Set-based Meta-Interpolation for Few-Task Meta-Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Meta-learning approaches enable machine learning systems to adapt to new tasks given few examples by leveraging knowledge from related tasks. However, a large number of meta-training tasks are still required for generalization to unseen tasks during meta-testing, which introduces a critical bottleneck for real-world problems that come with only few tasks, due to various reasons including the difficulty and cost of constructing tasks. Recently, several task augmentation methods have been proposed to tackle this issue using domain-specific knowledge to design augmentation techniques to densify the meta-training task distribution. However, such reliance on domain-specific knowledge renders these methods inapplicable to other domains. While Manifold Mixup based task augmentation methods are domain-agnostic, we empirically find them ineffective on non-image domains. To tackle these limitations, we propose a novel domain-agnostic task augmentation method, Meta-Interpolation, which utilizes expressive neural set functions to densify the meta-training task distribution using bilevel optimization. We empirically validate the efficacy of Meta-Interpolation on eight datasets spanning across various domains such as image classification, molecule property prediction, text classification and speech recognition. Experimentally, we show that Meta-Interpolation consistently outperforms all the relevant baselines. Theoretically, we prove that task interpolation with the set function regularizes the meta-learner to improve generalization.

1 Introduction

The ability to learn a new task given only a few examples is crucial for artificial intelligence. Recently, meta-learning [39, 3] has emerged as a viable method to achieve this objective and enables machine learning systems to quickly adapt to a new task by leveraging knowledge from other related tasks seen during meta-training. Although existing meta-learning methods can efficiently adapt to new tasks with few data samples, a large dataset of meta-training tasks is still required to learn meta-knowledge that can be transferred to unseen tasks. For many real-world applications, such extensive collections of meta-training tasks may be unavailable. Such scenarios give rise to the few-task meta-learning problem where a meta-learner can easily memorize the meta-training tasks but fail to generalize well to unseen tasks. The few-task meta-learning problem usually results from the difficulty in task generation and data collection. For instance, in the medical domain, it is infeasible to collect a large amount of data to construct extensive meta-training tasks due to privacy concerns. Moreover, for natural language processing, it is not straightforward to split a dataset into tasks, and hence entire datasets are treated as tasks [30].

Several works have been proposed to tackle the few-task meta-learning problem using task augmentation techniques such as clustering a dataset into multiple tasks [30], leveraging strong image

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

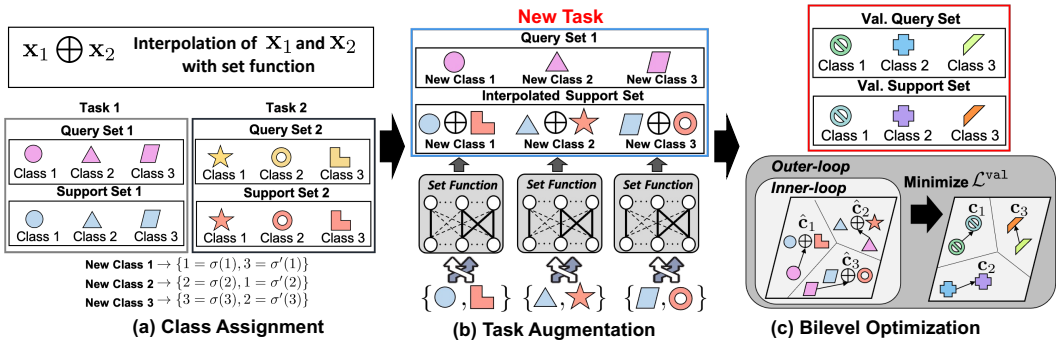


Figure 1: **Concept.** Three-way one-shot classification problem. **(a)** A new class is assigned to a pair of classes sampled without replacement from the pool of meta-training tasks. **(b)** The support sets are interpolated with a set function and paired with a query set. **(c)** Bilevel optimization of the set function and meta-learner.

augmentation methods such as vertical flipping to construct new classes [32], and the employment of Manifold Mixup [44] for densifying the meta-training task distribution [49, 50]. However, majority of these techniques require domain-specific knowledge to design such task augmentations and hence cannot be applied to other domains. While Manifold Mixup based methods [49, 50] are domain-agnostic, we empirically find them ineffective for mitigating meta-overfitting in few-task meta-learning especially in non-image domains such as chemical and text, and that they sometimes degrade generalization performance.

In this work, we focus solely on *domain-agnostic* task augmentation methods that can densify the meta-training task distribution to prevent meta-overfitting and improve generalization at meta-testing for few-task meta-learning. To tackle the limitations already discussed, we propose a novel domain-agnostic task augmentation method for metric based meta-learning models. Our method, Meta-Interpolation, utilizes expressive neural set functions to interpolate two tasks and the set functions are trained with bilevel optimization so that a meta-learner trained on the interpolated tasks generalizes to tasks in the meta-validation set. As a consequence of end-to-end training, the learned augmentation strategy is tailored to each specific domain without the need for specialized domain knowledge.

For example, for K-way classification, we sample two tasks consisting of support and query sets and assign a new class k to each pair of classes $\{\sigma(k), \sigma'(k)\}$ for $k = 1, \ldots, K$, where σ, σ' are permutations on $\{1, \ldots, K\}$ as depicted in Figure 1a. Hidden representations of the support set with classes $\sigma(k)$ and $\sigma\prime(k)$ are then transformed into a single support set using a set function that maps a set of two vectors to a single vector. We refer to the output of the set function as the interpolated support set and these are used to compute class prototypes. As shown in Figure 1b, the interpolated support set is paired with a query set (query set 1 in Figure 1a)), randomly selected from the two tasks to obtain a new task. Lastly, we optimize the set function so that a meta-learner trained on the augmented task can minimize the loss on the meta-validation tasks as illustrated in Figure 1c.

To verify the efficacy of our method, we empirically show that it significantly improves the performance of Prototypical Networks [40] on the few-task meta-learning problem across multiple domains. Our method outperforms the relevant baselines on eight few-task meta-learning benchmark datasets spanning image classification, chemical property prediction, text classification, and automatic speech recognition. Furthermore, our theoretical analysis shows that our task interpolation method with the set function regularizes the meta-learner and improves generalization performance.

Our contribution is threefold:

- We propose a novel domain-agnostic task augmentation method, Meta-Interpolation, which leverages expressive set functions to densify the meta-training task distribution for the few-task meta-learning problem.
- We theoretically analyze our model and show that it regularizes the meta-learner for better generalization.
- Through extensive experiments, we show that Meta-Interpolation significantly improves the performance of Prototypical Network on various domains such as image, text, and chemical molecule classification, and speech recognition on the few-task meta-learning problem.

2 Related Work

Meta-Learning The two mainstream approaches to meta-learning are gradient based [10, 33, 14, 24, 12, 36, 37] and metric based meta-learning [45, 40, 42, 29, 26, 6, 38]. The former formulates meta-knowledge as meta-parameters such as the initial model parameters and performs bilevel optimization to estimate the meta-parameters so that a meta-learner can generalize to unseen tasks with few gradient steps. The latter learns an embedding space where classification is performed by measuring the distance between a query and a set of class prototypes. In this work, we focus on metric based meta-learning with *fewer number* of meta-training tasks, i.e., few-task meta-learning. We propose a novel task augmentation method that densifies the meta-training task distribution and mitigates overfitting due to the fewer number of meta-training tasks for better generalization to unseen tasks.

Task Augmentation for Few-Task Meta-learning Several methods have been proposed to augment the number of meta-training tasks to mitigate overfitting in the context of few-task meta-learning. Ni et al. [32] apply strong data augmentations such as vertical flip to images to create a new class. For text classification, Murty et al. [30] split meta-training tasks into latent reasoning categories by clustering data with a pretrained language model. However, they require domain-specific knowledge to design such augmentations, and hence the resulting augmentation techniques are inapplicable to other domains where there is no well-defined data augmentation or pretrained model. In order to tackle this limitation, Manifold Mixup-based task augmentations have also been proposed. MetaMix [49] interpolates support and query sets with Manifold Mixup [44] to construct a new query set. MLTI [50] performs Manifold Mixup [44] on support and query sets from two tasks for task augmentation. Although these methods are domain-agnostic, we empirically find that they are not effective in some domains and can degrade generalization performance. In contrast, we propose to train an expressive neural set function to interpolate two tasks with bilevel optimization to find optimal augmentation strategies tailored specifically to each domain.

3 Method

Preliminaries In meta-learning, we are given a finite set of tasks $\{\mathcal{T}_t\}_{t=1}^{T}$, which are i.i.d samples from an unknown task distribution $p(\mathcal{T})$. Each task $\mathcal{T}_t$ consists of a support set $\mathcal{D}_t^s = \{(\mathbf{x}_{t,i}^s, y_{t,i}^s)\}_{i=1}^{N_s}$ and a query set $\mathcal{D}_t^q = \{(\mathbf{x}_{t,i}^q, y_{t,i}^q)\}_{i=1}^{N_q}$, where $\mathbf{x}_{t,i}$ and $y_{t,i}$ denote a data point and its corresponding label respectively. Given a predictive model $\hat{f}_{\theta,\lambda} \coloneqq f_{\theta_L}^L \circ \cdots \circ f_{\theta_{l+1}}^{l+1} \circ \varphi_\lambda \circ f_{\theta_l}^l \circ \cdots \circ f_{\theta_1}^1$ with L layers, we want to estimate the parameter θ that minimizes the meta-training loss and generalizes to query sets $\mathcal{D}_*^q$ sampled from an unseen task $\mathcal{T}_*$ using the support set $\mathcal{D}_*^s$, where λ is a hyperparameter for the function φ_λ. In this work, we primarily focus on metric based meta-learning methods rather than gradient based meta-learning methods due to efficiency and empirically higher performance over the gradient based methods on the tasks we consider.

Problem Statement In this work, we focus solely on *few-task* meta-learning. Here, the number of meta-training tasks drawn from the meta-training distribution is extremely small and the goal of a meta-learner is to learn meta-knowledge from such limited tasks that can be transferred to unseen tasks during meta-testing. The key challenges here are preventing the meta-learner from overfitting on the meta-training tasks and generalizing to unseen tasks drawn from a meta-test set.

Metric Based Meta-Learning The goal of metric based meta-learning is to learn an embedding space induced by $\hat{f}_{\theta,\lambda}$, where we perform classification by computing distances between data points and class prototypes. We adopt Prototypical Network (ProtoNet) [40] for $\hat{f}_{\theta,\lambda}$, where φ_λ is the identity function. Specifically, for each task $\mathcal{T}_t$ with its corresponding support $\mathcal{D}_t^s$ and query $\mathcal{D}_t^q$ sets, we compute class prototypes $\{\mathbf{c}_k\}_{k=1}^{K}$ as the average of the hidden representation of the support samples belonging to the class k as follows:

$$\mathbf{c}_k \coloneqq \frac{1}{N_k} \sum_{\substack{(\mathbf{x}_{t,i}^s, y_{t,i}^s) \in \mathcal{D}_t^s \\ y_{t,i}=k}} \hat{f}_{\theta,\lambda}(\mathbf{x}_{t,i}^s) \in \mathbb{R}^D \tag{1}$$

where N_k denotes the number of instances belonging to the class k. Given a metric $d(\cdot,\cdot) : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$, we compute the probability of a query point $\mathbf{x}_{t,i}^q$ being assigned to the class k by measuring the distance between the hidden representation $\hat{f}_{\theta,\lambda}(\mathbf{x}_{t,i}^q)$ and the class prototype $\mathbf{c}_k$ followed by

Algorithm 1 Meta-training

Require: Tasks $\{\mathcal{T}_t^{\text{train}}\}_{t=1}^{T}$ $\{\mathcal{T}_t^{\text{val}'}\}_{t'=1}^{T'}$, learning rate $\alpha, \eta \in \mathbb{R}_+$, update period S, and batch size B.

Initialize parameters θ, λ
for all $i \leftarrow 1, \ldots, M$ **do**
  $\mathcal{L}_{tr} \leftarrow 0$
  for all $j \leftarrow 1, \ldots, B$ **do**
    Sample two tasks $\mathcal{T}_{t_1} = \{\mathcal{D}_{t_1}^s, \mathcal{D}_{t_1}^q\}$ and $\mathcal{T}_{t_2} = \{\mathcal{D}_{t_2}^s, \mathcal{D}_{t_2}^q\}$ from $\{\mathcal{T}_t^{\text{train}}\}_{t=1}^{T}$.
    $\hat{\mathcal{D}}^s \leftarrow$ Interpolate$(\mathcal{D}_{t_1}^s, \mathcal{D}_{t_2}^s, \varphi_\lambda)$ with Eq.3.
    $\hat{\mathcal{T}} \leftarrow \{\hat{\mathcal{D}}^s, \mathcal{D}_{t_1}^q\}$
    $\mathcal{L}_{tr} \mathrel{+}= \frac{1}{2B}\mathcal{L}_{\text{singleton}}(\lambda, \theta, \mathcal{T}_{t_1})$
    $\mathcal{L}_{tr} \mathrel{+}= \frac{1}{2B}\mathcal{L}_{\text{mix}}(\lambda, \theta, \hat{\mathcal{T}})$
  end for
  $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}_{tr}}{\partial \theta}$
  if $\texttt{mod}(i, S) = 0$ **then**
    $g \leftarrow$ HyperGrad$(\theta, \lambda, \{\mathcal{T}_{t'}^{\text{val}}\}_{t'=1}^{T'}, \alpha, \frac{\partial \mathcal{L}_{tr}}{\partial \theta})$
    $\lambda \leftarrow \lambda - \eta \cdot g$
  end if
end for
return θ, λ

Algorithm 2 HyperGrad [27]

Require: model parameter θ, hyperparamter λ, validation tasks $\{\mathcal{T}_{t'}^{\text{val}}\}_{t'=1}^{T'}$, learning rate α, gradient of training loss w.r.t θ $\frac{\partial \mathcal{L}_{tr}}{\partial \theta}$, batch size B', and the number of iterations for Neumann series $q \in \mathbb{N}$.

$\mathcal{L}_V \leftarrow 0$
for all $i \leftarrow 1, \ldots, B'$ **do**
  Sample a task $\mathcal{T}_t$ from $\{\mathcal{T}_{t'}^{\text{val}}\}_{t'=1}^{T'}$.
  $\mathcal{L}_V \mathrel{+}= \frac{1}{B'}\mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T})$
end for
$\mathbf{v}_1 \leftarrow \frac{\partial \mathcal{L}_V}{\partial \theta}$
Initialize $\mathbf{p} \leftarrow \texttt{deepcopy}(\mathbf{v}_1)$
for all $j \leftarrow 1, \ldots, q$ **do**
  $\mathbf{v}_1 \mathrel{-}= \alpha \cdot \text{grad}(\frac{\partial \mathcal{L}_{tr}}{\partial \theta}, \theta, \text{grad_outputs} = \mathbf{v}_1)$
  $\mathbf{p} \mathrel{+}= \mathbf{v}_1$
end for
$\mathbf{v}_2 \leftarrow \text{grad}(\frac{\mathcal{L}_{tr}}{\partial \theta}, \lambda, \text{grad_outputs} = \alpha \mathbf{p})$
return $\frac{\partial \mathcal{L}_V}{\partial \lambda} - \mathbf{v}_2$

softmax. With the class probability, we compute the cross-entropy loss for ProtoNet as follows:

$$\mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}_t) := \sum_{i,k} \mathbb{1}_{\{y_{t,i}=k\}} \cdot \log \frac{\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}_{t,i}^q), \mathbf{c}_k))}{\sum_{k'} \exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}_{t,i}^q), \mathbf{c}_{k'}))} \tag{2}$$

where $\mathbb{1}$ is an indicator function. At meta-test time, a test query is assigned a label based on the minimal distance to a class prototype, i.e., $y_*^q = \arg\min_k d(\hat{f}_{\theta,\lambda}(\mathbf{x}_*^q), \mathbf{c}_k)$. However, optimizing $\frac{1}{T}\sum_{t=1}^{T} \mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}_t)$ w.r.t θ is prone to overfitting since we are given only a small number of meta-training tasks. The meta-learner tends to memorize the meta-training tasks, which limits its generalization to new tasks at meta-test time [51, 35].

Meta-Interpolation for Task Augmentation In order to tackle the meta-overfitting problem with a small number of tasks, we propose a novel data-driven domain-agnostic task augmentation framework which enables the meta-learner trained on few tasks to generalize to unseen few-shot classification tasks. Several methods have been proposed to densify the meta-training tasks. However, they heavily depend on the augmentation of images [32] or need a pretrained language model for task augmentation [30]. Although Manifold Mixup based methods [49, 50] are domain-agnostic, we empirically find them ineffective in certain domains. Instead, we optimize expressive neural set functions to augment tasks to enhance the generalization of a meta-learner to unseen tasks. As a consequence of end-to-end training, the learned augmentation strategy is tailored to each domain.

Specifically, let $\varphi_\lambda : \mathbb{R}^{n \times d} \to \mathbb{R}^d$ be a set function which maps a set of d dimensional vectors with cardinality n to a d dimensional vector. In all our experiments, we use Set Transformer [23] for φ_λ. Given a pair of tasks $\mathcal{T}_{t_1} = \{\mathcal{D}_{t_1}^s, \mathcal{D}_{t_1}^q\}$ and $\mathcal{T}_{t_2} = \{\mathcal{D}_{t_2}^s, \mathcal{D}_{t_2}^q\}$ with corresponding support and query sets for K way classification, we construct new classes by choosing K pairs of classes from the two tasks. We sample permutations σ_{t_1} and σ_{t_2} on $\{1, \ldots, K\}$ for each task $\mathcal{T}_{t_1}$ and $\mathcal{T}_{t_2}$ respectively and assign class k to the pair $\{\sigma_{t_1}(k), \sigma_{t_2}(k)\}$ for $k = 1, \ldots, K$. For the newly assigned class k, we pair two instances from classes $\sigma_{t_1}(k)$ and $\sigma_{t_2}(k)$ and interpolate their hidden representations with the set function φ_λ. The class prototypes for class k are computed using the output of φ_λ as follows:

$$\begin{gathered}
S_k := \{(\{\mathbf{x}_{t_1,i}^s, \mathbf{x}_{t_2,j}^s\}, k) \mid (\mathbf{x}_{t_1,i}^s, y_{t_1,i}^s) \in \mathcal{D}_{t_1}^s, y_{t_1,i}^s = \sigma_{t_1}(k), (\mathbf{x}_{t_2,j}^s, y_{t_2,j}^s) \in \mathcal{D}_{t_2}^s, y_{t_2,j}^s = \sigma_{t_2}(k)\} \\
\mathbf{h}_{t_1,i}^{s,l} := (f_{\theta_l}^l \circ \cdots \circ f_{\theta_1}^1)(\mathbf{x}_{t_1,i}^s), \quad \mathbf{h}_{t_2,j}^{s,l} := (f_{\theta_l}^l \circ \cdots \circ f_{\theta_1}^1)(\mathbf{x}_{t_2,j}^s) \in \mathbb{R}^d \\
\hat{\mathbf{c}}_k := \frac{1}{|S_k|} \sum_{(\{\mathbf{x}_{t_1,i}^s, \mathbf{x}_{t_2,j}^s\}, k) \in S_k} \left(f_{\theta_L}^L \circ \cdots \circ f_{\theta_{l+1}}^{l+1}\right)\left(\varphi_\lambda(\{\mathbf{h}_{t_1,i}^{s,l}, \mathbf{h}_{t_2,j}^{s,l}\})\right) \in \mathbb{R}^D \\
\hat{\mathcal{D}}^s := \{\hat{\mathbf{c}}_1, \ldots, \hat{\mathbf{c}}_K\}
\end{gathered} \tag{3}$$

where we define $\hat{\mathcal{D}}^s$ to be the set of all the interpolated prototypes $\hat{\mathbf{c}}_k$ for $k = 1, \ldots, K$. For queries, we do not perform any interpolation. Instead, we use $\mathcal{D}_{t_1}^q$ as the query set and compute its hidden

representation $\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t_1,i}) \in \mathbb{R}^D$. We then measure the distance between the query with $y^q_{t_1,i} = \sigma_{t_1}(k)$ and the interpolated prototype of class k to compute the loss as follows:

$$\mathcal{L}_{\text{mix}}(\lambda, \theta, \hat{\mathcal{T}}) := -\sum_{i,k} \mathbb{1}_{\{y^q_{t_1,i}=\sigma_{t_1}(k)\}} \cdot \log \frac{\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t_1,i}), \hat{\mathbf{c}}_k))}{\sum_{k'} \exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t_1,i}), \hat{\mathbf{c}}_{k'}))} \tag{4}$$

where $\hat{\mathcal{T}} = \{\hat{\mathcal{D}}^s, \mathcal{D}^q_{t_1}\}$. The intuition behind interpolating only support sets is to construct harder tasks that a meta-learner cannot memorize. Alternatively, we can interpolate only query sets. However, this is computationally more expensive since the size of query sets is usually larger than that of support sets. In Section 5, we empirically show that interpolating either support or query sets achieves higher training loss than interpolating both, which empirically supports the intuition. Lastly, we also use the original task $\mathcal{T}_{t_1}$ to evaluate the loss $\mathcal{L}_{\text{singleton}}(\lambda, \theta, \mathcal{T}_{t_1})$ in Eq. 2 by passing the corresponding support and query set to $\hat{f}_{\theta,\lambda}$. The additional forward pass enriches the diversity of the augmented tasks and makes meta-training consistent with meta-testing since we do not perform any task augmentation in the meta-testing stage.

Optimization Since jointly optimizing θ, the parameters of ProtoNet, and λ, the parameters of the set function φ_λ, with few tasks is prone to overfitting, we consider λ as hyperparameters and perform bilevel optimization with meta-training and meta-validation tasks as follows:

$$\lambda^* := \arg\min_{\lambda} \frac{1}{T'} \sum_{t=1}^{T'} \mathcal{L}_{\text{singleton}}(\lambda, \theta^*(\lambda); \mathcal{T}^{\text{val}}_t) \tag{5}$$

$$\theta^*(\lambda) := \arg\min_{\theta} \frac{1}{2T} \sum_{t=1}^{T} \mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}^{\text{train}}_t) + \mathcal{L}_{\text{mix}}(\lambda, \theta; \hat{\mathcal{T}}_t) \tag{6}$$

where $\mathcal{T}^{\text{train}}_t, \mathcal{T}^{\text{val}}_t, \hat{\mathcal{T}}_t$ denote the meta-training, meta-validation, and interpolated task, respectively. Since computing the exact gradient w.r.t λ is intractable due to the long inner optimization steps in Eq. 6, we leverage the implicit function theorem to approximate the gradient as Lorraine et al. [27]. Moreover, we alternately update θ and λ for computational efficiency as described in Algo. 1 and 2.

4 Theoretical Analysis

In this section, we theoretically investigate the behavior of the Set Transformer and how it induces a distribution dependent regularization, which is then shown to have the ability to control the Rademacher complexity for better generalization. To analyze the behavior of the Set Transformer, we first define it concretely with the attention mechanism $A(Q, K, V) = \text{softmax}(\sqrt{d^{-1}}QK^\top)V$. Given $h, h' \in \mathbb{R}^d$, define $H_1^{\{h,h'\}} = [h, h']^\top \in \mathbb{R}^{2\times d}$ and $H_1^{\{h\}} = h^\top \in \mathbb{R}^{1\times d}$. Then, for any $r \in \{\{h, h'\}, \{h\}\}$, the output of the Set Transformer $\varphi_\lambda(r)$ is defined as follows:

$$\varphi_\lambda(r) = A(Q_2, K_2^r, V_2^r)^\top \in \mathbb{R}^d. \tag{7}$$

where $Q_2 = SW_2^Q + b_2^Q$, $Q_1^r = H_1^r W_1^Q + \mathbf{1}_2 b_1^Q$, $K_j^r = H_j^r W_j^K + \mathbf{1}_2 b_j^K$, $V_j^r = H_j^r W_j^V + \mathbf{1}_2 b_j^V$ (for $j \in \{1, 2\}$), and $H_2^r = A(Q_1^r, K_1^r, V_1^r) \in \mathbb{R}^{n\times d}$. Q_j, K_j, V_j denote query, key, and value for the attention mechanism for $j = 1, 2$, respectively. Here, $\mathbf{1}_2 = [1, \dots, 1]^\top \in \mathbb{R}^n$, $W_j^Q, W_j^K, W_j^V \in \mathbb{R}^{d\times d}$, $b_j^Q, b_j^K, b_j^V \in \mathbb{R}^{1\times d}$, $Q_1^r, K_j^r, V_j^r \in \mathbb{R}^{n\times d}$, and $Q_2 \in \mathbb{R}^{1\times d}$. Let $l \in \{1, \dots, L\}$.

Our analysis will show the importance of the following quantity of the Set Transformer in our method:

$$\alpha_{ij}^{(t,t')} = p_2^{(t,t',i,j)}(1 - p_1^{(t,t',i,j)}) + (1 - p_2^{(t,t',i,j)})(1 - \tilde{p}_1^{(t,t',i,j)}), \tag{8}$$

where $p_1^{(t,t',i,j)} = \text{softmax}(\sqrt{d^{-1}}Q_1^{\{h_{t,i},h_{t',j}\}}(K_1^{\{h_{t,i},h_{t',j}\}})^\top)_{1,1}$, $\tilde{p}_1^{(t,t',i,j)} = \text{softmax}(\sqrt{d^{-1}}Q_1^{\{h_{t,i},h_{t',j}\}}(K_1^{\{h_{t,i},h_{t',j}\}})^\top)_{2,1}$, $p_2^{(t,t',i,j)} = \text{softmax}(\sqrt{d^{-1}}Q_2(K_2^{\{h_{t,i},h_{t',j}\}})^\top)_{1,1}$ with $h_{t,i} = \phi_\theta^l(\mathbf{x}^s_{t,i})$ and $\phi_\theta^l = f_{\theta_l}^l \circ \cdots \circ f_{\theta_1}^1$. For a matrix $A \in \mathbb{R}^{m\times n}$, $A_{i,j}$ denotes the entry for i-th row and j-th column of the matrix A.

We now introduce the additional notation and problem setting to present our results. Define $W = (W_1^V W_2^V)^\top \in \mathbb{R}^{d\times d}$, $b = (b_1^V W_2^V + b_2^V)^\top \in \mathbb{R}^d$, $L_t(\mathbf{c}) = -\frac{1}{n}\sum_{i=1}^n \log \frac{\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \mathbf{c}_{y^q_{t,i}}))}{\sum_{k'} \exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \mathbf{c}_{k'}))}$,

and $I_{t,k} = \{i \in [N_s^{(t)}] : y_{t,i}^s = k\}$, where $N_s^{(t)} = |\mathcal{D}_t^s|$. We also define the empirical measure $\mu_{t,k} = \frac{1}{|I_{t,k}|}\sum_{i\in I_{t,k}} \delta_i$ over the index $i \in [N_s^{(t)}]$ with the Dirac measures δ_i. Let $U[K]$ be the uniform distribution over $\{1, \dots, K\}$. For any function φ and point u in its domain, we define the j-th order tensor $\partial^j \varphi(u) \in \mathbb{R}^{d\times d\times\cdots\times d}$ by $\partial^j \varphi(u)_{i_1 i_2 \cdots i_j} = \frac{\partial^j}{\partial u_{i_1} u_{i_2} \cdots \partial u_{i_j}} \varphi(u)$. For example, $\partial^1 \varphi(u)$ and $\partial^2 \varphi(u)$ are the gradient and the Hessian of φ evaluated at u. For any j-th order tensor $\partial^j \varphi(u)$, we define the vectorization of the tensor by $\mathrm{vec}[\partial^j \varphi(u)] \in \mathbb{R}^{d^j}$. For an vector $a \in \mathbb{R}^d$, we define $a^{\otimes j} = a \otimes a \otimes \cdots \otimes a \in \mathbb{R}^{d^j}$ where $\otimes$ represents the Kronecker product. We assume that $\partial^r g\left(W\phi_{\theta_l}^l(\mathbf{x}_{t,i}^s) + b\right) = 0$ for all $r \geq 2$, where $g := f_{\theta_L}^L \circ \cdots \circ f_{\theta_{l+1}}^{l+1}$. This assumption is satisfied, for example, if g represents a deep neural network with ReLU activations. This assumption is also satisfied in the simpler special case considered in the proposition below.

The following theorem shows that $\mathcal{L}_{\text{mix}}(\lambda, \theta, \hat{\mathcal{T}}_{t,t'})$ is approximately $\mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}_t)$ plus regularization terms on the directional derivatives of $\phi_{\theta_l}^l$ on the direction of $W(\phi_{\theta_l}^l(\mathbf{x}_{t',j}^s) - \phi_{\theta_l}^l(\mathbf{x}_{t,i}^s))$:

Theorem 1. *For any $J \in \mathbb{N}_+$, if $c \mapsto d(y, c)$ is J-times differentiable for all y, then the J-th order approximation of $\mathcal{L}_{mix}(\lambda, \theta, \hat{\mathcal{T}}_{t,t'})$ is given by $\mathcal{L}_{singleton}(\lambda, \theta; \mathcal{T}_t) + \sum_{j=1}^J \frac{1}{j!}\mathrm{vec}[\partial^j L_t(\mathbf{c})]^\top \Delta^{\otimes j}$, where $\Delta = [\Delta_1^\top, \dots, \Delta_K^\top]^\top$ and*

$$\Delta_k^\top = \mathbb{E}_{\substack{i\sim\mu_{t,k},\\ j\sim\mu_{t',\sigma(k)}}}\left[\alpha_{ij}^{(t,t')} \partial g\left(W\phi_{\theta_l}^l(\mathbf{x}_{t,i}^s) + b\right) W\left(\phi_{\theta_l}^l(\mathbf{x}_{t',j}^s) - \phi_{\theta_l}^l(\mathbf{x}_{t,i}^s)\right)\right].$$

To illustrate the effect of this data-dependent regularization, we now consider the following special case that is used by Yao et al. [50] for ProtoNet: $\mathcal{L}(\lambda, \theta; \mathcal{T}_t) = \frac{1}{n}\sum_{i=1}^n \mathcal{L}_i(\lambda, \theta; \mathcal{T}_t)$ where $\mathcal{L}_i(\lambda, \theta; \mathcal{T}_t) = \frac{1}{1+\exp(\langle(\mathbf{x}_{t,i}^q - (\mathbf{c}_1' + \mathbf{c}_2')/2, \theta\rangle)}$ where $\mathbf{c}_k' := \frac{1}{N_{t,k}}\sum_{(\mathbf{x}_{t,i}^s, \mathbf{y}_{t,i}^s)\in\mathcal{D}_t^s} \mathbb{1}_{\{\mathbf{y}_{t,i}=k\}}\mathbf{x}_{t,i}^s$, and $\langle\cdot,\cdot\rangle$ denotes dot product. Define $c = \frac{1}{n}\sum_{i=1}^n \frac{1}{4}\frac{\psi(z_{t,i})(\psi(z_{t,i})-0.5)}{1+\exp(z_{t,i})}$, where $\psi(z_{t,i}) = \frac{\exp(z_{t,i})}{1+\exp(z_{t,i})}$ and $z_{t,i} = \langle \mathbf{x}_{t,i}^q - (\mathbf{c}_1' + \mathbf{c}_2')/2, \theta\rangle$. Note that $c > 0$ if θ is no worse than the random guess; e.g., $\mathcal{L}_i(\lambda, 0; \mathcal{T}_t) > \mathcal{L}_i(\lambda, \theta; \mathcal{T}_t)$ for all $i \in [n]$. We write $\|v\|_M^2 = v^\top M v$ for any positive semi-definite matrix M. In this special case, we consider that α is balanced: i.e., $\mathbb{E}_{t',\sigma}[\sum_{k=1}^2 \frac{1}{|I_{t',\sigma(k)}|}\sum_{j\in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|}\sum_{i\in I_{t,k}} \alpha_{ij}^{(t,t')}(\phi_{\theta_l}^l(\mathbf{x}_{t',j}^s) - \phi_{\theta_l}^l(\mathbf{x}_{t,i}^s))] = 0$ for all t. This is used to prevent the Set Transformer from over-fitting to the training sets; i.e., in such simple special cases, the Set Transformer without any restriction is too expressive relative to the rest of the model (and may memorize the training sets without using the rest of the model). The following proposition shows that the additional regularization term is simplified to the form of $c\|\theta\|_M^2$ in this special case:

Proposition 1. *In the special case explained above, the second approximation of $\mathbb{E}_{t',\sigma}[\mathcal{L}_{mix}(\lambda, \theta, \hat{\mathcal{T}}_{t,t'})]$ is given by $\mathcal{L}_{singleton}(\lambda, \theta; \mathcal{T}_t) + c\|\theta\|^2_{\mathbb{E}_{t',\sigma}[\delta_{t,t',\sigma}\delta_{t,t',\sigma}^\top]}$, where $\delta_{t,t',\sigma} = \mathbb{E}_{k\sim U[2]}\mathbb{E}_{i\sim\mu_{t,k}, j\sim\mu_{t',\sigma(k)}}[\alpha_{ij}^{(t,t')}(\mathbf{x}_{t',j}^s - \mathbf{x}_{t,i}^s)]$.*

In the above regularization form, we have an implicit regularization effect on $\|\theta\|_\Sigma^2$ where $\Sigma = \mathbb{E}_{\mathbf{x},\mathbf{x}'}[(\mathbf{x} - \mathbf{x}')(\mathbf{x} - \mathbf{x}')^\top]$. The following theorem shows that this implicit regularization can reduce the Rademacher complexity for better generalization:

Proposition 2. *Let $\mathcal{F}_R = \{\mathbf{x} \mapsto \theta^\top \mathbf{x} : \|\theta\|_\Sigma^2 \leq R\}$ with $\mathbb{E}_{\mathbf{x}}[\mathbf{x}] = 0$. Then, $\mathcal{R}_n(\mathcal{F}_R) \leq \frac{\sqrt{R}\sqrt{\mathrm{rank}(\Sigma)}}{\sqrt{n}}$.*

All the proofs are presented in Appendix B.

5 Experiments

We now demonstrate the efficacy of our set-based task augmentation method on multiple few-task benchmark datasets and compare against the relevant baselines.

Datasets We perform classification on eight datasets to validate our method. (1), (2), & (3) Metabolism [17], NCI [31] and Tox21 [18]: these are binary classification datasets for predicting the properties of chemical molecules. For Metabolism, we use three subdatasets for meta-training, meta-validation, and meta-testing, respectively. For NCI, we use four subdatasets for meta-training, two for meta-validation and the remaining three for meta-testing. For Tox21, we use six subdatasets

Table 1: Average accuracy of 5 runs and $\pm 95\%$ confidence interval for few shot classification on non-image domains – Tox21, NCI, GLUE-SciTail dataset, and ESC-50 datasets. ST stands for Set Transformer.

	Chemical			**Text**	**Speech**
	Metabolism	**Tox21**	**NCI**	**GLUE-SciTail**	**ESC-50**
Method	5-shot	5-shot	5-shot	4-shot	5-shot
ProtoNet	$63.62 \pm 0.56\%$	$64.07 \pm 0.80\%$	$80.45 \pm 0.48\%$	$72.59 \pm 0.45\%$	$69.05 \pm 1.48\%$
MetaReg	$66.22 \pm 0.99\%$	$64.40 \pm 0.65\%$	$80.94 \pm 0.34\%$	$72.08 \pm 1.33\%$	$74.95 \pm 1.78\%$
MetaMix	$68.02 \pm 1.57\%$	$65.23 \pm 0.56\%$	$79.46 \pm 0.38\%$	$72.12 \pm 1.04\%$	$71.99 \pm 1.41\%$
MLTI	$65.44 \pm 1.14\%$	$64.16 \pm 0.23\%$	$81.12 \pm 0.70\%$	$71.65 \pm 0.70\%$	$70.62 \pm 1.96\%$
ProtoNet+ST	$66.26 \pm 0.65\%$	$64.98 \pm 1.25\%$	$81.20 \pm 0.30\%$	$72.37 \pm 0.56\%$	$71.54 \pm 1.56\%$
Meta-Interpolation	$\mathbf{72.92} \pm 1.89\%$	$\mathbf{67.54} \pm 0.40\%$	$\mathbf{82.86} \pm 0.26\%$	$\mathbf{73.64} \pm 0.59\%$	$\mathbf{79.22} \pm 0.84\%$

for meta-training, two for meta-validation, and four for meta-testing. (4) GLUE-SciTail [30]: it consists of four natural language inference datasets where we predict whether a hypothesis sentence contradicts a premise sentence. We use MNLI [47] and QNLI [46] for meta-training, SNLI [5] and RTE [46] for meta-validation, and SciTail [20] for meta-testing. (5) ESC-50 [34]: this is an automatic speech recognition dataset. We make a 20/15/15 split out of 50 base classes for meta-training/validation/testing and sample 5 classes from each spilt to construct a 5-way classification task. (6) Rainbow MNIST (RMNIST) [11]: this is a 10-way classification dataset. Following Yao et al. [50], we construct each task by applying compositions of image transformations to the images of the MNIST [9] dataset. (7) & (8) Mini-ImageNet-S [45] and CIFAR100-FS [22]: these are 5-way classification datasets where we choose 12/16/20 classes out of 100 base classes for meta-training/validation/testing, respectively and sample 5 classes from each split to construct a task.

Note that Metabolism, Tox21, NCI, GLUE-SciTail, and ESC-50 are real-world few-task meta-learning datasets with a very small number of tasks. For Mini-ImageNet-S and CIFAR100-FS, following Yao et al. [50], we artificially reduce the number of tasks from the original datasets for few-task meta-learning. RMNIST is synthetically generated by applying augmentations to MNIST.

Implementation Details For RMNIST, Mini-ImageNet-S, and CIFAR100-FS, we use four convolutional blocks with each block consisting of a convolution, ReLU, batch normalization [19], and max pooling. For Metabolism, Tox21, and NCI, we convert the chemical molecules into SMILES format and extract a 1024 bit fingerprint feature using RDKit [15] where each bit captures a fragment of the molecule. We use two blocks of affine transformation, batch normalization, and Leaky ReLU, and affine transformation for the last layer. For GLUE-SciTail dataset, we stack 3 fully connected layers with ReLU on the pretrained language model ELECTRA [8]. For ESC-50 dataset, we pass raw audio signal to the pretrained VGGish [16] feature extractor to obtain an embedding vector. We use the feature vector as input to the classifier which is exactly the same as the one used for Metabolism, Tox21, and NCI. For our Meta-Interpolation, we use Set Transformer [23] for the set function φ_λ.

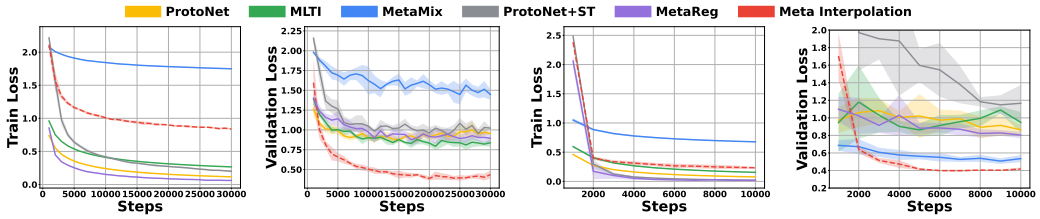
Baselines We compare our method against following *domain-agnostic* baselines.

1. **ProtoNet** [40]: Vanilla ProtoNet trained on Eq. 2 by fixing φ_λ to be the identity function.
2. **MetaReg** [2]: ProtoNet with ℓ_2 regularization where element-wise coefficients are learned with bilevel optimization.
3. **MetaMix** [49]: ProtoNet trained with support sets and mixed query sets where we interpolate one instance from the support sets and the other from the original query sets with Manifold Mixup.
4. **MLTI** [50]: ProtoNet trained with Manifold Mixup based task augmentation. We sample two tasks and interpolate two query sets and support sets with Manifold Mixup, respectively.
5. **ProtoNet+ST** ProtoNet and Set Transformer (φ_λ) trained with bilevel optimization but without task augmentation for $\mathcal{L}_{\text{mix}}(\lambda, \theta, \hat{\mathcal{T}}_t)$ in Eq. 6.
6. **Meta-Interpolation** Our full model learning to interpolate support sets from two tasks using bilevel optimization and training the ProtoNet with both the original and interpolated tasks.

Results As shown in Table 1, Meta-Interpolation consistently outperforms all the domain-agnostic task augmentation and regularization baselines on non-image domains. Notably, it significantly improves the performance on ESC-50, which is a challenging datatset that only contains 40 examples per class. In addition, Meta-Interpolation effectively tackles the Metabolism and GLUE-SciTail datasets which have an extremely small number of meta-training tasks: three and two meta-training tasks, respectively. Contrarily, the baselines do not achieve consistent improvements across all the

Table 2: Average accuracy of 5 runs and $\pm 95\%$ confidence interval for few shot classification on image domains — Rainbow MNIST, Mini-ImageNet, and CIFAR100. ST stands for Set Transformer.

Method	**RMNIST** 1-shot	**Mini-ImageNet-S** 1-shot	**Mini-ImageNet-S** 5-shot	**CIFAR-100-FS** 1-shot	**CIFAR-100-FS** 5-shot
ProtoNet	$75.35 \pm 1.43\%$	$39.14 \pm 0.78\%$	$51.17 \pm 0.57\%$	$38.05 \pm 1.56\%$	$52.63 \pm 0.74\%$
MetaReg	$76.40 \pm 0.56\%$	$39.36 \pm 0.45\%$	$50.94 \pm 0.67\%$	$37.74 \pm 0.70\%$	$52.73 \pm 1.26\%$
MetaMix	$76.54 \pm 0.72\%$	$38.25 \pm 0.09\%$	$52.38 \pm 0.52\%$	$36.13 \pm 0.63\%$	$52.52 \pm 0.89\%$
MLTI	$79.40 \pm 0.75\%$	$39.69 \pm 0.47\%$	$52.73 \pm 0.51\%$	$38.81 \pm 0.55\%$	$53.41 \pm 0.83\%$
ProtoNet+ST	$77.38 \pm 2.05\%$	$38.93 \pm 1.03\%$	$48.92 \pm 0.67\%$	$38.03 \pm 0.85\%$	$50.72 \pm 0.92\%$
Meta Interpolation	$\mathbf{83.24} \pm 1.39\%$	$\mathbf{40.28} \pm 0.48\%$	$\mathbf{53.06} \pm 0.33\%$	$\mathbf{41.48} \pm 0.45\%$	$\mathbf{54.94} \pm 0.80\%$



(a) Train RMNIST (b) Val. RMNIST (c) Train NCI (d) Val. NCI

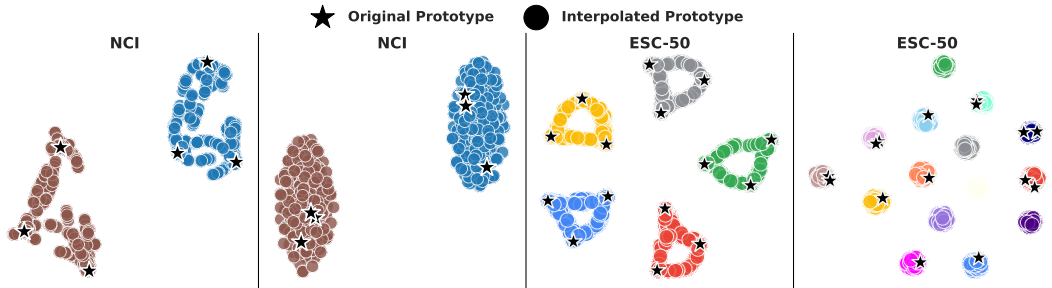
Figure 2: **(a)**$\sim$**(d)** Meta-train and meta-validation loss on RMNIST and NCI for ProtoNet, MLTI, MetaMix, ProtoNet+ST, and Meta Interpolation.

domains and tasks considered. For example, MetaReg is effective on the speech domain (ESC-50) and Metabolism, but does not work on the chemical (Tox21 and NCI) and text (GLUE-SciTail) domains. Similarly, MetaMix and MLTI achieve performance improvements on some datasets but degrade the test accuracy on others. This empirical evidence supports the hypothesis that the optimal task augmentation strategy varies across domains and justifies the motivation for Meta-Interpolation which learns augmentation strategies tailored to each domain.

We provide additional experimental results on the image domain in Table 2. Again, Meta-Interpolation outperforms all the baselines. In contrast to the previous experiments, MetaReg hurts the generalization performance on all the image datasets except on RMNIST. Note that Manifold Mixup-based augmentation methods, MetaMix and MLTI, marginally improve the generalization performance for 1-shot classification on Mini-ImageNet-S and CIFAR-100-FS, although they boost the accuracy on 5-shot experiments. This suggests that different task augmentation strategies are required for 1-shot and 5-shot for the same dataset. Meta-Interpolation on the other hand learns task augmentation strategies tailored for each task and dataset and consistently improves the performance of the vanilla ProtoNet for all the experiments on the image datasets.

Moreover, we plot the meta-training and meta-validation loss on RMNIST and NCI dataset in Figure 2. Meta-Interpolation obtains higher training loss but much lower validation loss than the others on both datasets. This implies that interpolating only support sets constructs harder tasks that a meta-learner cannot memorize and regularizes the meta-learner for better generalization. ProtoNet overfits to the meta-training tasks on both datasets. MLTI mitigates the overfitting issue on RMNIST but is not effective on the NCI dataset where it shows high validation loss in Figure 2d. On the other hand, MetaMix, which constructs a new query set by interpolating a support and query set with Manifold Mixup, results in generating overly difficult tasks which causes underfitting on RMNIST where the training loss is not properly minimized in Figure 2a. However, this augmentation strategy is effective for tackling meta-overfitting on NCI where the validation loss is lower than ProtoNet. The loss curve of ProtoNet+ST supports the claim that increasing the model size and using bilevel optimization cannot handle the few-task meta-learning problem. It shows higher validation loss on both RMNIST and NCI as presented in Figure 2b and 2d. Similarly, MetaReg which learns coefficients for ℓ_2 regularization fails to prevent meta-overfitting on both datasts.

Lastly, we empirically show that the performance gains mostly come from the task augmentation with Meta-Interpolation, rather than from bilevel optimization or the introduction of extra parameters with the set function. As shown in Table 1 and 2, ProtoNet+ST, which is Meta Interpolation but trained without any task augmentation, significantly degrades the performance of ProtoNet on Mini-ImageNet and CIFAR-100-FS. On the other datasets, the ProtoNet+ST obtains marginal improvement or largely



(a) MLTI (b) Meta-Interp. (c) MLTI (d) Meta-Interp.

Figure 3: Visualization of original and interpolated tasks from NCI ((**a**) and (**b**)) and ESC-50 ((**c**) and (**d**)).

underperforms the other baselines. Thus, the task augmentation strategy of interpolating two support sets with the set function φ_λ is indeed crucial for tackling the few-task meta-learning problem.

Ablation Study We further perform ablation studies to verify the effectiveness of each component of Meta-Interpolation. In Table 3, we show experimental results on the ESC-50 dataset while removing various components of our model. Firstly, we train our model without any task interpolation but keep the set function φ_λ, denoted as w/o Interpolation. The model without task interpolation significantly underperforms the full task-augmentation model, Meta-Interpolation, which shows that the improvements come from task interpolation, rather than the extra parameters introduced by the set encoding layer. Moreover, bilevel optimization is shown to be effective for estimating λ, which are the parameters of the set function. Jointly training ProtoNet and the set the function without bilevel optimization, denoted as w/o Bilevel, largely degrades the test accuracy by 15%. Lastly, we remove the loss $\mathcal{L}_{\text{singleton}}(\lambda, \theta, \mathcal{T}_t^{\text{train}})$ for inner optimization in Eq. 6, denoted as w/o $\mathcal{L}_{\text{singleton}}(\lambda, \theta, \mathcal{T}_t^{\text{train}})$. This hurts the generalization performance since it decreases the diversity of tasks and causes inconsistency between meta-training and meta-testing, since we do not perform any interpolation for support sets at meta-test time.

Table 3: Ablation study on ESC-50.

Model	Accuracy
Meta-Interpolation	**79.22** $\pm$ 0.84
w/o Interpolation	71.54 $\pm$ 1.56
w/o Bilevel	63.01 $\pm$ 2.06
w/o $\mathcal{L}_{\text{singleton}}(\lambda, \theta, \mathcal{T}_t^{\text{train}})$	78.01 $\pm$ 1.56

Visualization In Figure 3, we present the t-SNE [43] visualizations of the original and interpolated tasks with MLTI and Meta-Interpolation, respectively. Following Yao et al. [50], we sample three original tasks from NCI and ESC-50 dataset, where each task is a two-way five-shot and five-way five-shot classification problem, respectively. The tasks are interpolated with MLTI or Meta-Interpolation to construct 300 additional tasks and represented as a set of all class prototypes. To visualize the prototypes, we first perform Principal Component Analysis [13] (PCA) to reduce the dimension of each prototype. The first 50 principal components are then used to compute the t-SNE visualizations. As shown in Figure 3b and 3d, Meta-Interpolation successfully learns an expressive neural set function that densifies the task distribution. The task augmentations with MLTI, however, do not cover a wide embedding space as shown in Figure 3a and 3c as the mixup strategy allows to generate tasks only on the simplex defined by the given set of tasks.

6 Conclusion

We proposed a novel domain-agnostic task augmentation method, Meta Interpolation, to tackle the meta-overfitting problem in few-task meta-learning. Specifically, we leveraged expressive neural set functions to interpolate a given set of tasks and trained the interpolating function using bilevel optimization, so that the meta-learner trained with the augmented tasks generalizes to meta-validation tasks. Since the set function is optimized to minimize the loss on the validation tasks, it allows us to tailor the task augmentation strategy to each specific domain. We empirically validated the efficacy of our proposed method on various domains, including image classification, chemical property prediction, text classification, and automatic speech recognition, showing that Meta-Interpolation achieves consistent improvements across all domains. This is in stark contrast to the baselines which improve generalization in certain domains but degenerate performance in others. Furthermore, our theoretical analysis shed light on how Meta-Interpolation regularizes the meta-learner and improves its generalization performance. Lastly, we discussed the limitation of our method in Appendix C.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. MetaReg: Towards domain generalization using meta-regularization. *Advances in neural information processing systems*, 31, 2018.

[3] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. Learning a synaptic learning rule. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 2, pages 969–vol. IEEE, 1991.

[4] Charles Blair. The computational complexity of multi-level linear programs. *Annals of Operations Research*, 34, 1992.

[5] Samuel Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, 2015.

[6] Kaidi Cao, Maria Brbic, and Jure Leskovec. Concept learners for few-shot learning. In *International Conference on Learning Representations*, 2021.

[7] Richard Li-Yang Chen, Amy Cohn, and Ali Pinar. An implicit optimization approach for survivable network design. In *2011 IEEE network science workshop*, pages 180–187. IEEE, 2011.

[8] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020.

[9] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[11] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. In *International Conference on Machine Learning*, pages 1920–1930. PMLR, 2019.

[12] Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent. In *International Conference on Learning Representations*, 2020.

[13] Karl Pearson F.R.S. Liii. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11): 559–572, 1901. doi: 10.1080/14786440109462720.

[14] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, 2019.

[15] Greg Landrum. RDKit: Open-source cheminformatics software., 2018. https://github.com/rdkit/rdkit.

[16] Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. CNN architectures for large-scale audio classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, pages 131–135. IEEE, 2017.

[17] Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor W Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Therapeutics data commons: machine learning datasets and tasks for therapeutics. *arXiv e-prints*, pages arXiv–2102, 2021.

[18] Ruili Huang, Menghang Xia, Dac-Trung Nguyen, Tongan Zhao, Srilatha Sakamuru, Jinghua Zhao, Sampada A Shahane, Anna Rossoshek, and Anton Simeonov. Tox21challenge to build predictive models of nuclear receptor and stress response pathways as mediated by exposure to environmental chemicals and drugs. *Frontiers in Environmental Science*, 3:85, 2016.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[20] Tushar Khot, Ashish Sabharwal, and Peter Clark. SciTail: A textual entailment dataset from science question answering. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[22] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. In *https://www.cs.toronto.edu/ kriz/cifar.html*. Citeseer, 2009.

[23] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019.

[24] Yoonho Lee and Seungjin Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *International Conference on Machine Learning*, pages 2927–2936. PMLR, 2018.

[25] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics.

[26] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sungju Hwang, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. In *International Conference on Learning Representations*, 2019.

[27] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR, 2020.

[28] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[29] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

[30] Shikhar Murty, Tatsunori B 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.

[31] NCI. NCI dataset, 2018. https://github.com/GRAND-Lab/graph_datasets.

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

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

[34] Karol J. Piczak. ESC: Dataset for Environmental Sound Classification. In *Proceedings of the 23rd Annual ACM Conference on Multimedia*, pages 1015–1018. ACM Press, 2015. ISBN 978-1-4503-3459-4. doi: 10.1145/2733373.2806390. URL http://dl.acm.org/citation.cfm?doid=2733373.2806390.

[35] Janarthanan Rajendran, Alexander Irpan, and Eric Jang. Meta-learning requires meta-augmentation. *Advances in Neural Information Processing Systems*, 33:5705–5715, 2020.

[36] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019.

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

[38] Victor Garcia Satorras and Joan Bruna Estrach. Few-shot learning with graph neural networks. In *International Conference on Learning Representations*, 2018.

[39] Jurgen Schmidhuber. Evolutionary principles in self-referential learning. *On learning how to learn: The meta-meta-... hook.) Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*, 1(2), 1987.

[40] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[42] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1199–1208, 2018.

[43] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[44] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447. PMLR, 2019.

[45] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[46] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

[47] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, 2018.

[48] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

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

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

[51] Mingzhang Yin, George Tucker, Mingyuan Zhou, Sergey Levine, and Chelsea Finn. Meta-learning without memorization. In *International Conference on Learning Representations*, 2020.

[52] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Checklist

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] Please see limitation from Section 6
   (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]
   (b) Did you include complete proofs of all theoretical results? [Yes]
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]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We run experiments 5 times with different random seeds and show the 95% confidence interval on Table 1 and 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)? [No]
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] We describe the datasets and cite the creators in Section 5.
   (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? [N/A]
   (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]

Appendix

A Organization

The appendix is organized as follows: In Section B, we provide proofs of the theorem and propositions in Section 4. We discucss the limitations of our method in Section C. In Section D, we provide detailed descriptions of the experimental setup used in the main paper together with the exact data splits for meta-training, meta-validation and meta-testing for all the datasets used. Additionally, we specify the exact architecture of the Prototypical Networks used for all the experiments and further describe the Set Transformer in detail in Section D.3. All hyperparameters are detailed in Section D.4. Finally, in Section E, we explore the effect of the number of tasks for interpolation on the generalization performance of Meta-Interpolation.

B Proofs

B.1 Proof of Theorem 1

Proof. Define $\phi \coloneqq \phi^l_{\theta_l} = f^l_{\theta_l} \circ \cdots \circ f^1_{\theta_1}$, and $g \coloneqq f^L_{\theta_L} \circ \cdots \circ f^{l+1}_{\theta_{l+1}}$. Define the dimensionality as $\phi(\mathbf{x}^s_{t,i}) \in \mathbb{R}^d$, and $g(\phi(\mathbf{x}^s_{t,i})) \in \mathbb{R}^D$. From the definition, since we use $\hat{f}_{\theta,\lambda}$ in both training and testing time, we have

$$\mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}_t) = -\frac{1}{n}\sum_{i=1}^{n} \log \frac{\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \mathbf{c}_{y^q_{t,i}}))}{\sum_{k'} \exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \mathbf{c}_{k'}))}$$

where

$$\mathbf{c}_k = \frac{1}{N_{t,k}} \sum_{(\mathbf{x}^s_{t,i}, y^s_{t,i}) \in \mathcal{D}^s_t} \mathbb{1}_{\{y^s_{t,i}=k\}} (g \circ \varphi_\lambda)(\{\phi(\mathbf{x}^s_{t,i})\})$$

$$N_{t,k} = \sum_{(\mathbf{x}^s_{t,i}, y^s_{t,i}) \in \mathcal{D}^s_t} \mathbb{1}_{\{y^s_{t,i}=k\}}$$

In the analysis of the loss functions, without the loss of generality, we can set the permutation σ_t on $\{1, \ldots, K\}$ to be the identity since the every combination can be realized by one permutation σ instead of the two permutations $\sigma_t, \sigma_{t'}$. Therefore, using the definition of the $\hat{\mathcal{T}}_{t,t'}$, we can write the corresponding loss by

$$\mathcal{L}_{\text{mix}}(\lambda, \theta, \hat{\mathcal{T}}_{t,t'}) = -\frac{1}{n}\sum_{i=1}^{n} \log \frac{\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \hat{\mathbf{c}}_{y^q_{t,i}}))}{\sum_{k'} \exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \hat{\mathbf{c}}_{k'}))}$$

where

$$\hat{\mathbf{c}}_k \coloneqq \frac{1}{|S_k|} \sum_{(\{\mathbf{x}^s_{t,i}, \mathbf{x}^s_{t',j}\}, k) \in S_k} (g \circ \varphi_\lambda)(\{\phi(\mathbf{x}^s_{t,i}), \phi(\mathbf{x}^s_{t',j})\})$$

$$S_k \coloneqq \{(\{\mathbf{x}^s_{t,i}, \mathbf{x}^s_{t',j}\}, k) \mid (\mathbf{x}^s_{t,i}, y^s_{t,i}) \in \mathcal{D}^s_t, y^s_{t,i} = k, (\mathbf{x}^s_{t',j}, y^s_{t',j}) \in \mathcal{D}^s_{t'}, y^s_{t',j} = \sigma(k)\}$$

This can be rewritten as:

$$\hat{\mathbf{c}}_k = \frac{1}{N_{t,t',k}} \sum_{(\mathbf{x}^s_{t',j}, y^s_{t',j}) \in \mathcal{D}^s_{t'}} \mathbb{1}_{\{y^s_{t',i}=\sigma(k)\}} \sum_{(\mathbf{x}^s_{t,i}, y^s_{t,i}) \in \mathcal{D}^s_t} \mathbb{1}_{\{y^s_{t,i}=k\}} (g \circ \varphi_\lambda)(\{\phi(\mathbf{x}^s_{t,i}), \phi(\mathbf{x}^s_{t',j})\})$$

$$\begin{aligned} N_{t,t',k} &= \sum_{(\mathbf{x}^s_{t',j}, y^s_{t',j}) \in \mathcal{D}^s_{t'}} \mathbb{1}_{\{y^s_{t',i}=\sigma(k)\}} \sum_{(\mathbf{x}^s_{t,i}, y^s_{t,i}) \in \mathcal{D}^s_t} \mathbb{1}_{\{y^s_{t,i}=k\}} \\ &= \left(\sum_{(\mathbf{x}^s_{t',j}, y^s_{t',j}) \in \mathcal{D}^s_{t'}} \mathbb{1}_{\{y^s_{t',i}=\sigma(k)\}} \right) \left(\sum_{(\mathbf{x}^s_{t,i}, y^s_{t,i}) \in \mathcal{D}^s_t} \mathbb{1}_{\{y^s_{t,i}=k\}} \right) \\ &= N_{t',k} N_{t,k} \end{aligned}$$

Thus,

$$\hat{\mathbf{c}}_k = \frac{1}{N_{t',k}} \sum_{(\mathbf{x}^s_{t',j}, y^s_{t',j}) \in \mathcal{D}^s_{t'}} \mathbb{1}_{\{y^s_{t',j} = \sigma(k)\}} \left(\frac{1}{N_{t,k}} \sum_{(\mathbf{x}^s_{t,i}, y^s_{t,i}) \in \mathcal{D}^s_t} \mathbb{1}_{\{y^s_{t,i} = k\}} (g \circ \varphi_\lambda)(\{\phi(\mathbf{x}^s_{t,i}), \phi(\mathbf{x}^s_{t',j})\}) \right).$$

Define the set

$$I_{t,k} \coloneqq \{i : y^s_{t,i} = k\}.$$

Then,

$$\hat{\mathbf{c}}_k = \frac{1}{|I_{t',\sigma(k)}|} \sum_{j \in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} (g \circ \varphi_\lambda)(\{\phi(\mathbf{x}^s_{t,i}), \phi(\mathbf{x}^s_{t',j})\}).$$

Summarizing the computation so far, we have that

$$\mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}_t) = -\frac{1}{n} \sum_{i=1}^{n} \log \frac{\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \mathbf{c}_{y^q_{t,i}}))}{\sum_{k'} \exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \mathbf{c}_{k'}))}$$

$$\mathbf{c}_k = \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} (g \circ \varphi_\lambda)(\{\phi(\mathbf{x}^s_{t,i})\})$$

and

$$\mathcal{L}_{\text{mix}}(\lambda, \theta, \hat{\mathcal{T}}_{t,t'}) = -\frac{1}{n} \sum_{i=1}^{n} \log \frac{\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \hat{\mathbf{c}}_{y^q_{t,i}}))}{\sum_{k'} \exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}), \hat{\mathbf{c}}_{k'}))}$$

$$\hat{\mathbf{c}}_k = \frac{1}{|I_{t',\sigma(k)}|} \sum_{j \in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} (g \circ \varphi_\lambda)(\{\phi(\mathbf{x}^s_{t,i}), \phi(\mathbf{x}^s_{t',j})\})$$

with

$$I_{t,k} = \{i : y^s_{t,i} = k\}.$$

We now analyze the relationship of $\varphi_\lambda(\{h, h'\})$ and $\varphi_\lambda(\{h\})$. From this definition, we first compute $\varphi_\lambda(\{h, h'\})$ as follows. By writing $\sigma_1 = \text{softmax}(\sqrt{d^{-1}} Q_1^{\{h,h'\}} (K_1^{\{h,h'\}})^\top)$ and $\sigma_2 = \text{softmax}(\sqrt{d^{-1}} Q_2 (K_2^{\{h,h'\}})^\top)$,

$$\begin{aligned}
\varphi_\lambda(\{h, h'\}) &= A(Q_2, K_2^{\{h,h'\}}, V_2^{\{h,h'\}}) \\
&= \text{softmax}(\sqrt{d^{-1}} Q_2 (K_2^{\{h,h'\}})^\top)(H_2^{\{h,h'\}} W_2^V + \mathbf{1}_2 b_2^V) \\
&= \text{softmax}(\sqrt{d^{-1}} Q_2 (K_2^{\{h,h'\}})^\top)(A(Q_1^{\{h,h'\}}, K_1^{\{h,h'\}}, V_1^{\{h,h'\}}) W_2^V + \mathbf{1}_2 b_2^V) \\
&= \sigma_2(\sigma_1(H^{\{h,h'\}} W_1^V + \mathbf{1}_2 b_1^V) W_2^V + \mathbf{1}_2 b_2^V) \\
&= \sigma_2 \sigma_1 H^{\{h,h'\}} W_1^V W_2^V + \sigma_2 \sigma_1 \mathbf{1}_2 b_1^V W_2^V + \sigma_2 \mathbf{1}_2 b_2^V.
\end{aligned}$$

Here, by writing $p_1 = p_1^{(t,t',i,j)}$, $p_1' = \tilde{p}_1^{(t,t',i,j)}$, and $p_2 = p_2^{(t,t',i,j)}$, we can rewrite that

$$\sigma_1 = \text{softmax}(\sqrt{d^{-1}} Q_1^{\{h,h'\}} (K_1^{\{h,h'\}})^\top) = \begin{bmatrix} p_1 & 1 - p_1 \\ p_1' & 1 - p_1' \end{bmatrix} \in \mathbb{R}^{2 \times 2}$$

$$\sigma_2 = \text{softmax}(\sqrt{d^{-1}} Q_2 (K_2^{\{h,h'\}})^\top) = [p_2 \quad 1 - p_2] \in \mathbb{R}^{1 \times 2}$$

Thus, by letting $\bar{p} = p_2 p_1 + (1 - p_2) p_1' \in [0, 1]$,

$$\sigma_2 \sigma_1 = [p_2 \quad 1 - p_2] \begin{bmatrix} p_1 & 1 - p_1 \\ p_1' & 1 - p_1' \end{bmatrix} = [p_2 p_1 + (1 - p_2) p_1' \quad p_2(1 - p_1) + (1 - p_2)(1 - p_1')] = [\bar{p} \quad 1 - \bar{p}].$$

Using these,

$$\sigma_2 \sigma_1 H^{\{h,h'\}} = [\bar{p} \quad 1 - \bar{p}] \begin{bmatrix} h^\top \\ (h')^\top \end{bmatrix} = \bar{p} h^\top + (1 - \bar{p})(h')^\top$$

$$\sigma_2 \sigma_1 \mathbf{1}_2 = [\bar{p} \quad 1 - \bar{p}] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1,$$

and

$$\sigma_2 \mathbf{1}_2 = [p_2 \quad 1-p_2] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1.$$

Thus, by letting $W = (W_1^V W_2^V)^\top \in \mathbb{R}^{d\times d}$ and $b = (b_1^V W_2^V + b_2^V)^\top \in \mathbb{R}^d$, and by defining $\alpha = 1 - \bar{p}$, we have that

$$\varphi_\lambda(\{h, h'\}) = W\left(h + \alpha(h' - h)\right) + b.$$

For $\varphi_\lambda(\{h\})$, similarly, by writing $\sigma_1 = \mathrm{softmax}(\sqrt{d^{-1}} Q_1^{\{h\}} (K_1^{\{h\}})^\top)$ and $\sigma_2 = \mathrm{softmax}(\sqrt{d^{-1}} Q_2 (K_2^{\{h\}})^\top)$,

$$\begin{aligned} \varphi_\lambda(\{h\}) &= A(Q_2, K_2^{\{h\}}, V_2^{\{h\}}) \\ &= \sigma_2\sigma_1 h^\top W_1^V W_2^V + \sigma_2\sigma_1 b_1^V W_2^V + \sigma_2 b_2^V. \end{aligned}$$

Here, since $\sigma_1 = \mathrm{softmax}(\sqrt{d^{-1}} Q_1^{\{h\}} (K_1^{\{h\}})^\top) \in \mathbb{R}$ and $\sigma_2 = \mathrm{softmax}(\sqrt{d^{-1}} Q_2 (K_2^{\{h\}})^\top) \in \mathbb{R}$, we have $\sigma_1 = \sigma_2 = 1$ and thus,

$$\varphi_\lambda(\{h\}) = Wh + b.$$

Therefore, we have that

$$\begin{aligned} &\varphi_\lambda(\{h, h'\}) = W\left(h + \alpha(h' - h)\right) + b, \\ &\varphi_\lambda(\{h\}) = Wh + b. \end{aligned}$$

where W and b does *not* depend on the (h, h'). Using these and by defining $h_{t,i} = \phi(\mathbf{x}_{t,i}^s)$, we have that

$$\mathbf{c}_k = \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} g(h_{t,i}^\top W + b) \in \mathbb{R}^D$$

$$\hat{\mathbf{c}}_k \coloneqq \frac{1}{|I_{t',\sigma(k)}|} \sum_{j \in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} g\left(W\left[h_{t,i} + \alpha_{ij}^{(t,t')}(h_{t',j} - h_{t,i})\right] + b\right) \in \mathbb{R}^D.$$

With these preparation, we are now ready to prove the regularization form. Fix t, t' and write $\alpha_{ij} = \alpha_{ij}^{(t,t')}$. Define a vector α such that $\alpha = (\alpha_{ij})_{i,j}$. Then,

$$\hat{c}_k(\alpha) \coloneqq \frac{1}{|I_{t',\sigma(k)}|} \sum_{j \in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} g\left(W\left[h_{t,i} + \alpha_{ij}(h_{t',j} - h_{t,i})\right] + b\right)$$

Then, from the results of the calculations above, we have that

$$\begin{aligned} \hat{c}_k(\alpha) &= \hat{\mathbf{c}}_k, \\ \hat{c}_k(0) &= \mathbf{c}_k. \end{aligned}$$

Using the assumptions that $\partial^r g(z) = 0$ for all $r \geq 2$, for any J, the J-th approximation of $\hat{c}_k(\alpha)$ is given by

$$\hat{c}_k(\alpha)_J = \hat{c}_k(0) + \frac{1}{|I_{t',\sigma(k)}|} \sum_{j \in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} \alpha_{ij} \partial g\left(W h_{t,i} + b\right) W(h_{t',j} - h_{t,i})$$

Let $\bar{\gamma} \geq 1$ to be set later and define

$$\Delta_k \coloneqq \frac{1}{|I_{t',\sigma(k)}|} \sum_{j \in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} \frac{\alpha_{ij}}{\bar{\gamma}} \partial g\left(W h_{t,i} + b\right) W(h_{t',j} - h_{t,i}).$$

Then,

$$\hat{c}_k(\alpha)_J = \hat{c}_k(0) + \bar{\gamma}\Delta_k.$$

Define $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_K)$ and $\Delta = (\Delta_1, \ldots, \Delta_K)$. For any given t, define

$$L_t(\mathbf{c}) \coloneqq -\frac{1}{n} \sum_{i=1}^n \log \frac{\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}_{t,i}^q), \mathbf{c}_{y_{t,i}^q}))}{\sum_{k'} \exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}_{t,i}^q), \mathbf{c}_{k'}))}.$$

Then, we have that

$$L_t(\mathbf{c}+\bar{\gamma}\Delta) = -\frac{1}{n}\sum_{i=1}^{n}\log\frac{\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}),(\mathbf{c}+\bar{\gamma}\Delta)_{y^q_{t,i}}))}{\sum_{k'}\exp(-d(\hat{f}_{\theta,\lambda}(\mathbf{x}^q_{t,i}),(\mathbf{c}+\bar{\gamma}\Delta)_{k'}))} = \mathcal{L}_{\text{mix}}(\lambda,\theta,\hat{\mathcal{T}}_{t,t'}),$$

$$L_t(\mathbf{c}) = \mathcal{L}_{\text{singleton}}\left(\lambda,\theta;\mathcal{T}_t\right).$$

The J-th approximation of $\mathcal{L}_{\text{mix}}(\lambda,\theta,\hat{\mathcal{T}}_{t,t'})$ is given by

$$\begin{aligned}\mathcal{L}_{\text{mix}}(\lambda,\theta,\hat{\mathcal{T}}_{t,t'})_J = L_t(\mathbf{c}+\bar{\gamma}\Delta)_J &= L_t(\mathbf{c}) + \sum_{j=1}^{J}\frac{1}{j!}\varphi^{(j)}(0)\\ &= \mathcal{L}_{\text{singleton}}\left(\lambda,\theta;\mathcal{T}_t\right) + \sum_{j=1}^{J}\frac{1}{j!}\varphi^{(j)}(0),\end{aligned}$$

where $\varphi^{(j)}$ is the j-th order derivative of $\varphi:\gamma\mapsto L_t(\mathbf{c}+\gamma\Delta)$. Here, for any $j\in\mathbb{N}^+$, by defining $b'=\mathbf{c}+\gamma\Delta\in\mathbb{R}^{KD}$,

$$\varphi^{(j)}(\gamma) = \sum_{i_1=1}^{KD}\sum_{i_2=1}^{KD}\cdots\sum_{i_j=1}^{KD}\frac{\partial^j L_t(b')}{\partial b'_{i_1}\partial b'_{i_2}\cdots\partial b'_{i_j}}\Delta_{i_1}\Delta_{i_2}\cdots\Delta_{i_j}.$$

Then, by using the vectorization of the tensor $\text{vec}[\partial^j L_t(b')]\in\mathbb{R}^{(KD)^j}$, we can rewrite this equation as

$$\varphi^{(j)}(\gamma) = \text{vec}[\partial^j L_t(\mathbf{c}+\gamma\Delta)]^\top\Delta^{\otimes j}, \tag{9}$$

where $\Delta^{\otimes j}=\Delta\otimes\Delta\otimes\cdots\otimes\Delta\in\mathbb{R}^{(KD)^j}$. By combining these with $\bar{\gamma}=1$,

$$\mathcal{L}_{\text{mix}}(\lambda,\theta,\hat{\mathcal{T}}_{t,t'})_J = \mathcal{L}_{\text{singleton}}\left(\lambda,\theta;\mathcal{T}_t\right) + \sum_{j=1}^{J}\frac{1}{j!}\,\text{vec}[\partial^j L_t(\mathbf{c})]^\top\Delta^{\otimes j},$$

where

$$\Delta_k = \frac{1}{|I_{t',\sigma(k)}|}\sum_{j\in I_{t',\sigma(k)}}\frac{1}{|I_{t,k}|}\sum_{i\in I_{t,k}}\alpha_{ij}^{(t,t')}\partial g\left(Wh_{t,i}+b\right)W(h_{t',j}-h_{t,i}).$$

□

B.2 Proof of Proposition 1

Proof. We now apply our general regularization form theorem to this special case from the previous paper [50] on the ProtoNet loss. In this special case, we have that $\phi(\mathbf{x})=\mathbf{x}$, $W=I, b=0, h_{t,i}=\mathbf{x}^s_{t,i}$, $g(\mathbf{x})=\mathbf{x}^\top\theta$, and

$$L_t(\mathbf{c}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{1+\exp(\langle\mathbf{x}^q_{t,i},\theta\rangle-\mathbf{c}_1/2-\mathbf{c}_2/2)}.$$

Here, for $\mathbf{c}=(\mathbf{c}_1,\mathbf{c}_2)$ with

$$\mathbf{c}_1 = \frac{1}{N_{t,1}}\sum_{(\mathbf{x}^s_{t,i},y^s_{t,i})\in\mathcal{D}^s_t}\mathbb{1}_{\{y^s_{t,i}=1\}}\varphi_\lambda(\{\mathbf{x}^s_{t,i}\})^\top\,\theta = \theta^\top\mathbf{c}'_1 \tag{10}$$

and

$$\mathbf{c}_2 = \frac{1}{N_{t,2}}\sum_{(\mathbf{x}^s_{t,i},y^s_{t,i})\in\mathcal{D}^s_t}\mathbb{1}_{\{y^s_{t,i}=2\}}\varphi_\lambda(\{\mathbf{x}^s_{t,i}\})^\top\,\theta = \theta^\top\mathbf{c}'_2, \tag{11}$$

we recover that

$$L_t(\mathbf{c}) = \mathcal{L}\left(\lambda,\theta;\mathcal{T}_t\right).$$

Thus, to instantiate our general theorem to this special case, we only need to compute the $\partial^j L_t(\mathbf{c})$ and $\partial g(h_{t,i}^\top W + b)$ up to the second order approximation.

$$\partial g(h_{t,i}^\top W + b) = \partial g(\mathbf{x}_{t,i}^s) = \theta$$

$$\partial^j g(h_{t,i}^\top W + b) = \partial^j g(\mathbf{x}_{t,i}^s) = 0 \qquad \forall j \geq 2$$

Define $z_{t,i} \coloneqq \langle \mathbf{x}_{t,i}^q - (\mathbf{c}_1' + \mathbf{c}_2')/2, \theta \rangle$. Since $\mathbf{c}_1 = \theta^\top \mathbf{c}_1'$ and $\mathbf{c}_2 = \theta^\top \mathbf{c}_2'$ by Equation 10 and 11,

$$\begin{aligned}
\langle \mathbf{x}_{t,i}^q - (\mathbf{c}_1' + \mathbf{c}_2')/2, \theta \rangle &= \langle \mathbf{x}_{t,i}^q, \theta \rangle - (\langle \mathbf{c}_1'/2, \theta \rangle + \langle \mathbf{c}_2'/2, \theta \rangle) \\
&= \langle \mathbf{x}_{t,i}^q, \theta \rangle - \theta^\top \mathbf{c}_1'/2 - \theta^\top \mathbf{c}_2'/2 \\
&= \langle \mathbf{x}_{t,i}^q, \theta \rangle - \mathbf{c}_1/2 - \mathbf{c}_2/2
\end{aligned}$$

Thus, we have $L_t(\mathbf{c}) = \frac{1}{n} \sum_{i=1}^n \frac{1}{1+\exp(z_{t,i})}$. Then,

$$\frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_1} = -\frac{1}{n} \sum_{i=1}^n [1 + \exp(z_{t,i})]^{-2} \exp(z_{t,i}) \frac{\partial z_{t,i}}{\partial \mathbf{c}_1} = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} [1 + \exp(z_{t,i})]^{-2} \exp(z_{t,i})$$

Similarly,

$$\frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_2} = -\frac{1}{n} \sum_{i=1}^n [1 + \exp(z_{t,i})]^{-2} \exp(z_{t,i}) \frac{\partial z_{t,i}}{\partial \mathbf{c}_2} = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} [1 + \exp(z_{t,i})]^{-2} \exp(z_{t,i})$$

For the second order, by defining the logistic function $\psi(z_{t,i}) = \frac{\exp(z_{t,i})}{1+\exp(z_{t,i})}$,

$$\begin{aligned}
\frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_1 \partial \mathbf{c}_1} &= \frac{1}{n} \sum_{i=1}^n \frac{-2}{2} [1 + \exp(z_{t,i})]^{-3} \exp(z_{t,i}) \frac{\partial \exp(z_{t,i})}{\partial \mathbf{c}_1} + \frac{1}{2} [1 + \exp(z_{t,i})]^{-2} \frac{\partial \exp(z_{t,i})}{\partial \mathbf{c}_1} \\
&= \frac{1}{n} \sum_{i=1}^n \frac{1}{2} [1 + \exp(z_{t,i})]^{-3} \exp(z_{t,i})^2 - \frac{1}{4} [1 + \exp(z_{t,i})]^{-2} \exp(z_{t,i}) \\
&= \frac{1}{n} \sum_{i=1}^n \frac{1}{2} [1 + \exp(z_{t,i})]^{-2} \exp(z_{t,i}) (\psi(z_{t,i}) - 0.5) \\
&= \frac{1}{n} \sum_{i=1}^n \frac{1}{2} \frac{\psi(z_{t,i})(\psi(z_{t,i}) - 0.5)}{1 + \exp(z_{t,i})}
\end{aligned}$$

Similarly,

$$\frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_2 \partial \mathbf{c}_2} = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} \frac{\psi(z_{t,i})(\psi(z_{t,i}) - 0.5)}{1 + \exp(z_{t,i})}$$

$$\frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_1 \partial \mathbf{c}_2} = \frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_2 \partial \mathbf{c}_1} = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} \frac{\psi(z_{t,i})(\psi(z_{t,i}) - 0.5)}{1 + \exp(z_{t,i})}$$

Therefore, the second approximation of $\mathbb{E}_{t',\sigma}[\mathcal{L}_{\text{mix}}(\lambda, \theta, \hat{\mathcal{T}}_{t,t'})]$ is

$$\begin{aligned}
&\mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}_t) + \mathbb{E}_{t',\sigma} \left[\sum_{j=1}^2 \frac{1}{j!} \operatorname{vec}[\partial^j L_t(\mathbf{c})]^\top \Delta^{\otimes j} \right] \\
&= \mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}_t) + \mathbb{E}_{t',\sigma} \left[\operatorname{vec}[\partial L_t(\mathbf{c})]^\top \Delta + \frac{1}{2} \operatorname{vec}[\partial^2 L_t(\mathbf{c})]^\top \Delta^{\otimes 2} \right]
\end{aligned}$$

where $\Delta = [\Delta_1^\top, \Delta_2^\top]^\top$ with

$$\begin{aligned}
\Delta_k &= \frac{1}{|I_{t',\sigma(k)}|} \sum_{j \in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} \alpha_{ij}^{(t,t')} (h_{t',j} - h_{t,i})^\top \theta \\
&= \theta^\top \delta_{t,t',\sigma,k}
\end{aligned}$$

where

$$\delta_{t,t',\sigma,k} \coloneqq \frac{1}{|I_{t',\sigma(k)}|} \sum_{j \in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} \alpha_{ij}^{(t,t')} (h_{t',j} - h_{t,i}).$$

For the first order term,

$$\begin{aligned}
\mathrm{vec}[\partial L_t(\mathbf{c})]^\top \Delta &= \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} [1 + \exp(z_{t,i})]^{-2} \exp(z_{t,i}) (\theta^\top \delta_{t,t',\sigma,k} + \theta^\top \delta_{t,t',\sigma,k}) \\
&= \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} [1 + \exp(z_{t,i})]^{-2} \right) \exp(z_{t,i}) \theta^\top \delta_{t,t',\sigma}
\end{aligned}$$

where

$$\delta_{t,t',\sigma} = \sum_{k=1}^{2} \frac{1}{|I_{t',\sigma(k)}|} \sum_{j \in I_{t',\sigma(k)}} \frac{1}{|I_{t,k}|} \sum_{i \in I_{t,k}} \alpha_{ij}^{(t,t')} (h_{t',j} - h_{t,i}).$$

For the second order term,

$$\begin{aligned}
\mathrm{vec}[\partial^2 L_t(\mathbf{c})]^\top \Delta^{\otimes 2} &= [\Delta^\top \otimes \Delta^\top] \ \mathrm{vec}[\partial^2 L_t(\mathbf{c})] \\
&= \Delta^\top \partial^2 L_t(\mathbf{c}) \Delta \\
&= \begin{bmatrix} \Delta_1^\top & \Delta_2^\top \end{bmatrix} \begin{bmatrix} \frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_1 \partial \mathbf{c}_1} & \frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_1 \partial \mathbf{c}_2} \\ \frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_2 \partial \mathbf{c}_1} & \frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_2 \partial \mathbf{c}_2} \end{bmatrix} \begin{bmatrix} \Delta_1 \\ \Delta_2 \end{bmatrix} \\
&= \Delta_1^2 \frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_1 \partial \mathbf{c}_1} + \Delta_2^2 \frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_2 \partial \mathbf{c}_2} + 2 \Delta_1 \Delta_2 \frac{\partial L_t(\mathbf{c})}{\partial \mathbf{c}_1 \partial \mathbf{c}_2} \\
&= \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \frac{\psi(z_{t,i})(\psi(z_{t,i}) - 0.5)}{1 + \exp(z_{t,i})} \right) \left(\Delta_1^2 + \Delta_2^2 + 2 \Delta_1 \Delta_2 \right) \\
&= \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \frac{\psi(z_{t,i})(\psi(z_{t,i}) - 0.5)}{1 + \exp(z_{t,i})} \right) (\Delta_1 + \Delta_2)^2 \\
&= \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \frac{\psi(z_{t,i})(\psi(z_{t,i}) - 0.5)}{1 + \exp(z_{t,i})} \right) \left(\theta^\top \delta_{t,t',\sigma} \right)^2
\end{aligned}$$

Since $\mathbb{E}_{t',\sigma}[\delta_{t,t',\sigma}] = 0$ with the α being balanced, the second approximation of $\mathbb{E}_{t',\sigma}[\mathcal{L}_{\text{mix}}(\lambda, \theta, \hat{\mathcal{T}}_{t,t'})]$ becomes:

$$\begin{aligned}
&\mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}_t) + \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{4} \frac{\psi(z_{t,i})(\psi(z_{t,i}) - 0.5)}{1 + \exp(z_{t,i})} \right) \mathbb{E}_{t',\sigma} \left[\left(\theta^\top \delta_{t,t',\sigma} \right)^2 \right]. \\
&= \mathcal{L}_{\text{singleton}}(\lambda, \theta; \mathcal{T}_t) + \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{4} \frac{\psi(z_{t,i})(\psi(z_{t,i}) - 0.5)}{1 + \exp(z_{t,i})} \right) \theta^\top \mathbb{E}_{t',\sigma} \left[\delta_{t,t',\sigma} \delta_{t,t',\sigma}^\top \right] \theta.
\end{aligned}$$

□

B.3 Proof of Proposition 2

Proof. Let $\xi_1, \ldots, \xi_n$ be independent uniform random variables taking values in $\{-1, 1\}$; i.e., Rademacher variables. We first bound the empirical Rademacher complexity as follows:

$$\begin{aligned}
\hat{\mathcal{R}}_n(\mathcal{F}_R) &= \mathbb{E}_\xi \sup_{f \in \mathcal{F}_R} \frac{1}{n} \sum_{i=1}^{n} \xi_i f(\mathbf{x}_i) \\
&= \mathbb{E}_\xi \sup_{\theta : \|\theta\|_\Sigma^2 \le R} \frac{1}{n} \sum_{i=1}^{n} \xi_i \theta^\top \mathbf{x}_i \\
&= \mathbb{E}_\xi \sup_{\theta : \|\theta\|_\Sigma^2 \le R} \mathbb{E}_{\mathbf{x}'} \frac{1}{n} \sum_{i=1}^{n} \xi_i \theta^\top (\mathbf{x}_i - \mathbf{x}')
\end{aligned}$$

$$
\begin{aligned}
&\leq \frac{1}{n}\mathbb{E}_{\xi} \sup_{\theta:\theta^\top\Sigma\theta\leq R} \|\Sigma^{1/2}\theta\|_2 \mathbb{E}_{\mathbf{x}'} \left\| \sum_{i=1}^{n} \xi_i \Sigma^{\dagger/2}(\mathbf{x}_i - \mathbf{x}') \right\|_2 \\
&\leq \frac{\sqrt{R}}{n}\mathbb{E}_{\mathbf{x}'}\mathbb{E}_{\xi}\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}\xi_i\xi_j(\Sigma^{\dagger/2}(\mathbf{x}_i-\mathbf{x}'))^\top(\Sigma^{\dagger/2}(\mathbf{x}_j-\mathbf{x}'))} \\
&\leq \frac{\sqrt{R}}{n}\sqrt{\mathbb{E}_{\mathbf{x}'}\mathbb{E}_{\xi}\sum_{i=1}^{n}\sum_{j=1}^{n}\xi_i\xi_j(\Sigma^{\dagger/2}(\mathbf{x}_i-\mathbf{x}'))^\top(\Sigma^{\dagger/2}(\mathbf{x}_j-\mathbf{x}'))} \\
&= \frac{\sqrt{R}}{n}\sqrt{\mathbb{E}_{\mathbf{x}'}\sum_{i=1}^{n}(\Sigma^{\dagger/2}(\mathbf{x}_i-\mathbf{x}'))^\top(\Sigma^{\dagger/2}(\mathbf{x}_i-\mathbf{x}'))} \\
&= \frac{\sqrt{R}}{n}\sqrt{\mathbb{E}_{\mathbf{x}'}\sum_{i=1}^{n}(\mathbf{x}_i-\mathbf{x}')^\top\Sigma^{\dagger}(\mathbf{x}_i-\mathbf{x}')}
\end{aligned}
$$

where $\Sigma^\dagger$ denotes the Moore–Penrose inverse of Σ. Using this bound on the empirical Rademacher complexity, we now bound the Rademacher complexity as follows:

$$
\begin{aligned}
\mathcal{R}_n(\mathcal{F}_R) = \mathbb{E}_{\mathbf{x}_1,\ldots,\mathbf{x}_n}\hat{\mathcal{R}}_n(\mathcal{F}_R) &\leq \mathbb{E}_{\mathbf{x}_1,\ldots,\mathbf{x}_n}\frac{\sqrt{R}}{n}\sqrt{\sum_{i=1}^{n}\mathbb{E}_{\mathbf{x}'}(\mathbf{x}_i-\mathbf{x}')^\top\Sigma^{\dagger}(\mathbf{x}_i-\mathbf{x}')} \\
&\leq \frac{\sqrt{R}}{n}\sqrt{\sum_{i=1}^{n}\mathbb{E}_{\mathbf{x}_i,\mathbf{x}'}(\mathbf{x}_i-\mathbf{x}')^\top\Sigma^{\dagger}(\mathbf{x}_i-\mathbf{x}')} \\
&= \frac{\sqrt{R}}{n}\sqrt{\sum_{i=1}^{n}\sum_{k,l}(\Sigma^{\dagger})_{kl}\mathbb{E}_{\mathbf{x}_i,\mathbf{x}'}(\mathbf{x}_i-\mathbf{x}')_k(\mathbf{x}_i-\mathbf{x}')_l} \\
&= \frac{\sqrt{R}}{n}\sqrt{\sum_{i=1}^{n}\sum_{k,l}(\Sigma^{\dagger})_{kl}(\Sigma)_{kl}} \\
&= \frac{\sqrt{R}}{n}\sqrt{\sum_{i=1}^{n}\mathrm{tr}(\Sigma\Sigma^{\dagger})} \\
&= \frac{\sqrt{R}}{n}\sqrt{\sum_{i=1}^{n}\mathrm{rank}(\Sigma)} \\
&= \frac{\sqrt{R}\sqrt{\mathrm{rank}(\Sigma)}}{\sqrt{n}}
\end{aligned}
$$

□

C Limitation

Although we have shown promising results in various domains, our method requires extra computation for bilevel optimization to estimate λ, the parameters of the set function φ_λ. Reducing the computational cost for bilevel optimization will be important for future work.

D Experimental Setup

D.1 Dataset Description

Metabolism We use the following split for the metabolism dataset:

$$\begin{aligned}\texttt{Meta-Train} &= \{\texttt{CYP1A2_Veith}, \texttt{CYP3A4_Veith}, \texttt{CYP2C9_Substrate_CarbonMangels}\}\\ \texttt{Meta-Validation} &= \{\texttt{CYP2D6_Veith}, \texttt{CYP2D6_Substrate_CarbonMangels}\}\\ \texttt{Meta-Test} &= \{\texttt{CYP2C19_Veith}, \texttt{CYP2C9_Veith}, \texttt{CYP3A4_Substrate_CarbonMangels}\}\end{aligned}$$

Following Yao et al. [50], we balance each subdataset by selecting 1000 samples from each subdataset. Each data sample is processed by extracting a 1024-bit fingerprint feature from the SMILES representation of each chemical compound using the RDKit [15] library.

Tox21 We use the following split for the metabolism dataset[1]:

$$\begin{aligned}\texttt{Meta-Train} &= \{\texttt{NR-AR}, \texttt{NR-AR-LBD}, \texttt{NR-AhR}, \texttt{NR-Aromatase}, \texttt{NR-ER}, \texttt{NR-ER-LBD}\}\\ \texttt{Meta-Validation} &= \{\texttt{NR-PPAR-gamma}, \texttt{SR-ARE}\}\\ \texttt{Meta-Test} &= \{\texttt{SR-ATAD5}, \texttt{SR-HSE}, \texttt{SR-MMP}, \texttt{SR-p53}\}\end{aligned}$$

NCI We download the dataset from the github repository [2] and use the following splits for meta-training, meta-validation, and meta-testing:

$$\begin{aligned}\texttt{Meta-Train} &= \{41, 47, 83, 109\}\\ \texttt{Meta-Validation} &= \{81, 145\}\\ \texttt{Meta-Test} &= \{1, 33, 123\}\end{aligned}$$

GLUE-SciTail We use Hugging Face Datasets library [25] to download MNLI, QNLI, SNLI, RTE, and SciTail datasets and tokenize it ELECTRA tokenizer with setting maximum length to 128. We list meta-train, meta-validation, and meta-test split as follows:

$$\begin{aligned}\texttt{Meta-Train} &= \{\texttt{MNLI}, \texttt{QNLI}\}\\ \texttt{Meta-Validation} &= \{\texttt{SNLI}, \texttt{RTE}\}\\ \texttt{Meta-Test} &= \{\texttt{SciTail}\}\end{aligned}$$

ESC-50 We download ESC-50 dataset [34] from the github repository[3] and use the meta-training, meta-validation, and meta-test split as follows:

Meta-train set:

$$\texttt{dog}, \texttt{rooster}, \texttt{pig}, \texttt{cow}, \texttt{frog}, \texttt{cat}, \texttt{hen}, \texttt{insects}, \texttt{sheep}, \texttt{crow},$$
$$\texttt{rain}, \texttt{sea_waves}, \texttt{crackling_fire}, \texttt{crickets}, \texttt{chirping_birds},$$
$$\texttt{water_drops}, \texttt{wind}, \texttt{pouring_water}, \texttt{toilet_flush}, \texttt{thunderstorm}$$

Meta-validation set:

$$\texttt{crying_baby}, \texttt{sneezing}, \texttt{clapping}, \texttt{breathing}, \texttt{coughing}, \texttt{footsteps},$$
$$\texttt{laughing}, \texttt{brushing_teeth}, \texttt{snoring}, \texttt{drinking_sipping}, \texttt{door_wood_knock},$$
$$\texttt{mouse_click}, \texttt{keyboard_typing}, \texttt{door_wood_creaks}, \texttt{can_opening}$$

Meta-test set:

$$\texttt{washing_machine}, \texttt{vacuum_cleaner}, \texttt{clock_alarm}, \texttt{clock_tick},$$
$$\texttt{glass_breaking}, \texttt{helicopter}, \texttt{chainsaw}, \texttt{siren},$$
$$\texttt{car_horn}, \texttt{engine}, \texttt{train}, \texttt{church_bells}, \texttt{airplane}, \texttt{fireworks}, \texttt{hand_saw}$$

[1] https://tdcommons.ai/single_pred_tasks/tox/#tox21

[2] https://github.com/GRAND-Lab/graph_datasets/tree/master/Graph_Repository

[3] https://github.com/karolpiczak/ESC-50

RMNIST Following Yao et al. [50], we create RMNIST dataset by changing the size, color, and angle of the images from the original MNIST dataset. To be specific, we merge training and test split of MNIST and randomly select 5,600 images for each class and split into 56 subdatasets where each class has 100 examples. We only choose 16/6/10 subdatasets for meta-train, meta-validation, and meta-test split, respectively and do not use the rest of them. Each subdatset with the corresponding composition of image transformations is considered a distinct task. Following are the specific splits.

Meta-train set:

$$(\texttt{red}, \texttt{full}, 90^\circ), (\texttt{indigo}, \texttt{full}, 0^\circ), (\texttt{blue}, \texttt{full}, 270^\circ), (\texttt{orange}, \texttt{half}, 270^\circ),$$
$$(\texttt{green}, \texttt{full}, 90^\circ), (\texttt{green}, \texttt{full}, 270^\circ), (\texttt{orange}, \texttt{full}, 180^\circ), (\texttt{red}, \texttt{full}, 180^\circ),$$
$$(\texttt{green}, \texttt{full}, 0^\circ), (\texttt{orange}, \texttt{full}, 0^\circ), (\texttt{violet}, \texttt{full}, 270^\circ), (\texttt{orange}, \texttt{half}, 90^\circ),$$
$$(\texttt{violet}, \texttt{half}, 180^\circ), (\texttt{orange}, \texttt{full}, 90^\circ), (\texttt{violet}, \texttt{full}, 180^\circ), (\texttt{blue}, \texttt{full}, 90^\circ)$$

Meta-validation set:

$$(\texttt{indigo}, \texttt{half}, 270^\circ), (\texttt{blue}, \texttt{full}, 0^\circ), (\texttt{yellow}, \texttt{half}, 180^\circ),$$
$$(\texttt{yellow}, \texttt{half}, 0^\circ), (\texttt{yellow}, \texttt{half}, 90^\circ), (\texttt{violet}, \texttt{half}, 0^\circ)$$

Meta-test set:

$$(\texttt{yellow}, \texttt{full}, 270^\circ), (\texttt{red}, \texttt{full}, 0^\circ), (\texttt{blue}, \texttt{half}, 270^\circ), (\texttt{blue}, \texttt{half}, 0^\circ), (\texttt{blue}, \texttt{half}, 180^\circ),$$
$$(\texttt{red}, \texttt{half}, 270^\circ), (\texttt{violet}, \texttt{full}, 90^\circ), (\texttt{blue}, \texttt{half}, 90^\circ), (\texttt{green}, \texttt{half}, 270^\circ), (\texttt{red}, \texttt{half}, 90^\circ)$$

Mini-ImageNet-S Following Yao et al. [50], we reduce the number of meta-training tasks by choosing subset of the original meta-training classes. We specify the classes used for meta-training tasks as follows:

$$\texttt{n03017168}, \texttt{n07697537}, \texttt{n02108915}, \texttt{n02113712}, \texttt{n02120079}, \texttt{n04509417},$$
$$\texttt{n02089867}, \texttt{n03888605}, \texttt{n04258138}, \texttt{n03347037}, \texttt{n02606052}, \texttt{n06794110}$$

For meta-validation and meta-testing classes, we use the same classes as the original Mini-ImageNet.

CIFAR-100-FS Similar to Mini-ImageNet-S, we choose 12/16/20 classes for meta-training, meta-validation, meta-test classes. Followings are the specific classes for each split:

$$\texttt{Meta-Train} = \{0, \ldots, 11\}$$
$$\texttt{Meta-Validation} = \{64, \ldots, 79\}$$
$$\texttt{Meta-Test} = \{80, \ldots, 99\}$$

D.2 Prototypical Network Architecture

We summarize neural network architectures for ProtoNet on each datasets in Table 4, 5, 6, 7, 8, and 9. For Glue-SciTail, we use pretrained ELECTRA-small [8] which we download from Hugging Face [48].

Table 4: Conv4 architecture for RMNIST.

Output Size	**Layers**
$3 \times 28 \times 28$	Input Image
$32 \times 14 \times 14$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
$32 \times 7 \times 7$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
$32 \times 3 \times 3$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
$32 \times 1 \times 1$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
32	Flatten

Table 5: Conv4 architecture for Mini-ImageNet-S.

Output Size	**Layers**
$3 \times 84 \times 84$	Input Image
$32 \times 42 \times 42$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
$32 \times 21 \times 21$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
$32 \times 10 \times 10$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
$32 \times 5 \times 5$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
800	Flatten

Table 6: Conv4 architecture for CIFAR-100-FS.

Output Size	**Layers**
$3 \times 32 \times 32$	Input Image
$32 \times 16 \times 16$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
$32 \times 8 \times 8$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
$32 \times 4 \times 4$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
$32 \times 2 \times 2$	conv2d(3×3, stride 1, padding 1), BatchNorm2D, ReLU, Maxpool(2×2, stride 2)
128	Flatten

Table 7: Fully connected networks for Metabolism, Tox21, and NCI.

Output Size	**Layers**
1024	Input SMILE
500	Linear(1024, 500, bias=True), BatchNorm1D, LeakyReLU
500	Linear(500, 500, bias=True), BatchNorm1D, LeakyReLU
500	Linear(500, 500, bias=True)

Table 8: Fully connected networks for ESC-50.

Output Size	**Layers**
650	Input VGGish feature
500	Linear(650, 500, bias=True), BatchNorm1D, LeakyReLU
500	Linear(500, 500, bias=True), BatchNorm1D, LeakyReLU
500	Linear(500, 500, bias=True)

Table 9: ELECTRA-small for GLUE-SciTail.

Output Size	**Layers**
128	Input sentence
128×256	ElectraModel("google/electra-small-discriminator")
256	[CLS] embedding
256	Linear(256, 256, bias=True), ReLU
256	Linear(256, 256, bias=True), ReLU
256	Linear(256, 256, bias=True)

D.3 Set Transformer

We describe Set Transformer [23], φ_λ, in more detail. Let $X \in \mathbb{R}^{n \times d}$ be a stack of n d-dimensional row vectors. Let $W^Q_{1,j}, W^K_{1,j}, W^V_{1,j} \in \mathbb{R}^{d \times d_k}$ be weight matrices for self-attention and let $b^Q_{1,j}, b^K_{1,j}, b^V_{1,j} \in \mathbb{R}^{d_k}$ be bias vectors for $j = 1, \ldots, 4$. For encoding an input X, we compute

self-attention as follows:

$$
\begin{aligned}
Q_1^{(j)} &= XW_{1,j}^Q + \mathbf{1}(b_{1,j}^Q)^\top \in \mathbb{R}^{n\times d_k} \\
K_1^{(j)} &= XW_{1,j}^Q + \mathbf{1}(b_{1,j}^K)^\top \in \mathbb{R}^{n\times d_k} \\
V_1^{(j)} &= XW_{1,j}^K + \mathbf{1}(b_{i,j}^V)^\top \in \mathbb{R}^{n\times d_k} \\
A_1^{(j)}(X) &= \text{LayerNorm}\left(Q_1^{(j)} + \text{softmax}\left(Q_1^{(j)}(K_1^{(j)})^\top/\sqrt{d_k}\right)V_1^{(j)}\right) \in \mathbb{R}^{n\times d_k} \\
O_1(X) &= \text{Concat}(A_1^{(1)}(X),\ldots,A_1^{(4)}(X)) \in \mathbb{R}^{n\times d_h}
\end{aligned}
\tag{12}
$$

where $\mathbf{1} = (1,\ldots,1)^\top \in \mathbb{R}^n$ is a vector of ones, $d_h = 4d_k$, and softmax is applied for each row. After self-attention, we add a skip connection with layer normalization [1] as follows:

$$
g_1(X) \coloneqq \left(\text{LayerNorm}\left(O_1(X)\right) + \text{ReLU}(W_1O_1(X) + \mathbf{1}b_1^\top)\right) \tag{13}
$$

where $W_1 \in \mathbb{R}^{d_h\times d_h}, b_1 \in \mathbb{R}^{d_h}$. Similarly, we compute another self-attention on top of $g_1(X)$ with $W_{2,j}^Q, W_{2,j}^K, W_{2,j}^V \in \mathbb{R}^{d_k\times d_k}$ weight matrices for self-attention and let $b_{2,j}^Q, b_{2,j}^K, b_{2,j}^V \in \mathbb{R}^{d_k}$ be bias vectors, for $j = 1\ldots,4$.

$$
\begin{aligned}
Q_2^{(j)} &= g_1(X)W_{2,j}^Q + \mathbf{1}(b_{2,j}^Q)^\top \in \mathbb{R}^{n\times d_k} \\
K_2^{(j)} &= g_1(X)W_{2,j}^Q + \mathbf{1}(b_{2,j}^K)^\top \in \mathbb{R}^{n\times d_k} \\
V_2^{(j)} &= g_1(X)W_{2,j}^K + \mathbf{1}(b_{2,j}^V)^\top \in \mathbb{R}^{n\times d_k} \\
A_2^{(j)}(g_1(X)) &= \text{LayerNorm}\left(Q_2^{(j)} + \text{softmax}\left(Q_2^{(j)}(K_2^{(j)})^\top/\sqrt{d_k}\right)V_2^{(j)}\right) \in \mathbb{R}^{n\times d_k} \\
O_2(g_1(X)) &= \text{Concat}(A_2^{(1)}(g_1(X)),\ldots,A_2^{(4)}(g_1(X)) \in \mathbb{R}^{n\times d_h}
\end{aligned}
\tag{14}
$$

After self-attention, we also add a skip connection after the second self-attention with dropout [41].

$$
(g_2\circ g_1)(X) \coloneqq \text{Dropout}(H_2(X)) \tag{15}
$$

$$
H_2(X) = \text{LayerNorm}(\left(O_2(g_1(X)) + \text{ReLU}(O_2(g_1(X))W_2 + \mathbf{1}b_2^\top)\right)) \tag{16}
$$

where $W_2 \in \mathbb{R}^{d_h\times d_h}, b_2 \in \mathbb{R}^{d_h}$.

After encoding X with two-layers of self-attention, we perform pooling with attention. Let $S \in \mathbb{R}^{1\times d_h}$ be learnable parameters and $W_{3,j}^Q, W_{3,j}^K, W_{3,j}^V \in \mathbb{R}^{d_h\times d_k}$ be weight matrices for the pooling-attention and let $b_{3,j}^Q, b_{3,j}^K, b_{3,j}^V \in \mathbb{R}^{d_k}$ be bias vectors, for $j = 1,\ldots,4$. We pool $(g_2\circ g_1)(X)$ as follows:

$$
\begin{aligned}
Q_3^{(j)} &= SW_{3,j}^Q + \mathbf{1}(b_{3,j}^Q)^\top \in \mathbb{R}^{1\times d_k} \\
K_3^{(j)} &= (g_2\circ g_1)(X)W_{3,j}^Q + \mathbf{1}(b_{3,j}^K)^\top \in \mathbb{R}^{n\times d_k} \\
V_3^{(j)} &= (g_2\circ g_1)(X)W_{3,j}^K + \mathbf{1}(b_{3,j}^V)^\top \in \mathbb{R}^{n\times d_k} \\
A_3^{(j)}((g_2\circ g_1)(X)) &= \text{LayerNorm}\left(Q_3^{(j)} + \text{softmax}\left(Q_3^{(j)}(K_3^{(j)})^\top/\sqrt{d_k}\right)V_3^{(j)}\right) \in \mathbb{R}^{1\times d_k} \\
O_3((g_2\circ g_1)(X)) &= \text{Concat}\left(A_3^{(1)}((g_2\circ g_1)(X)),\ldots,A_3^{(4)}((g_2\circ g_1)(X))\right) \in \mathbb{R}^{1\times d_h}
\end{aligned}
\tag{17}
$$

After pooling, we add another skip connection with dropout as follows:

$$
\begin{aligned}
(g_3\circ g_2\circ g_1)(X) &\coloneqq \text{Dropout}\left(H_3(X)\right) \\
H_3(X) &= \text{LayerNorm}\left(O_3\left((g_2\circ g_1)(X)\right) + \text{ReLU}(O_3((g_2\circ g_1)(X))W_3 + \mathbf{1}b_3^\top)\right)
\end{aligned}
\tag{18}
$$

where $W_3 \in \mathbb{R}^{d_h\times d_h}, b_3 \in \mathbb{R}^{d_h}$. Finally, we perform affine-transformation after the pooling as follows:

$$
(g_4\circ g_3\circ g_2\circ g_1)(X) \coloneqq \left((g_3\circ g_2\circ g_1)(X)W_4 + \mathbf{1}b_4^\top\right)^\top \tag{19}
$$

where $W_4 \in \mathbb{R}^{d_h\times d}, b_4 \in \mathbb{R}^d$.

To summarize, Set Transformer is the set function $\varphi_\lambda \coloneqq g_4\circ g_3\circ g_2\circ g_1 : \mathbb{R}^{n\times d} \to \mathbb{R}^d$ that can handle a set with arbitrary cardinality $n \in \mathbb{N}$.

D.4 Hyperparameters

In Table 10 and 11, we summarize all the hyperparameters for each datasets, where MI stands for Mini-ImageNet-S. For CIFAR-100-FS, we use the same hyperparameters for both 1-hot and 5-shot.

Table 10: Hyperparameters for non-image domains.

Hyperparameters	Metabolism	Tox21	NCI	GLUE-SciTail	ESC-50
learning rate α	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$3 \cdot 10^{-5}$	$1 \cdot 10^{-3}$
optimizer	Adam [21]	Adam	Adam	AdamW [28]	Adam
scheduler	none	none	none	linear	none
batch size	4	4	4	1	4
query size for meta-training	10	10	10	10	5
maximum training iterations	10,000	10,000	10,000	50,000	10,000
number of episodes for meta-test	3,000	3,000	3,000	3,000	3,000
hyper learning rate η	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$
hyper optimizer	Adam	Adam	Adam	Adam	Adam
hyper scheduler	linear	linear	linear	linear	linear
update period S	100	100	100	1,000	100
input size for set function d	500	500	500	256	500
hidden size for set function d_h	1,024	1,024	500	1,024	1,024
layer for interpolation l	1	1	1	2	2
iterations for Neumann series q	5	5	5	10	5
distance metric $d(\cdot,\cdot)$	Euclidean	Euclidean	Euclidean	Euclidean	Euclidean

Table 11: Hyperparameters for image domains.

Hyperparameters	RMNIST	MI (1-shot)	MI (5-shot)	CIFAR-100-FS (1-shot, 5-shot)
learning rate α	$1 \cdot 10^{-1}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
optimizer	SGD	Adam	Adam	Adam
scheduler	none	none	none	none
batch size	4	4	4	4
query size for meta-training	1	15	15	15
maximum training iterations	10,000	50,000	50,000	50,000
number of episodes for meta-test	3,000	3,000	3,000	3,000
hyper learning rate η	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$
hyper optimizer	Adam	Adam	Adam	Adam
hyper scheduler	linear	linear	linear	linear
update period S	1000	1,000	1,000	1,000
input size for set function d	1,568	500	500	256
hidden size for set function d_h	1,568	14,112	56,448	8,192
layer for interpolation l	2	2	1	1
iterations for Neumann series q	5	5	5	5
distance metric $d(\cdot,\cdot)$	Euclidean	Euclidean	Euclidean	Euclidean

E Effect of Cardinality for Interpolation

Since the set function φ_λ can handle sets of arbitrary cardinality $n \in \mathbb{N}$, we plot the test accuracy on the ESC-50 dataset with varying set sizes from one to five. As shown in Figure 4, for sets of cardinality one, where we do not perform any interpolation, we observe significant degradation in performance. This suggests that the interpolation of instances is crucial for better generalization. On the other hand, for set sizes from two to five, the gain is marginal with increasing cardinality. Furthermore, increasing the set size introduces extra computational overhead. Thus, we set the cardinality to two for task interpolation in all the experiments.

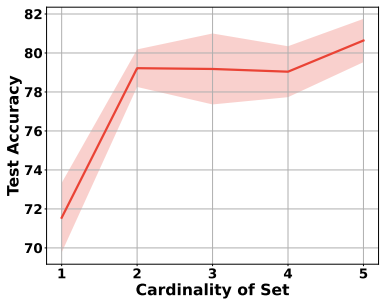


Figure 4: Acc. as a function of set size.

Table 12: Accuracy on ESC50 dataset as varying the number of meta-training tasks.

Model	5 Tasks	10 Tasks	15 Tasks	20 Tasks
ProtoNet	51.41 ± 3.93	60.63 ± 3.61	65.49 ± 2.05	69.05 ± 1.49
MLTI	58.98 ± 3.54	61.60 ± 2.04	66.29 ± 2.41	70.62 ± 1.96
Meta-Interpolation	$\mathbf{72.74 \pm 0.84}$	$\mathbf{74.78 \pm 1.43}$	$\mathbf{77.47 \pm 1.33}$	$\mathbf{79.22 \pm 0.96}$

F Additional Experiments

F.1 First-order MAML

As stated in introduction, we solely focus on metric based meta-learning due to the efficiency and better empirical performance over the gradient based methods. Moreover it is challenging to combine our method with Model-Agnostic Meta-Learning (MAML) [10] since it yields a tri-level optimization problem which requires to differentiate through second order derivatives and tri-level optimizati is still known to be challengin problem [4, 7]. Thus, instead of using the original MAML, we have performed additional experiments on ESC50 dataset with first-order MAML (FOMAML) which approximates the Hessian with a zero matrix. As shown in Table 13, the experimental results show that our Meta-Interpolation with the first-order MAML outperforms MLTI, which again confirms the general effectiveness of our set-based augmentation scheme. However, it largely underperforms our original Meta-Interpolation framework with metric-based meta-learning.

Table 13: First-order MAML (FOMAML) on ESC50.

Method	Accuracy
FOMAML	72.14 ± 0.73
FOMAML + MLTI	71.52 ± 0.55
FOMAML + Meta-Interpolation	76.68 ± 1.02
ProtoNet + Meta-Interpolation	79.22 ± 0.96

F.2 Effect of the number of meta-training tasks

We analyze the effect of the number of meta-training tasks on ESC50 dataset. As shown in Table 12, Meta-Interpolation consistently outperforms the baselines by large margins, regardless of the number of the tasks.

F.3 Location of interpolation

Contrary to Manifold Mixup, we fix the location of interpolation as shown in Table 10. Otherwise we cannot use the same architecture of Set Transformer to interpolate output of different layers since hidden dimension of each layer is different. Moreover, we report the test accuracy for each layer to be interpolated. Interpolating hidden representation of support sets from layer 2, which is our model used in the main experiments on ESC50 dataset, achieves the best performance.

Table 14: Accuracy for different location of interpolation on ESC-50.

Layer	Accuracy
Input Layer	66.83 ± 1.31
Layer 1	74.04 ± 2.05
Layer 2	$\mathbf{79.22 \pm 0.96}$
Layer 3	77.62 ± 1.46

F.4 Ablation for set functions

We have performed additional experiment on ESC50 dataset with different set function, DeepSets [52] to show the general effectiveness of our method regardless of the set encoding scheme. In Table 15, Meta-Interpolation with DeepSets improves the generalization performance of ProtoTypical Network and the model with Set Transformer further boost the performance as a consequence of higher-order pairwise interactions among the set elements via the attention mechanism.

Table 15: Performance of different set functions on ESC50.

Set Function	Accuracy
ProtoNet	69.05 ± 1.69
DeepSets	74.26 ± 1.77
Set Transformer	$\mathbf{79.22 \pm 0.96}$

F.5 Ablation for interpolation strategy

We empirically verify our interpolation strategy that mixes only support sets. We compare our method to various interpolation strategy including the one mixing a support set with zero mean and unit variance gaussian noise. In Table 16, we empirically show that the interpolation strategy which mixes only support sets outperforms the others. Note that interpolating a support set with gaussian noise works well on ESC50 dataset though we find that it significantly degrades the performance of Prototypical Network on RMNIST, from 75.35 ± 1.63 to 69.60 ± 1.60, which shows a necessity of interpolating two support sets.

Table 16: Performance of different interpolation on ESC50.

ESC50	
Interpolation Strategy	**Accuracy**
Query	78.19 ± 0.84
Query+Support	76.87 ± 0.94
Support	$\mathbf{79.22} \pm \mathbf{0.96}$
Support+ Noise	78.27 ± 1.24
RMNIST	
Interpolation Strategy	**Accuracy**
Support	$\mathbf{75.35} \pm \mathbf{1.63}$
Support+ Noise	69.60 ± 1.60