Adversarial Task Up-sampling for Meta-learning

Yichen Wu[1,2]*, **Long-Kai Huang**[2]†, **Ying Wei**[1]†
[1]City University of Hong Kong, [2]Tencent AI Lab
{wuyichen.am97, hlongkai}@gmail.com, yingwei@cityu.edu.hk

Abstract

The success of meta-learning on existing benchmarks is predicated on the assumption that the distribution of meta-training tasks covers meta-testing tasks. Frequent violation of the assumption in applications with either insufficient tasks or a very narrow meta-training task distribution leads to memorization or learner overfitting. Recent solutions have pursued augmentation of meta-training tasks, while it is still an open question to generate both correct and sufficiently imaginary tasks. In this paper, we seek an approach that up-samples meta-training tasks from the task representation via a task up-sampling network. Besides, the resulting approach named Adversarial Task Up-sampling (ATU) suffices to generate tasks that can maximally contribute to the latest meta-learner by maximizing an adversarial loss. On few-shot sine regression and image classification datasets, we empirically validate the marked improvement of ATU over state-of-the-art task augmentation strategies in the meta-testing performance and also the quality of up-sampled tasks.

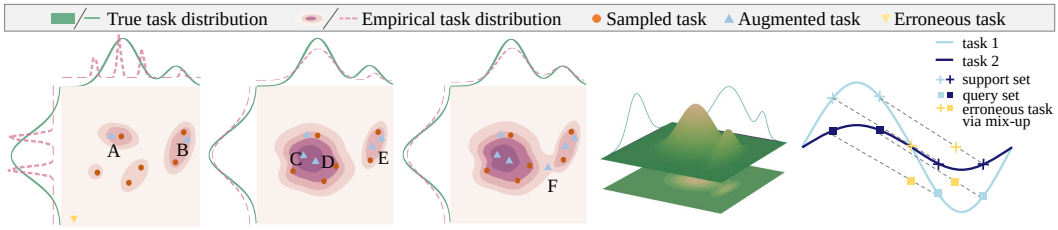
1 Introduction

The past few years have seen the burgeoning development of meta-learning, *a.k.a.* learning to learn, which draws upon the meta-knowledge learned from previous tasks (i.e., *meta-training tasks*) to expedite the learning of novel tasks (i.e., *meta-testing tasks*) with a few examples. A sufficient number and diversity of meta-training tasks are pivotal for the generalization capability of the meta-knowledge, so that (1) they cover the true task distribution (i.e., environment [4]) from which meta-testing tasks are sampled, discouraging learner overfitting [23] and (2) the meta-knowledge empowers fast adaptation via the support set for each task, avoiding memorization overfitting [44]. Notwithstanding up to millions of meta-training tasks in benchmark datasets [24, 31], real-world applications such as drug discovery [40] and medical image diagnosis [14] usually have access to only thousands or hundreds of tasks, which puts the meta-knowledge at high risk of learner and memorization overfitting.

While early attempts towards improving the generalization capability of the meta-knowledge revolve around regularization methods that limit the capacity of the meta-knowledge [11, 44], recent works on augmentation of meta-training tasks have shown a marked improvement [20, 40, 43]. The objective of task augmentation is to draw the empirical task distribution which is formed by assembling Dirac delta functions located in each meta-training task closer to the true task distribution. Consequently, achieving this objective requires a qualified task augmentation approach to simultaneously possess the following three properties: (1) *task-aware*: the augmented tasks comply with the true task distribution, being not erroneous to lead the meta-knowledge astray (tasks A and B in Figure 1a); (2) *task-imaginary*: the augmented tasks cover a substantial portion of the true distribution, embracing task diversity which task-awareness is nonetheless inadequate to guarantee (tasks C, D, and E in Figure 1b); (3) *model-adaptive*: the augmented tasks are timely in improving the current meta-knowledge, to which the meta-knowledge before augmentation struggles to generalize (task F in Figure 1c).

*Part of the work was done when the author interned in Tencent AI Lab.
†Corresponding author: Long-Kai Huang and Ying Wei

36th Conference on Neural Information Processing Systems (NeurIPS 2022).



(a) task-aware (b) task-imaginary (c) model-adaptive (d) true task distribution (e) an example erroneous task

Figure 1: Pictorial illustration of the three characteristics possessed by a qualified task augmentation approach, i.e., being (a) task-aware, (b) task-imaginary, and (c) model-adaptive. (d) shows the true task distribution that task augmentation aims to approximate, and (e) presents an example erroneous task violation of the true task distribution of sinusoidal functions $y = w\sin(x)$ ($w \in [0, 2]$).

Unfortunately, developing such a qualified task augmentation approach remains challenging. First, the task-aware methods sacrifice task diversity for task-awareness – they establish task-awareness by injection of the same random noise to labels of the support and query set [23], rotation of both support and query images [20], or mix-up of support and query examples within each task [40], all of which result in augmented tasks that are within the immediate vicinity of sampled meta-training tasks as shown in Figure 1a. Second, the task-imaginary method [43] that mixes up both support examples of two distinct tasks and their query examples in the feature space compromises on task-awareness – the resulting examples are even multi-modal in Figure 1e and constitute an erroneous task that fails to comply with the task distribution. Third, how to adaptively augment tasks that maximally improve the meta-knowledge and thereby the performance on meta-testing tasks remains unexplored.

To this end, we propose the Adversarial Task Up-sampling (ATU) framework to augment tasks that are aware of the task distribution, imaginary, and adaptive to the current meta-knowledge. Grounded on gradient-based meta-learning algorithms that are generally applicable to either regression or classification problems, ATU takes the initialization of the base learner as the meta-knowledge. Concretely, ATU consists of a task up-sampling network whose input is a task itself and outputs are augmented tasks. To ensure that the augmented tasks are imaginary and meanwhile faithful to the underlying task distribution, we train the up-sampling network to minimize the Earth Mover Distance between augmented tasks and the local task distribution characterized by a set of sampled tasks. Besides, we enforce the up-sampling network to produce challenging tasks that complement the current initialization, by maximizing the loss of the model adapted from the initialization on their query examples and minimizing the similarity between the gradient of the initialization with respect to their support examples and that of their query examples.

In summary, our main contributions are three-fold: (1) we present the first task-level augmentation network that learns to generate tasks that simultaneously meet the qualifications of being task-aware, task-imaginary, and model-adaptive; (2) we provide a theoretical analysis to justify that the proposed ATU framework indeed promotes task-awareness; (3) we conduct comprehensive experiments covering both regression and classification problems and a total of five datasets, where the proposed ATU improves the generalization ability of gradient-based meta-learning algorithms by up to 3.81%.

2 Related Work

As a paradigm that effectively adapts the meta-knowledge learned from past tasks to accelerate the learning of new ones, meta-learning has sparked considerable interest in many scenarios [38, 26, 36, 35], especially for few-shot learning. It falls into four major strands based on what the meta-knowledge is, i.e., optimizer-based methods [3, 37], feed-forward methods [24, 10, 39], metric-based methods [27, 29, 33] and gradient-based meth-

Table 1: Summary of existing task augmentation strategies.

Method	Task-aware	Task-imaginary	Model-adaptive
MetaAug [23]	✓	✗	✗
MetaMix [40]	✓	✗	✗
Meta-Maxup [20]	✓	✗	✗
MLTI [43]	✗	✓	✗
ATU	✓	✓	✓

ods [9, 15, 42, 12, 6], where the inner optimizer, the mapping function from the support set to the task-specific model, the distance metric measuring the similarity between samples, and the parameter initialization are formulated as the meta-knowledge that enables quick adaptation to a task within a small number of steps. Our method is primarily evaluated on gradient-based methods which enjoy wide adoption and applicability in either classification or regression problems.

Within-task Overfitting. Few-shot learning puts meta-learning, especially gradient-based methods which require optimization of high-dimensional parameters within each task, at risk of within-task overfitting. Some works tackle the problem with various ways of reducing the number of parameters to adapt in the inner loop: only updating the head [22] or the feature extractor [21], learning data-dependent latent generative embeddings of parameters [25] or context parameters [48] to adapt, imposing gradient dropout [32], and generating stochastic input-dependent perturbations [13]. The other bunch of works alleviates the problem through data augmentation within each task. Ni et al. [20] applies standard augmentation like Random Crop and CutMix onto support samples. Sun et al. [28] and Zhang et al. [47] proposed to generate more data within a class via a ball generator and generative adversarial networks, respectively. These techniques designed for within-task overfitting, however, have been proved to lend little support to meta-overfitting which we focus on.

Meta-overfitting. Distinguished from traditional overfitting within a task, two types of meta-overfitting including memorization and learner overfitting have been pinpointed in [44, 23]. Despite meta-regularization techniques [11, 44] that limit the capacity of the meta-learner, task augmentation strategies [23, 19, 20, 16, 43, 40] have emerged as more effective solutions to meta-overfitting. Table 1 presents a summary of these strategies, except the strategies of large rotation [16] being part of Meta-Maxup [20] and DReCa [19] applicable to natural language inference tasks only. MetaAug [23] augments a task by adding a random noise on labels of both support and query sets, and MetaMix [40] mixes support and query examples within a task. Such within-task augmentation guarantees the validity of augmented tasks, i.e., being task-aware, though it almost does not alter the mapping from the support to the query set, i.e., generating limited imaginary tasks beyond meta-training tasks. Meta-Maxup [20] and MLTI [43] approach this problem via the cross-task mixup method, unfavorably at the expense of erroneous tasks. Our work seeks a novel task augmentation framework capable of generating tasks that not only meet the task-awareness and task-imagination needs but also adapt to maximally benefit the up-to-the-minute meta-learner.

3 Preliminaries

3.1 Meta-Learning Problem and Gradient-Based Meta-Learning

Meta-Learning model f are trained and evaluated on episodes of few-shot learning tasks. Assume the task distribution is $p(\mathcal{T})$. A few-shot learning task T_i i.i.d. sampled from $p(\mathcal{T})$ consists of a support set $D_i^s = (X_i^s, Y_i^s) = \{(x_{i,j}^s, y_{i,j}^s)\}_{j=1}^{K^s}$ and a query set $D_i^q = (X_i^q, Y_i^q) = \{(x_{i,j}^q, y_{i,j}^q)\}_{j=1}^{K^q}$, where X_i^s and Y_i^s, (X_i^q and Y_i^q) are the collection of inputs and labels in support (query) set, and K^s (K^q) is the size of support (query) set.

The most representative gradient-based meta-learning algorithm is MAML [9]. MAML aims to learn an initialization parameter θ_0 of the model f that can be adapted to any new task after a few steps of gradient update. Concretely, given a specific task D_i^s, D_i^q and a parametric model f_θ, MAML initializes the model parameter θ by θ_0 and updates θ by performing gradient descent on the support set D_i^s. It then optimizes the initialization parameter θ_0 by minimizing the loss $\mathcal{L}$ estimated on the query set D_i^q. The objective of MAML can be formulated as

$$\min_{\theta_0} \mathbb{E}_{T_i \sim p(\mathcal{T})}\, \mathcal{L}(\phi_i, D_i^q), \qquad \text{s.t.} \quad \phi_i = \theta_0 - \alpha \nabla_{\theta_0} \mathcal{L}(\theta_0, D_i^s). \tag{1}$$

3.2 Earth Mover's Distance

To estimate the distance between two tasks, we use Earth-Mover Distance (EMD). Earth-Mover Distance, a.k.a. Wasserstein metric, is a distance measure of two probability distributions or two sets of points, and is widely used in image retrieval and point cloud up-sampling works [46, 45]. Given two sets S_1 and S_2 with the same size, EMD calculates their distances as:

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \frac{1}{\|S_1\|} \sum_{s \in S_1} \|s - \phi(s)\|_2 \tag{2}$$

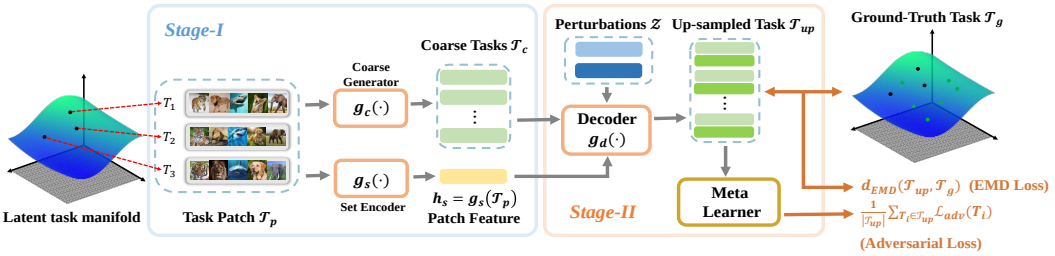


Figure 2: Illustration of the ATU algorithm with a task up-sampling network. The task up-sampling network consists of a set encoder $g_s(\cdot)$ which extracts a set feature of the input task patch, a coarse task generator $g_c(\cdot)$ which generates coarse tasks given the task patch, and a decoder $g_d(\cdot)$ which generates fine tasks from coarse tasks based on random perturbance and the set feature.

where ϕ is a bijective projection mapping S_1 to S_2. The value of EMD in (2) can be obtained by solving the linear programming problem w.r.t. ϕ.

4 Adversarial Task Up-sampling

In practice, the task distribution $p(\mathcal{T})$ is unknown and we optimize the meta parameter θ_0 with an empirical estimation of Eq. (1) over of meta-training tasks $\{T_i\}_{i=1}^{N_T}$ as

$$\min_{\theta_0} \frac{1}{N_T} \sum_{i=1}^{N_T} \mathcal{L}(\phi_i, D_i^q), \qquad \text{s.t.} \quad \phi_i = \theta_0 - \alpha \nabla_{\theta_0} \mathcal{L}(\theta_0, D_i^s) \tag{3}$$

Given a finite set of meta-training tasks, the empirical task distribution may deviate from the true task distribution. The meta-model trained on such a finite set of tasks will cause memorization or learner overfitting [44, 23], which hurts the generalization to new tasks. To alleviate this problem, we propose a new task up-sampling network to generate a sufficient number of diverse tasks such that the empirical task distribution formed by the original meta-training tasks and the augmented tasks together is closer to the true task distribution. To achieve this, the tasks generated by the task up-sampling network should match the true task distribution and cover a large fraction of it. However, since the true task distribution and its underlying manifold are unknown, we cannot provide the task up-sampling network with explicit information about it. Instead, we generate new tasks by performing Task Up-sampling (TU) from a set of training tasks that implicitly comprise the latent task manifold information. The idea of Task Up-sampling is inspired by point cloud up-sampling methods [46, 45], which generate up-sampled points lying on the latent distribution (i.e., the shape) of the given local point patch. Similar to the point cloud up-sampling algorithm, our augmentation network receives a task patch consisting of a set of tasks $\mathcal{T}_p = \{T_i\}_{i=1}^{N_p}$ where N_p is the set size, and generates up-sampled tasks $\mathcal{T}_{up} = \{\hat{D}^s, \hat{D}^q\}$ that are uniformly distributed over the same underlying task distribution as the task patch.

Due to the high complexity of task generation, it is infeasible to directly generate up-sampling tasks without sacrificing the quality of the tasks. Inspired by [46], we propose a two-stage generation strategy to generate the up-sampled tasks. In the first stage, we produce a sparse set of tasks, aiming at recovering the *global* task distribution of the task patch. The tasks obtained in the first stage are called coarse tasks. In the second stage, we generate multiple tasks for each coarse task, aiming to characterize the *local* task distribution around each coarse task. To guide the second generation stage, we use the patch features of the input task set as input to provide global task information and also multiple random noise vectors as input to provide directional perturbations to generate diverse tasks around the coarse task. The generation process is summarized in Fig. 2.

Our proposed task up-sampling network consists of 3 components, namely, a coarse task generator $g_c(\cdot)$, a set encoder $g_s(\cdot)$, and a decoder $g_d(\cdot)$. The coarse task generator is similar to a set auto-encoder. It first encodes the information of the whole input task set and then decodes it to generate $r_c N_p$ coarse tasks $\mathcal{T}_c = \{T_i^c\}$. The set encoder, denoted by $g_s(\cdot)$, extracts the set information of the input task patch as a patch feature h_s to provide global information in the second-stage generation. For each task T_i^c in $\mathcal{T}_c$, the decoder $g_d(\cdot)$ generates r_d tasks located around T_i^c in the task manifold

by taking as input r_d random perturbations $\{z_i\}_{i=1}^r$ that are i.i.d. noise sampled from a uniform distribution and the set feature h_s as input. In general, we use the same perturbations for each coarse task T_i^c in $\mathcal{T}_c$. Finally, we obtain the up-sampled task set $\mathcal{T}_{up}$ consisting of rN_p tasks as $\mathcal{T}_{up} = g_d(\mathcal{T}_c, \mathcal{Z}, h_s)$, where $r = r_c \times r_d$ is the up-sampling ratio. We denote the task augmentation network by $G_{\theta_g}(\mathcal{T}_p, \mathcal{Z})$ where θ_g is the trainable parameters of the task augmentation network.

In each iteration of the training phase, we construct rN_p tasks to form the ground truth tasks set $\mathcal{T}_g$ (e.g., randomly select from the meta training task set). Then we sample N_p tasks from $\mathcal{T}_g$ to form the task patch $\mathcal{T}_p$ and randomly sample r perturbation noise vectors to form the perturbation set $\mathcal{Z} = \{z_i\}_{i=1}^r$. By feeding $\mathcal{T}_p$ and $\mathcal{Z}$ to the task augmentation network, we obtain the up-sampled task set $\mathcal{T}_{up} = G_{\theta_g}(\mathcal{T}_p, \mathcal{Z})$. To train the task augmentation network, we apply EMD loss between the up-sampled task set $\mathcal{T}_{up}$ and the ground-truth task set $\mathcal{T}_g$ to encourage the generated task set to have the same distribution as the true task distribution. However, it may still be insufficient to make the up-sampled tasks cover a significant fraction of the true task distribution. In this case, the up-sampling tasks provide limited additional information compared with the original meta-training tasks, and thus the meta-learner has restricted benefit from the generated tasks. To generate more informative tasks for the meta-learner, we want the generated tasks to be difficult for the current meta model θ_0. Following [41], we measure the difficulty of the a task for θ_0 by the loss estimated on query set w.r.t. to ϕ_i, i.e. $\mathcal{L}(\phi_i, \hat{D}_i^q)$, and the gradient similarity between the support and query sets w.r.t. θ_0, i.e. $\langle \nabla_{\theta_0}\mathcal{L}(\theta_0, \hat{D}_i^s), \nabla_{\theta_0}\mathcal{L}(\theta_0, \hat{D}_i^q)\rangle$. The large loss and small gradient similarity indicate a difficult task. Therefore, we want to maximize the following objective function to generate informative tasks:

$$\mathcal{L}_{adv}(\theta_0, (\hat{D}_i^s, \hat{D}_i^q)) = \eta_1 \mathcal{L}(\phi_i, \hat{D}_i^q) - \eta_2 \langle \nabla_{\theta_0}\mathcal{L}(\theta_0, \hat{D}_i^s), \nabla_{\theta_0}\mathcal{L}(\theta_0, \hat{D}_i^q)\rangle, \tag{4}$$

where η_1 and η_2 are two hyperparameters that control the strength of the two terms in $\mathcal{L}_{adv}$. We call this loss adversarial loss because it aims to increase the difficulty of the up-sampling tasks for the meta-learner while the meta-learner is trained to minimize the loss on the generated difficult tasks. And we named the proposed algorithm as Adversarial Task Up-sampling (ATU).

Together with the EMD loss, we obtain the objective to train the task up-sampling network:

$$\mathcal{L}_{ATU}(\theta_g, \mathcal{T}_p) = d_{EMD}(\mathcal{T}_{up}, \mathcal{T}_g) - \frac{1}{rN_p} \sum_{\hat{T}_i \in \mathcal{T}_{up}} \mathcal{L}_{adv}(\theta_0, (\hat{D}_i^s, \hat{D}_i^q)). \tag{5}$$

Note that the gradient of Eq. (5) will not be backpropagated to the meta model θ_0 and the meta model will be updated by minimizing the meta loss in Eq. (3) on the up-sampled tasks $\mathcal{T}_{up}$. We summarize the proposed ATU in Algorithm 1 in Appendix A.

4.1 ATU on Regression and Classification Problem

Before introducing the details of regression and classification tasks, let us first review the Eq. (2), where S_1 and S_2 can be understood as the set of up-sampled tasks $\mathcal{T}_{up}$ and the set of ground-truth tasks $\mathcal{T}_g$, respectively. Each point s in either set represents the embedding of a task.

Regression Tasks. We consider a simple regression problem: sinusoidal regression, which is widely used to evaluate the effectiveness of the meta-learning methods. In sinusoidal regression problem, before feeding a task $T_i = (D_i^s, D_i^q)$ to the task up-sampling network, we need to present the embedding of a sine regression task at first. In this paper, we combine all samples of the support set and query set as the embedding of a sine regression task, i.e., $s = [x_1^s, y_1^s, x_2^s, y_2^s, ..., x_{K^s}^s, y_{K^s}^s, x_1^q, y_1^q, x_2^q, y_2^q, ..., x_{K^q}^q, y_{K^q}^q] \in \mathbb{R}^{2(K^s+K^q)}$, where we sort the support set and query set such that $x_1^s \leq x_2^s \leq ... \leq x_{K^s}^s$ and $x_1^q \leq x_2^q \leq ... \leq x_{K^q}^q$. This sorting could make the task input is invariant to the permutation of data in support and query sets and thus the extracted feature of each task is permutation-invariant, which simplifies the design of the task up-sampling network. To generate the coarse tasks, we first use the set encoder $g_s(\cdot)$ to extract the set feature h_s and directly generate the coarse tasks from the set feature h_s. Then we generate the up-sampled tasks $\mathcal{T}_{up}$ utilizing the decoder $g_d(\cdot)$ with perturbations z. Consequently, the dimension of generated tasks $\mathcal{T}_{up}$ is $(rN_p, 2(K^s, K^q))$, and that of ground truth tasks $\mathcal{T}_g$ is also $(rN_p, 2(K^s, K^q))$, where r is the up-sampling ratio and N_p is the set size of a task patch. For the regression problem, we add an extra EMD loss on the support and query set for each generated task $\hat{T}_i \in \mathcal{T}_{up}$ to encourage the points in the generated support set and query set to follow the same sinusoidal distribution, and the objective is shown in Eq. (6).

$$\mathcal{L}_{ATU}(\theta_g, \mathcal{T}_p) = d_{EMD}(\mathcal{T}_{up}, \mathcal{T}_g) + \eta_3 \frac{1}{rN_p} \sum_{\hat{T}_i \in \mathcal{T}_{up}} d_{EMD}(\hat{D}_i^s, \hat{D}_i^q) - \frac{1}{rN_p} \sum_{\hat{T}_i \in \mathcal{T}_{up}} \mathcal{L}_{adv}(\theta_0, (\hat{D}_i^s, \hat{D}_i^q)), \tag{6}$$

where $\hat{T}_i = (\hat{D}_i^s, \hat{D}_i^q)$ is obtained by transforming each of $\mathcal{T}_{up}$ back to a support set and a query set.

Classification Tasks. Dissimilar to regression tasks, classification tasks' labels of each class are randomly given under the mutually-exclusive setting [44]. For example, for an N-way K^s-shot classification problem with K^q query samples for each class, the label y in episodic-based meta-training is a randomly chosen value from $\{0, 1, ..., N-1\}$. In light of the fact that the label y is not semantically meaningful, we only use the images x to represent the embedding of an N-way classification task. Concretely, we reshape each task into a task pool of (K^s+K^q) tasks, each of which is N-way 1-shot and represented as $s = [x_1, x_2, ..., x_N] \in \mathbb{R}^{Nd}$ where d is the dimension of each image. Then, we can split the task into (K^s+K^q) N-way 1-shot classification tasks without query examples. We represent the task by concatenating the input from N classes in a fixed order (based on classes). We treat the (K^s+K^q) tasks as a task patch and feed them to the task up-sampling network.

Since the task distribution of the image classification problem is extremely complex, it is impractical to generate the coarse tasks from a set feature. Instead, we use the original tasks as the coarse tasks and generate the up-sampled tasks by a more informative perturbance around the original tasks. To achieve this, we generate the perturbation by randomly sampling extra K_M images from N different classes in the base set. Then for the image x_i in a class of a task in the coarse tasks $\mathcal{T}_c$, we subtract it from the K_M images to obtain K_M residual images and their corresponding set features (i.e., obtained from $g_s(\cdot)$), concatenate the set features with a noise vector, and use an attention network to obtain a residual images feature x_i^{res} for the image x_i given K_M residual images. Finally, we generate the image as $x_i^u = x_i + x_i^{res}$ for the augmented task. We repeat this process for all images in a coarse task to get an augmented task and apply the r noise vector to get r up-sampled tasks. The dimension of generated tasks $\mathcal{T}_{up}$ and ground truth tasks $\mathcal{T}_g$ are both $(r(K^s + K^q), Nd)$, where $\mathcal{T}_g$ is obtained by mixing up with images in the memory bank. The whole training objective function is Eq. (5). More details of network structures and training details are shown in Appendix B.

5 Theoretical Analysis

We will introduce the formal definition of an up-sampled task that conforms to task-awareness, based on which we present the essential property of our proposed ATU framework in maximizing the task-awareness, compared to previous task augmentation approaches.

Definition 1 (Task-aware Up-sampling)**.** Suppose that we are given a set of N_p tasks $\{\mathbf{X}_i, \mathbf{Y}_i\}_{i=1}^{N_p}$ from which we up-sample a new task T_{up}. For each i-th task, its ground-truth parameter that map the input $\mathbf{X}_i$ to the output $\mathbf{Y}_i$ is θ_i, i.e., $\mathbf{Y}_i = f_{\theta_i}(\mathbf{X}_i)$. The up-sampled task $T_{up} = \{\mathbf{X}_u, \mathbf{Y}_u\}$ is defined to be task-aware, if and only if $\theta_u = g(\theta_1, \cdots, \theta_{N_u})$ and $\mathbf{Y}_u = f_{\theta_u}(\mathbf{X}_u)$ where g is the up-sampling function and θ_u is the up-sampled parameter.

This definition states two prerequisites a task-aware up-sampling has to meet: (1) the up-sampling is performed in the functional space, which is to relate N_u parameters via g; (2) the mapping between the input and the output of an up-sampled task satisfies f_{θ_u}.

Property 1 (Task-awareness Maximization)**.** Consider $N_u = 2$, $g(\theta_1, \theta_2) = (1-\lambda)\theta_1 + \lambda\theta_2$, $f_{\theta_1}(\cdot) = \mathbf{W}_1$, and $f_{\theta_2}(\cdot) = \mathbf{W}_2$. The proposed ATU algorithm that pursues an up-sampled task $T_{up} = \{\mathbf{X}_u, \mathbf{Y}_u\}$ via minimizing the EMD loss between T_1 and T_2 maximizes the task-awareness, i.e., minimizing the distance between $\mathbf{Y}_u$ and $f_{\theta_u}(\mathbf{X}_u)$.

Proof. According to the definition of EMD (Eq. (2)), it solves: $\phi^* = \arg\min_{\phi \in \mathbf{\Phi}} \sum_j \|\mathbf{x}_{1,j} - \mathbf{x}_{2,\phi(j)}\|_2$, where $\mathbf{\Phi} = \{\{1, \cdots, n_1\} \mapsto \{1, \cdots, n_2\}\}$ denotes the set containing all possible bijective assignments, each of which gives one-to-one correspondence between T_1 and T_2. Based on the optimal assignments ϕ^*, the EMD is known to be defined as $d_{EMD} = \frac{1}{min\{n_1, n_2\}} \sum_j \|\mathbf{x}_{1,j} - \mathbf{x}_{2,\phi^*(j)}\|_2$. In light of the difficulty in mathematically formulating a possible up-sampled task $\tilde{T}_u$ that lies in the local manifold of $\{T_1, T_2\}$, we reasonably assume a simplified way of characterizing an up-sampled task $\tilde{T}_u$ to be $\tilde{\mathbf{y}}_{u,j} = \boldsymbol{\alpha}_{1,j}^T \mathbf{Y}_1 + \boldsymbol{\alpha}_{2,j}^T \mathbf{Y}_2$, $\tilde{\mathbf{x}}_{u,j} = \boldsymbol{\alpha}_{1,j}^T \mathbf{X}_1 + \boldsymbol{\alpha}_{2,j}^T \mathbf{X}_2$, $\forall j$, where each sample is a convex

combination of samples from both T_1 and T_2. The combination coefficients $\boldsymbol{\alpha}_{1,j}, \boldsymbol{\alpha}_{2,j} \in \mathbb{R}^{(K^s+K^q)\times 1}$, $\sum_k^{K^s+K^q} \alpha_{1,jk}=1, \sum_k \alpha_{2,jk}=1, \alpha_{1,jk}, \alpha_{2,jk} \geq 0, \forall k$. Different combination coefficients lead to a set of up-sampled task candidates $\{\tilde{T}_u\}$. We evaluate the task-awareness property of each candidate $\tilde{T}_u$, i.e., the distance between $\tilde{\mathbf{Y}}_u$ and $f_{\theta_u}(\tilde{\mathbf{X}}_u)$, to be $\|\tilde{\mathbf{Y}}_u - f_{\theta_u}(\tilde{\mathbf{X}}_u)\|_2 = \sum_j \|\tilde{\mathbf{y}}_{u,j} - f_{\theta_u}(\tilde{\mathbf{x}}_{u,j})\|_2 = \sum_j \|(\mathbf{W}_1 - \mathbf{W}_2)[\lambda \alpha_{1,j}^T \mathbf{X}_1 - (1-\lambda)\alpha_{2,j}^T \mathbf{X}_2]\|_2 =$ LHS. (See Appendix E.)

Note that LHS $\leq \sum_j \|\mathbf{W}_1 - \mathbf{W}_2\|_2 (\lambda \|\tilde{\mathbf{x}}_{u,j} - \mathbf{x}_{2,\phi_2(j)}\|_2 + \|\mathbf{X}_2\|_2)$ and LHS $\leq \sum_j \|\mathbf{W}_1 - \mathbf{W}_2\|_2 ((1-\lambda)\|\mathbf{x}_{1,\phi_1(j)} - \tilde{\mathbf{x}}_{u,j}\|_2 + \|\mathbf{X}_1\|_2)$. (See Appendix E.) By combining the two inequalities above, we have LHS$\leq \sum_j \|\mathbf{W}_1 - \mathbf{W}_2\|_2 \min\{\lambda \|\tilde{\mathbf{x}}_{u,j} - \mathbf{x}_{2,\phi_2(j)}\|_2 + \|\mathbf{X}_2\|_2, (1-\lambda)\|\lambda \mathbf{x}_{1,\phi_1(j)} - \tilde{\mathbf{x}}_{u,j}\|_2 + \|\mathbf{X}_1\|_2\}$. In practice, it is easy to normalize all the tasks in the feature space, which leads to $\|\mathbf{X}_1\|_2 = \|\mathbf{X}_2\|_2$. Therefore, by minimizing the EMD loss $\mathrm{d}_{EMD} = \min\{\min_{\phi_2} \sum_j \|\tilde{\mathbf{x}}_{u,j} - \mathbf{x}_{2,\phi_2(j)}\|_2, \min_{\phi_1} \sum_j \|\tilde{\mathbf{x}}_{u,j} - \mathbf{x}_{1,\phi_1(j)}\|_2\}$, the proposed task up-sampling network identifies from the candidate set $\{\tilde{T}_u\}$ the task T_u that has the minimal distance between $\mathbf{Y}_u$ and $f_{\theta_u}(\mathbf{X}_u)$; in other words, the task-awareness is maximized. □

Previous task augmentation approaches directly mix up two tasks without minimizing the EMD loss, i.e., $\mathbf{y}_{u,j} = (1-\lambda)\mathbf{y}_{1,j} + \lambda \mathbf{y}_{2,j}, \mathbf{x}_{u,j} = (1-\lambda)\mathbf{x}_{1,j} + \lambda \mathbf{x}_{2,j}$. In this case, the task-awareness is unwarranted as we have illustrated in Section 1, provided that $\|\mathbf{Y}_u - f_{\theta_u}(\mathbf{X}_u)\|_2 = \sum_j \|(1-\lambda)\mathbf{y}_{1,j} + \lambda \mathbf{y}_{2,j} - [(1-\lambda)\mathbf{W}_1 + \lambda \mathbf{W}_2][(1-\lambda)\mathbf{x}_{1,j} + \lambda \mathbf{x}_{2,j}]\|_2 = \sum_j \lambda^2 (1-\lambda)^2 \|(\mathbf{W_1} - \mathbf{W}_2)(\mathbf{x}_{1,j} - \mathbf{x}_{2,j})\|_2$.

6 Experiments

To evaluate the effectiveness of ATU, we conduct extensive experiments to answer the following questions: **Q1:** How does ATU perform compared to state-of-the-art task-augmentation-based and regularization meta-learning methods? **Q2:** Whether can the proposed ATU consistently improve performance for different meta-learning methods? **Q3:** What does up-sampled task by ATU looks like? **Q4:** What is the influence of increasing the task number within meta-training data on the performance improvement of ATU? **Benchmarks.** We compared ATU with state-of-the-art task augmentation strategies for meta-learning, including MetaAug [23], MetaMix [40], Meta-Maxup [20], MLTI [43], and regularization methods, including MetaDropout [13], TAML [11], and Meta-Reg [44]for both regression and classification problems. We also consider a variant of ATU which removes the adversarial loss $\mathcal{L}_{adv}$ and trains the task augmentation network only through the EMD loss. We denote this variant by TU. To validate the consistent effect of ATU in improving different meta-learners, we apply ATU and AU on MAML [15], MetaSGD [15] and ANIL [22]. We also consider cross-domain settings where the meta-testing tasks are from different domains.

Table 2: MSE with $\pm 95\%$ confidence intervals on sinusoidal regression.

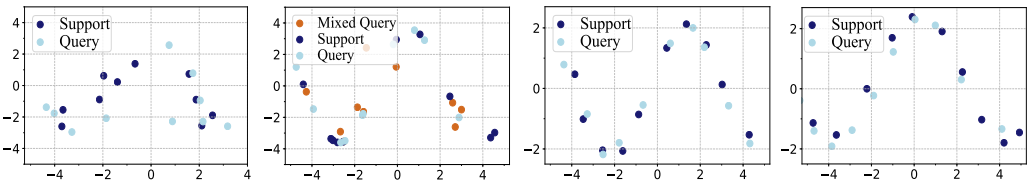
Model	10-shot	20-shot	30-shot
DropGrad [32]	0.91 ± 0.17	0.62 ± 0.12	0.55 ± 0.13
MetaAug [23]	0.93 ± 0.18	0.65 ± 0.14	0.58 ± 0.12
Meta-Learner: MAML			
MAML [9]	0.93 ± 0.18	0.65 ± 0.13	0.58 ± 0.12
MetaMix [40]	0.81 ± 0.17	0.58 ± 0.12	0.56 ± 0.11
MLTI [43]	0.92 ± 0.17	0.65 ± 0.13	0.62 ± 0.12
TU	0.84 ± 0.16	0.55 ± 0.12	0.47 ± 0.10
ATU	$\mathbf{0.70 \pm 0.14}$	$\mathbf{0.47 \pm 0.13}$	$\mathbf{0.42 \pm 0.11}$
Meta-Learner: MetaSGD			
MetaSGD [15]	0.70 ± 0.17	0.49 ± 0.11	0.42 ± 0.09
MetaMix [40]	0.60 ± 0.15	0.37 ± 0.09	0.37 ± 0.08
MLTI [43]	0.66 ± 0.16	0.51 ± 0.11	0.44 ± 0.10
TU	0.54 ± 0.11	0.36 ± 0.08	0.31 ± 0.08
ATU	$\mathbf{0.49 \pm 0.10}$	$\mathbf{0.34 \pm 0.08}$	$\mathbf{0.29 \pm 0.08}$

Table 3: MSE with $\pm 95\%$ confidence intervals on cross-domain sinusoidal regression.

Cross-domain	Frequency [0.4,0.8]	Aplitude [5.0,6.0]	Phase $[-\pi,0]$
Meta-Learner: MAML			
MAML [9]	1.78 ± 0.35	3.52 ± 0.35	3.12 ± 0.52
MetaMix [40]	1.67 ± 0.30	3.60 ± 0.28	3.14 ± 0.54
MLTI [43]	1.92 ± 0.42	3.56 ± 0.37	3.66 ± 0.63
TU	1.70 ± 0.34	3.22 ± 0.31	2.88 ± 0.48
ATU	$\mathbf{1.58 \pm 0.35}$	$\mathbf{2.92 \pm 0.29}$	$\mathbf{2.58 \pm 0.48}$
Meta-Learner: MetaSGD			
MetaSGD [15]	2.24 ± 0.46	2.42 ± 0.32	2.73 ± 0.56
MetaMix [40]	1.77 ± 0.35	2.50 ± 0.27	2.46 ± 0.48
MLTI [43]	1.80 ± 0.42	2.56 ± 0.28	2.54 ± 0.54
TU	$\mathbf{1.64 \pm 0.38}$	2.37 ± 0.25	$\mathbf{2.04 \pm 0.46}$
ATU	1.71 ± 0.40	$\mathbf{2.19 \pm 0.23}$	2.53 ± 0.62

6.1 Regression

Experimental Setup. Following [15], we construct the K-shot regression task by sampling from the target sine curve $y(x) = Asin(\omega x + b)$, where the amplitude $A \in [0.1, 5.0]$, the frequency $\omega \in [0.8, 1.2]$, the phase $b \in [0, \pi]$ and x is sampled from $[-5.0, 5.0]$. In the meta-training phase, each task contains K support and K target (K=10) examples. We adopt mean squared error (MSE) as the loss function. For the base model f_θ, we adopt a small neural network, which consists of an input layer of size 1, 2 hidden layers of size 40 with ReLU and an output layer of size 1. We use one



(a) MLTI (b) MetaMix (c) TU (d) ATU

Figure 3: The augmented regression tasks generated by different augmentation-based methods.

gradient update with a fixed step size $\alpha = 0.01$ in inner loop, and use Adam as the outer-loop optimizer following [9, 15]. Moreover, the meta-learner is trained on 240,000 tasks with meta batch-size being 4. In meta-testing stage , we randomly sample 100 sine curves as meta-test tasks, each task containing K support samples and 100 query examples. The data points x in query set are evenly distributed on $[-5.0, 5.0]$. The averaged MSE with 95% confidence intervals upon these 100 sine curves with K=10, 20, 30 are reported in Table 2. We also perform cross-domain experiments by sampling 100 sine curves which have different frequencies, amplitudes or phases from the tasks in meta-training set and report the results in Table 3. More settings about the up-sampling networks are listed in Appendix C.

Performance. The results in Table 2 and Table 3 show that ATU consistently outperforms the baseline methods MAML, MetaMix, and MLTI in different K-shot (K$\in \{10, 20, 30\}$) settings and various domain settings. These results validate that the tasks generated by ATU can better approximate the true task distribution and provide more information to the meta-learner than MetaMix and MLTI, thus enabling better generalization of the model. We further verify the superiority of the proposed methods by visualizing the augmented tasks generated by the proposed methods and the baseline methods. The visualization results in Fig. 3 show that the points in the tasks generated by TU and ATU fit the sine curve well, while the points in the tasks generated by MLTI and MetaMix deviate from the sine curve. This indicates that augmented tasks generated by TU and ATU match the true task distribution. It is noteworthy that the support set and query set generated by ATU differ significantly from those generated by TU, which indicates that the task generated by ATU is more difficult . This, together with the results that ATU outperforms TU in most experiments, demonstrates the effectiveness of the adversarial losses in generating informative tasks to improve generalization of the meta-learner.

In Fig. 4, we also visualize the adaptation of meta-learner trained by different task augmentation methods for a 10-shot meta-test regression task. Compared to the MAML trained on original meta-training tasks, the MAML trained on tasks generated by TU fits the ground-truth sinusoid after only one update. And ATU performs even better than TU. This again validates that the augmented tasks generated by TU and ATU are more informative for the meta-learner to learn the meta knowledge from the true task distribution.

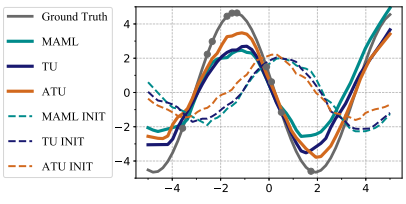


Figure 4: Initialization (dotted) and one-step adaptation (solid) regression curves of MAML,TU and ATU when K=10.

6.2 Classification

Experimental setup. We follow MLTI to evaluate the performance of task augmentation algorithms for few-shot classification problem with limited number of base classes in meta-training set under non-label-sharing settings. We consider four datasets (base classes number): miniImagenet-S (12), ISIC [18] (4), Dermnet-S (30), and Tabular Murris [5] (57) covering classification tasks on general natural images, medical images, and gene data. Note that the miniImagenet-S and Dermnet-S are constructed by limiting the base classes of miniImangenet [33] and Dermnet [1], respectively. We construct N-way K-shot tasks, setting $N = 5$ for miniImagenet-S, Derment-S, Tabular Murris and $N = 2$ for ISIC dataset due to its limited number of base classes and setting $K = 1$ or $K = 5$. Recall that TAU relies on extra K_M images to generate augmented tasks in image classification problem. To make the training process more efficient, we set K_M to 3 and the upsampling rate r be 2. More details of these datasets and the settings of the task augmentation networks are listed in Appendix D.

Table 4: Average accuracy under different settings of few-shot classification and various datasets.

Model	miniImagenet-S		ISIC		DermNet-S		Tabular Murris	
	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
MAML [9]	38.27%	52.14%	57.59%	65.24%	43.47%	60.56%	79.08%	88.55%
Meta-Reg [44]	38.35%	51.74%	58.57%	68.45%	45.01%	60.92%	79.18%	89.08%
TAML [11]	38.70%	52.75%	58.39%	66.09%	45.73%	61.14%	79.82%	89.11%
Meta-Dropout [13]	38.32%	52.53%	58.40%	67.32%	44.30%	60.86%	78.18%	89.25%
MetaMix [40]	39.43%	54.14%	60.34%	69.47%	46.81%	63.52%	81.06%	89.75%
Meta-Maxup [20]	39.28%	53.02%	58.68%	69.16%	46.10%	62.64%	79.56%	88.88%
MLTI [43]	41.58%	55.22%	61.79%	70.69%	48.03%	64.55%	81.73%	91.08%
TU	42.16%	56.33%	62.03%	73.97%	48.07%	64.81%	81.88%	91.15%
ATU	**42.60%**	**56.78%**	**62.84%**	**74.50%**	**48.33%**	**65.16%**	**82.04%**	**91.42%**

Performance. We show the performance on the four datasets in Table 4. On all four datasets, the proposed ATU consistently outperforms the baseline methods, including the augmentation-based methods (i.e. MetaMix, Meta-Maxup and MLTI) and regularization-based methods (Meta-Reg, TAML and Meta-Dropout). And TU achieves the second best performance on all experiments. We also observe that our method achieves a large improvement on the ISIC dataset which consists of only 4 base classes, indicating the effectiveness of our method in limited tasks scenarios. We further evaluate the effectiveness of the proposed ATU on improving the generalization for different backbone meta-learner by conducting experiments under 1-shot setting to compare the performance of MLTI and ATU in improving the performance of the backbone meta-learner MetaSGD and ANIL. The results are presented in Table 5. ATU again consistently outperforms MLTI. All these results validate the superiority of the proposed ATU and TU over the existing baselines in generating informative tasks to improve the performance of different backbone meta-learners. We also evaluate the performance of ATU in cross-domain adaptaion settings. In Table 6, we present the results of the experiment that apply the meta-model trained on miniImageNet-S to Dermnet-S, and vice versa. ATU improves the generalization performance of MAML (the backbone meta-learner in this experiment) by a large margin. This indicates ATU can consistently improve the backbone meta-model's generalization ability under the challenging cross-domain settings.

Table 5: Comparison of compatibility with different backbone meta-learning algorithms on 1-shot classification.

Method	mini-S	ISIC	Derm-S	TM
MetaSGD [15]	37.88%	58.79%	42.07%	81.55%
MetaSGD+MLTI	39.58%	61.57%	45.49%	83.31%
MetaSGD+ATU	**40.52%**	**62.84%**	**46.78%**	**83.84%**
ANIL [22]	38.02%	59.48%	44.58%	75.67%
ANIL+MLTI	39.15%	61.78%	46.79%	77.11%
ANIL+ATU	**39.27%**	**62.12%**	**47.03%**	**77.23%**

Table 6: Cross-domain adaptation experiments between mini-S and Dermnet-S. A→B denotes that the backbone meta-model is meta-trained on A and meta-tested on B.

Model	mini-S→ Derm-S		Derm-S→ mini-S	
	1-shot	5-shot	1-shot	5-shot
MAML [9]	34.46%	50.36%	28.78%	41.29%
MAML+ATU	**36.86%**	**51.98%**	**30.68%**	**46.72%**
MetaSGD [15]	31.07%	49.07%	28.17%	41.83%
MetaSGD+ATU	**37.75%**	**54.60%**	**30.78%**	**44.01%**

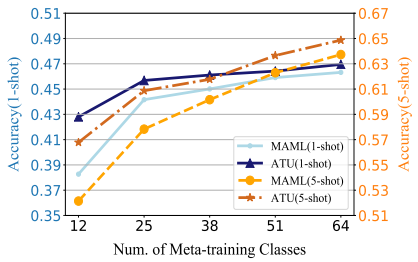


Figure 5: The averaged accuracy on the miniImagenet dataset with different number of tasks.

Figure 6: T-SNE visualization of original and up-sampled tasks on 1-shot miniImagenet-S setting.

Effect of the number of meta-training tasks. We conduct experiments to analyze the change in the performance improvement of ATU over MAML with the number of meta-training tasks in 1-shot and 5-shot settings. The results presented in Fig. 5 show that ATU significantly improves the performance of MAML by about 4.5%-5% when the number of base classes is 12, while the improvement decreases with the number of base classes increasing on both 1-shot and 5-shot settings. When the number of base classes increases, the number of training tasks increases rapidly and the empirical task distribution constructed from the meta-training tasks becomes closer and closer to the true latent task distribution. Therefore, the extra information provided by tasks generated by ATU becomes less. However, even if all available base classes are used in the meta-training (i.e., 64 meta-training classes), our proposed ATU could still help to improve the performance of MAML.

Visualization of the generated tasks. We visualize the up-sampled tasks by t-SNE to evaluate their generation quality for MAML under the 1-shot miniImagenet-S setting. Concretely, we up-sample 100 tasks for 5 original tasks via ATU by using different perturbations for each task. In order to visualize the relationship between generated tasks and original tasks using t-SNE, we represent each task by concatenating the vector of the support and query sets. The results presented in Fig. 6 show that the up-sampled tasks stay near the original tasks, which means they are matching the true task distribution. This indicates the generated tasks are task-aware. Besides, we can observe that the augmented tasks are diverse and cover a substantial portion of the original tasks. This demonstrates the task imaginary property of the augmented tasks. These two observations suggest that the proposed ATU is a qualified task augmentation algorithm.

Effect of the extra $d_{EMD}(\hat{D}_i^s, \hat{D}_i^q)$ in regression tasks. As presented in Section 4.1, we propose to apply an extra EMD loss $d_{EMD}(\hat{D}_i^s, \hat{D}_i^q)$ on the support and query set for each generated task to encourage the points in the generated support set and the query set to follow the same sine curve. In Fig. 3, we have visualized the tasks generated by the Task Up-sampling Network trained with the extra EMD loss. Here we provide additional visualization result for tasks generated by the Task Up-sampling Network trained without the extra EMD loss in Fig. 7. It can be seen that the support set and the query set are not on the same sinusoid, indicating that the generated tasks need additional supervision to avoid them being too difficult or not even valid tasks.

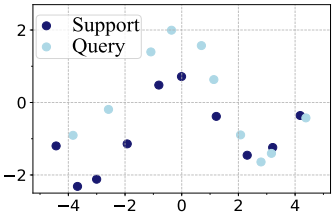


Figure 7: Visualization of the up-sampled task generated by ATU when $\eta_3 = 0$ in Eq. (6).

7 Conclusion and Limitation

In this paper, we propose the first task-level up-sampling network that learns to generate tasks that simultaneously meet the qualifications of being task-aware, task-imaginary, and model-adaptive. The proposed Adversarial Task Up-sampling (ATU) takes a set of tasks as input and learn to up-sample tasks complying with the true task distribution while being informative to improve the generalization of the meta-learner. We theoretically justify that ATU promotes task-awareness and empirically verify that ATU improves the generalization of various backbone meta-learner for both regression and classification tasks on five datasets. **Limitations.** Our theoretical results are obtained under some strong assumptions, but all the experiments and visualization outcomes validate our method's effectiveness in real settings.

8 Acknowledgement

This work is sponsored by the Tencent AI Lab Gift Fund (Project 9229073) and CityU Strategic Interdisciplinary Research Grant (Project 7020064).

References

[1] Dermnet dataset, 2016. `http://www.dermnet.com/`.

[2] Fungi dataset, 2018. `https://www.kaggle.com/c/fungi-challenge-fgvc-2018`.

[3] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, pages 3981–3989, 2016.

[4] Jonathan Baxter. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning*, 28(1):7–39, 1997.

[5] Kaidi Cao, Maria Brbic, and Jure Leskovec. Concept learners for few-shot learning. In *ICLR*, 2020.

[6] Can Chen, Xi Chen, Chen Ma, Zixuan Liu, and Xue Liu. Gradient-based bi-level optimization for deep learning: A survey. *arXiv preprint arXiv:2207.11719*, 2022.

[7] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *ICLR*, 2018.

[8] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *CVPR*, pages 3606–3613, 2014.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017.

[10] David Ha, Andrew M Dai, and Quoc V Le. Hypernetworks. In *ICLR*, 2017.

[11] Muhammad Abdullah Jamal and Guo-Jun Qi. Task agnostic meta-learning for few-shot learning. In *CVPR*, pages 11719–11727, 2019.

[12] Hae Beom Lee, Hayeon Lee, Donghyun Na, Saehoon Kim, Minseop Park, Eunho Yang, and Sung Ju Hwang. Learning to balance: Bayesian meta-learning for imbalanced and out-of-distribution tasks. In *ICLR*, 2020.

[13] Hae Beom Lee, Taewook Nam, Eunho Yang, and Sung Ju Hwang. Meta dropout: Learning to perturb latent features for generalization. In *ICLR*, 2019.

[14] Xiaomeng Li, Lequan Yu, Yueming Jin, Chi-Wing Fu, Lei Xing, and Pheng-Ann Heng. Difficulty-aware meta-learning for rare disease diagnosis. In *MICCAI*, pages 357–366, 2020.

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

[16] Jialin Liu, Fei Chao, and Chih-Min Lin. Task augmentation by rotating for meta-learning. *arXiv preprint arXiv:2003.00804*, 2020.

[17] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[18] Md Ashraful Alam Milton. Automated skin lesion classification using ensemble of deep neural networks in isic 2018: Skin lesion analysis towards melanoma detection challenge. *arXiv preprint arXiv:1901.10802*, 2019.

[19] Shikhar Murty, Tatsunori Hashimoto, and Christopher D Manning. Dreca: A general task augmentation strategy for few-shot natural language inference. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1113–1125, 2021.

[20] Renkun Ni, Micah Goldblum, Amr Sharaf, Kezhi Kong, and Tom Goldstein. Data augmentation for meta-learning. In *ICML*, pages 8152–8161, 2021.

[21] Jaehoon Oh, Hyungjun Yoo, ChangHwan Kim, and Se-Young Yun. Boil: Towards representation change for few-shot learning. In *ICLR*, 2020.

[22] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *ICLR*, 2019.

[23] Janarthanan Rajendran, Alex Irpan, and Eric Jang. Meta-learning requires meta-augmentation. *arXiv preprint arXiv:2007.05549*, 2020.

[24] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[25] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *ICLR*, 2018.

[26] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-weight-net: Learning an explicit mapping for sample weighting. *NeurIPS*, 32, 2019.

[27] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, pages 4080–4090, 2017.

[28] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *CVPR*, pages 403–412, 2019.

[29] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, pages 1199–1208, 2018.

[30] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? In *ECCV*, pages 266–282. Springer, 2020.

[31] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. In *ICLR*, 2019.

[32] Hung-Yu Tseng, Yi-Wen Chen, Yi-Hsuan Tsai, Sifei Liu, Yen-Yu Lin, and Ming-Hsuan Yang. Regularizing meta-learning via gradient dropout. In *ACCV*, 2020.

[33] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *NeurIPS*, 29:3630–3638, 2016.

[34] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[35] Renzhen Wang, Kaiqin Hu, Yanwen Zhu, Jun Shu, Qian Zhao, and Deyu Meng. Meta feature modulator for long-tailed recognition. *arXiv preprint arXiv:2008.03428*, 2020.

[36] Renzhen Wang, Xixi Jia, Quanziang Wang, and Deyu Meng. Learning to adapt classifier for imbalanced semi-supervised learning. *arXiv preprint arXiv:2207.13856*, 2022.

[37] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *ICML*, pages 3751–3760, 2017.

[38] Yichen Wu, Jun Shu, Qi Xie, Qian Zhao, and Deyu Meng. Learning to purify noisy labels via meta soft label corrector. In *AAAI*, pages 10388–10396, 2021.

[39] Jin Xu, Jean-Francois Ton, Hyunjik Kim, Adam Kosiorek, and Yee Whye Teh. Metafun: Meta-learning with iterative functional updates. In *ICML*, pages 10617–10627, 2020.

[40] 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 *ICML*, pages 11887–11897, 2021.

[41] Huaxiu Yao, Yu Wang, Ying Wei, Peilin Zhao, Mehrdad Mahdavi, Defu Lian, and Chelsea Finn. Meta-learning with an adaptive task scheduler. *NeurIPS*, 34, 2021.

[42] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *ICML*, pages 7045–7054, 2019.

[43] Huaxiu Yao, Linjun Zhang, and Chelsea Finn. Meta-learning with fewer tasks through task interpolation. 2022.

[44] Mingzhang Yin, George Tucker, Mingyuan Zhou, Sergey Levine, and Chelsea Finn. Meta-learning without memorization. In *ICLR*, 2020.

[45] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In *CVPR*, pages 2790–2799, 2018.

[46] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. Pcn: Point completion network. In *2018 International Conference on 3D Vision (3DV)*, pages 728–737, 2018.

[47] Ruixiang Zhang, Tong Che, Zoubin Ghahramani, Yoshua Bengio, and Yangqiu Song. Metagan: An adversarial approach to few-shot learning. *NeurIPS*, 2:8, 2018.

[48] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *ICML*, pages 7693–7702, 2019.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] .
   (b) Did you describe the limitations of your work? [Yes] . See Section 8
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] .
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] . See Section 5
   (b) Did you include complete proofs of all theoretical results? [Yes] . See Section 5
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] . We exploit the open-source datasets and will release the code.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)?[Yes] . See Section 6.2 and Appendix C/D.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)?[Yes] . See Section 6.2.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] . See Appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators?[Yes] . See Section 6.2.
   (b) Did you mention the license of the assets?[N/A] .
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] .
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] .
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] .
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] .
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] .

A Pseudo-codes

We present the pseudo-codes for the task upsampling network $g(\cdot)$ (Algorithm 1), and the meta-training algorithm for the regression task (Algorithm 2) and the classification task (Algorithm 3) in 1-step MAML with ATU. For regression task, we randomly sample a batch of tasks as the ground-truth task set $\mathcal{T}_g$ and construct the task patch by down-sampling (FPS sampling). For classification tasks, we construct (K^s+K^q) tasks in one shot to obtain a task patch from a K^s-shot classification task with K^q query samples. We assume the local task distribution to be smooth, and construct the ground-truth tasks by perform mixup for each image in each task with a nearest image in K_M images. We name the set of K_M images by *memory bank* and denote it by $\mathcal{I}_M$. The K_M images are randomly sampled from classes different from those in the input tasks. It worth note that ATU is only applied in the meta-training phase and, therefore, the meta-testing phase remains the same as the original 1-step MAML. It is direct to extend 1-step MAML to multi-step MAML and extend MAML to ANIL, Meta-SGD without modifying ATU.

Algorithm 1 Task Up-sampling Network $g(\cdot)$

Require: up-sampling ratio $r = r_c \times r_d$ for the coarse generator and decoder, respectively

Input: a task patch $\mathcal{T}_p = \{T_i\}_{i=1}^{N_p}$
Extract the set feature for the input task patch $h_s = g_s(\mathcal{T}_p)$
Generate a set of coarse tasks $\mathcal{T}_c = g_c(\mathcal{T}_p)$ with set size $r_c N_p$
Sample a set of perturbations $\mathcal{Z}$ in size r_d
Generate the up-sampled task set $\mathcal{T}_{up} = g_d(\mathcal{T}_c, \mathcal{Z}, h_s)$
Output: a set of up-sampled tasks $\mathcal{T}_{up}$

Algorithm 2 Meta-training of 1-step MAML with ATU for regression tasks

Require: distribution over meta-training tasks $p(\mathcal{T})$; inner-loop and outer-loop learning rates α,β; hyperparameters η_1, η_2, η_3 in Eq. (4) and Eq. (6); batch size of tasks B; task patch size N_p; up-sampling ratio $r = r_c \times r_d$ for the task up-sampling network

Randomly initialize the parameters θ_0 of the meta-model.
while not converge **do**
  Randomly sample a batch of tasks as $\mathcal{T}_g$ with batch size rN_p
  Perform down-sampling (FPS sampling) on $\mathcal{T}_g$ to construct the local task patch $\mathcal{T}_p$
  Generate the augmented task set through our Task Up-sampling Network as $\mathcal{T}_{up} = g(\mathcal{T}_p)$
  Randomly split the up-sampled task set $\mathcal{T}_{up}$ into n batches $\{\mathcal{T}_{batch}\}$, each with B tasks (i.e., $n = |\mathcal{T}_{up}|/B$)
  for each task batch $\mathcal{T}_{batch}$ in $\mathcal{T}_{up}$ **do**
    for each task $T_i \in \mathcal{T}_{batch}$ **do**
      Perform inner-loop update of MAML as $\phi_i = \theta_0 - \alpha \nabla_{\theta_0} \mathcal{L}(f_{\theta_0}, D_i^s)$
    end for
    Calculate $\mathcal{L}(f_{\phi_i}, D_i^q)$ and $\mathcal{L}_{adv}(\theta_0, D_i^s, D_i^q)$
    Update the meta-model parameter θ_0 as $\theta_0 \leftarrow \theta_0 - \beta \frac{1}{B} \sum_{i=1}^{n} \nabla_{\theta_0} \mathcal{L}(f_{\phi_i}, D_i^q)$
  end for
  Calculate the objective function in Eq. (6) and perform backpropagation to update the Task Up-sampling Network
end while

B Network Architecture of the Task Up-sampling Network

In this section, we provide the network architectures for the Task Up-sampling Network for both regression and classification tasks. As shown in Fig. 8, the set encoder $g_s(\cdot)$ of the Task Up-Sampling Network for regression tasks consists of 2 convolution layers followed by a max-pooling layer to extract the permutation-invariant feature for the input task patch. The dimension of the set feature is 1024. The coarse task generator $g_c(\cdot)$ consists of a set encoder to extract the set feature for the input patch, followed by 3 linear layers to generate coarse tasks from the set feature. The set encoder in coarse generator is the same as $g_s(\cdot)$. The output of the last layers is reshaped into $r_c N_p$ coarse tasks. By concatenating the coarse task and a r_d-dimension noise vector, we obtain the input of the

Algorithm 3 Meta-training of 1-step MAML with ATU for classification tasks (N-way K^s-shot)

Require: distribution over meta-training tasks $p(\mathcal{T})$; inner-loop and outer-loop learning rates α, β; hyperparameters η_1, η_2 in Eq. (4); batch size of tasks B; task patch size N_p; Beta distribution $Beta(\delta_1, \delta_2)$; up-sampling ratio r ($r = r_c \times r_d$, $r_c = 1$)

Randomly initialize the parameters θ_0 of the meta model
while not converge **do**
  Randomly sample a batch of tasks $\mathcal{T}_{batch}$ with B tasks.
  for each task $T_i \in \mathcal{T}_{batch}$ **do**
    Reshape T_i as the task patch $\mathcal{T}_p$
    Randomly sample extra K_M images which consists of images not belong to any class in T_i
    Construct the $\mathcal{T}_g = (\hat{C}_0, ..., \hat{C}_N)$: Sample $\lambda \sim Beta(\delta_1, \delta_2)$. For the image in each class C_j in T_i, generate a new image as $\hat{C}_j = \lambda \times C_j + (1 - \lambda) \times X_j$, where X_j is the nearest image (measured by euclidean distance) to the image in the class C_j.
    Generate up-sampling task set through Task Up-sampling Network as $\mathcal{T}_{up} = g(\mathcal{T}_p)$
    Randomly sample one task $\hat{T}_i$ from $\mathcal{T}_{up}$
    Perform inner-loop update of MAML as $\phi_i = \theta_0 - \alpha \nabla_{\theta_0} \mathcal{L}(f_{\theta_0}, \hat{D}_i^s)$
  end for
  Calculate $\mathcal{L}(f_{\phi_i}, D_i^q)$ and $\mathcal{L}_{adv}(\theta_0, D_i^s, D_i^q)$
  Update the meta model parameter θ_0 as $\theta_0 \leftarrow \theta_0 - \beta \frac{1}{B} \sum_{i=1}^{n} \nabla_{\theta_0} \mathcal{L}(f_{\phi_i}, \hat{D}_i^q)$
  Calculate the objective function in Eq. (5) and perform backpropagation to update the Task Up-sampling Network
end while

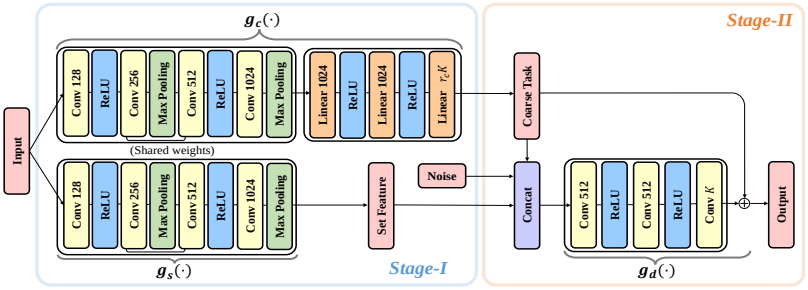


Figure 8: The up-sampling network of the regression task.

decoder $g_d(\cdot)$. The decoder consists of 3 convolution layers. We then use the output of the last layer as residual added to the coarse tasks to obtain the up-sampling tasks.

The network structure of Task Up-sampling Network for classification tasks is presented in Fig. 9. We directly use the input task patch as the coarse tasks and, therefore, the coarse task generator $g_c(\cdot)$ is an identity function. The set encoder $g_s(\cdot)$ consists of 2 convolution layers, each followed by a Batch Norm layer. We use the K_M images in the memory bank as the perturbation, concatenating with a $(N_p \times K_M)$-dimension noise vector to obtain the input to the decoder. The decoder consists of a attention module and a mapping module. The attention module is constructed by 3 convolution layers, followed by 3 linear layers. The attention block generates the attention scores. The mapping module, which consists of 3 convolution layers with xxx filters, maps the K_M perturbation to K_M residual features. We perform weighted sum of the K_M residual features with the attention scores to generate r final residual features and add them to the coarse tasks to obtain the up-sampling 1-shot tasks. We then construct r augmented tasks by stacking $K^s + K^q$ 1-shot tasks.

C Setups and Additional Experiment Results for Regression Tasks

C.1 Setups of Hyperparameter

The hyperparameters of the ATU are listed in Table 8.

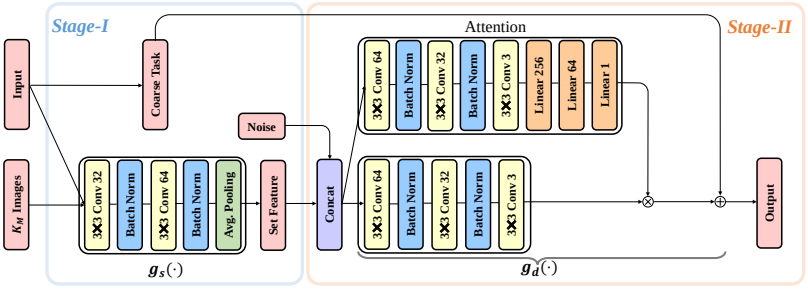


Figure 9: The up-sampling network of the classification task.

C.2 Effect of Augmentation Ratio

In the regression task, we assume the combination of the augmented and original tasks will better approximate the real task distribution and, therefore, update the meta-model not only with the augmented tasks generated by the Task Up-sampled Network, but also with the original meta-training tasks. We define the augmentation ratio as the proportion of augmented tasks among all the tasks. Note that the experiment results shown in Table 2 are obtained by setting the augmented ratio as 0.2. We present the results with other ratios in Table 7. It can be observed that the performance with positive ratio is better than the performance with ratio$= 0$, which means the meta-model is trained without augmented tasks. These results indicates that the augmented tasks are more informative than the original tasks in training a better meta-model.

Table 7: Ablation study on the augmentation ratio of the sine regression task.

Augmentation ratio	TU performance (10-shot)
0	0.93 ± 0.18
0.2	0.84 ± 0.16
0.4	0.89 ± 0.17
0.6	0.91 ± 0.18

Table 8: Hyperparameters of the sine regression task in Table 2.

Hyperparameters	ATU
maximum training iterations	3750
up-sampling ratio r (r_c, r_d)	8 (2, 4)
loss weights (η_1, η_2, η_3)	$(8e^{-3}, 4e^{-3}, 3e^{-1})$
size of $\mathcal{T}_g$	64

D Setups and Additional Experiment Results for Classification Tasks

D.1 Introduction and Hyperparameters of the Four Datasets

We provide the detailed information of the datasets and hyperparameters of the classification tasks for obtaining the results in Tabel 4 in this section. We construct the 4 datasets following the settings in MLTI [43].

miniImagenet-S. Compared with miniImagenet, miniImagenet-S has fewer meta-training classes so as to limit the task number. The specific meta-training classes of miniImagenet-S include:

n03017168, n07697537, n02108915, n02113712, n02120079, n04509417,n02089867, n03888605, n04258138, n03347037, n02606052, n06794110

We use four convolutional blocks and a classifier as the base learner [43, 9], and each convolutional block contains a convolutional layer, a batch normalization layer and a ReLU activation layer. In order to analyze the effect of the number of meta-training tasks, we add more classes for meta-training according to the following sequence:

n03476684, n02966193, n13133613, n03337140, n03220513, n03908618,n01532829, n04067472, n02074367, n03400231, n02108089, n01910747, n02747177, n02795169, n04389033, n04435653, n02111277, n02108551,n04443257, n02101006, n02823428, n03047690, n04275548, n04604644, n02091831, n01843383, n02165456, n03676483, n04243546, n03527444, n01770081, n02687172, n09246464, n03998194, n02105505, n01749939, n04251144, n07584110, n07747607, n04612504, n01558993, n03062245, n04296562, n04596742, n03838899, n02457408, n13054560, n03924679, n03854065, n01704323, n04515003, n03207743

ISIC. ISIC skin dataset [18] was provided by ISIC2018 Challenge, in which 7 disease classes and 10015 dermoscopic images are included. Following [43, 14], *Nevus, Malanoma, Benign Keratoses, Basal Cell Carcinoma*, the four categories with the largest number of images, are as meta-training classes; the rest *Dermatofibroma, Pigmented Bowen's, Benign Keratoses* are as meta-testing classes. We re-scale the size of each medical image to $84 \times 84 \times 3$ and adopt the same 4-layer convolutional as the base model like miniImagenet-S.

DermNet-S. Dermnet-S are part of the public Dermnet Skin Disease Altas, in which 625 different fine-grained categories are included. Dermnet-S chooses the top-30 classes for meta-training. The concrete meta-training classes and meta-testing classes are:

- **Meta-training classes:** *Seborrheic Keratoses Ruff, Herpes Zoster, Atopic Dermatitis Adult Phase, Psoriasis Chronic Plaque, Eczema Hand, Seborrheic Dermatitis, Keratoacanthoma, Lichen Planus, Epidermal Cyst, Eczema Nummular, Tinea (Ringworm) Versicolor, Tinea (Ringworm) Body, Lichen Simplex Chronicus, Scabies, Psoriasis Palms Soles, Malignant Melanoma, Candidiasis large Skin Folds, Pityriasis Rosea, Granuloma Annulare, Erythema Multiforme, Seborrheic Keratosis Irritated, Stasis Dermatitis and Ulcers, Distal Subungual Onychomycosis, Allergic Contact Dermatitis, Psoriasis, Molluscum Contagiosum, Acne Cystic, Perioral Dermatitis, Vasculitis, Eczema Fingertips.*

- **Meta-testing classes:** *Warts, Ichthyosis Sex Linked, Atypical Nevi, Venous Lake, Erythema Nodosum, Granulation Tissue, Basal Cell Carcinoma Face, Acne Closed Comedo, Scleroderma, Crest Syndrome, Ichthyosis Other Forms, Psoriasis Inversus, Kaposi Sarcoma, Trauma, Polymorphous Light Eruption, Dermagraphism, Lichen Sclerosis Vulva, Pseudomonas, Cutaneous Larva Migrans, Psoriasis Nails, Corns, Lichen Sclerosus Penis, Staphylococcal Folliculitis, Chilblains Perniosis, Psoriasis Erythrodermic, Squamous Cell Carcinoma Ear, Basal Cell Carcinoma Ear, Ichthyosis Dominant, Erythema Infectiosum, Actinic Keratosis Hand, Basal Cell Carcinoma Lid, Amyloidosis, Spiders, Erosio Interdigitalis Blastomycetica, Scarlet Fever, Pompholyx, Melasma, Eczema Trunk Generalized, Metastasis, Warts Cryotherapy, Nevus Spilus, Basal Cell Carcinoma Lip, Enterovirus, Pseudomonas Cellulitis, Benign Familial Chronic Pemphigus, Pressure Urticaria, Halo Nevus, Pityriasis Alba, Pemphigus Foliaceous, Cherry Angioma, Chapped Fissured Feet, Herpes Buttocks, Ridging Beading.*

Tabular Murris. The Tabular Murris is a gene dataset (i.e., 2866-dim features) including 105,960 cells of 124 cell types extracted from 23 organs. Following [43, 5], the concrete training/validation/testing split is:

- **Meta-training classes:** *BAT, MAT,Limb Muscle, Trachea, Heart, Spleen, GAT, SCAT, Mammary Gland, Liver, Kidney, Bladder, Brain Myeloid, Brain Non-Myeloid, Diaphragm.*

- **Meta-validation classes:** *Skin, Lung, Thymus, Aorta*

- **Meta-testing organs:** *Large Intestine, Marrow, Pancreas, Tongue*

Unlike the base model for the other 3 datasets, we use two fully connected blocks and a linear layer as the backbone network, where each fully connected block includes a linear layer, a batch normalization layer, a ReLu activation layer, and a dropout layer. The dropout ratio and the feature channels of the linear layer are set 0.2, 64, which is the same as the settings of [43, 5].

For a fair comparison with MLTI, we adopt the same MetaMix strategy as MLTI to augment the query set for the four datasets. We set the augmentation ratio as 1 in classification tasks and do not use the original sampled tasks in meta-training because we empirically find that ATU obtains better performance with higher augmentation ratio. The other settings are the same with MLTI. More details about the hyperparameters are listed in Table 9.

Table 9: Hyperparameters of Tabel 4

Hyperparameters(ATU)	miniImagenet-S	ISIC	Dermnet-S	Tabular Murris
inner-loop learning rate	0.01	0.01	0.01	0.01
outer-loop learning rate	0.001	0.001	0.001	0.001
$Beta(\delta_1, \delta_2)$	(3,5)	(2,2)	(2,2)	(2,2)
Number of steps in inner loop	5	5	5	5
batch size	4	4	4	4
query size in meta-training tasks	15	15	15	15
maximum training iterations	50,000	50,000	50,000	50,000
adversarial loss weights η ($\eta_1 = \eta_2$)	3	3	0.5	0.5
up-sampling ratio r	2	2	2	2

Table 10: Ablation study on the memory bank size K_M in the classification task.

K_M	TU (mini-S 1-shot)
3	42.16 ± 0.73%
5	42.20 ± 0.76%
7	42.28 ± 0.72%

Table 11: Sensitivity analysis of the adversarial loss weights on the classification task.

Adversarial weights	ATU (mini-S 1-shot)
$\eta_1 = \eta_2 = 1$	42.38 ±0.82%
$\eta_1 = \eta_2 = 3$	42.60 ±0.84%
$\eta_1 = \eta_2 = 5$	41.67 ±0.79%

D.2 Ablation Study

Effect of K_M. As shown in Table 10, the classification performance increases as the memory bank size K_M increases. But the performance gain is not very significant for a large K_M. Considering the training efficiency, we set $K_M = 3$ in all classification experiments.

Table 12: The averaged accuracy with 95% confidence intervals of various interpolation task augmentation methods and our task up-sampling method on miniImagenet-s (5-shot).

Task generation method	miniImagenet-S (5-shot)
Naive Baseline[1]	$53.49 \pm 0.74\%$
Naive Baseline[2]	$50.25 \pm 0.71\%$
Naive Baseline[3]	$53.91 \pm 0.78\%$
TU	$56.33 \pm 0.79\%$

Effect of η_1,η_2. We choose different (η_1,η_2) to explore the sensitivity of model performance to adversarial loss weights. It can be observed from the results in Table 11 that the adversarial loss weights have a large influence on the performance of the model and it achieves the best performance when setting $\eta_1 = \eta_2 = 3$.

Effect of Augmented Task Generation Strategies. Due to the high complexity of the classification tasks' distribution, we assume its latent task distribution is smooth and construct the ground-truth task manifold $\mathcal{T}_g$ via mixing all image in each class of T_i with its corresponding nearest image in the memory bank (the sampled K_M images) (see Algorithm 3). Under this assumption, one naive method is to generate augmented tasks in the same way as the generation of ground-truth tasks where we can directly mix the images in the tasks with the K_M images in the memory bank. To verify the effectiveness and necessity of training a Task Up-sampling Network, we compare TU with 3 naive methods in Table 12: (1) Naive Baseline[1]:for each image of task T_i, we randomly choose one image in the memory bank to mix; (2) Naive Baseline[2]: for all images in a class of task T_i, we randomly choose one image in the memory bank to mix; (3) Naive Baseline[3]: for all images in a class of task T_i, we choose the nearest image in the memory bank to mix. The Naive Baseline[3] is the method that we used to construct $\mathcal{T}_g$. The results in Table 12 show that TU outperforms the other 3 baselines by a large margin. TU outperforms Naive Baseline[3] because the tasks generated by TU match the local task distribution better than those generated by just mixing with the images in the memory bank.

Moreover, the tasks generated by TU is more diverse and informative. Taking this into consideration, we set the augmentation ratio to be 1 and do not use the original tasks in the meta-training.

D.3 Visualization of the generated Classification tasks of $\mathcal{T}_p$.

We visualize part of the images in a generated classification task in Fig. 10. The three images in the top row are the selected extra K_M images and $\hat{T}_1, \hat{T}_2, \hat{T}_3$ are three 5-way 1-shot tasks generated by ATU.

Figure 10: Visualization of part up-sampled classification tasks (i.e., $\mathcal{T}_{up}$) generated by ATU.

Table 13: Complete results of Table 6 with 95% confidence interval under the cross-domain setting.

Model	miniImagenet-S→ DermNet-S 1-shot	miniImagenet-S→ DermNet-S 5-shot	DermNet-S→ miniImagenet-S 1-shot	DermNet-S→ miniImagenet-S 5-shot
MAML [9]	$34.46 \pm 0.63\%$	$50.36 \pm 0.64\%$	$28.78 \pm 0.55\%$	$41.29 \pm 0.64\%$
MAML+ATU	**$36.86 \pm 0.64\%$**	**$51.98 \pm 0.62\%$**	**$30.68 \pm 0.68\%$**	**$46.72 \pm 0.73\%$**
MetaSGD [15]	$31.07 \pm 0.57\%$	$49.07 \pm 0.59\%$	$28.17 \pm 0.53\%$	$41.83 \pm 0.67\%$
MetaSGD+ATU	**$37.75 \pm 0.65\%$**	**$54.60 \pm 0.58\%$**	**$30.78 \pm 0.58\%$**	**$44.01 \pm 0.68\%$**

D.4 Complete Results with Confidence Interval

We list the complete results with 95% confidence interval in Table 14,15,13, which are corresponding to the Tabel 4, 5, 6 in Section 6.2.

E Proof of Property 1.

Property 1 (Task-awareness Maximization). Consider $N_u = 2$, $g(\theta_1, \theta_2) = (1-\lambda)\theta_1 + \lambda\theta_2$, $f_{\theta_1}(\cdot) = \mathbf{W}_1$, and $f_{\theta_2}(\cdot) = \mathbf{W}_2$. The proposed ATU algorithm that pursues an up-sampled task $T_u = \{\mathbf{X}_u, \mathbf{Y}_u\}$ via minimizing the EMD loss between T_1 and T_2 maximizes the task-awareness, i.e., minimizing the distance between $\mathbf{Y}_u$ and $f_{\theta_u}(\mathbf{X}_u)$.

Proof. According to the definition of EMD (Eq. (2)), it solves: $\phi^* = \arg\min_{\phi \in \mathbf{\Phi}} \sum_j \|\mathbf{x}_{1,j} - \mathbf{x}_{2,\phi(j)}\|_2$, where $\mathbf{\Phi} = \{\{1, \cdots, n\} \mapsto \{1, \cdots, n\}\}$ denotes the set containing all possible bijective assignments, each of which gives one-to-one correspondence between T_1 and T_2. Based on the optimal assignments ϕ^*, the EMD is known to be defined as $d_{EMD} = \frac{1}{n}\sum_j \|\mathbf{x}_{1,j} - \mathbf{x}_{2,\phi^*(j)}\|_2$. In light of the difficulty in mathematically formulating a possible up-sampled task $\tilde{T}_u$ that lies in the local manifold of $\{T_1, T_2\}$, we reasonably assume a simplified way of characterizing an

Table 14: Complete classification results of Table 4 with 95% confidence interval.

Setting	Model	miniImagenet-S	ISIC	DermNet-S	Tabular Murris
1-shot	MAML [9]	$38.27 \pm 0.74\%$	$57.59 \pm 0.79\%$	$43.47 \pm 0.83\%$	$79.08 \pm 0.91\%$
	Meta-Reg [44]	$38.35 \pm 0.76\%$	$58.57 \pm 0.94\%$	$45.01 \pm 0.83\%$	$79.18 \pm 0.87\%$
	TAML [11]	$38.70 \pm 0.77\%$	$58.39 \pm 1.00\%$	$45.73 \pm 0.84\%$	$79.82 \pm 0.87\%$
	Meta-Dropout [13]	$38.32 \pm 0.75\%$	$58.40 \pm 1.02\%$	$44.30 \pm 0.84\%$	$78.18 \pm 0.93\%$
	MetaMix [40]	$39.43 \pm 0.77\%$	$60.34 \pm 1.03\%$	$46.81 \pm 0.81\%$	$81.06 \pm 0.86\%$
	Meta-Maxup [20]	$39.28 \pm 0.77\%$	$58.68 \pm 0.86\%$	$46.10 \pm 0.82\%$	$79.56 \pm 0.89\%$
	MLTI [43]	$41.58 \pm 0.72\%$	$61.79 \pm 1.00\%$	$48.03 \pm 0.79\%$	$81.73 \pm 0.89\%$
	TU	$42.16 \pm 0.76\%$	$62.03 \pm 0.95\%$	$48.07 \pm 0.83\%$	$81.88 \pm 0.90\%$
	ATU	$\mathbf{42.60 \pm 0.77\%}$	$\mathbf{62.84 \pm 0.98\%}$	$\mathbf{48.33 \pm 0.81\%}$	$\mathbf{82.04 \pm 0.94\%}$
5-shot	MAML [9]	$52.14 \pm 0.65\%$	$65.24 \pm 0.77\%$	$60.56 \pm 0.74\%$	$88.55 \pm 0.60\%$
	Meta-Reg [44]	$51.74 \pm 0.68\%$	$68.45 \pm 0.81\%$	$60.92 \pm 0.69\%$	$89.08 \pm 0.61\%$
	TAML [11]	$52.75 \pm 0.70\%$	$66.09 \pm 0.71\%$	$61.14 \pm 0.72\%$	$89.11 \pm 0.59\%$
	Meta-Dropout [13]	$52.53 \pm 0.69\%$	$67.32 \pm 0.92\%$	$60.86 \pm 0.73\%$	$89.25 \pm 0.59\%$
	MetaMix [40]	$54.14 \pm 0.73\%$	$69.47 \pm 0.60\%$	$63.52 \pm 0.73\%$	$89.75 \pm 0.58\%$
	Meta-Maxup [20]	$53.02 \pm 0.72\%$	$69.16 \pm 0.61\%$	$62.64 \pm 0.72\%$	$88.88 \pm 0.57\%$
	MLTI [43]	$55.22 \pm 0.76\%$	$70.69 \pm 0.68\%$	$64.55 \pm 0.74\%$	$91.08 \pm 0.54\%$
	TU	$56.33 \pm 0.69\%$	$73.97 \pm 0.70\%$	$64.81 \pm 0.72\%$	$91.15 \pm 0.60\%$
	ATU	$\mathbf{56.78 \pm 0.73\%}$	$\mathbf{74.50 \pm 0.90\%}$	$\mathbf{65.16 \pm 0.75\%}$	$\mathbf{91.42 \pm 0.61\%}$

Table 15: Complete results of Table 5 with 95% confidence interval under different backbones.

Method	miniImagenet-S	ISIC	DermNet-S	Tabular Murris
MetaSGD [15]	$37.88 \pm 0.74\%$	$58.79 \pm 0.82\%$	$42.07 \pm 0.83\%$	$81.55 \pm 0.91\%$
MetaSGD+MLTI	$39.58 \pm 0.76\%$	$61.57 \pm 1.10\%$	$45.49 \pm 0.83\%$	$83.31 \pm 0.87\%$
MetaSGD+ATU	$\mathbf{40.52 \pm 0.78\%}$	$\mathbf{62.84 \pm 1.01\%}$	$\mathbf{46.78 \pm 0.84\%}$	$\mathbf{83.84 \pm 0.90\%}$
ANIL [22]	$38.02 \pm 0.75\%$	$59.48 \pm 1.00\%$	$44.58 \pm 0.85\%$	$75.67 \pm 0.99\%$
ANIL+MLTI	$39.15 \pm 0.73\%$	$61.78 \pm 1.24\%$	$46.79 \pm 0.77\%$	$77.11 \pm 1.00\%$
ANIL+ATU	$\mathbf{39.27 \pm 0.76\%}$	$\mathbf{62.12 \pm 0.98\%}$	$\mathbf{47.03 \pm 0.85\%}$	$\mathbf{77.23 \pm 0.99\%}$

up-sampled task $\tilde{T}_u$ to be $\tilde{\mathbf{y}}_{u,j} = \boldsymbol{\alpha}_{1,j}^T \mathbf{Y}_1 + \boldsymbol{\alpha}_{2,j}^T \mathbf{Y}_2$, $\tilde{\mathbf{x}}_{u,j} = \boldsymbol{\alpha}_{1,j}^T \mathbf{X}_1 + \boldsymbol{\alpha}_{2,j}^T \mathbf{X}_2$, $\forall j$, where each sample is a convex combination of samples from both T_1 from T_2. The combination coefficients $\boldsymbol{\alpha}_{1,j}^T, \boldsymbol{\alpha}_{2,j}^T \in \mathbb{R}^{(K^s+K^q)\times 1}$, $\sum_k^{K^s+K^q} \alpha_{1,jk} = 1$, and $\sum_k^{K^s+K^q} \alpha_{2,jk} = 1$. Different combination coefficients lead to a set of up-sampled task candidates $\{\tilde{T}_u\}$. We evaluate the task-awareness property of each candidate $\tilde{T}_u$, i.e., the distance between $\tilde{\mathbf{Y}}_u$ and $f_{\theta_u}(\tilde{\mathbf{X}}_u)$, to be

$$\|\tilde{\mathbf{Y}}_u - f_{\theta_u}(\tilde{\mathbf{X}}_u)\|_2 = \textstyle\sum_j \|\tilde{\mathbf{y}}_{u,j} - f_{\theta_u}(\tilde{\mathbf{x}}_{u,j})\|_2$$

$$= \textstyle\sum_j \|\boldsymbol{\alpha}_{1,j}^T \mathbf{Y}_1 + \boldsymbol{\alpha}_{2,j}^T \mathbf{Y}_2 - [(1-\lambda)\mathbf{W}_1 + \lambda \mathbf{W}_2][\boldsymbol{\alpha}_{1,j}^T \mathbf{X}_1 + \boldsymbol{\alpha}_{2,j}^T \mathbf{X}_2]\|_2$$

$$= \textstyle\sum_j \|\boldsymbol{\alpha}_{1,j}^T \mathbf{X}_1 \mathbf{W}_1 + \boldsymbol{\alpha}_{2,j}^T \mathbf{X}_2 \mathbf{W}_2 - [(1-\lambda)\mathbf{W}_1 + \lambda \mathbf{W}_2][\boldsymbol{\alpha}_{1,j}^T \mathbf{X}_1 + \boldsymbol{\alpha}_{2,j}^T \mathbf{X}_2]\|_2$$

$$= \textstyle\sum_j \|(\mathbf{W}_1 - \mathbf{W}_2)[\lambda \alpha_{1,j}^T \mathbf{X}_1 - (1-\lambda)\alpha_{2,j}^T \mathbf{X}_2]\|_2 = \text{LHS}$$

Note that

$$\begin{aligned} LHS &= \sum_j \|(\mathbf{W}_1 - \mathbf{W}_2)[\lambda \tilde{\mathbf{x}}_{u,j} - \alpha_{2,j}^T \mathbf{X}_2]\|_2 \\ &= \sum_j \|\mathbf{W}_1 - \mathbf{W}_2[\lambda \tilde{\mathbf{x}}_{u,j} - \lambda \mathbf{x}_{2,\phi_2(j)} + \lambda \mathbf{x}_{2,\phi_2(j)} - \alpha_{2,j}^T \mathbf{X}_2]\|_2 \\ &\leq \sum_j \|\mathbf{W}_1 - \mathbf{W}_2\|_2 (\lambda \|\tilde{\mathbf{x}}_{u,j} - \mathbf{x}_{2,\phi_2(j)}\|_2 + \|\mathbf{X}_2\|_2), \end{aligned}$$

where the last inequality follows the triangle inequality and the fact that $0 \leq \lambda \leq 1$. Similarly, we have

$$\begin{aligned} LHS &= \sum_j \|(\mathbf{W}_1 - \mathbf{W}_2)[\alpha_{1,j}^T \mathbf{X}_1 - (1-\lambda)\tilde{\mathbf{x}}_{u,j}]\|_2 \\ &= \sum_j \|\mathbf{W}_1 - \mathbf{W}_2[\alpha_{1,j}^T \mathbf{X}_1 - (1-\lambda)\mathbf{x}_{1,\phi_1(j)} + (1-\lambda)\mathbf{x}_{1,\phi_1(j)} - (1-\lambda)\tilde{\mathbf{x}}_{u,j}]\|_2 \\ &\leq \sum_j \|\mathbf{W}_1 - \mathbf{W}_2\|_2((1-\lambda)\|\mathbf{x}_{1,\phi_1(j)} - \tilde{\mathbf{x}}_{u,j}\|_2 + \|\mathbf{X}_1\|_2). \end{aligned}$$

In practice, it is easy to normalize all the tasks in the feature space, which leads to $\|\mathbf{X}_1\|_2 = \|\mathbf{X}_2\|_2$ Therefore, by minimizing the EMD loss

$$d_{EMD} = \min\{\min_{\phi_2} \sum_j \|\tilde{\mathbf{x}}_{u,j} - \mathbf{x}_{2,\phi_2(j)}\|_2, \min_{\phi_1} \sum_j \|\tilde{\mathbf{x}}_{u,j} - \mathbf{x}_{1,\phi_1(j)}\|_2\},$$

the proposed task up-sampling network identifies from the candidate set $\{\tilde{T}_u\}$ the task T_u that has the minimal distance between $\mathbf{Y}_u$ and $f_{\theta_u}(\mathbf{X}_u)$; in other words, the task-awareness is maximized.

□

Previous task augmentation approaches directly mix up two tasks without minimizing the EMD loss, i.e., $\mathbf{y}_{u,j} = (1-\lambda)\mathbf{y}_{1,j} + \lambda\mathbf{y}_{2,j}, \mathbf{x}_{u,j} = (1-\lambda)\mathbf{x}_{1,j} + \lambda\mathbf{x}_{2,j}$. In this case, the task-awareness is unwarranted as we have illustrated in Section 1, provided that $\|\mathbf{Y}_u - f_{\theta_u}(\mathbf{X}_u)\|_2 = \sum_j \|(1-\lambda)\mathbf{y}_{1,j} + \lambda\mathbf{y}_{2,j} - [(1-\lambda)\mathbf{W}_1 + \lambda\mathbf{W}_2][(1-\lambda)\mathbf{x}_{1,j} + \lambda\mathbf{x}_{2,j}]\|_2 = \sum_j \lambda^2(1-\lambda)^2\|(\mathbf{W_1} - \mathbf{W}_2)(\mathbf{x}_{1,j} - \mathbf{x}_{2,j})\|_2$.

Table 16: 1-shot meta-training on MiniImangenet-S and meta-testing on various meta-datasets.

mini-S →	Derm-S	CUB	Aircraft	Fungi	Texture
MAML	34.46%	39.81%	27.92%	30.06%	26.29%
MAML+ATU	36.86%(↑**2.40%**)	40.67%(↑**0.86%**)	30.11%(↑**2.19%**)	32.81%(↑**2.75%**)	27.28%(↑**2.40%**)
MetaSGD	31.07%	39.94%	28.71%	30.96%	25.75%
MetaSGD+ATU	37.75%(↑**6.68%**)	42.52%(↑**2.58%**)	30.22%(↑**1.51%**)	32.52%(↑**1.56%**)	28.61%(↑**2.86%**)

Table 17: 5-shot meta-training on MiniImangenet-S and meta-testing on various meta-datasets.

mini-S →	Derm-S	CUB	Aircraft	Fungi	Texture
MAML	50.36%	57.02%	36.63%	40.96%	36.61%
MAML+ATU	51.98%(↑**1.62%**)	61.04%(↑**4.02%**)	40.19%(↑**3.56%**)	43.59%(↑**2.63%**)	37.60%(↑**0.99%**)
MetaSGD	49.07%	55.87%	37.94%	39.76%	33.84%
MetaSGD+ATU	54.60%(↑**5.53%**)	60.37%(↑**4.50%**)	39.11%(↑**1.17%**)	42.77%(↑**3.01%**)	36.59%(↑**2.75%**)

Table 18: 1-shot meta-training on DermNet-S and meta-testing on various meta-datasets.

Derm-S →	mini-S	CUB	Aircraft	Fungi	Texture
MAML	28.78%	35.10%	28.03%	26.71%	26.17%
MAML+ATU	30.68%(↑**1.90%**)	36.37%(↑**1.27%**)	29.31%(↑**1.28%**)	27.16%(↑**0.45%**)	27.11%(↑**0.94%**)
MetaSGD	28.17%	32.69%	26.07%	25.19%	25.02%
MetaSGD+ATU	30.78%(↑**2.61%**)	35.86%(↑**3.17%**)	31.56%(↑**5.49%**)	28.07%(↑**2.88%**)	28.04%(↑**3.02%**)

Table 19: 5-shot meta-training on DermNet-S and meta-testing on various meta-datasets.

Derm-S →	mini-S	CUB	Aircraft	Fungi	Texture
MAML	41.29%	53.44%	38.30%	35.04%	37.01%
MAML+ATU	46.27%(↑**4.98%**)	54.99%(↑**1.55%**)	41.22%(↑**2.92%**)	35.45%(↑**0.41%**)	39.04%(↑**2.03%**)
MetaSGD	41.83%	52.32%	37.27%	36.74%	35.27%
MetaSGD+ATU	44.01%(↑**2.18%**)	58.52%(↑**6.20%**)	43.28%(↑**6.01%**)	38.28%(↑**1.54%**)	38.28%(↑**3.01%**)

Table 20: Computational cost analysis.

	Pre-train	Ordinary training	Total
MAML	–	35,936 s	35,936 s
Ours	13,512 s	45,926 s	59,438 s

F More experiments under the cross-domain setting.

For a more detailed analysis of how the model behaves in a cross-domain setting, we conduct more experiments meta-tested on meta-datasets, as shown in Table 16, 17, 18, 19.

G Computational cost for the method

The tasks are generated on the fly during meta-training. Our method includes two stages: (1) pre-training the augmentation network and (2) meta-training of the meta-learner and the augmentation network jointly. For fair comparison with MAML which trains for 50k iterations, we pre-train the augmentation network for 10k iterations, and meta-train for 40k iterations.

In summary, our method's (TU) computation cost is **1.65x** of the vanilla MAML. The breakdown of the computation cost is listed in Table 20.

H Experiments on the limited meta-datasets.

In order to further valid the effectiveness of the proposed method, we have conducted experiments to evaluate the three suggested baselines (including Baseline++ [7], RFS [30], and ProtoNet [27]) on the setting of limited tasks, and show the comparison results in Table 21. We construct the dataset CUB-S, Fungi-S, Aircraft-S, and Texture-S similarly to miniImagenet-S. The details of their construction are listed as follows.

CUB-S. CUB [34] is a bird image dataset including 11,788 photos of 200 bird species. In this paper, we randomly select 48 species with 60 images in each species. We devide them into meta-training/meta-validation/meta-testing sets as 12/16/20 species.

- **Meta-training classes:** *Savannah Sparrow, Dark eyed Junco, Black footed Albatross, Henslow Sparrow, Cape Glossy Starling, Black throated Sparrow, Northern Waterthrush, Hooded Warbler, Baltimore Oriole, Scarlet Tanager, Cerulean Warbler, Downy Woodpecker.*
- **Meta-validation classes:** *Mockingbird, Vermilion Flycatcher, Cape May Warbler, Prothonotary Warbler, White crowned Sparrow, Ovenbird, Pomarine Jaeger, Indigo Bunting, Blue winged Warbler, Chipping Sparrow, Horned Grebe, Fox Sparrow, Green Violetear, Nashville Warbler, Least Tern, Marsh Wren.*
- **Meta-testing classes:** *Rose breasted Grosbeak, Nighthawk, Long tailed Jaeger, Bronzed Cowbird, California Gull, Ivory Gull, Northern Fulmar, Brown Pelican, Ring billed Gull, Great Grey Shrike, White breasted Nuthatch, Mourning Warbler, Sage Thrasher, Horned Puffin, Pied Kingfisher, Shiny Cowbird, Scott Oriole, Red eyed Vireo, Song Sparrow, Winter Wren.*

Table 21: Complete classification results of Table 4 with 95% confidence interval.

Setting	Model	CUB-S	Fungi-S	Aircraft-S	Texture-S
1-shot	Protonet [27]	$35.35 \pm 0.70\%$	$26.01 \pm 0.51\%$	$30.26 \pm 0.56\%$	$26.52 \pm 0.53\%$
	Protonet+MLTI	$36.17 \pm 0.72\%$	$28.80 \pm 0.57\%$	$33.26 \pm 0.68\%$	$28.28 \pm 0.56\%$
	Protonet+TU	$38.35 \pm 0.71\%$	$30.91 \pm 0.59\%$	$34.87 \pm 0.70\%$	$29.02 \pm 0.57\%$
	Baseline++ [7]	$43.98 \pm 0.84\%$	$32.97 \pm 0.74\%$	$36.28 \pm 0.79\%$	$31.36 \pm 0.58\%$
	RFS [30]	$43.96 \pm 0.82\%$	$33.05 \pm 0.70\%$	$33.68 \pm 0.80\%$	$31.47 \pm 0.59\%$
	MAML	$41.58 \pm 0.90\%$	$29.63 \pm 0.64\%$	$34.54 \pm 0.72\%$	$33.79 \pm 0.69\%$
	MAML+MLTI	$44.77 \pm 0.88\%$	$31.34 \pm 0.65\%$	$37.76 \pm 0.73\%$	$34.51 \pm 0.69\%$
	MAML+TU	$47.23 \pm 0.96\%$	$33.21 \pm 0.68\%$	$39.79 \pm 0.80\%$	$34.82 \pm 0.67\%$
	MAML+ATU	$\mathbf{48.33 \pm 0.96\%}$	$\mathbf{33.66 \pm 0.70\%}$	$\mathbf{41.31 \pm 0.82\%}$	$\mathbf{35.26 \pm 0.73\%}$
5-shot	Protonet [27]	$55.30 \pm 0.75\%$	$34.06 \pm 0.63\%$	$50.49 \pm 0.66\%$	$33.37 \pm 0.58\%$
	Protonet+MLTI	$56.69 \pm 0.77\%$	$34.44 \pm 0.56\%$	$51.77 \pm 0.64\%$	$35.34 \pm 0.56\%$
	Protonet+TU	$58.32 \pm 0.76\%$	$35.56 \pm 0.62\%$	$52.24 \pm 0.66\%$	$37.20 \pm 0.59\%$
	Baseline++ [7]	$54.41 \pm 0.75\%$	$44.49 \pm 0.75\%$	$45.84 \pm 0.77\%$	$40.31 \pm 0.61\%$
	RFS [30]	$55.40 \pm 0.74\%$	$46.55 \pm 0.77\%$	$47.05 \pm 0.78\%$	$40.91 \pm 0.60\%$
	MAML	$57.97 \pm 0.85\%$	$37.10 \pm 0.65\%$	$43.62 \pm 0.69\%$	$39.47 \pm 0.63\%$
	MAML+MLTI	$63.89 \pm 0.81\%$	$45.64 \pm 0.74\%$	$55.05 \pm 0.72\%$	$40.62 \pm 0.67\%$
	MAML+TU	$64.41 \pm 0.82\%$	$46.99 \pm 0.83\%$	$55.85 \pm 0.70\%$	$41.38 \pm 0.65\%$
	MAML+ATU	$\mathbf{65.56 \pm 0.80\%}$	$\mathbf{47.91 \pm 0.80\%}$	$\mathbf{56.90 \pm 0.71\%}$	$\mathbf{42.52 \pm 0.62\%}$

Fungi-S. Fungi [2] dataset contains 1,500 wild mushroom species with over 100,000 fungi images. We select the sepcies with greater than 150 images and then randomly choose 100 species, where each species contains 150 images. We split them into meta-training/meta-validation/meta-testing sets with 12/16/20 species.

- **Meta-training classes:** *Suillus granulatus, Phaeolus schweinitzii, Cystoderma amianthinum, Pycnoporellus fulgens, Psathyrella candolleana, Meripilus giganteus, Phellinus pomaceus, Laccaria laccata, Laccaria proxima, Amanita excelsa, Ganoderma pfeifferi, Clitopilus prunulus.*
- **Meta-validation classes:** *Agaricus impudicus, Daedaleopsis confragosa, Fomitopsis pinicola, Cortinarius anserinus, Mucidula mucida, Trametes versicolor, Stropharia cyanea, Ramaria stricta, Radulomyces confluens, Gliophorus psittacinus, Psathyrella spadiceogrisea, Coprinopsis lagopus, Daedalea quercina, Amanita muscaria, Armillaria lutea, Vuilleminia comedens.*
- **Meta-testing classes:** *Hygrocybe ceracea, Trametes hirsuta, Polyporus tuberaster, Lacrymaria lacrymabunda, Fistulina hepatica, Gymnopus dryophilus, Amanita rubescens, Fuscoporia ferrea, Craterellus undulatus, Tricholoma scalpturatum, Mycena pura, Russula depallens, Bjerkandera adusta, Trametes gibbosa, Tremella mesenterica, Cerioporus varius, Amanita fulva, Xylodon paradoxus, Cuphophyllus virgineus, Cortinarius flexipes.*

Aircraft-S. Aircraft [17] is a fine-grained image dataset that contains 102 categories of aircraft. We randomly choose 100 variants with 100 images in each variant and split them into meta-training/meta-validation/meta-testing with 12/16/20 categories respectively.

- **Meta-training classes:** *MD-90, 737-600, A310, An-12, DR-400, Falcon-900, DC-3, Challenger-600, Fokker-70, Cessna-172, 747-400, ERJ-145.*
- **Meta-validation classes:** *737-900, A340-600, 737-800, 737-400, L-1011, A330-200, Gulfstream-V, 737-500, A340-200, ATR-72, MD-11, CRJ-700, EMB-120, Fokker-100, DC-6, 737-700.*
- **Meta-testing classes:** *707-320, PA-28, Cessna-208, F-A-18, DHC-8-300, ERJ-135, Tornado, BAE-146-200, A321, ATR-42, Saab-2000, Tu-134, Fokker-50, A380, MD-80, Gulfstream-IV, Yak-42, 747-100, 767-400, Embraer-Legacy-600.*

Texture-S. Texture [8] dataset contains 47 classes with 5640 images in total, where each class has 120 images. We randomly split them into meta-training/meta-validation/meta-testing with 12/7/10 classes respectively.

- **Meta-training classes:** *pitted, woven, crosshatched, crystalline, sprinkled, lacelike, bubbly, marbled, dotted, bumpy, striped, zigzagged.*
- **Meta-validation classes:** *wrinkled, grid, perforated, cobwebbed, honeycombed, cracked, blotchy.*
- **Meta-testing classes:** *fibrous, matted, scaly, chequered, flecked, paisley, braided, polka-dotted, interlaced, meshed.*