Few-Shot Learning via Embedding Adaptation with Set-to-Set Functions

Han-Jia Ye[*]
Nanjing University
yehj@lamda.nju.edu.cn

Hexiang Hu
USC
hexiangh@usc.edu

De-Chuan Zhan
Nanjing University
zhandc@lamda.nju.edu.cn

Fei Sha[†]
USC & Google
fsha@google.com

Abstract

Learning with limited data is a key challenge for visual recognition. Many few-shot learning methods address this challenge by learning an instance embedding function from seen classes and apply the function to instances from unseen classes with limited labels. This style of transfer learning is task-agnostic: the embedding function is not learned optimally discriminative with respect to the unseen classes, where discerning among them leads to the target task. In this paper, we propose a novel approach to adapt the instance embeddings to the target classification task with a set-to-set *function, yielding embeddings that are task-specific and are discriminative. We empirically investigated various instantiations of such set-to-set functions and observed the Transformer is most effective — as it naturally satisfies key properties of our desired model. We denote this model as* FEAT *(few-shot embedding adaptation w/ Transformer) and validate it on both the standard few-shot classification benchmark and four extended few-shot learning settings with essential use cases,* i.e.*, cross-domain, transductive, generalized few-shot learning, and low-shot learning. It archived consistent improvements over baseline models as well as previous methods, and established the new state-of-the-art results on two benchmarks.*

1. Introduction

Few-shot visual recognition [10, 23, 24, 27, 49] emerged as a promising direction in tackling the challenge of learning new visual concepts with limited annotations. Concretely, it distinguishes two sets of visual concepts: SEEN and UNSEEN ones. The target task is to construct visual classifiers to identify classes from the UNSEEN where each class has a very small number of exemplars ("few-shot"). The main idea is to discover transferable visual knowledge in the SEEN classes, which have ample labeled instances, and leverage it to construct the desired classifier. For example, state-of-the-art approaches for few-shot learn-

ing [40, 43, 46, 49] usually learn a discriminative instance embedding model on the SEEN categories, and apply it to visual data in UNSEEN categories. In this common embedding space, non-parametric classifiers (*e.g.*, nearest neighbors) are then used to avoid learning complicated recognition models from a small number of examples.

Such approaches suffer from one important limitation. Assuming a common embedding space implies that the discovered knowledge – discriminative visual features – on the SEEN classes are equally effective for *any* classification tasks constructed for an arbitrary set of UNSEEN classes. In concrete words, suppose we have two different target tasks: discerning "cat" versus "dog" and discerning "cat" versus "tiger". Intuitively, each task uses a different set of discriminative features. Thus, the most desired embedding model first needs to be able to extract discerning features for either task at the same time. This could be a challenging aspect in its own right as the current approaches are agnostic to what those "downstream" target tasks are and could accidentally de-emphasize selecting features for future use. Secondly, even if both sets of discriminative features are extracted, they do not necessarily lead to the optimal performance for a *specific* target task. The most useful features for discerning "cat" versus "tiger" could be irrelevant and noise to the task of discerning "cat" versus "dog"!

What is missing from the current few-shot learning approaches is an *adaptation* strategy that tailors the visual knowledge extracted from the SEEN classes to the UNSEEN ones in a target task. In other words, we desire separate embedding spaces where each one of them is customized such that the visual features are most discriminative for a given task. Towards this, we propose a few-shot model-based embedding adaptation method that adjusts the instance embedding models derived from the SEEN classes. Such model-based embedding adaptation requires a *set-to-set function*: a function mapping that takes all instances from the few-shot support set and outputs the set of adapted support instance embeddings, with elements in the set co-adapting with each other. Such output embeddings are then assembled as the prototypes for each visual category and serve as the nearest neighbor classifiers. Figure 1 qualitatively illustrates

---

[*]Work mostly done when the author was a visiting scholar at USC.
[†]On leave from USC

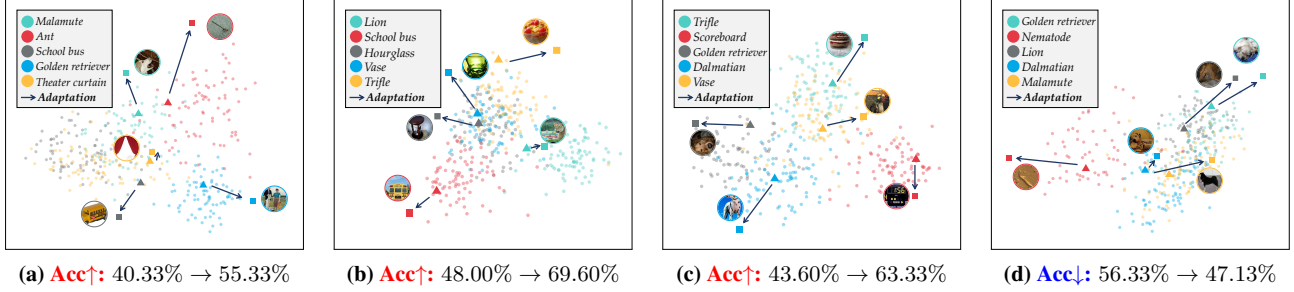| (a) **Acc↑:** 40.33% → 55.33% | (b) **Acc↑:** 48.00% → 69.60% | (c) **Acc↑:** 43.60% → 63.33% | (d) **Acc↓:** 56.33% → 47.13% |

Figure 1: Qualitative visualization of model-based embedding adaptation procedure (implemented using FEAT) on test tasks (refer to § 5.2.2 for more details). Each figure shows the locations of PCA projected support embeddings (class prototypes) before and after the adaptation of FEAT. Values below are the 1-shot 5-way classification accuracy before and after the the adaptation. Interestingly, the embedding adaptation step of FEAT pushes the support embeddings apart from the clutter and toward their own clusters, such that they can better fits the test data of its categories. (Best view in colors!)

the embedding adaptation procedure (as results of our best model). These class prototypes spread out in the embedding space toward the samples cluster of each category, indicating the effectiveness of embedding adaptation.

In this paper, we implement the set-to-set transformation using a variety of function approximators, including bidirectional LSTM [16] (Bi-LSTM), deep sets [56], graph convolutional network (GCN) [21], and Transformer [29, 47]. Our experimental results (refer to § 5.2.1) suggest that Transformer is the most parameter efficient choice that at the same time best implements the key properties of the desired set-to-set transformation, including *contextualization*, *permutation invariance*, *interpolation* and *extrapolation* capabilities (see § 4.1). As a consequence, we choose the set-to-set function instantiated with Transformer to be our final model and denote it as FEAT (**F**ew-shot **E**mbedding **A**daptation with **T**ransformer). We further conduct comprehensive analysis on FEAT and evaluate it on many extended tasks, including few-shot domain generalization, transductive few-shot learning, and generalized few-shot learning. Our overall contribution is three-fold.

- We formulate the few-shot learning as a model-based embedding adaptation to make instance embeddings task-specific, via using a set-to-set transformation.

- We instantiate such set-to-set transformation with various function approximators, validating and analyzing their few-shot learning ability, task interpolation ability, and extrapolation ability, *etc*. It concludes our model (FEAT) that uses the Transformer as the set-to-set function.

- We evaluate our FEAT model on a variety of extended few-shot learning tasks, where it achieves superior performances compared with strong baseline approaches.

2. Related Work

Methods specifically designed for few-shot learning fall broadly into two categories. The first is to control how a classifier for the target task should be constructed. One fruitful idea is the meta-learning framework where the classifiers are optimized *in anticipation* that a future update due to data from a new task performs well on that task [2, 3, 10, 13, 26, 32, 36, 40], or the classifier itself is directly meta-predicted by the new task data [35, 53].

Another line of approach has focused on learning generalizable instance embeddings [1, 5, 6, 17, 22, 31, 42, 46, 49] and uses those embeddings on simple classifiers such as nearest neighbor rules. The key assumption is that the embeddings capture all necessarily discriminative representations of data such that simple classifiers are sufficed, hence avoiding the danger of overfitting on a small number of labeled instances. Early work such as [22] first validated the importance of embedding in one-shot learning, whilst [49] proposes to learn the embedding with a soft nearest neighbor objective, following a meta-learning routine. Recent advances have leveraged different objective functions for learning such embedding models, *e.g.*, considering the class prototypes [43], decision ranking [46], and similarity comparison [45]. Most recently, [41] utilizes the graph convolution network [21] to unify the embedding learning.

Our work follows the second school of thoughts. The main difference is that we do not assume the embeddings learned on SEEN classes, being agnostic to the target tasks, are necessarily discriminative for those tasks. In contrast, we propose to *adapt* those embeddings for each target task *with a set-to-set function* so that the transformed embeddings are better aligned with the discrimination needed in those tasks. We show empirically that such task-specific embeddings perform better than task-agnostic ones. MetaOptNet [25] and CTM [28] follow the same spirit of learning task-specific embedding (or classifiers) via either explicitly optimization of target task or using concentrator and projector to make distance metric task-specific.
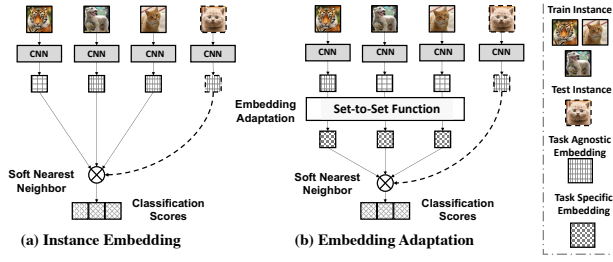
Figure 2: Illustration of the proposed **F**ew-Shot **E**mbedding **A**daptation **T**ransformer (FEAT). Existing methods usually use the same embedding function $\mathbf{E}$ for all tasks. We propose to adapt the embeddings to each target few-shot learning task with a set-to-set function such as Transformer, BiLSTM, DeepSets, and GCN.

3. Learning Embedding for Task-agnostic FSL

In the standard formulation of few-shot learning (FSL) [10, 49], a task is represented as a M-shot N-way classification problem with N classes sampled from a set of visual concepts $\mathcal{U}$ and M (training/support) examples per class. We denote the training set (also referred as support sets in the literature) as $\mathcal{D}_{\mathbf{train}} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{NM}$, with the instance $\mathbf{x}_i \in \mathbb{R}^D$ and the one-hot labeling vector $\mathbf{y}_i \in \{0,1\}^N$. We will use "support set" and "training set" interchangeably in the paper. In FSL, M is often small (*e.g.*, $M = 1$ or $M = 5$). The goal is to find a function f that classifies a test instance $\mathbf{x}_{\mathbf{test}}$ by $\hat{\mathbf{y}}_{\mathbf{test}} = f(\mathbf{x}_{\mathbf{test}}; \mathcal{D}_{\mathbf{train}}) \in \{0,1\}^N$.

Given a small number of training instances, it is challenging to construct complex classifiers $f(\cdot)$. To this end, the learning algorithm is also supplied with additional data consisting of ample labeled instances. These additional data are drawn from visual classes $\mathcal{S}$, which does not overlap with $\mathcal{U}$. We refer to the original task as *the target task* which discerns N UNSEEN classes $\mathcal{U}$. To avoid confusion, we denote the data from the SEEN classes $\mathcal{S}$ as $\mathcal{D}^{\mathcal{S}}$.

To learn $f(\cdot)$ using $\mathcal{D}^{\mathcal{S}}$, we synthesize many M-shot N-way FSL tasks by sampling the data in the meta-learning manner [10, 49]. Each sampling gives rise to a task to classify a test set instance $\mathbf{x}_{\mathbf{test}}^{\mathcal{S}}$ into one of the N SEEN classes by $f(\cdot)$, where the test instances set $\mathcal{D}_{\mathbf{test}}^{\mathcal{S}}$ is composed of the labeled instances with the same distribution as $\mathcal{D}_{\mathbf{train}}^{\mathcal{S}}$. Formally, the function $f(\cdot)$ is learnt to minimize the averaged error over those sampled tasks

$$f^* = \arg\min_f \sum_{(\mathbf{x}_{\mathbf{test}}^{\mathcal{S}}, \mathbf{y}_{\mathbf{test}}^{\mathcal{S}}) \in \mathcal{D}_{\mathbf{test}}^{\mathcal{S}}} \ell(f(\mathbf{x}_{\mathbf{test}}^{\mathcal{S}}; \mathcal{D}_{\mathbf{train}}^{\mathcal{S}}), \mathbf{y}_{\mathbf{test}}^{\mathcal{S}}) \tag{1}$$

where the loss $\ell(\cdot)$ measures the discrepancy between the prediction and the true label. For simplicity, we have assumed we only synthesize one task with test set $\mathcal{D}_{\mathbf{test}}^{\mathcal{S}}$. The optimal f^* is then applied to the original target task.

We consider the approach based on learning embeddings

Algorithm 1 Training strategy of embedding adaptation

Require: Seen class set $\mathcal{S}$
1: **for all** iteration = 1,...,MaxIteration **do**
2:     Sample N-way M-shot $(\mathcal{D}_{\mathbf{train}}^{\mathcal{S}}, \mathcal{D}_{\mathbf{test}}^{\mathcal{S}})$ from $\mathcal{S}$
3:     Compute $\phi_{\mathbf{x}} = \mathbf{E}(\mathbf{x})$, for $\mathbf{x} \in \mathcal{X}_{\mathbf{train}}^{\mathcal{S}} \cup \mathcal{X}_{\mathbf{test}}^{\mathcal{S}}$
4:     **for all** $(\mathbf{x}_{\mathbf{test}}^{\mathcal{S}}, \mathbf{y}_{\mathbf{test}}^{\mathcal{S}}) \in \mathcal{D}_{\mathbf{test}}^{\mathcal{S}}$ **do**
5:         Compute $\{\psi_{\mathbf{x}}; \forall \mathbf{x} \in \mathcal{X}_{\mathbf{train}}^{\mathcal{S}}\}$ with $\mathbf{T}$ via Eq. 3
6:         Predict $\hat{\mathbf{y}}_{\mathbf{test}}^{\mathcal{S}}$ with $\{\psi_{\mathbf{x}}\}$ as Eq. 4
7:         Compute $\ell(\hat{\mathbf{y}}_{\mathbf{test}}^{\mathcal{S}}, \mathbf{y}_{\mathbf{test}}^{\mathcal{S}})$ with Eq. 1
8:     **end for**
9:     Compute $\nabla_{\mathbf{E}, \mathbf{T}} \sum_{(\mathbf{x}_{\mathbf{test}}^{\mathcal{S}}, \mathbf{y}_{\mathbf{test}}^{\mathcal{S}}) \in \mathcal{D}_{\mathbf{test}}^{\mathcal{S}}} \ell(\hat{\mathbf{y}}_{\mathbf{test}}^{\mathcal{S}}, \mathbf{y}_{\mathbf{test}}^{\mathcal{S}})$
10:     Update $\mathbf{E}$ and $\mathbf{T}$ with $\nabla_{\mathbf{E}, \mathbf{T}}$ use SGD
11: **end for**
12: **return** Embedding function $\mathbf{E}$ and set function $\mathbf{T}$.

for FSL [43, 49] (see Figure 2 (a) for an overview). In particular, the classifier $f(\cdot)$ is composed of two elements. The first is an embedding function $\phi_{\mathbf{x}} = \mathbf{E}(\mathbf{x}) \in \mathbb{R}^d$ that maps an instance $\mathbf{x}$ to a representation space. The second component applies the nearest neighbor classifiers in this space:

$$\hat{\mathbf{y}}_{\mathbf{test}} = f(\phi_{\mathbf{x}_{\mathbf{test}}}; \{\phi_{\mathbf{x}}, \forall(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\mathbf{train}}\}) \tag{2}$$
$$\propto \exp\left(\mathbf{sim}(\phi_{\mathbf{x}_{\mathbf{test}}}, \phi_{\mathbf{x}})\right) \cdot \mathbf{y}, \forall(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\mathbf{train}}$$

Note that only the embedding function is learned by optimizing the loss in Eq. 1. For reasons to be made clear in below, we refer this embedding function as *task-agnostic*.

4. Adapting Embedding for Task-specific FSL

In what follows, we describe our approach for few-shot learning (FSL). We start by describing the main idea (§ 4.1, also illustrated in Figure 2), then introduce the set-to-set adaptation function (§ 4.2). Last are learning (§ 4.3) and implementations details (§ 4.4).

4.1. Adapting to Task-Specific Embeddings

The key difference between our approach and traditional ones is to learn *task-specific* embeddings. We argue that the embedding $\phi_{\mathbf{x}}$ is not ideal. In particular, the embeddings do not necessarily highlight the most discriminative representation for a specific target task. To this end, we introduce an adaption step where the embedding function $\phi_{\mathbf{x}}$ (more precisely, its values on instances) is transformed. This transformation is a *set-to-set* function that *contextualizes* over the image instances of a set, to enable strong co-adaptation of each item. Instance functions fails to have such co-adaptation property. Furthermore, the set-to-set-function receives instances as bags, or sets without orders, requiring the function to output the set of refined instance embeddings

3

while being *permutation-invariant*. Concretely,

$$\{\psi_{\mathbf{x}} \; ; \; \forall \mathbf{x} \in \mathcal{X}_{\mathbf{train}}\} = \mathbf{T}\left(\{\phi_{\mathbf{x}} \; ; \; \forall \mathbf{x} \in \mathcal{X}_{\mathbf{train}}\}\right) \quad (3)$$
$$= \mathbf{T}\left(\pi\{\phi_{\mathbf{x}} \; ; \; \forall \mathbf{x} \in \mathcal{X}_{\mathbf{train}}\}\right)$$

where $\mathcal{X}_{\mathbf{train}}$ is a set of all the instances in the training set $\mathcal{D}_{\mathbf{train}}$ for the target task. $\pi(\cdot)$ is a permutation operator over a set. Thus the set of *adapted* embedding will not change if we apply a permutation over the input embedding set. With *adapted* embedding $\psi_{\mathbf{x}}$, the test instance $\mathbf{x}_{\mathbf{test}}$ can be classified by computing nearest neighbors w.r.t. $\mathcal{D}_{\mathbf{train}}$:

$$\hat{\mathbf{y}}_{\mathbf{test}} = f(\phi_{\mathbf{x}_{\mathbf{test}}}; \{\psi_{\mathbf{x}}, \forall(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\mathbf{train}}\}) \quad (4)$$

Our approach is generally applicable to different types of task-agnostic embedding function $\mathbf{E}$ and similarity measure $\mathbf{sim}(\cdot, \cdot)$, *e.g.*, the (normalized) cosine similarity [49] or the negative distance [43]. Both the embedding function $\mathbf{E}$ and the set transformation function $\mathbf{T}$ are optimized over synthesized FSL tasks sampled from $\mathcal{D}^{\mathcal{S}}$, sketched in Alg. 1. Its key difference from conventional FSL is in the *line 4 to line 8* where the embeddings are transformed.

4.2. Embedding Adaptation via Set-to-set Functions

Next, we explain various choices as the instantiations of the set-to-set embedding adaptation function.

Bidirectional LSTM (BILSTM) [16, 49] is one of the common choice to instantiate the set-to-set transformation, where the addition between the input and the hidden layer outputs of each BILSTM cell leads to the adapted embedding. It is notable that the output of the BILSTM suppose to depend on the order of the input set. Note that using BILSTM as embedding adaptation model is similar but different from the fully conditional embedding [49], where the later one contextualizes both training and test instance embedding altogether, which results in a transductive setting.

DeepSets [56] is inherently a permutation-invariant transformation function. It is worth noting that DEEPSETS aggregates the instances in a set into a holistic *set vector*. We consider two components to implement such DeepSets transformation, an instance centric *set vector* combined with a set context vector. For $\mathbf{x} \in \mathcal{X}_{\mathbf{train}}$, we define its complementary set as $\mathbf{x}^{\complement}$. Then we implement the DEEPSETS by:

$$\psi_{\mathbf{x}} = \phi_{\mathbf{x}} + g([\phi_{\mathbf{x}}; \sum_{\mathbf{x}_{i'} \in \mathbf{x}^{\complement}} h(\phi_{\mathbf{x}_{i'}})]) \quad (5)$$

In Eq. 5, g and h are two-layer multi-layer perception (MLP) with ReLU activation which map the embedding into another space and increase the representation ability of the embedding. For each instance, embeddings in its complementary set is first combined into a *set vector* as the context, and then this vector is concatenated with the input

embedding to obtain the residual component of adapted embedding. This conditioned embedding takes other instances in the set into consideration, and keeps the "set (permutation invariant)" property. In practice, we find using the maximum operator in Eq. 5 works better than the sum operator suggested in [56].

Graph Convolutional Networks (GCN) [21, 41] propagate the relationship between instances in the set. We first construct the degree matrix A to represent the similarity between instances in a set. If two instances come from the same class, then we set the corresponding element in A to 1, otherwise to 0. Based on A, we build the "normalized" adjacency matrix S for a given set with added self-loops $S = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$. I is the identity matrix, and D is the diagonal matrix whose elements are equal to the sum of elements in the corresponding row of $A + I$.

Let $\Phi^0 = \{\phi_{\mathbf{x}} \; ; \; \forall \mathbf{x} \in \mathcal{X}_{\mathbf{train}}\}$, the relationship between instances could be propagated based on S, *i.e.*,

$$\Phi^{t+1} = \mathbf{ReLU}(S\Phi^t W) \; , \; t = 0, 1, \dots, T-1 \quad (6)$$

W is a projection matrix for feature transformation. In GCN, the embedding in the set is transformed based on Eq. 6 multiple times, and the final Φ^T gives rise to the $\{\psi_{\mathbf{x}}\}$.

Transformer. [47] We use the *Transformer* architecture [47] to implement $\mathbf{T}$. In particular, we employ self-attention mechanism [29, 47] to transform each instance embedding with consideration to its contextual instances. Note that it naturally satisfies the desired properties of $\mathbf{T}$ because it outputs refined instance embeddings and is permutation invariant. We denote it as **F**ew-Shot **E**mbedding **A**daptation with **T**ransformer (FEAT).

Transformer is a store of triplets in the form of (query $\mathcal{Q}$, key $\mathcal{K}$, and value $\mathcal{V}$). To compute proximity and return values, those points are first linearly mapped into some space $K = W_K^\top \left[\phi_{\mathbf{x}_k}; \forall \mathbf{x}_k \in \mathcal{K} \right] \in \mathbb{R}^{d \times |\mathcal{K}|}$, which is also the same for $\mathcal{Q}$ and $\mathcal{V}$ with W_Q and W_V respectively. Transformer computes what is the right value for a query point — the query $\mathbf{x}_q \in \mathcal{Q}$ is first matched against a list of keys K where each key has a value V. The final value is then returned as the sum of all the values *weighted* by the proximity of the key to the query point, *i.e.* $\psi_{\mathbf{x}_q} = \phi_{\mathbf{x}_q} + \sum_k \alpha_{qk} V_{:,k}$, where

$$\alpha_{qk} \propto \exp\left(\frac{\phi_{\mathbf{x}_q}^\top W_Q \cdot K}{\sqrt{d}}\right)$$

and $V_{:,k}$ is the k-th column of V. In the standard FSL setup, we have $\mathcal{Q} = \mathcal{K} = \mathcal{V} = \mathcal{X}_{\mathbf{train}}$.

4.3. Contrastive Learning of Set-to-Set Functions

To facilitate the learning of embedding adaptation, we apply a contrastive objective in addition to the general one.

It is designed to make sure that instances embeddings *after adaptation* is similar to the same class neighbors and dissimilar to those from different classes. Specifically, the embedding adaptation function $\mathbf{T}$ is applied to instances of each n of the N class in $\mathcal{D}_{\mathbf{train}}^{\mathcal{S}} \cup \mathcal{D}_{\mathbf{test}}^{\mathcal{S}}$, which gives rise to the transformed embedding $\psi_{\mathbf{x}}'$ and class centers $\{\mathbf{c}_n\}_{n=1}^{N}$. Then we apply the contrastive objective to make sure training instances are close to its own class center than other centers. The total objective function (together with Eq. 1) is shown as following:

$$\mathcal{L}(\hat{\mathbf{y}}_{\mathbf{test}}, \mathbf{y}_{\mathbf{test}}) = \ell(\hat{\mathbf{y}}_{\mathbf{test}}, \mathbf{y}_{\mathbf{test}}) \tag{7}$$
$$+ \lambda \cdot \ell\big(\mathbf{softmax}\left(\mathbf{sim}(\psi_{\mathbf{x}_{\mathbf{test}}}', \mathbf{c}_n)\right), \mathbf{y}_{\mathbf{test}}\big)$$

This contrastive learning makes the set transformation extract common characteristic for instances of the same category, so as to preserve the category-wise similarity.

4.4. Implementation details

We consider three different types of convolutional networks as the backbone for instance embedding function $\mathbf{E}$: 1) A 4-layer convolution network (ConvNet) [43, 46, 49] and 2) the 12-layer residual network (ResNet) used in [25], and 3) the Wide Residual Network (WideResNet) [40, 55]. We apply an additional pre-training stage for the backbones over the SEEN classes, based on which our re-implemented methods are further optimized. To achieve more precise embedding, we average the same-class instances in the training set before the embedding adaptation with the set-to-set transformation. Adam [20] and SGD are used to optimize ConvNet and ResNet variants respectively. Moreover, we follow the most standard implementations for the four set-to-set functions — BiLSTM [16], DeepSets [56], Graph Convolutional Networks (GCN) [21] and Transformer (FEAT) [47]. We refer readers to supplementary material (SM) for complete details and ablation studies of each set-to-set functions. Our implementation is available at https://github.com/Sha-Lab/FEAT.

5. Experiments

In this section, we first evaluate a variety of models for embedding adaptation in § 5.2 with standard FSL. It concludes that FEAT (with Transformer) is the most effective approach among different instantiations. Next, we perform ablation studies in § 5.2.2 to analyze FEAT in details. Eventually, we evaluate FEAT on many extended few-shot learning tasks to study its general applicability (§ 5.3). This study includes few-shot domain generalization, transductive few-shot learning, generalized few-shot learning, and large-scale low-shot learning (refer to SM).

5.1. Experimental Setups

Datasets. *Mini*ImageNet [49] and *Tiered*ImageNet [38] datasets are subsets of the ImageNet [39]. *Mini*ImageNet includes a total number of 100 classes and 600 examples per class. We follow the setup provided by [36], and use 64 classes as SEEN categories, 16 and 20 as two sets of UNSEEN categories for model validation and evaluation respectively. *Tiered*ImageNet is a large-scale dataset with more categories, which contains 351, 97, and 160 categories for model training, validation, and evaluation, respectively. In addition to these, we investigate the OfficeHome [48] dataset to validate the generalization ability of FEAT across domains. There are four domains in OfficeHome, and two of them ("Clipart" and "Real World") are selected, which contains 8722 images. After randomly splitting all classes, 25 classes serve as the seen classes to train the model, and the remaining 15 and 25 classes are used as two UNSEEN for evaluation. Please refer to SM for more details.

Evaluation protocols. Previous approaches [10, 43, 46] usually follow the original setting of [49] and evaluate the models on 600 sampled target tasks (15 test instances per class). In a later study [40], it was suggested that such an evaluation process could potentially introduce high variances. Therefore, we follow the new and more trustworthy evaluation setting to evaluate both baseline models and our approach on 10,000 sampled tasks. We report the mean accuracy (in %) as well as the 95% confidence interval.

Baseline and embedding adaptation methods. We re-implement the prototypical network (ProtoNet) [43] as a task-agnostic embedding baseline model. This is known as a very strong approach [8] when the backbone architecture is deep, *i.e.*, residual networks [15]. As suggested by [33], we tune the scalar temperature carefully to scale the logits of both approaches in our re-implementation. As mentioned, we implement the embedding adaptation model with four different function approximators, and denote them as BILSTM, DEEPSETS, GCN, and FEAT (*i.e.* Transformer). The concrete details of each model are included in the SM.

Backbone pre-training. Instead of optimizing from scratch, we apply an additional pre-training strategy as suggested in [35, 40]. The backbone network, appended with a **softmax** layer, is trained to classify all SEEN classes with the cross-entropy loss (*e.g.*, 64 classes in the *Mini*ImageNet). The classification performance over the penultimate layer embeddings of sampled 1-shot tasks from the model validation split is evaluated to select the best pre-trained model, whose weights are then used to initialize the embedding function $\mathbf{E}$ in the few-shot learning.

5.2. Standard Few-Shot Image Classification

We compare our proposed FEAT method with the instance embedding baselines as well as previous methods on

Table 1: Few-shot classification accuracy on *Mini*ImageNet. ★ CTM [28] and SimpleShot [51] utilize the ResNet-18. (see SM for the full table with confidence intervals and WRN results.).

Setups →	**1-Shot 5-Way**		**5-Shot 5-Way**	
Backbone →	ConvNet	ResNet	ConvNet	ResNet
MatchNet [49]	43.40	-	51.09	-
MAML [10]	48.70	-	63.11	-
ProtoNet [43]	49.42	-	68.20	-
RelationNet [45]	51.38	-	67.07	-
PFA [35]	54.53	59.60	67.87	73.74
TADAM [33]	-	58.50	-	76.70
MetaOptNet [25]	-	62.64	-	78.63
CTM [28]	-	64.12	-	80.51
SimpleShot [51]	49.69	62.85	66.92	80.02
Instance embedding				
ProtoNet	52.61	62.39	71.33	80.53
Embedding adaptation				
BILSTM	52.13	63.90	69.15	80.62
DEEPSETS	54.41	64.14	70.96	80.93
GCN	53.25	64.50	70.59	81.65
FEAT	**55.15**	**66.78**	**71.61**	**82.05**

Table 2: Few-shot classification accuracy and 95% confidence interval on *Tiered*ImageNet with the ResNet backbone.

Setups →	**1-Shot 5-Way**	**5-Shot 5-Way**
ProtoNet [43]	53.31 $_{\pm 0.89}$	72.69 $_{\pm 0.74}$
RelationNet [45]	54.48 $_{\pm 0.93}$	71.32 $_{\pm 0.78}$
MetaOptNet [25]	65.99 $_{\pm 0.72}$	81.56 $_{\pm 0.63}$
CTM [28]	68.41 $_{\pm 0.39}$	84.28 $_{\pm 1.73}$
SimpleShot [51]	69.09 $_{\pm 0.22}$	84.58 $_{\pm 0.16}$
Instance embedding		
ProtoNet	68.23 $_{\pm 0.23}$	84.03 $_{\pm 0.16}$
Embedding adaptation		
BILSTM	68.14 $_{\pm 0.23}$	84.23 $_{\pm 0.16}$
DEEPSETS	68.59 $_{\pm 0.24}$	84.36 $_{\pm 0.16}$
GCN	68.20 $_{\pm 0.23}$	84.64 $_{\pm 0.16}$
FEAT	**70.80** $_{\pm 0.23}$	**84.79** $_{\pm 0.16}$

Table 3: Number of parameters introduced by each set-to-set function in additional to the backbone's parameters.

	BILSTM	DEEPSETS	GCN	FEAT
ConvNet	25K	82K	33K	16K
ResNet	2.5M	8.2M	3.3M	1.6M

the standard *Mini*ImageNet [49] and *Tiered*ImageNet [38] benchmarks, and then perform detailed analysis on the ablated models. We include additional results with CUB [50] dataset in SM, which shares a similar observation.

5.2.1. Main Results

Comparison to previous State-of-the-arts. Table 1 and Table 2 show the results of our method and others on the *Mini*ImageNet and *Tiered*ImageNet. First, we observe that the best embedding adaptation method (FEAT) outperforms the instance embedding baseline on both datasets, indicating the effectiveness of learning task-specific embedding space. Meanwhile, the FEAT model performs significantly better than the current state-of-the-art methods on *Mini*ImageNet dataset. On the *Tiered*ImageNet, we observe that the ProtoNet baseline is already better than some previous state-of-the-arts based on the 12-layer ResNet backbone. This might due to the effectiveness of the pre-training stage on the *Tiered*ImageNet as it is larger than *Mini*ImageNet and a fully converged model can be itself very effective. Based on this, all embedding adaptation approaches further improves over ProtoNet almost in all cases, with FEAT achieving the best performances among all approaches. Note that here our pre-training strategy is most similar to the one used in PFA [35], while we further fine-tune the backbone. Temperature scaling of the logits influences the performance a lot when fine-tuning over the pre-trained weights. Additionally, we list some recent methods (SimpleShot [51], and CTM [28]) using different backbone

architectures such as ResNet-18 for reference.

Comparison among the embedding adaptation models. Among the four embedding adaptation methods, BILSTM in most cases achieves the worst performances and sometimes even performs worse than ProtoNet. This is partially due to the fact that BILSTM can not easily implement the required permutation invariant property (also shown in [56]), which confuses the learning process of embedding adaptation. Secondly, we find that DEEPSETS and GCN have the ability to adapt discriminative task-specific embeddings but do not achieve consistent performance improvement over the baseline ProtoNet especially on *Mini*ImageNet with the ConvNet backbone. A potential explanation is that, such models when jointly learned with the backbone model, can make the optimization process more difficult, which leads to the varying final performances. In contrast, we observe that FEAT can consistently improve ProtoNet and other embedding adaptation approaches in all cases, without additional bells and whistles. It shows that the Transformer as a set-to-set function can implement rich interactions between instances, which provides its high expressiveness to model the embedding adaptation process.

Interpolation and extrapolation of classification ways. Next, we study different set-to-set functions on their capability of interpolating and extrapolating across the number of classification ways. To do so, we train each variant of em-
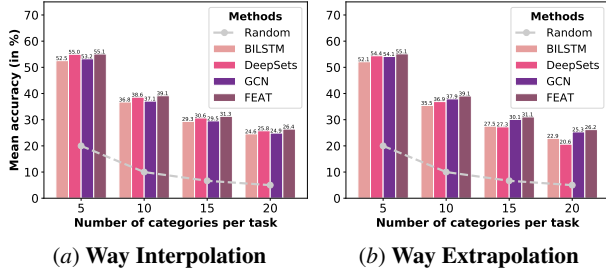
(a) **Way Interpolation**   *(b)* **Way Extrapolation**

Figure 3: Interpolation and **Extrapolation** of few-shot tasks from the "way" perspective. First, We train various embedding adaptation models on 1-shot 20-way (a) or 5-way (b) classification tasks and evaluate models on unseen tasks with different number of classes ($N=\{5, 10, 15, 20\}$). It shows that FEAT is superior in terms of way interpolation and extrapolation ability.

bedding adaptation functions with both 1-shot 20-way and 1-shot 5-way tasks, and measure the performance change as a function to the number of categories in the test time. We report the mean accuracies evaluated on few-shot classification with $N = \{5, 10, 15, 20\}$ classes, and show results in Figure 3. Surprisingly, we observe that FEAT *achieves almost the same numerical performances in both extrapolation and interpolation scenarios*, which further displays its strong capability of learning the set-to-set transformation. Meanwhile, we observe that DEEPSETS works well with interpolation but fails with extrapolation as its performance drops significantly with the larger N. In contrast, GCN achieves strong extrapolation performances but does not work as effectively in interpolation. BILSTM performs the worst in both cases, as it is by design not permutation invariant and may have fitted an arbitrary dependency between instances.

Parameter efficiency. Table 3 shows the number of additional parameters each set-to-set function has introduced. From this, we observe that with both ConvNet and ResNet backbones, FEAT has the smallest number of parameters compared with all other approaches while achieving best performances from various aspects (as results discussed above), which highlights its high parameter efficiency.

All above, we conclude that: 1) learning embedding adaptation with a set-to-set model is very effective in modeling task-specific embeddings for few-shot learning 2) FEAT is the most parameter-efficient function approximater that achieves the best empirical performances, together with nice permutation invariant property and strong interpolation/extrapolation capability over the classification way.

5.2.2. Ablation Studies

We analyze FEAT and its ablated variants on the *Mini*ImageNet dataset with ConvNet backbone.

How does the embedding adaptation looks like qualita-

tively? We sample four few-shot learning tasks and learn a principal component analysis (PCA) model (that projects embeddings into 2-D space) using the instance embeddings of the test data. We then apply this learned PCA projection to both the support set's pre-adapted and post-adapted embeddings. The results are shown in Figure 1 (the beginning of the paper). In three out of four examples, post-adaptation embeddings of FEAT improve over the pre-adaption embeddings. Interestingly, we found that the embedding adaptation step of FEAT has the tendency of pushing the support embeddings apart from the clutter, such that they can better fit the test data of its categories. In the negative example where post-adaptation degenerates the performances, we observe that the embedding adaptation step has pushed two support embeddings "Golden Retriever" and "Lion" too close to each other. It has qualitatively shown that the adaptation is crucial to obtain superior performances and helps to contrast against task-agnostic embeddings.

5.3. Extended Few-Shot Learning Tasks

In this section, we evaluate FEAT on 3 different few-shot learning tasks. Specifically, cross-domain FSL, transductive FSL [30, 38], and generalized FSL [7]. We overview the setups briefly and please refer to SM for details.

FS Domain Generalization assumes that examples in UN-SEEN support and test set can come from the different domains, *e.g.*, sampled from different distributions [9, 19]. The example of this task can be found in Figure 4. It requires a model to recognize the intrinsic property than texture of objects, and is de facto analogical recognition.

Transductive FSL. The key difference between standard and transductive FSL is whether test instances arrive one at a time or all simultaneously. The latter setup allows the structure of unlabeled test instances to be utilized. Therefore, the prediction would depend on both the training (support) instances and all the available test instances in the target task from UNSEEN categories.

Generalized FSL. Prior works assumed the test instances coming from unseen classes only. Different from them, the *generalized FSL* setting considers test instances from both SEEN and UNSEEN classes [37]. In other words, during the model evaluation, while support instances all come from $\mathcal{U}$, the test instances come from $\mathcal{S} \cup \mathcal{U}$, and the classifier is required to predict on both SEEN and UNSEEN categories. .

5.3.1. Few-Shot Domain Generalization

We show that FEAT learns to adapt *the intrinsic structure of tasks*, and **generalizes across domains**, *i.e.*, predicting test instances even when the visual appearance is changed.

Setups. We train the FSL model in the standard domain and evaluate with cross-domain tasks, where the N-categories are aligned but domains are different. In detail, a model is

	C → C	C → R
Supervised	$34.38_{\pm 0.16}$	$29.49_{\pm 0.16}$
ProtoNet	$35.51_{\pm 0.16}$	$29.47_{\pm 0.16}$
FEAT	$\mathbf{36.83}_{\pm 0.17}$	$\mathbf{30.89}_{\pm 0.17}$

(a) Few-shot domain generalization

	1-Shot	5-Shot
TPN [30]	55.51	69.86
TEAM [34]	56.57	72.04
FEAT	$\mathbf{57.04}_{\pm 0.20}$	$\mathbf{72.89}_{\pm 0.16}$

(b) Transductive few-shot learning

	SEEN	UNSEEN	COMBINED
Random	$1.56_{\pm 0.00}$	$20.00_{\pm 0.00}$	$1.45_{\pm 0.00}$
ProtoNet	$41.73_{\pm 0.03}$	$48.64_{\pm 0.20}$	$35.69_{\pm 0.03}$
FEAT	$\mathbf{43.94}_{\pm 0.03}$	$\mathbf{49.72}_{\pm 0.20}$	$\mathbf{40.50}_{\pm 0.03}$

(c) Generalized few-shot learning

Table 4: We evaluate our model on three additional few-shot learning tasks: **(a)** Few-shot domain generalization, **(b)** Transductive few-shot learning, and **(c)** Generalized few-shot learning. We observe that FEAT consistently outperform all previous methods or baselines.
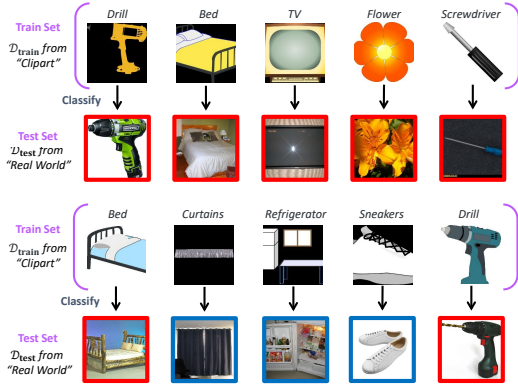

Figure 4: Qualitative results of **few-shot domain-generalization** for FEAT. Correctly classified examples are shown in **red** boxes and incorrectly ones are shown in **blue** boxes. We visualize one task that FEAT succeeds (top) and one that fails (bottom).

trained on tasks from the "Clipart" domain of OfficeHome dataset [48], then the model is required to generalize to both "Clipart (**C**)" and "Real World (**R**)" test instances. In other words, we need to classify complex real images by seeing only a few sketches (Figure 4 gives an overview of data).

Results. Table 4 (a) gives the quantitative results and Figure 4 qualitatively examines it. Here, the "supervised" denotes a model trained with the standard classification strategy and then its penultimate layer's output feature is used as the nearest neighbor classifier. We observe that ProtoNet can outperform this baseline on tasks when evaluating instances from "Clipart" but not ones from "real world". However, FEAT improves over "real world" few-shot classification even only seeing the support data from "Clipart".

5.3.2. Transductive Few-Shot Learning

We show that without additional efforts in modeling, FEAT outperforms existing methods in transductive FSL.

Setups. We further study this semi-supervised learning setting to see how well FEAT can incorporate test instances into joint embedding adaptation. Specifically, we use the unlabeled test instances to augment the key and value sets of Transformer (refer to SM for details), so that the embedding

adaptation takes relationship of all test instances into consideration. We evaluate this setting on the transductive protocol of *Mini*ImageNet [38]. With the adapted embedding, FEAT makes predictions based on Semi-ProtoNet [38].

Results. We compare with two previous approaches, TPN [30] and TEAM [34]. The results are shown in Table 4 (b). We observe that FEAT improves its standard FSL performance (refer to Table 1) and also outperforms previous semi-supervised approaches by a margin.

5.3.3. Generalized Few-Shot Learning

We show that FEAT performs well on generalized few-shot classification of both SEEN and UNSEEN classes.

Setups. In this scenario, we evaluate not only on classifying test instances from a N-way M-shot task from UN-SEEN set $\mathcal{U}$, but also on all available SEEN classes from $\mathcal{S}$. To do so, we hold out 150 instances from each of the 64 seen classes in *Mini*ImageNet for validation and evaluation. Next, given a 1-shot 5-way training set $\mathcal{D}_{\mathbf{train}}$, we consider three evaluation protocols based on different class sets [7]: UNSEEN measures the mean accuracy on test instances only from $\mathcal{U}$ (5-Way few-shot classification); SEEN measures the mean accuracy on test instances only from $\mathcal{S}$ (64-Way classification); COMBINED measures the mean accuracy on test instances from $\mathcal{S} \cup \mathcal{U}$ (69-Way mixed classification).

Results. The results can be found in Table 4 (c). We observe that again FEAT outperforms baseline ProtoNet. To calibrate the prediction score on SEEN and UNSEEN classes [7, 52], we select a constant seen/unseen class probability over the validation set, and subtract this calibration factor from seen classes' prediction score. Then we take the prediction with maximum score value after calibration.

6. Discussion

A common embedding space fails to tailor discriminative visual knowledge for a target task especially when there are a few labeled training data. We propose to do embedding adaptation with a set-to-set function and instantiate it with transformer (FEAT), which customizes task-specific embedding spaces via a self-attention architecture. The adapted embedding space leverages the relationship between target

task training instances, which leads to discriminative instance representations. FEAT achieves the state-of-the-art performance on benchmarks, and its superiority can generalize to tasks like cross-domain, transductive, and generalized few-shot classifications.

References

[1] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. Label-embedding for attribute-based classification. In *CVPR*, pages 819–826, 2013. 2

[2] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *NIPS*, pages 3981–3989. 2016. 2

[3] A. Antoniou, H. Edwards, and A. J. Storkey. How to train your MAML. In *ICLR*, 2019. 2

[4] L. J. Ba, R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 13

[5] S. Changpinyo, W.-L. Chao, B. Gong, and F. Sha. Synthesized classifiers for zero-shot learning. In *CVPR*, pages 5327–5336, 2016. 2

[6] S. Changpinyo, W.-L. Chao, and F. Sha. Predicting visual exemplars of unseen classes for zero-shot learning. In *ICCV*, pages 3496–3505, 2017. 2

[7] W.-L. Chao, S. Changpinyo, B. Gong, and F. Sha. An empirical study and analysis of generalized zero-shot learning for object recognition in the wild. In *ECCV*, pages 52–68, 2016. 7, 8

[8] W.-Y. Chen, Y.-C. Liu, Z. Kira, Y.-C. F. Wang, and J.-B. Huang. A closer look at few-shot classification. In *ICLR*, 2019. 5

[9] N. Dong and E. P. Xing. Domain adaption in one-shot learning. In *ECML PKDD*, pages 573–588, 2018. 7

[10] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017. 1, 2, 3, 5, 6, 14, 15

[11] G. Ghiasi, T.-Y. Lin, and Q. V. Le. Dropblock: A regularization method for convolutional networks. In *NeurIPS*, pages 10750–10760. 2018. 13

[12] S. Gidaris and N. Komodakis. Dynamic few-shot visual learning without forgetting. In *CVPR*, pages 4367–4375, 2018. 18

[13] L.-Y. Gui, Y.-X. Wang, D. Ramanan, and J. M. F. Moura. Few-shot human motion prediction via meta-learning. In *ECCV*, pages 441–459, 2018. 2

[14] B. Hariharan and R. B. Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *ICCV*, pages 3037–3046, 2017. 18

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 5, 18

[16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 2, 4, 5, 11

[17] K. Hsu, S. Levine, and C. Finn. Unsupervised learning via meta-learning. In *ICLR*, 2019. 2

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. 13

[19] B. Kang and J. Feng. Transferable meta learning across domains. In *UAI*, pages 177–187, 2018. 7

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5, 14

[21] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 2, 4, 5, 12

[22] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015. 2

[23] B. M. Lake, R. Salakhutdinov, J. Gross, and J. B. Tenenbaum. One shot learning of simple visual concepts. In *CogSci*, 2011. 1

[24] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 1

[25] K. Lee, S. Maji, A. Ravichandran, and S. Soatto. Meta-learning with differentiable convex optimization. In *CVPR*, pages 10657–10665, 2019. 2, 5, 6, 13, 15

[26] Y. Lee and S. Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *ICML*, pages 2933–2942, 2018. 2

[27] F.-F. Li, R. Fergus, and P. Perona. One-shot learning of object categories. *TPAMI*, 28(4):594–611, 2006. 1

[28] H. Li, D. Eigen, S. Dodge, M. Zeiler, and X. Wang. Finding task-relevant features for few-shot learning by category traversal. In *CVPR*, pages 1–10, 2019. 2, 6

[29] Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. A structured self-attentive sentence embedding. In *ICLR*, 2017. 2, 4

[30] Y. Liu, J. Lee, M. Park, S. Kim, E. Yang, S. J. Hwang, and Y. Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. In *ICLR*, 2019. 7, 8, 17, 18

[31] L. Metz, N. Maheswaranathan, B. Cheung, and J. Sohl-Dickstein. Learning unsupervised learning rules. *CoRR*, abs/1804.00222, 2018. 2

[32] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018. 2

[33] B. N. Oreshkin, P. R. López, and A. Lacoste. TADAM: task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, pages 719–729. 2018. 5, 6, 11, 15

[34] L. Qiao, Y. Shi, J. Li, Y. Wang, T. Huang, and Y. Tian. Transductive episodic-wise adaptive metric for few-shot learning. In *ICCV*, pages 3603–3612, 2019. 8, 17, 18

[35] S. Qiao, C. Liu, W. Shen, and A. L. Yuille. Few-shot image recognition by predicting parameters from activations. In *CVPR*, pages 7229–7238, 2018. 2, 5, 6, 13, 14, 15

[36] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017. 2, 5

[37] M. Ren, R. Liao, E. Fetaya, and R. S. Zemel. Incremental few-shot learning with attention attractor networks. In *NeurIPS*, pages 5276–5286, 2019. 7

[38] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel. Meta-learning for semi-supervised few-shot classification. In *ICLR*, 2018. 5, 6, 7, 8, 14, 17, 18

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F.-F. Li. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 5, 18

[40] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *ICLR*, 2019. 1, 2, 5, 13, 14, 15

[41] V. G. Satorras and J. B. Estrach. Few-shot learning with graph neural networks. In *ICLR*, 2018. 2, 4, 12

[42] T. R. Scott, K. Ridgeway, and M. C. Mozer. Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning. In *NeurIPS*, pages 76–85. 2018. 2

[43] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. In *NIPS*, pages 4080–4090. 2017. 1, 2, 3, 4, 5, 6, 11, 13, 15, 18

[44] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014. 13

[45] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, pages 1199–1208, 2018. 2, 6, 15

[46] E. Triantafillou, R. S. Zemel, and R. Urtasun. Few-shot learning through an information retrieval lens. In *NIPS*, pages 2252–2262. 2017. 1, 2, 5, 13, 14

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 6000–6010. 2017. 2, 4, 5, 12, 13, 14, 16

[48] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan. Deep hashing network for unsupervised domain adaptation. In *CVPR*, pages 5385–5394, 2017. 5, 8, 14, 17

[49] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *NIPS*, pages 3630–3638. 2016. 1, 2, 3, 4, 5, 6, 11, 12, 13, 14, 15

[50] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 6, 14

[51] Y. Wang, W.-L. Chao, K. Q. Weinberger, and L. van der Maaten. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. *CoRR*, abs/1911.04623, 2019. 6, 14, 15

[52] Y.-X. Wang, R. B. Girshick, M. Hebert, and B. Hariharan. Low-shot learning from imaginary data. In *CVPR*, pages 7278–7286, 2018. 8, 18

[53] X.-S. Wei, P. Wang, L. Liu, C. Shen, and J. Wu. Piecewise classifier mappings: Learning fine-grained learners for novel categories with few examples. *TIP*, 28(12):6116–6125, 2019. 2

[54] H.-J. Ye, H. Hu, D.-C. Zhan, and F. Sha. Learning embedding adaptation for few-shot learning. *CoRR*, abs/1812.03664, 2018. 13

[55] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016. 5, 13

[56] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *NIPS*, pages 3394–3404. 2017. 2, 4, 5, 6, 12

Supplementary Material

A. Details of Baseline Methods

In this section, we describe two important embedding learning baselines *i.e.*, Matching Network (MatchNet) [49] and Prototypical Network (ProtoNet) [43], to implement the prediction function $f(\mathbf{x_{test}}; \mathcal{D_{train}})$ in the few-shot learning framework.

MatchNet and ProtoNet. Both MatchNet and ProtoNet stress the learning of the embedding function $\mathbf{E}$ from the source task data $\mathcal{D}^{\mathcal{S}}$ with a meta-learning routine similar to Alg. 1 in the main text. We omit the super-script $\mathcal{S}$ since the prediction strategies can apply to tasks from both SEEN and UNSEEN sets.

Given the training data $\mathcal{D_{train}} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{NM}$ of an M-shot N-way classification task, we can obtain the embedding of each training instance based on the function $\mathbf{E}$:[1]

$$\phi(\mathbf{x}_i) = \mathbf{E}(\mathbf{x}_i), \ \forall \mathbf{x}_i \in \mathcal{X_{train}} \quad (8)$$

To classify a test instance $\mathbf{x_{test}}$, we perform the nearest neighbor classification , *i.e.*,

$$\hat{\mathbf{y}}_{\mathbf{test}} \propto \exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_i})\right) \cdot \mathbf{y}_i \quad (9)$$

$$= \frac{\exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_i})\right)}{\sum_{\mathbf{x}_{i'} \in \mathcal{X_{train}}} \exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_{i'}})\right)} \cdot \mathbf{y}_i$$

$$= \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D_{train}}} \frac{\exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_i})\right)}{\sum_{\mathbf{x}_{i'} \in \mathcal{X_{train}}} \exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_{i'}})\right)} \cdot \mathbf{y}_i$$

Here, MatchNet finds the most similar training instance to the test one, and assigns the label of the nearest neighbor to the test instance. Note that $\mathbf{sim}$ represents the cosine similarity, and $\gamma > 0$ is the scalar temperature value over the similarity score, which is found important empirically [33]. During the experiments, we tune this temperature value carefully, ranging from the reciprocal of $\{0.1, 1, 16, 32, 64, 128\}$.[2]

The ProtoNet has two key differences compared with the MatchNet. First, when $M > 1$ in the target task, ProtoNet computes the mean of the same class embeddings as the class center (prototype) in advance and classifies a test instance by computing its similarity to the nearest class center (prototype). In addition, it uses the negative distance rather

---

[1]In the following, we use $\phi(\mathbf{x}_i)$ and $\phi_{\mathbf{x}_i}$ exchangeably to represent the embedding of an instance $\mathbf{x}_i$ based on the mapping ϕ.

[2]In experiments, we find the temperature scale over logits influences the model training a lot when we optimize based on pre-trained weights.

than the cosine similarity as the similarity metric:

$$\mathbf{c}_n = \frac{1}{M} \sum_{\mathbf{y}_i = n} \phi(\mathbf{x}_i), \ \forall n = 1, \dots, N \quad (10)$$

$$\hat{\mathbf{y}}_{\mathbf{test}} \propto \exp\left(\gamma \cdot \|\phi_{\mathbf{x_{test}}} - \mathbf{c}_n\|_2^2\right) \cdot \mathbf{y}_n$$

$$= \sum_{n=1}^{N} \frac{\exp\left(-\gamma\|\phi_{\mathbf{x_{test}}} - \mathbf{c}_n\|_2^2\right)}{\sum_{n'=1}^{N} \exp\left(-\gamma\|\phi_{\mathbf{x_{test}}} - \mathbf{c}_{n'}\|_2^2\right)} \mathbf{y}_n \quad (11)$$

Similar to the aforementioned scalar temperature for Match-Net, in Eq. 11 we also consider the scale γ. Here we abuse the notation by using $\mathbf{y}_i = n$ to enumerate the instances with label n, and denote $\mathbf{y}_n$ as the one-hot coding of the n-th class. Thus Eq. 11 outputs the probability to classify $\mathbf{x_{test}}$ to the N classes.

In the experiments, we find ProtoNet incorporates better with FEAT. When there is more than one shot in each class, we average all instances per class in advance by Eq. 10 before inputting them to the set-to-set transformation. This pre-average manner makes more precise embedding for each class and facilitates the "downstream" embedding adaptation. We will validate this in the additional experiments.

B. Details of the Set-to-Set Functions

In this section, we provide details about four implementations of the set-to-set embedding adaptation function $\mathbf{T}$, *i.e.*, the BILSTM, DEEPSETS, GCN, and the TRANSFORMER. The last one is the key component in our **F**ew-shot **E**mbedding **A**daptation with **T**ransformer (FEAT) approach. Then we will introduce the configuration of the multi-layer/multi-head transformer, and the setup of the transformer for the transductive Few-Shot Learning (FSL).

B.1. BiLSTM as the Set-to-Set Transformation

Bidirectional LSTM (BILSTM) [16, 49] is one of the common choice to instantiate the set-to-set transformation, where the addition between the input and the hidden layer outputs of each BILSTM cell leads to the adapted embedding. In detail, we have

$$\{\overleftarrow{\phi}(\mathbf{x}), \overrightarrow{\phi}(\mathbf{x})\} = \mathbf{BILSTM}(\{\phi(\mathbf{x})\}); \ \forall \mathbf{x} \in \mathcal{X_{train}} \quad (12)$$

Where $\overleftarrow{\phi}(\mathbf{x})$ and $\overrightarrow{\phi}(\mathbf{x})$ are the hidden layer outputs of the two LSTM models for each instance embedding in the input set. Then we get the adapted embedding as

$$\psi(\mathbf{x}) = \phi(\mathbf{x}) + \overleftarrow{\phi}(\mathbf{x}) + \overrightarrow{\phi}(\mathbf{x}) \quad (13)$$

It is notable that the output of the BILSTM suppose to depend on the order of the input set. Vinyals *et al.* [49] propose to use the Fully Conditional Embedding to encode the context of both the test instance and the support set
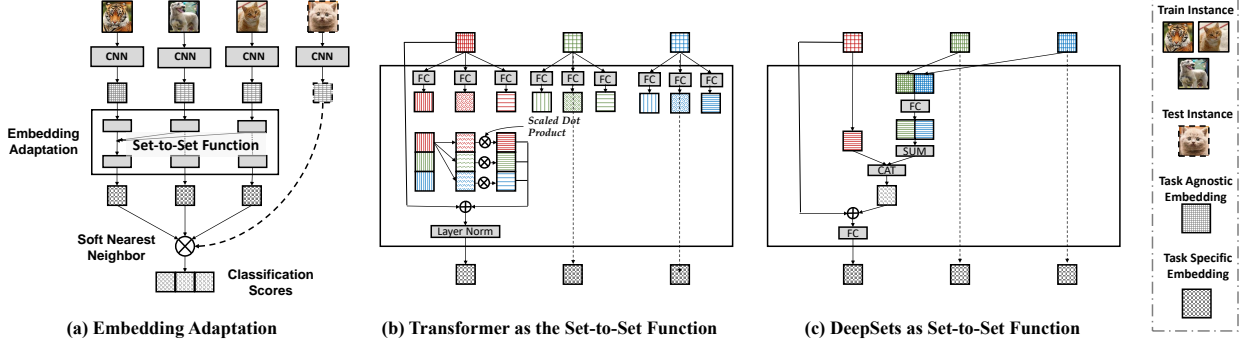
Figure 5: Illustration of two embedding adapation methods considered in the paper. (a) shows the main flow of Few-Shot Embedding Adaptation, while (b) and (c) demonstrate the workflow of Transformer and DeepSets respectively.

instances based on BILSTM and LSTM w/ Attention module. Different from [49], we apply the set-to-set embedding adaptation only over the support set, which leads to a fully inductive learning setting.

B.2. DeepSets as the Set-to-Set Transformation

Deep sets [56] suggests a generic aggregation function over a set should be the transformed sum of all elements in this set. Therefore, a very simple set-to-set transformation baseline involves two components, an instance centric representation combined with a set context representation. For any instance $\mathbf{x} \in \mathcal{X}_{\mathbf{train}}$, we define its complementary set as $\mathbf{x}^{\complement}$. Then we implement the set transformation by:

$$\psi(\mathbf{x}) = \phi(\mathbf{x}) + g([\phi(\mathbf{x}); \sum_{\mathbf{x}_{i'} \in \mathbf{x}^{\complement}} h(\phi(\mathbf{x}_{i'}))]) \qquad (14)$$

In Eq. 14, g and h are transformations which map the embedding into another space and increase the representation ability of the embedding. Two-layer multi-layer perception (MLP) with ReLU activation is used to implement these two mappings. For each instance, embeddings in its complementary set are first combined into a vector as the context, and then this vector is concatenated with the input embedding to obtain the residual component of the adapted embedding. This conditioned embedding takes other instances in the set into consideration, and keeps the "set (permutation invariant)" property. Finally, we determine the label with the newly adapted embedding ψ as Eq. 11. An illustration of the DeepSets notation in the embedding adaptation can be found in Figure 5 (c). The summation operator in Eq. 14 could also be replaced as the maximum operator, and we find the maximum operator works better than summation operator in our experiments.

B.3. GCN as the Set-to-Set Transformation

Graph Convolutional Networks (GCN) [21, 41] propagate the relationship between instances in the set. We first

construct a degree matrix $A \in \mathbb{R}^{NK \times NK}$ to represent the similarity between instances in a set. If two instances $\mathbf{x}_i$ and $\mathbf{x}_j$ come from the same class, then we set the corresponding element A_{ij} in A to 1, otherwise we have $A_{ij} = 0$. Based on A, we build the "normalized" adjacency matrix S for a given set with added self-loops $S = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$. $I \in \mathbb{R}^{NK \times NK}$ is the identity matrix, and D is the diagonal matrix whose elements are equal to the sum of elements in the corresponding row of $A + I$, *i.e.*, $D_{ii} = \sum_j A_{ij} + 1$ and $D_{ij} = 0$ if $i \neq j$. Let $\Phi^0 = \{\phi_{\mathbf{x}} ; \forall \mathbf{x} \in \mathcal{X}_{\mathbf{train}}\}$ be the concatenation of all the instance embeddings in the training set $\mathcal{X}_{\mathbf{train}}$. We use the super-script to denote the generation of the instance embedding matrix. The relationship between instances could be propagated based on S, *i.e.*,

$$\Phi^{t+1} = \mathbf{ReLU}(S\Phi^t W) , \ t = 0, 1, \dots, T-1 \qquad (15)$$

W is a learned a projection matrix for feature transformation. In GCN, the embedding in the set is transformed based on Eq. 15 multiple times (we propagate the embedding set two times during the experiments), and the final propagated embedding set Φ^T gives rise to the $\psi_{\mathbf{x}}$.

B.4. Transformer as the Set-to-Set Transformation

In this section, we describe in details about our **F**ew-Shot **E**mbedding **A**daptation w/ **T**ransformer (FEAT) approach, specifically how to use the transformer architecture [47] to implement the set-to-set function $\mathbf{T}$, where self-attention mechanism facilitates the instance embedding adaptation with consideration of the contextual embeddings.

As mentioned before, the transformer is a store of triplets in the form of (query, key, and value). Elements in the query set are the ones we want to do the transformation. The transformer first matches a query point with each of the keys by computing the "query" – "key" similarities. Then the proximity of the key to the query point is used to weight the corresponding values of each key. The transformed input acts as a residual value which will be added to the input.

Basic Transformer. Following the definitions in [47], we use $\mathcal{Q}$, $\mathcal{K}$, and $\mathcal{V}$ to denote the set of the query, keys, and values, respectively. All these sets are implemented by different combinations of task instances.

To increase the flexibility of the transformer, three sets of linear projections ($W_Q \in \mathbb{R}^{d \times d'}$, $W_K \in \mathbb{R}^{d \times d'}$, and $W_V \in \mathbb{R}^{d \times d'}$) are defined, one for each set.[3] The points in sets are first projected by the corresponding projections

$$
\begin{aligned}
Q &= W_Q^\top \left[\phi_{\mathbf{x}_q}; \quad \forall \mathbf{x}_q \in \mathcal{Q} \right] \in \mathbb{R}^{d' \times |\mathcal{Q}|} \\
K &= W_K^\top \left[\phi_{\mathbf{x}_k}; \quad \forall \mathbf{x}_k \in \mathcal{K} \right] \in \mathbb{R}^{d' \times |\mathcal{K}|} \\
V &= W_V^\top \left[\phi_{\mathbf{x}_v}; \quad \forall \mathbf{x}_v \in \mathcal{V} \right] \in \mathbb{R}^{d' \times |\mathcal{V}|}
\end{aligned}
\tag{16}
$$

$|\mathcal{Q}|$, $|\mathcal{K}|$, and $|\mathcal{V}|$ are the number of elements in the sets $\mathcal{Q}$, $\mathcal{K}$, and $\mathcal{V}$ respectively. Since there is a one-to-one correspondence between elements in $\mathcal{K}$ and $\mathcal{V}$ we have $|\mathcal{K}| = |\mathcal{V}|$.

The similarity between a query point $\mathbf{x}_q \in \mathcal{Q}$ and the list of keys $\mathcal{K}$ is then computed as "attention":

$$
\alpha_{qk} \propto \exp\left(\frac{\phi_{\mathbf{x}_q}^\top W_Q \cdot K}{\sqrt{d}} \right); \ \forall \mathbf{x}_k \in \mathcal{K}
\tag{17}
$$

$$
\alpha_{q,:} = \mathbf{softmax}\left(\frac{\phi_{\mathbf{x}_q}^\top W_Q \cdot K}{\sqrt{d}} \right) \in \mathbb{R}^{|\mathcal{K}|}
\tag{18}
$$

The k-th element α_{qk} in the vector $\alpha_{q,:}$ reveals the particular proximity between $\mathbf{x}_k$ and $\mathbf{x}_q$. The computed attention values are then used as weights for the final embedding $\mathbf{x}_q$:

$$
\tilde{\psi}_{\mathbf{x}_q} = \sum_k \alpha_{qk} V_{:,k}
\tag{19}
$$

$$
\psi_{\mathbf{x}_q} = \tau\left(\phi_{\mathbf{x}_q} + W_{\mathbf{FC}}^\top \tilde{\psi}_{\mathbf{x}_q} \right)
\tag{20}
$$

$V_{:,k}$ is the k-th column of V. $W_{\mathbf{FC}} \in \mathbb{R}^{d' \times d}$ is the projection weights of a fully connected layer. τ completes a further transformation, which is implemented by the dropout [44] and layer normalization [4]. The whole flow of transformer in our FEAT approach can be found in Figure 5 (b). With the help of transformer, the embeddings of all training set instances are adapted (we denote this approach as FEAT).

Multi-Head Multi-Layer Transformer. Following [47], an extended version of the transformer can be built with multiple parallel attention heads and stacked layers. Assume there are totally H heads, the transformer *concatenates* multiple attention-transformed embeddings, and then uses a linear mapping to project the embedding to the original embedding space (with the original dimensionality). Besides, we can take the transformer as a feature encoder of the input query instance. Therefore, it can be applied

over the input query *multiple times* (with different sets of parameters), which gives rise to the multi-layer transformer. We discuss the empirical performances with respect to the change number of heads and layers in § D.

B.5. Extension to transductive FSL

Facilitated by the flexible set-to-set transformer in Eq. 20, our adaptation approach can naturally be extended to the transductive FSL setting.

When classifying test instance $\mathbf{x}_{\mathbf{test}}$ in the transdutive scenario, other test instances $\mathcal{X}_{\mathbf{test}}$ from the N categories would also be available. Therefore, we enrich the transformer's query and key/value sets

$$
\mathcal{Q} = \mathcal{K} = \mathcal{V} = \mathcal{X}_{\mathbf{train}} \cup \mathcal{X}_{\mathbf{test}}
\tag{21}
$$

In this manner, the embedding adaptation procedure would also consider the structure among unlabeled test instances. When the number of shots $K > 1$, we average the embedding of labeled instances in each class first before combining them with the test set embeddings.

C. Implementation Details

Backbone architecture. We consider three backbones, as suggested in the literature, as the instance embedding function $\mathbf{E}$ for the purpose of fair comparisons. We resize the input image to $84 \times 84 \times 3$ before using the backbones.

- **ConvNet.** The 4-layer convolution network [43, 46, 49] contains 4 repeated blocks. In each block, there is a convolutional layer with 3×3 kernel, a Batch Normalization layer [18], a **ReLU**, and a Max pooling with size 2. We set the number of convolutional channels in each block as 64. A bit different from the literature, we add a global max pooling layer at last to reduce the dimension of the embedding. Based on the empirical observations, this will not influence the results, but reduces the computation burden of later transformations a lot.

- **ResNet.** We use the 12-layer residual network in [25].[4] The DropBlock [11] is used in this ResNet architecture to avoid over-fitting. A bit different from the ResNet-12 in [25], we apply a global average pooling after the final layer, which leads to 640 dimensional embeddings.[5]

- **WRN.** We also consider the Wide residual network [40, 55]. We use the WRN-28-10 structure as in [35, 40], which sets the depth to 28 and width to 10. After a global

---

[3]For notation simplicity, we omit the bias in the linear projection here.

[4]The source code of the ResNet is publicly available on https://github.com/kjunelee/MetaOptNet

[5]We use the ResNet backbone with input image size $80 \times 80 \times 3$ from [35] in the old version of our paper [54], whose source code of ResNet is publicly available on https://github.com/joe-siyuan-qiao/FewShot-CVPR. Empirically we find the ResNet-12 [25] works better than our old ResNet architecture.

average pooling in the last layer of the backbone, we get a 640 dimensional embedding for further prediction.

Datasets. Four datasets, *Mini*ImageNet [49], *Tiered*ImageNet [38], Caltech-UCSD Birds (CUB) 200-2011 [50], and OfficeHome [48] are investigated in this paper. Each dataset is split into three parts based on different non-overlapping sets of classes, for model training (a.k.a. meta-training in the literature), model validation (a.k.a. meta-val in the literature), and model evaluation (a.k.a. meta-test in the literature). The CUB dataset is initially designed for fine-grained classification. It contains in total 11,788 images of birds over 200 species. On CUB, we randomly sampled 100 species as SEEN classes, another two 50 species are used as two UNSEEN sets for model validation and evaluation [46]. For all images in the CUB dataset, we use the provided bounding box to crop the images as a pre-processing [46]. Before input into the backbone network, all images in the dataset are resized based on the requirement of the network.

Pre-training strategy. As mentioned before, we apply an additional pre-training strategy as suggested in [35, 40]. The backbone network, appended with a **softmax** layer, is trained to classify all classes in the SEEN class split (*e.g.*, 64 classes in the *Mini*ImageNet) with the cross-entropy loss. In this stage, we apply image augmentations like random crop, color jittering, and random flip to increase the generalization ability of the model. After each epoch, we validate the performance of the pre-trained weights based on its few-shot classification performance on the model validation split. Specifically, we randomly sample 200 1-shot N-way few-shot learning tasks (N equals the number of classes in the validation split, *e.g.*, 16 in the *Mini*ImageNet), which contains 1 instance per class in the support set and 15 instances per class for evaluation. Based on the penultimate layer instance embeddings of the pre-trained weights, we utilize the nearest neighbor classifiers over the few-shot tasks and evaluate the quality of the backbone. We select the pre-trained weights with the best few-shot classification accuracy on the validation set. The pre-trained weights are used to initialize the embedding backbone **E**, and the weights of the whole model are then optimized together during the model training.

Transformer Hyper-parameters. We follow the architecture as presented in [47] to build our FEAT model. The hidden dimension d' for the linear transformation in our FEAT model is set to 64 for ConvNet and 640 for ResNet/WRN. The dropout rate in transformer is set as 0.5. We empirically observed that the shallow transformer (with one set of projection and one stacked layer) gives the best overall performance (also studied in § D.2).

Optimization. Following the literature, different optimizers are used for the backbones during the model training. For the ConvNet backbone, stochastic gradient descent with Adam [20] optimizer is employed, with the initial learning rate set to be 0.002. For the ResNet and WRN backbones, vanilla stochastic gradient descent with Nesterov acceleration is used with an initial rate of 0.001. We fix the weight decay in SGD as 5e-4 and momentum as 0.9. The schedule of the optimizers is tuned over the validation part of the dataset. As the backbone network is initialized with the pre-trained weights, we scale the learning rate for those parameters by 0.1.

D. Additional Experimental Results

In this section, we will show more experimental results over the *Mini*ImageNet/CUB dataset, the ablation studies, and the extended few-shot learning.

D.1. Main Results

The full results of all methods on the *Mini*ImageNet can be found in Table 5. The results of MAML [10] optimized over the pre-trained embedding network are also included. We re-implement the ConvNet backbone of MAML and cite the MAML results over the ResNet backbone from [40]. It is also noteworthy that the FEAT gets the best performance among all popular methods and baselines.

We also investigate the Wide ResNet (WRN) backbone over *Mini*ImageNet, which is also the popular one used in [35, 40]. SimpleShot [51] is a recent proposed embedding-based few-shot learning approach that takes full advantage of the pre-trained embeddings. We cite the results of PFA [35], LEO [40], and SimpleShot [51] from their papers. The results can be found in Table 6. We re-implement ProtoNet and our FEAT approach with WRN. It is notable that in this case, our FEAT achieves *much higher* promising results than the current state-of-the-art approaches. Table 7 shows the classification results with WRN on the *Tiered*ImageNet data set, where our FEAT still keeps its superiority when dealing with 1-shot tasks.

Table 8 shows the 5-way 1-shot and 5-shot classification results on the CUB dataset based on the ConvNet backbone. The results on CUB are consistent with the trend on the *Mini*ImageNet dataset. Embedding adaptation indeed assists the embedding encoder for the few-shot classification tasks. Facilitated by the set function property, the DEEPSETS works better than the BILSTM counterpart. Among all the results, the transformer based FEAT gets the top tier results.

D.2. Ablation Studies

In this section, we perform further analyses for our proposed FEAT and its ablated variants classifying in the Pro-

Table 5: Few-shot classification accuracy± 95% confidence interval on *Mini*ImageNet with ConvNet and ResNet backbones. Our implementation methods are measured over 10,000 test trials.

Setups →	1-Shot 5-Way		5-Shot 5-Way	
Backbone Network →	ConvNet	ResNet	ConvNet	ResNet
MatchNet [49]	$43.40_{\pm 0.78}$	-	$51.09_{\pm 0.71}$	-
MAML [10]	$48.70_{\pm 1.84}$	-	$63.11_{\pm 0.92}$	-
ProtoNet [43]	$49.42_{\pm 0.78}$	-	$68.20_{\pm 0.66}$	-
RelationNet [45]	$51.38_{\pm 0.82}$	-	$67.07_{\pm 0.69}$	-
PFA [35]	$54.53_{\pm 0.40}$	-	$67.87_{\pm 0.20}$	-
TADAM [33]	-	$58.50_{\pm 0.30}$	-	$76.70_{\pm 0.30}$
MetaOptNet [25]	-	$62.64_{\pm 0.61}$	-	$78.63_{\pm 0.46}$
Baselines				
MAML	$49.24_{\pm 0.21}$	$58.05_{\pm 0.10}$	$67.92_{\pm 0.17}$	$72.41_{\pm 0.20}$
MatchNet	$52.87_{\pm 0.20}$	$65.64_{\pm 0.20}$	$67.49_{\pm 0.17}$	$78.72_{\pm 0.15}$
ProtoNet	$52.61_{\pm 0.20}$	$62.39_{\pm 0.21}$	$71.33_{\pm 0.16}$	$80.53_{\pm 0.14}$
Embedding Adaptation				
BILSTM	$52.13_{\pm 0.20}$	$63.90_{\pm 0.21}$	$69.15_{\pm 0.16}$	$80.63_{\pm 0.14}$
DEEPSETS	$54.41_{\pm 0.20}$	$64.14_{\pm 0.22}$	$70.96_{\pm 0.16}$	$80.93_{\pm 0.14}$
GCN	$53.25_{\pm 0.20}$	$64.50_{\pm 0.20}$	$70.59_{\pm 0.16}$	$81.65_{\pm 0.14}$
Ours: FEAT	$\mathbf{55.15}_{\pm 0.20}$	$\mathbf{66.78}_{\pm 0.20}$	$\mathbf{71.61}_{\pm 0.16}$	$\mathbf{82.05}_{\pm 0.14}$

Table 6: Few-shot classification performance with **Wide ResNet (WRN)-28-10 backbone** on *Mini*ImageNet dataset (mean accuracy±95% confidence interval). Our implementation methods are measured over 10,000 test trials.

Setups →	1-Shot 5-Way	5-Shot 5-Way
PFA [35]	$59.60_{\pm 0.41}$	$73.74_{\pm 0.19}$
LEO [40]	$61.76_{\pm 0.08}$	$77.59_{\pm 0.12}$
SimpleShot [51]	$63.50_{\pm 0.20}$	$80.33_{\pm 0.14}$
ProtoNet (Ours)	$62.60_{\pm 0.20}$	$79.97_{\pm 0.14}$
Ours: FEAT	$\mathbf{65.10}_{\pm 0.20}$	$\mathbf{81.11}_{\pm 0.14}$

Table 7: Few-shot classification performance with **Wide ResNet (WRN)-28-10 backbone** on *Tiered*ImageNet dataset (mean accuracy±95% confidence interval). Our implementation methods are measured over 10,000 test trials.

Setups →	1-Shot 5-Way	5-Shot 5-Way
LEO [40]	$66.33_{\pm 0.05}$	$81.44_{\pm 0.09}$
SimpleShot [51]	$69.75_{\pm 0.20}$	$\mathbf{85.31}_{\pm 0.15}$
Ours: FEAT	$\mathbf{70.41}_{\pm 0.23}$	$84.38_{\pm 0.16}$

Table 8: Few-shot classification performance with ConvNet backbone on CUB dataset (mean accuracy±95% confidence interval). Our implementation methods are measured over 10,000 test trials.

Setups →	1-Shot 5-Way	5-Shot 5-Way
MatchNet [49]	$61.16_{\pm 0.89}$	$72.86_{\pm 0.70}$
MAML [10]	$55.92_{\pm 0.95}$	$72.09_{\pm 0.76}$
ProtoNet [43]	$51.31_{\pm 0.91}$	$70.77_{\pm 0.69}$
RelationNet [45]	$62.45_{\pm 0.98}$	$76.11_{\pm 0.69}$
Instance Embedding		
MatchNet	$67.73_{\pm 0.23}$	$79.00_{\pm 0.16}$
ProtoNet	$63.72_{\pm 0.22}$	$81.50_{\pm 0.15}$
Embedding Adaptation		
BILSTM	$62.05_{\pm 0.23}$	$73.51_{\pm 0.19}$
DEEPSETS	$67.22_{\pm 0.23}$	$79.65_{\pm 0.16}$
GCN	$67.83_{\pm 0.23}$	$80.26_{\pm 0.15}$
Ours: FEAT	$\mathbf{68.87}_{\pm 0.22}$	$\mathbf{82.90}_{\pm 0.15}$

Table 9: Ablation studies on whether the embedding adaptation improves the discerning quality of the embeddings. After embedding adaptation, FEAT improves w.r.t. the before-adaptation embeddings a lot for Few-shot classification.

	1-Shot 5-Way	5-Shot 5-Way
Pre-Adapt	$51.60_{\pm 0.20}$	$70.40_{\pm 0.16}$
Post-Adapt	$55.15_{\pm 0.20}$	$71.61_{\pm 0.16}$

toNet manner, on the *Mini*ImageNet dataset, using the ConvNet as the backbone network.

Do the adapted embeddings improve the pre-adapted embeddings? We report few-shot classification results by
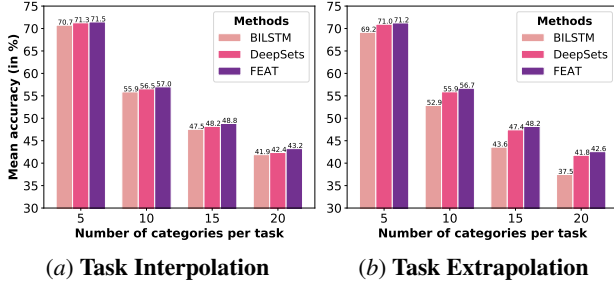
Figure 6: **Interpolation** and **Extrapolation** of few-shot tasks. We train different embedding adaptation models on *5-shot* 20-way or 5-way classification tasks and evaluate models on unseen tasks with different number of classes (N={*5, 10, 15, 20*}). It verifies both the interpolation and extrapolation ability of FEAT on a varying number of ways in few-shot classification.

Table 10: Ablation studies on the position to average the same-class embeddings when there are multiple shots per class in FEAT (tested on the 5-Way tasks with different numbers of shots). "Pre-Avg" and "Post-Avg" means we get the embedding center for each class before or after the set-to-set transformation, respectively.

Setups →	Pre-Avg	Post-Avg
5	$71.61_{\pm 0.16}$	$70.70_{\pm 0.16}$
15	$77.76_{\pm 0.14}$	$76.58_{\pm 0.14}$
30	$79.66_{\pm 0.13}$	$78.77_{\pm 0.13}$

Table 11: Ablation studies on the number of heads in the Transformer of FEAT (with number of layers fixes to one).

Setups →	1-Shot 5-Way	5-Shot 5-Way
1	$55.15_{\pm 0.20}$	$71.57_{\pm 0.16}$
2	$54.91_{\pm 0.20}$	$71.44_{\pm 0.16}$
4	$55.05_{\pm 0.20}$	$71.63_{\pm 0.16}$
8	$55.22_{\pm 0.20}$	$71.39_{\pm 0.16}$

Table 12: Ablation studies on the number of layers in the Transformer of FEAT (with number of heads fixes to one).

Setups →	1-Shot 5-Way	5-Shot 5-Way
1	$55.15_{\pm 0.20}$	$71.57_{\pm 0.16}$
2	$55.42_{\pm 0.20}$	$71.44_{\pm 0.16}$
3	$54.96_{\pm 0.20}$	$71.63_{\pm 0.16}$

using the pre-adapted embeddings of support data (*i.e.*, the embedding before adaptation), against those using adapted embeddings, for constructing classifiers. Table 9 shows that task-specific embeddings after adaptation improves over task-agnostic embeddings in few-shot classifications.

Can FEAT possesses the characteristic of the set function? We test three set-to-set transformation implementations, namely the BILSTM, the DEEPSETS, and the Transformer (FEAT), w.r.t. two important properties of the set function, *i.e.*, task interpolation and task extrapolation. In particular, the few-shot learning model is first trained with 5-shot 20-way tasks. Then the learned model is required to evaluate different 5-shot tasks with $N = \{5, 10, 15, 20\}$ (Extrapolation). Similarly, for interpolation, the model is trained with 5-shot 20-way tasks in advance and then evaluated on the previous multi-way tasks. The classification change results can be found in Figure 6 (a) and (b). BILSTM cannot deal with the size change of the set, especially in the task extrapolation. In both cases, FEAT still gets improvements in all configurations of N.

When to average the same-class embeddings? When there is more than one instance per class, *i.e.* $M > 1$, we average the instances in the same class and use the class center to make predictions as in Eq. 10. There are two positions to construct the prototypes in FEAT — before the set-to-set transformation (Pre-Avg) and after the set-to-set transformation (Post-Avg). In Pre-Avg, we adapt the embeddings of the centers, and a test instance is predicted based on its distance to the nearest adapted center; while in Post-Avg, the instance embeddings are adapted by the set-to-set function first, and the class centers are computed based on the adapted instance embeddings. We investigate the two choices in Table 10, where we fix the number of ways to 5 ($N = 5$) and change the number of shots (M) among $\{5, 15, 30\}$. The results demonstrate the Pre-Avg version performs better than the Post-Avg in all cases, which shows a more precise input of the set-to-set function by averaging the instances in the same class leads to better results. So we use the Pre-Avg strategy as a default option in our experiments.

Will deeper and multi-head transformer help? In our current implementation of the set-to-set transformation function, we make use of a shallow and simple transformer, *i.e.*, one layer and one head (set of projection). From [47], the transformer can be equipped with complex components using multiple heads and deeper stacked layers. We evaluate this augmented structure, with the number of attention heads increases to 2, 4, 8, as well as with the number of layers increases to 2 and 3. As in Table 11 and Table 12, we empirically observe that more complicated structures do not result in improved performance. We find that with more layers of transformer stacked, the difficulty of optimization increases and it becomes harder to train models until their convergence. Whilst for models with more heads, the models seem to over-fit heavily on the training data, even with the usage of auxiliary loss term (like the contrastive loss in

Table 13: Ablation studies on effects of the contrastive learning of the set-to-set function on FEAT.

Setups →	1-Shot 5-Way	5-Shot 5-Way
$\lambda = 10$	$53.92_{\pm 0.20}$	$70.41_{\pm 0.16}$
$\lambda = 1$	$54.84_{\pm 0.20}$	$71.00_{\pm 0.16}$
$\lambda = 0.1$	$\mathbf{55.15}_{\pm 0.20}$	$\mathbf{71.61}_{\pm 0.16}$
$\lambda = 0.01$	$54.67_{\pm 0.20}$	$71.26_{\pm 0.16}$

Table 14: Ablation studies on the prediction strategy (with cosine similarity or euclidean distance) of FEAT.

Setups →	1-Shot 5-Way		5-Shot 5-Way	
Backbone →	ConvNet	ResNet	ConvNet	ResNet
Cosine Similarity-based Prediction				
FEAT	$54.64_{\pm 0.20}$	$66.26_{\pm 0.20}$	$71.72_{\pm 0.16}$	$81.83_{\pm 0.15}$
Euclidean Distance-based Prediction				
FEAT	$55.15_{\pm 0.20}$	$66.78_{\pm 0.20}$	$71.61_{\pm 0.16}$	$82.05_{\pm 0.14}$

our approach). It might require some careful regularizations to prevent over-fitting, which we leave for future work.

The effectiveness of contrastive loss. Table 13 show the few-shot classification results with different weight values (λ) of the contrastive loss term for FEAT. From the results, we can find that the balance of the contrastive term in the learning objective can influence the final results. Empirically, we set $\lambda = 0.1$ in our experiments.

The influence of the prediction strategy. We investigate two embedding-based prediction ways for the few-shot classification, *i.e.*, based on the cosine similarity and the negative euclidean distance to measure the relationship between objects, respectively. We compare these two choices in Table 14. Two strategies in Table 14 only differ in their similarity measures. In other words, with more than one shot per class in the task training set, we average the same class embeddings first, and then make classification by computing the cosine similarity or the negative euclidean distance between a test instance and a class prototype. During the optimization, we tune the logits scale temperature for both these methods. We find that using the euclidean distance usually requires small temperatures (*e.g.*, $\gamma = \frac{1}{64}$) while a large temperature (*e.g.*, $\gamma = 1$) works well with the normalized cosine similarity. The former choice achieves a slightly better performance than the latter one.

D.3. Few-Shot Domain Generalization

We show that FEAT learns to adapt *the intrinsic structure of tasks*, and **generalize across domains**, *i.e.*, predicting test instances even when the visual appearance is changed.

Table 15: Cross-Domain 1-shot 5-way classification results of the FEAT approach.

	$\mathbf{C \to C}$	$\mathbf{C \to R}$	$\mathbf{R \to R}$
Supervised	$34.38_{\pm 0.16}$	$29.49_{\pm 0.16}$	$37.43_{\pm 0.16}$
ProtoNet	$35.51_{\pm 0.16}$	$29.47_{\pm 0.16}$	$37.24_{\pm 0.16}$
FEAT	$\mathbf{36.83}_{\pm 0.17}$	$\mathbf{30.89}_{\pm 0.17}$	$\mathbf{38.49}_{\pm 0.16}$

Table 16: Results of models for transductive FSL with ConvNet backbone on *Mini*ImageNet. We cite the results of Semi-ProtoNet and TPN from [38] and [34] respectively. For TEAM [34], the authors do not report the confidence intervals, so we set them to 0.00 in the table. FEAT† and FEAT‡ adapt embeddings with the joint set of labeled training and unlabeled test instances, while make prediction via ProtoNet and Semi-ProtoNet respectively.

Setups →	1-Shot 5-Way	5-Shot 5-Way
Standard		
ProtoNet	$52.61_{\pm 0.20}$	$71.33_{\pm 0.16}$
FEAT	$55.15_{\pm 0.20}$	$71.61_{\pm 0.16}$
Transductive		
Semi-ProtoNet [38]	$50.41_{\pm 0.31}$	$64.39_{\pm 0.24}$
TPN [30]	$55.51_{\pm 0.84}$	$69.86_{\pm 0.67}$
TEAM [34]	$56.57_{\pm 0.00}$	$72.04_{\pm 0.00}$
Semi-ProtoNet (Ours)	$55.50_{\pm 0.10}$	$71.76_{\pm 0.08}$
FEAT†	$56.49_{\pm 0.16}$	$72.65_{\pm 0.20}$
FEAT‡	$\mathbf{57.04}_{\pm 0.16}$	$\mathbf{72.89}_{\pm 0.20}$

Setups. We train a few-shot learning model in the standard domain and evaluate it with cross-domain tasks, where the N-categories are aligned but domains are different. In detail, a model is trained on tasks from the "Clipart" domain of OfficeHome dataset [48], then the model is required to generalize to both "Clipart (**C**)" and "Real World (**R**)" instances. In other words, we need to classify complex real images by seeing only a few sketches, or even based on the instances in the "Real World (**R**)" domain.

Results. Table 15 gives the quantitative results. Here, the "supervised" refers to a model trained with standard classification and then is used for the nearest neighbor classifier with its penultimate layer's output feature. We observe that ProtoNet can outperform this baseline on tasks when evaluating instances from "Clipart" but not ones from "real world". However, FEAT can improve over "real world" few-shot classification even only seeing the support data from "Clipart". Besides, when the support set and the test set of the target task are sampled from the same but new domains, *e.g.*, the training and test instances both come from "real world", FEAT also improves the classification accuracy w.r.t. the baseline methods. It verifies the domain generalization ability of the FEAT approach.

D.4. Additional Discussions on Transductive FSL

We list the results of the transductive few-shot classification in Table 16, where the unlabeled test instances arrive simultaneously, so that the common structure among the unlabeled test instances could be captured. We compare with three approaches, Semi-ProtoNet [38], TPN [30], and TEAM [34]. Semi-ProtoNet utilizes the unlabeled instances to facilitate the computation of the class center and makes predictions similar to the prototypical network; TPN meta learns a label propagation way to take the unlabeled instances relationship into consideration; TEAM explores the pairwise constraints in each task, and formulates the embedding adaptation into a semi-definite programming form. We cite the results of Semi-ProtoNet from [38], and cite the results of TPN and TEAM from [34]. We also re-implement Semi-ProtoNet with our pre-trained backbone (the same pre-trained ConvNet weights as the standard few-shot learning setting) for a fair comparison.

In this setting, our model leverages the unlabeled test instances to augment the transformer as discussed in § B.4 and the embedding adaptation takes the relationship of all test instances into consideration. Based on the adapted embedding by the joint set of labeled training instances and unlabeled test instances, we can make predictions with two strategies. First, we still compute the center of the labeled instances, while such adapted embeddings are influenced by the unlabeled instances (we denote this approach as $\mathrm{FEAT}^{\dagger}$, which works the same way as standard FEAT except the augmented input of the embedding transformation function); Second, we consider to take advantage of the unlabeled instances and use their adapted embeddings to construct a better class prototype as in Semi-ProtoNet (we denote this approach as $\mathrm{FEAT}^{\ddagger}$).

By using more unlabeled test instances in the transductive environment, $\mathrm{FEAT}^{\dagger}$ achieves further performance improvement compared with the standard FEAT, which verifies the unlabeled instances could assist the embedding adaptation of the labeled ones. With more accurate class center estimation, $\mathrm{FEAT}^{\ddagger}$ gets a further improvement. The performance gain induced by the transductive FEAT is more significant in the one-shot learning setting compared with the five-shot scenario, since the helpfulness of unlabeled instance decreases when there are more labeled instances.

D.5. More Generalized FSL Results

Here we show the full results of FEAT in the generalized few-shot learning setting in Table 17, which includes both the 1-shot and 5-shot performance. All methods are evaluated on instances composed by SEEN classes, UNSEEN classes, and both of them (COMBINED), respectively. In the 5-shot scenario, the performance improvement mainly comes from the improvement of over the UNSEEN tasks.

Table 17: Results of generalized FEAT with ConvNet backbone on *Mini*ImageNet. All methods are evaluated on instances composed by SEEN classes, UNSEEN classes, and both of them (COMBINED), respectively.

Measures →	SEEN	UNSEEN	COMBINED
1-shot learning			
ProtoNet	$41.73_{\pm 0.03}$	$48.64_{\pm 0.20}$	$35.69_{\pm 0.03}$
FEAT	$\mathbf{43.94}_{\pm 0.03}$	$\mathbf{49.72}_{\pm 0.20}$	$\mathbf{40.50}_{\pm 0.03}$
5-shot learning			
ProtoNet	$41.06_{\pm 0.03}$	$64.94_{\pm 0.17}$	$38.04_{\pm 0.02}$
FEAT	$\mathbf{44.94}_{\pm 0.03}$	$\mathbf{65.33}_{\pm 0.16}$	$\mathbf{41.68}_{\pm 0.03}$
Random Chance	1.56	20.00	1.45

Table 18: The top-5 low-shot learning accuracy over all classes on the large scale ImageNet [39] dataset (w/ ResNet-50).

UNSEEN	1-Shot	2-Shot	5-Shot	10-Shot	20-Shot
ProtoNet [43]	49.6	64.0	74.4	78.1	80.0
PMN [52]	53.3	65.2	75.9	80.1	82.6
FEAT	**53.8**	**65.4**	**76.0**	**81.2**	**83.6**
All	1-Shot	2-Shot	5-Shot	10-Shot	20-Shot
ProtoNet [43]	61.4	71.4	78.0	80.0	81.1
PMN [52]	64.8	72.1	78.8	81.7	83.3
FEAT	**65.1**	**72.5**	**79.3**	**82.1**	**83.9**
All w/ Prior	1-Shot	2-Shot	5-Shot	10-Shot	20-Shot
ProtoNet [43]	62.9	70.5	77.1	79.5	80.8
PMN [52]	63.4	70.8	77.9	80.9	82.7
FEAT	**63.8**	**71.2**	**78.1**	**81.3**	**83.4**

D.6. Large-Scale Low-Shot Learning

Similar to the generalized few-shot learning, the large-scale low-shot learning [12, 14, 52] considers the few-shot classification ability on both SEEN and UNSEEN classes on the full ImageNet [39] dataset. There are in total 389 SEEN classes and 611 UNSEEN classes [14]. We follow the setting (including the splits) of the prior work [14] and use features extracted based on the pre-trained ResNet-50 [15]. Three evaluation protocols are evaluated, namely the top-5 few-shot accuracy on the UNSEEN classes, on the combined set of both SEEN and UNSEEN classes, and the calibrated accuracy on weighted by selected set prior on the combined set of both SEEN and UNSEEN classes. The results are listed in Table 18. We observe that FEAT achieves better results than others, which further validates FEAT's superiority in generalized classification setup, a large scale learning setup.