the set of all possible magnitudes. In general, each image operation has its own range of transformation magnitude. For example, a rotation operation has the magnitude in degrees ranging from $-180°$ to $180°$, while a contrast operation has the magnitude in pixel intensity ranging from 0 to 1. We normalize these magnitudes in different ranges to the same interval to stabilize the learning of task transformations. We set $\mathcal{M} = [0, 1]$ such that given $m \in \mathcal{M}$, for a function g with the transformation range $[m^g_{\min}, m^g_{\max}]$, the actual magnitude is $m \cdot (m^g_{\max} - m^g_{\min}) + m^g_{\min}$. In other words, when $m = 0$, $g(x; m)$ gives the original input x; when $m = 1$, $g(x; m)$ gives the most transformed image. For example, $O(\cdot)$ could be a rotate-$90°$ operation with g being the rotate function and $m = 0.75$ ($m \in [0, 1]$ corresponds to a degree in $[-180°, 180°]$).

4.3 Sampling Strategy The task transformation layer samples a task transformation function τ from the distribution $p_\tau(\tau;\omega)$ which can be factored as the product of L conditional probability distributions, i.e.,

$$p_\tau(\tau;\omega) = \prod_{l=1}^{L} p(O_l|O_1, \ldots, O_{l-1}), \tag{4.5}$$

where L is a hyperparameter denoting the length of τ, and each conditional distribution $p(O_l|O_1, \ldots, O_{l-1})$ is supported on $|\mathcal{G}|$ operations with learnable magnitudes, where $|\cdot|$ denotes the size of a set. The previous operations $O_1, \ldots, O_{l-1}$ are sampled from the support of $p(O_k|O_1, \ldots, O_{k-1})$ with $1 \le k < l$. A task transformation function with length L is constructed by sampling an operation O_l from $p(O_l|O_1, \ldots, O_{l-1})$ with l staring from 1 to L. In summary, we design $p_\tau(\tau;\omega)$ to be a learnable distribution over the $|\mathcal{G}|^L$ task transformation functions with ω including all learnable magnitudes and probabilities of image operations.

It is straightforward to directly sample O_l from $p(O_l|O_1, \ldots, O_{l-1})$, but this sampling process is not differentiable with respect to the parameters in $p(O_l|O_1, \ldots, O_{l-1})$ which is a categorical distribution by design. To address this, we apply a differentiable relaxation on $p(O_l = O|O_1, \ldots, O_{l-1})$ via Gumbel-Softmax reparameterization [10]. Concretely, in the forward pass, we select $O_l = \arg\max_{O \in \mathcal{O}^L_l}(p_O + r_O)$, where $p_O = p(O_l = O|O_1, \ldots, O_{l-1})$, $\mathcal{O}^L_l$ is the support of $p(O_l|O_1, \ldots, O_{l-1})$ and by design $|\mathcal{O}^L_l| = |\mathcal{G}|$, $r_O = -\log(-\log(u))$, and $u \sim \text{Uniform}(0, 1)$. In the backward pass, we have all the operations involved as $O_l(\cdot) = \sum_{O \in \mathcal{O}^L_l} s_O O(\cdot)$, where s_O is calculated as

$$s_O = \frac{\exp((p_O + r_O)/\epsilon)}{\sum_{O \in \mathcal{O}^L_l} \exp((p_O + r_O)/\epsilon)}, \forall O \in \mathcal{O}^L_l, \tag{4.6}$$

where ϵ is the temperature of the softmax function, and it controls the sampling uncertainty: a larger ϵ will generate a task distribution with more randomly sampled task transformation functions. To sample diverse task transformation functions during training, we set ϵ to a large number, e.g. $\epsilon = 20$.

Discussion. To sample the l-th operation for a task transformation function, we need $O(|\mathcal{G}|^l)$ parameters to specify $p(O_l|O_1,\ldots,O_{l-1})$, and a total of $O(|\mathcal{G}|^L)$ parameters to determine the whole distribution $p_\tau(\tau;\omega)$. Although the additional parameters needed in our method is exponential with respect to L, the value of L is usually very small in practice. We find that using $L \leq 5$ is sufficient to achieve good performance. Moreover, $|\mathcal{G}|$ is usually in the order of tens or less. Hence, the number of additional learnable parameters induced by our method is negligible when it is compared with that of a meta-model.

4.4 Learning Objective With the task transformation layer, we name the new meta-learning framework as *learning to learn task transformations (L2TT)*, which optimizes a training task distribution by transforming randomly constructed tasks with a set of learnable task transformation functions. Given $p_\tau(\tau;\omega)$ and the distribution of randomly constructed tasks $p(\mathcal{T})$, the learning objective of our proposed method L2TT is:

$$f_{\theta^*}, \omega^* = \mathcal{B}\Big(\mathbb{E}_{\mathcal{T}'\sim\tilde{p}(\mathcal{T}')}\mathcal{L}(\mathcal{T}')\Big) \tag{4.7}$$

$$\text{s.t.} \quad \tilde{p}(\mathcal{T}') = \int_\tau \int_{\mathcal{T}} p(\mathcal{T}'|\mathcal{T},\tau)p(\mathcal{T})p_\tau(\tau;\omega), \tag{4.8}$$

$$\mathcal{L}(\mathcal{T}') = \frac{1}{n_Q}\sum_{(x,y)\in\mathcal{Q}'} \ell(f_{\hat{\theta}}(x), y), \tag{4.9}$$

$$f_{\hat{\theta}} = \mathcal{A}(f_\theta, \mathcal{S}'), \tag{4.10}$$

where $p(\mathcal{T}'|\mathcal{T},\tau)$ denotes the distribution of the transformed task $\mathcal{T}'$ given $\mathcal{T}$ and τ. By providing the transformed support set $\mathcal{S}'$ and the query set $\mathcal{Q}'$ in $\mathcal{T}'$ to the inner and outer loop learning procedures, respectively, we naturally embed task transformations in the learning to learn framework and can jointly optimize them with the meta-model. Eq. (4.8) shows the new task distribution $\tilde{p}(\mathcal{T}')$ depends on $p_\tau(\tau;\omega)$. By optimizing ω, we equivalently optimize the task distribution such that it is well tuned with the underlying model architecture and the meta-learning algorithm.

One of the learning outcomes is the optimized task transformation function distribution $p(\tau;\omega^*)$. If we average all the learned magnitudes with the corresponding probabilities over all the transformation functions, we obtain a new metric called the average task transformation magnitude (AvgM-TT). This metric indicates the overall task difficulty required by training a given meta-model with the provided meta-learning algorithm and the training dataset. As shown in the experiments, AvgM-TT provides a quantitative way of comparing between different meta-learning algorithms, meta-model architectures, and training datasets.

5 Experiments

We conduct experiments to answer the following research questions (RQs). **RQ1**: Is our L2TT meta-learning framework effective for different meta-learning algorithms and meta-model architectures? **RQ2**: How our method compares with other methods that can also change images in a task? **RQ3**: How the learned task transformations differ for various combinations of meta-learning algorithms and meta-model architectures? We also show the results of different designs of task transformations in ablation study.

5.1 Experimental Setup

Datasets. CIFAR-FS [2] is a lightweight yet challenging few-shot image classification benchmark, and it allows fast prototyping. The dataset consists of all 100 classes from CIFAR-100 [12], and the classes are split into 64, 16, and 20 for meta-training, meta-validation, and meta-testing respectively. There are 600 images of size 32×32 in each class. **miniImageNet** [21] is a challenging few-shot classification benchmark without demanding computational resources. It contains 100 classes with each having 600 images. Each image is down sampled to have the size of 84×84. We follow the dataset split from [21] and divide the dataset into three non-overlapping sets of classes, forming meta-training, meta-validation, and meta-testing sets. The class numbers in the three sets are 64, 16, and 20, respectively.

Baseline methods. For **RQ1** and **RQ3**, we select four meta-learning algorithms, including three metric-based meta-learning algorithms: R2D2 [2], MetaOptNet [13], and ProtoNet [26], and one gradient-based meta-learning algorithm MAML [8]. We select ResNet12 and CNN64 (4 64-filter convolutional layers) as the meta-model architectures. We follow the implementations in [18] and give the implementation details in the appendix. For **RQ2**, the most related methods for comparison are data augmentation methods. We only consider augmentation in the input pixel space for fair comparison since the image operations that we use all work in this space. Specifically, we include SimpleAug, AutoAugment [5], and MetaDA [18] in the experiments. SimpleAug uses the following transformation `RandomCrop`→ `ColorJitter`→`RandomHorizontalFlip` to transform each sample in a task, and it is the default augmentation method in many meta-learning algorithms [25, 2, 13, 31, 33]. AutoAugment contains sets of augmentation policies optimized for specific datasets, e.g., CIFAR-100 and ImageNet. Each policy is a chain of image operations. MetaDA [18] manually designed a set of augmentation policies which augment a task by transforming the samples in the support set, in the query set, or in the whole task. Since MetaDA adopts

CutMix [32] to change the query samples (called QC), the training objective changes from predicting the labels to additionally predicting the areas of the two image patches in a mixed image created by Cutmix. For fair comparison, we apply QC to SimpleAug, AutoAugment, and our method and get SimpleAug-QC, AutoAugment-QC, L2TT-QC, respectively.

5.2 Implementation Details

Task transformations. For SimpleAug-QC, we append QC to the end of the transformation in SimpleAug. To apply AutoAugment to meta-learning, we randomly sample an augmentation policy from AutoAugment as the task transformation function. Then, for AutoAugment-QC, we append QC to the end of the sampled policy. For MetaDA, we also randomly sample policies from its manually designed augmentation policies. For our proposed method, we select 18 augmentation functions from [9, 3] as the set of differentiable image operations $\mathcal{G}$. The details of these functions are show in the appendix. The hyperparameters L and ϵ specify how a task transformation function is constructed. We use the validation accuracy of a meta-trained model to select the best L and ϵ.

Meta-training. To meta-train our models, we use the SGD optimizer with 0.9 momentum and the weight decay of 5×10^{-4}. The learning rate starts at 0.1 and decays following a cosine annealing scheduler. We stop training at epoch 100. In each epoch, we sample 1000 batches of tasks with each batch containing 8 tasks. Each task is 5-way 5-shot; it contains randomly sampled 5 classes, and each class has 5 samples in the support set and 6 samples in the query set. The best model is selected using the validation accuracy. Details of the training settings are shown in the appendix.

In the evaluation phase, we report the 1-shot and 5-shot performance of a model using the average classification accuracy with 95% confidence interval over 10,000 randomly sampled test tasks. Our code is implemented with PyTorch [20]. All the experiments are conducted on the Nvidia RTX 8000 GPUs. We provide our code and additional supplementary material at `https://github.com/gtzheng/L2TT`.

5.3 Results For **RQ1**, we evaluate our proposed L2TT framework on the CIFAR-FS and miniImageNet datasets in five meta-learning settings specified by meta-model architectures and meta-learning algorithms. The Standard meta-learning framework from Eq. (3.1) to Eq. (3.3) is the default meta-learning framework for the five meta-learning settings, and we use SimpleAug as the data augmentation method. The 5-way few-shot classification accuracies with 95% confidence intervals are reported in Table 1, and the best results are highlighted in bold fonts. Compared with the Standard framework, our proposed L2TT framework achieves consistent performance improvement on the two datasets across all the five meta-learning settings with different meta-model architecture and meta-learning algorithms. This universal improvement verifies the effectiveness of our method in different meta-learning settings. Moreover, it also implies that the mismatch between training tasks, meta-learning algorithms, and meta-model architectures, is prevalent in many meta-learning settings. By learning to learn task transformations, we can optimize the training task distribution for a give meta-model architecture and a meta-learning algorithm to provide further performance improvement. The performance gains achieved by our method are larger when we use deeper meta-model architectures, e.g., the gains achieved with ResNet-12 are larger than those with CNN64.

In our method, a task transformation function is designed to have several image operations. This design choice relates our method to the widely used data augmentation methods and gives rise to **RQ2**. Although the goal of our method is different from a typical data augmentation method which aims to generate diverse samples, our method of learning to learn task transformations generates diverse samples like a typical data augmentation method does. We compare our method with three data augmentation methods: SimpleAug-QC, AutoAugment-QC, and MetaDA. For fair comparison, we also include QC in our method and denote the new method as L2TT-QC. Table 2 shows the 5-way few-shot classification accuracies with 95% confidence intervals of the five meta-models trained with these methods. We highlight the best results in bold fonts in Table 2. From the data augmentation perspective, we observe that L2TT-QC performs the best in all the five meta-learning settings and on the two datasets.

Moreover, among the methods compared in Table 2, MetaDA is a competitive method. It is specifically designed for meta-learning and has 9 augmentation policies manually designed with the expert knowledge about meta-learning, such as considering the support-query structure of a task in designing the policies. However, MetaDA does not consider the difference in various meta-learning settings. In contrast, our method learns task transformations which can be automatically optimized for a specific meta-learning setting. Additionally, MetaDA shows a tendency to overfit to the 5-shot setting since it outperforms AutoAugment-QC in almost all the 5-shot cases but is inferior to AutoAugment-QC in more than half of the 1-shot cases, especially in those with the miniImageNet dataset. Since all the models

Architecture	Meta-learning algorithm	Meta-learning framework	CIFAR-FS		miniImageNet	
			1-shot	5-shot	1-shot	5-shot
ResNet-12	R2D2	Standard	72.53±0.11	84.16±0.08	60.59±0.10	75.90±0.08
		L2TT	**75.96±0.11**	**86.72±0.08**	**63.56±0.11**	**78.25±0.08**
ResNet-12	ProtoNet	Standard	70.42±0.12	83.25±0.08	58.28±0.10	75.82±0.08
		L2TT	**73.63±0.11**	**85.76±0.08**	**60.82±0.11**	**78.16±0.08**
ResNet-12	MetaOptNet	Standard	71.56±0.12	84.03±0.08	60.51±0.10	76.34±0.08
		L2TT	**74.34±0.11**	**86.19±0.08**	**62.50±0.10**	**78.17±0.08**
CNN64	ProtoNet	Standard	61.10±0.12	79.28±0.09	47.43±0.10	70.55±0.08
		L2TT	**63.73±0.11**	**81.06±0.09**	**49.12±0.10**	**71.57±0.08**
CNN64	MAML	Standard	55.81±0.11	75.50±0.09	42.38±0.10	64.64±0.09
		L2TT	**58.50±0.11**	**76.16±0.09**	**47.70±0.10**	**64.75±0.09**

Table 1: Performance comparison between our proposed L2TT and the Standard meta-learning frameworks under different meta-learning algorithms and meta-model architectures on the CIFAR-FS and miniImageNet datasets.

Architecture	Meta-learning algorithm	Data augmentation method	CIFAR-FS		miniImageNet	
			1-shot	5-shot	1-shot	5-shot
ResNet-12	R2D2	SimpleAug-QC	76.01±0.11	86.21±0.08	64.24±0.11	78.42±0.08
		AutoAugment-QC	76.36±0.11	86.86±0.08	64.93±0.11	78.82±0.08
		MetaDA	76.00±0.11	87.25±0.08	63.78±0.10	78.96±0.08
		L2TT-QC	**77.40±0.11**	**87.76±0.08**	**65.82±0.10**	**80.36±0.08**
ResNet-12	ProtoNet	SimpleAug-QC	75.04±0.11	86.29±0.08	60.69±0.11	76.43±0.09
		AutoAugment-QC	75.51±0.11	86.57±0.08	61.94±0.11	77.41±0.09
		MetaDA	75.00±0.11	87.06±0.08	60.63±0.11	78.13±0.08
		L2TT-QC	**76.52±0.11**	**87.12±0.08**	**63.07±0.11**	**78.79±0.08**
ResNet-12	MetaOptNet	SimpleAug-QC	73.20±0.12	86.29±0.08	64.71±0.11	79.03±0.08
		AutoAugment-QC	73.37± 0.11	86.45±0.08	65.19±0.10	79.47±0.08
		MetaDA	73.97±0.11	86.98±0.08	64.00±0.10	79.43±0.08
		L2TT-QC	**76.65±0.11**	**87.84±0.08**	**65.60±0.10**	**80.99±0.08**
CNN64	ProtoNet	SimpleAug-QC	63.26±0.12	80.63±0.08	49.95±0.10	71.39±0.08
		AutoAugment-QC	62.74±0.12	79.87±0.09	49.90±0.10	69.53±0.09
		MetaDA	63.07±0.12	80.86±0.08	49.67± 0.10	71.39±0.08
		L2TT-QC	**63.75±0.11**	**81.41±0.08**	**50.67±0.10**	**71.76±0.08**
CNN64	MAML	SimpleAug-QC	58.80±0.12	76.67±0.10	48.57±0.10	66.03±0.09
		AutoAugment-QC	55.82±0.12	72.16±0.10	47.08±0.10	63.59±0.09
		MetaDA	60.20±0.12	77.36±0.09	47.51±0.10	66.79±0.09
		L2TT-QC	**61.20±0.12**	**78.12±0.09**	**48.91±0.10**	**66.90±0.08**

Table 2: Performance comparison between various data augmentation methods for different meta-learning algorithms and meta-model architectures on the CIFAR-FS and miniImageNet datasets.

are meta-trained in the same 5-way 5-shot setting, the results show that MetaDA cannot generalize well to a low-shot setting where the number of labeled samples in the support set of a task is small. Our method does not have the above limitation. More results can be found in the appendix.

For **RQ3**, we calculate AvgM-TT for the five meta-learning settings and show the results in Figure 1. By design, a higher magnitude indicates a more aggressive transformation on the images in a task. Among the first four metric-based settings in Figure 1, we observe that the AvgM-TTs for the miniImageNet dataset are higher than those for the CIFAR-FS dataset. This indicates that we generally need "harder" tasks for the larger and complex miniImageNet dataset than those for the smaller and simple CIFAR-FS dataset. However, for the gradient-based setting CNN64-MAML, the two AvgM-TTs on the two datasets are very similar, indicating that MAML is not very sensitive to training data with varying complexities. The two values are also larger than those in the four metric-based settings. This indicates that MAML is easier to suffer from overfitting than the three metric-based algorithms and needs "harder" tasks in training. We also observe that deeper meta-model architectures require "harder" tasks, as shown by the higher AvgM-TTs of ResNet12-ProtoNet than those of CNN64-ProtoNet on respective datasets. Overall, AvgM-TT indicates the overall complexity involving training datasets, model architectures, and meta-learning algorithms.

5.4 Ablation Study Task transformations are constructed by sampling from a distribution of task transformations. Different design choices affect the overall performance. We study two important hyperparameters L and ϵ of the distribution. The function length L

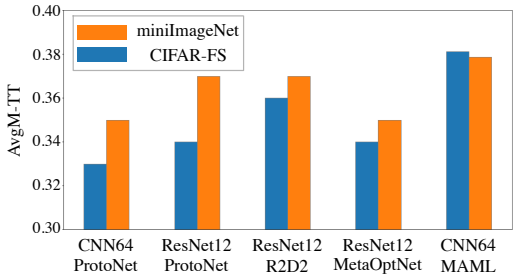

Figure 1: AvgM-TTs for different meta-learning algorithms and meta-model architectures on the miniImageNet and CIFAR-FS datasets.

L	Shot	ϵ		
		10	20	∞
1	1-shot	73.08±0.12	73.73±0.12	73.65±0.12
	5-shot	84.03±0.08	84.39±0.08	84.25±0.08
2	1-shot	75.30±0.11	75.92±0.11	72.65±0.11
	5-shot	86.05±0.08	86.30±0.08	84.87±0.08
3	1-shot	74.75±0.11	**75.96±0.11**	75.03±0.11
	5-shot	85.74±0.08	**86.72±0.08**	85.97±0.08
4	1-shot	75.11±0.11	72.04±0.11	73.44±0.11
	5-shot	86.16+-0.08	84.60±0.08	85.36±0.08

Table 3: Analysis on different design choices for task transformation functions. We study various length-L task transformation functions sampled with different levels of sampling uncertainty controlled by τ. We use R2D2 with ResNet-12 on the CIFAR-FS dataset.

controls the number of image operations in a task transformation. The temperature ϵ controls the uncertainty during the sampling of a task transformation. A larger ϵ will produce more random sampling results. We use $\epsilon = \infty$ to denote the uniform sampling of image operations. In other words, all possible task transformations have equal chance of being selected during training. We use ResNet-12 and R2D2 in our L2TT framework with different Ls and ϵs. The results are show in Table 3. We observe improved performance for each ϵ when L increases from 1 to 3. A larger L increases the representation power of a task transformation and offers more flexibility in how a training task distribution can be transformed. However, as shown by the results when $L = 4$, a too large L hurts the performance. We also note that uniformly sampling image operations will result in suboptimal performance, as shown by the results when $\epsilon = \infty$. This inferior performance with $\epsilon = \infty$ indicates that task transformations are not of equal importance, and this also justifies our probabilistic modeling of task transformations.

6 Conclusion

In this paper, we introduced a task transformation layer to address the mismatch between training tasks and a given meta-learning setting specified by the meta-model architecture and the meta-learning algorithm during meta-training. The added layer adjusts the difficulty of an input task by transforming the task to control the information flowing from the meta-model input to its output. We implemented the task transformation layer as a stochastic function with differentiable image operations as its building blocks, which leads to a new metric called AvgM-TT indicating the overall task difficulty required for training on a specific dataset in a certain meta-learning setting. Experimental results demonstrated the effectiveness of our method in improving the generalization performance of various meta-learning algorithms on different model architectures and datasets.

Acknowledgment

This work is supported in part by the US National Science Foundation under grants 2106913, 2008208, 1955151. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[2] Luca Bertinetto, Joao F. Henriques, Philip Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019.

[3] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[4] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[5] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019.

[6] Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. Randaugment: Practical automated data augmentation with a reduced search space. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18613–18624. Curran Associates, Inc., 2020.

[7] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, volume 70, pages 1126–1135, 2017.

[9] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. In *European Conference on Computer Vision*, pages 1–16. Springer, 2020.

[10] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[13] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[14] Yoonho Lee and Seungjin Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *International Conference on Machine Learning*, 2018.

[15] Zhenguo Li, Fengwei Zhou, Fei Chen, and Huang Li. Meta-SGD: Learning to learn quickly for few shot learning. *ArXiv*, abs/1707.09835, 2017.

[16] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In *Advances in Neural Information Processing Systems*, 2019.

[17] Aoming Liu, Zehao Huang, Zhiwu Huang, and Naiyan Wang. Direct differentiable augmentation search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12219–12228, 2021.

[18] Renkun Ni, Micah Goldblum, Amr Sharaf, Kezhi Kong, and Tom Goldstein. Data augmentation for meta-learning. In *International Conference on Machine Learning*, pages 8152–8161. PMLR, 2021.

[19] Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems 31*, pages 721–731. 2018.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

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

[22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[23] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

[24] Jin-Woo Seo, Hong-Gyu Jung, and Seong-Whan Lee. Self-augmentation: Generalizing deep networks to unseen classes for few-shot learning. *Neural Networks*, 138:140–149, 2021.

[25] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30*, pages 4077–4087. 2017.

[26] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[27] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[28] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29*, pages 3630–3638. 2016.

[29] Huaxiu Yao, Long-Kai Huang, Linjun Zhang, Ying Wei, Li Tian, James Zou, Junzhou Huang, et al. Improving generalization in meta-learning via task augmentation. In *International Conference on Machine Learning*, pages 11887–11897. PMLR, 2021.

[30] Huaxiu Yao, Linjun Zhang, and Chelsea Finn. Meta-learning with fewer tasks through task interpolation. In *International Conference on Learning Representations*, 2022.

[31] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8808–8817, 2020.

[32] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6023–6032, 2019.

[33] Chi Zhang, Yujun Cai, Guosheng Lin, and Chunhua Shen. Deepemd: Differentiable earth mover's distance for few-shot learning, 2020.

[34] Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial autoaugment. In *International Conference on Learning Representations*, 2020.

Appendix

A Meta-Learning Settings

Meta-Learning Algorithms. We use three metric-based meta-learning algorithms and one gradient-based meta-learning algorithm in the experiments. A metric-based algorithm first uses a feature extractor to get the embedding of each sample, and then learns a specifically designed classification head to classify samples based on their embeddings. A gradient-based algorithm uses gradient descent to learn a new classifier to classify samples. The four algorithms are described as follows:

- ProtoNet[25] is a metric-based meta-learning algorithm. It first calculates a class prototype from the support set of a task by averaging embeddings of the samples with the same label. Then, it searches the nearest class prototype for each sample in the query set and predicts the sample to have the same class label as the nearest class prototype.
- R2D2 [2] is the abbreviation for Ridge Regression Differentiable Discriminator, and it is a metric-based meta-learning algorithm. It adopts ridge regression as the classification head which is differentiable and has closed form solutions.
- MetaOptNet [13] is a metric-based meta-learning algorithm. It adopts an SVM classifier as the classification head in few-shot classification.
- MAML [8] is a gradient-based meta-learning algorithm. It aims to learn a model that can quickly generalize to new concepts with a few gradient descent steps.

Meta-Model Architectures. We call the meta-model architectures and backbones interchangeably. Essentially, they are feature extractors that convert inputs to their vector representations. In the experiments, we use the ResNet-12 backbone adopted in [19] and the four-layer convolutional backbone (CNN64) with 64 filters in each layer adopted in [25].

B Implementation Details Meta-Learning.

We follow the implementations in [18] to implement the above meta-learning algorithms and the backbones. Similar to [19], we adopt a learnable scaler to scale the outputs of each of the classification heads from the three metric-based meta-learning algorithms. For MAML, we adopt its first-order approximation in the experiments to achieve a good tradeoff between computational complexity and few-shot classification performance. Moreover, we use a 5-step gradient descent (10 steps in evaluation) with a learning rate of 0.01 in the inner loop of MAML, an Adam optimizer with a learning rate of 0.001 in the outer loop of MAML, and a cosine annealing scheduler with the minimum learning rate of 1×10^{-5} to control the learning rate in the outer loop.

Differentiable Image Operations. The image operations/functions used in our method are listed in Table A.1. Each function has its description in the "Description" column and its own transformation magnitude listed in the "Magnitude" column. We implement each image operation as a differentiable function in the sense that its output is differentiable with respect to the input. However, for functions that are not inherently differentiable, e.g., the `posterize` function, we use the straight-through estimator [9, 1] to approximate the corresponding gradient. Specifically, given an operation $O(x) = g(x; m)$ with an input x and a transformation magnitude m, we implement the operation as $O(x) = \text{StopGrad}(g(x; m) - x - m) + x + m$, where StopGrad is a stop gradient operation and treats its operand as a constant in the backward pass. We find that this method is simple and works well in our experiments.

L2TT Algorithm. The details of L2TT are shown in Algorithm 1.

Algorithm 1 L2TT

Input: training tasks distribution $p(\mathcal{T})$, parameters of the task transformation distribution ω, a meta-learning algorithm $\mathcal{E} = \{\mathcal{A}, \mathcal{B}\}$, a meta-model f_θ
Output: θ

```
//Outer loop
while in algorithm B do
    Randomly construct a task T such that T ~ p(T)
    Sample τ from p_τ(x; ω)
    T' = {x' | x' = τ(x), x ∈ T}
    Split T' such that T' = {S', Q'}
    //Inner loop
    f_θ̂ = A(f_θ, S')
    //Task loss
    L(T') = (1/n_Q) Σ_{(x,y)∈Q'} ℓ(f_θ̂(x), y)
    Use B to jointly optimize θ and ω with respect to L(T').
end while
return θ, ω
```

Data Augmentation Methods. We describe the two data augmentation methods used in our experiments: AutoAugment [5] and MetaDA [18].

- **AutoAugment.** There are 25 policies optimized for a selected dataset. Each policy is a sequence of transformations. In AutoAugment, each pol-

Function	Magnitude	Description	Function	Magnitude	Description
Shift_r	[0,1]	Change red channel value	Shift_x	[0,0.5]	Shift horizontally
Shift_g	[0,1]	Change green channel value	Shift_y	[0,0.5]	Shift vertically
Shift_b	[0,1]	Change blue channel value	Scale	[0.1,10]	Scale images
Brightness	[-1,1]	Change brightness	Self_mix	None	Self-mix up an image
Contrast	[1,10]	Change contrast	Posterize	[0,8]	Reduce number of bits for each color channel
Solarize	[0,1]	Invert pixels under a threshold value	Equalize	None	Equalize the histogram of an image
Hflip	None	Horizontally flip	Cutout_fixed1	[0,1]	Cutout 8 regions with a random size in an image
Vflip	None	Vertically flip	Cutout_fixed2	[0,8]	Cutout random number of regions with fixed size
Rotate	[-180,180]	Rotate an image	Sample_pairing	[0,1.0]	Combine two different images with random weights

Table A.1: Image operations used in our method.

icy is designed to have 2 transformations. For meta-models trained on the CIFAR-FS dataset, we use the set of policies optimized for CIFAR-10. For meta-models trained on the miniImageNet dataset, we use the set of policies optimized for ImageNet [22]. To use AutoAugment in training, we first apply the sequential transformation `RandomCrop`→`RandomHorizontalFlip` to an image. Then, we randomly sample a policy from the selected set of AutoAugment policies and apply it to the transformed image. Finally, we apply Cutout [7] with 16x16 pixels [5] to the image obtained from the previous step.

- **MetaDA.** There are four basic augmentation functions in MetaDA: SelfMix, Cutmix, random erase, and rotation. SelfMix [24] replaces a patch of an image with another patch from the same image. Cutmix [32] cuts an image patch from one image and pastes the patch to another image to construct a mixed image with the ground truth labels of the two original images mixed proportionally to reflect the area of the image patch in the mixed image. Rotation rotates an image with a degree which is a multiple of 90. Random erase (RE) randomly erases patches from an image. For each task, the augmentation functions can be applied to the support set (S), the query set (Q), or the whole task (T). We use the large-size pool defined in [18] as the set of augmentation policies. These policies are: Q-cutmix, Q-RE, S-RE, T-Rotation, Q-cutmix→T-rotation, Q-RE→T-Rotation, Q-RE→S-RE, Q-cutmix→Q-RE, Q-cutmix→S-RE.

Note that MetaDA includes Q-cutmix (QC) in the set of augmentation policies. Since QC changes the learning objective from minimizing classification loss to additionally predicting the area of an image patch in a mixed image, we cannot directly compare MetaDA with AutoAugment. For fair comparison, we append QC to the end of the policy sampled from AutoAugment.

Hyperparameter Settings. In Table A.2, we give the settings for the two important hyperparameters L and ϵ used in different meta-learning settings listed in Table 1. We select the optimal values for each meta-learning setting based on the validation performance. For metric-based meta-learning algorithms, we observe that L is larger for the settings with a ResNet-12 backbone than the one with a CNN64 backbone. In general, a task transformation with a large L indicates large variations in the images of a transformed task, and the transformed task can be considered as a "hard" task. Moreover, we observe that for MAML with the same CNN64 backbone, the optimal L is even larger than those for the metric-based algorithms with the deeper ResNet-12 backbone. This indicates that MAML is easier to suffer from overfitting when compared with the three metric-based meta-learning algorithms. For training MAML on the miniImageNet dataset, we set

$\epsilon = \infty$ which means uniformly sampling task transformations. This is because the task transformation distribution $p_\tau(\tau; \omega)$ tends to concentrate its probability mass on certain trivial task transformations that make little change to the images in a task, resulting in inferior performance. Hence, we set ϵ to infinity to circumvent this problem in this meta-learning setting.

For the experiments in Table 2, because of the introduction of QC, we find that setting $L = 1$ works well for all the meta-learning settings.

Architecture	Meta-learning algorithm	CIFAR		miniImageNet	
		L	ϵ	L	ϵ
ResNet-12	R2D2	3	20	3	20
ResNet-12	ProtoNet	3	20	3	20
ResNet-12	MetaOptNet	3	20	3	20
CNN64	ProtoNet	2	20	2	20
CNN64	MAML	5	40	4	∞

Table A.2: The values of L and ϵ used in Table 1.

C Additional Results

C.1 Meta-Learned Task Transformations. At the end of meta-training, we show the most probable task transformation with different lengths for ProtoNet-CNN64 and ProtoNet-ResNet12 meta-trained on the CIFAR-FS dataset. The results are shown in Table A.3. These task transformations reflect the difference between different meta-learning settings. For the shallow network CNN64, the meta-learned task transformations are not as diverse as the ones meta-learned with the deep network ResNet-12. For example, when $L = 1$, the maximum probability of sampling an operation is 0.09 in the ProtoNet-CNN64 setting, while the value decreases to 0.07 in the ProtoNet-ResNet12 setting. Since the probabilities of all the operations add up to 1, this means that operations other than Equalize are more likely to be sampled in the ProtoNet-ResNet12 setting than those in the ProtoNet-CNN64 setting. We observe the same trend for other values of L. With more operations involved in a task transformation, we can create "harder" tasks with more variations in the images of the tasks. From the meta-learned most probable task transformations, we can conclude that in order to obtain a well-trained meta-model, we need "harder" tasks for meta-training in the ProtoNet-ResNet12 setting than in the ProtoNet-CNN64 setting.

C.2 Performance Comparison on 10-way Tasks We evaluate how well our method (L2TT-QC) generalizes to a high-way setting by comparing the performance of L2TT-QC, AutoAugment-QC, and MetaDA on 10-way tasks generated from the CIFAR-FS and miniImageNet datasets. We exclude the results of MAML since it learns a fixed-way classifier during meta-training, and it cannot be directly evaluated in this setting. All the models are meta-trained on 5-way 5-shot tasks. The few-shot classification accuracy results are shown in Table A.4. We observe that our method generalizes well to this challenging setting and achieves the best performance in all the testing cases.

D Visualization of Transformed Images Figure A.1 shows three sets of images obtained by applying three task transformations with variable numbers of image operations to the original images in a sampled task. With more image operations, the obtained images show more variations than those obtained with less image operations.

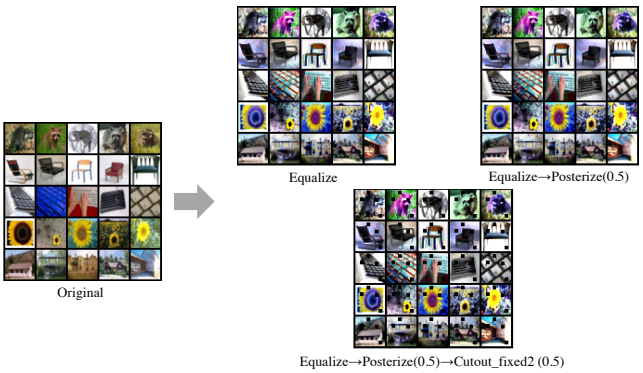

Figure A.1: Visualization of three task transformations with variable numbers of image operations.

E Visualization of Task Embeddings We visualize the original tasks and their transformed versions using t-SNE. We obtain each task embedding by averaging all the embeddings of the images in the same task. We sample 2000 original tasks from the training split of the CIFAR-FS dataset. For each task, we transform it with a task transformation sampled from $p_\tau(\tau; \omega)$. We use the backbone network meta-trained in the ProtoNet-CNN64 setting on the CIFAR-FS dataset to extract image embeddings. The visualization is shown in Figure A.2. We see that the meta-learned task transformations can generate not only tasks that are close to the original ones, but also tasks that are distant in the embedding space.

Architecture	Meta-learning algorithm	L	Task Transformation ($\cdots \rightarrow$Operation(Probability, Magnitude)$\rightarrow \cdots$)
CNN64	ProtoNet	1	Equalize(0.09,N/A)
		2	Equalize(0.10,N/A)→RandomContrast(0.06,0.24)
		3	Equalize(0.10,N/A)→RandomContrast(0.06,0.32)→Equalize(0.06,N/A)
ResNet-12	ProtoNet	1	Equalize(0.07,0.50)
		2	Equalize(0.08,N/A)→Posterize(0.06,0.50)
		3	Equalize(0.09,N/A)→Posterize(0.06,0.44)→Posterize(0.06,0.50)

Table A.3: Meta-learned task transformations with different lengths. We describe a task transformation as a sequence of operations. Each operation has a probability and a magnitude. For operations that do not have magnitudes, such as Equalize and Hflip, we set their magnitudes to "N/A".

Architecture	Meta-learning algorithm	Data augmentation method	CIFAR-FS		miniImageNet	
			1-shot	5-shot	1-shot	5-shot
ResNet-12	R2D2	AutoAugment-QC	63.01±0.08	76.66±0.06	48.16±0.07	65.11±0.06
		MetaDA	62.46±0.08	77.27±0.06	47.52±0.07	65.33±0.05
		L2TT-QC	**63.78±0.08**	**77.36±0.06**	**51.07±0.07**	**68.64±0.05**
ResNet-12	ProtoNet	AutoAugment-QC	62.08±0.08	76.33±0.06	45.25±0.07	63.12±0.06
		MetaDA	61.74±0.08	77.01±0.06	44.68±0.07	64.40±0.06
		L2TT-QC	**62.66±0.08**	**77.22±0.06**	**46.42±0.07**	**64.79±0.06**
ResNet-12	MetaOptNet	AutoAugment-QC	59.41±0.08	76.20±0.06	48.41±0.07	65.73±0.05
		MetaDA	59.54±0.08	77.00±0.06	47.23±0.07	66.08±0.05
		L2TT-QC	**62.61±0.08**	**77.48±0.06**	**48.70±0.07**	**67.84±0.05**
CNN64	ProtoNet	AutoAugment-QC	48.35±0.08	67.55±0.06	34.34±0.06	54.45±0.06
		MetaDA	48.74±0.08	69.09±0.06	34.25±0.06	56.75±0.06
		L2TT-QC	**49.28±0.08**	**69.11±0.06**	**34.90±0.06**	**57.46±0.05**

Table A.4: Few-shot classification accuracy comparison on 10-way tasks.

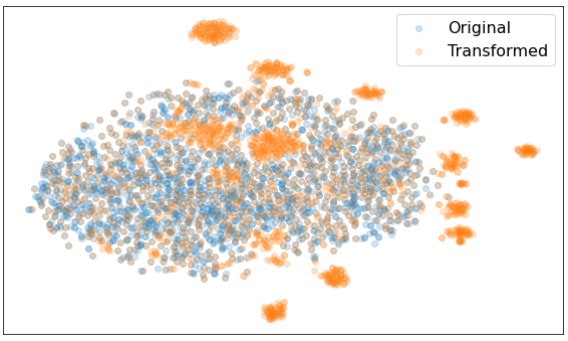

Figure A.2: Visualization of original and transformed tasks via t-SNE.