Learning Embedding Adaptation for Few-Shot Learning

Han-Jia Ye*
Nanjing University
Nanjing, China, 210023
yehj@lamda.nju.edu.cn

Hexiang Hu
USC
Los Angeles, CA 90089
hexiangh@usc.edu

De-Chuan Zhan
Nanjing University
Nanjing, China, 210023
zhandc@lamda.nju.edu.cn

Fei Sha †
Netflix
Los Angeles, CA 90028
fsha@netflix.com

Abstract

*Learning with limited data is a key challenge for visual recognition. 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 is the target task. In this paper, we propose a novel approach to adapt the embedding model to the target classification task, yielding embeddings that are task-specific and are discriminative. To this end, we employ a type of self-attention mechanism called Transformer to transform the embeddings from task-agnostic to task-specific by focusing on relating instances from the test instances to the training instances in both seen and unseen classes. We verify the effectiveness of our model 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 large scale low-shot learning). Our approach archived consistent improvements over baseline models and previous state-of-the-art methods.*

1. Introduction

Learning visual recognition systems with deep learning architectures [17, 25, 46] often starts with annotating a large number of labeled images from a set of pre-defined visual categories [8, 33, 59]. Despite its utility, such a paradigm encounters difficulty in use cases where one needs to learn new visual concepts with limited annotation. For instance, in the visual search of merchandises, the search algorithm has access only to a limited set of stock photos (provided by the merchants) for newly released products.

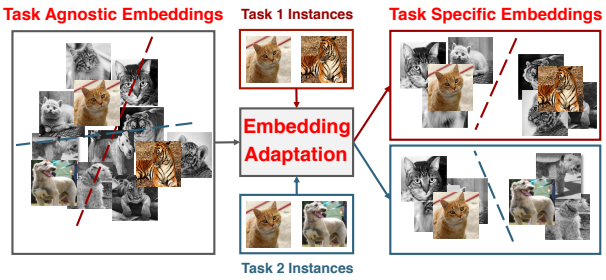


Figure 1: An illustration of the Embedding Adaptation in few-shot learning framework. Here, the colorful images are training instances and gray-dot are test instances. Instead of using task-agnostic embedding of each training instance to perform nearest neighbor classification, we propose to adapt embeddings to fit each few-shot learning task. This makes embeddings "task-specific".

To this end, few-shot learning has emerged as a promising approach in tackling this challenge [10, 26, 27, 30, 53]. Concretely, few-shot visual recognition 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 only 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 learning [44, 47, 50, 53] usually learn a discriminative instance embedding model from the SEEN categories, and apply the embedding model to vi-

*This work is mostly performed while Han-Jia Ye was a visiting scholar at USC.

†On leave from University of Southern California (feisha@usc.edu).

s 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 do suffer from an important limitation. Assuming a common embedding space implies that the discovered knowledge – discriminative visual features – on the SEEN classes are equally effective for *any* 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 lacking in the current approaches for few-shot learning 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. Figure 1 schematically illustrates this notion.

Towards this, we propose a new few-shot learning approach in this paper. The approach implements the above-mentioned adaptation strategy. Specifically, the adaptation algorithm transforms the embedding models derived from the SEEN classes, leveraging the few labeled instances in the target task. The transformation is obtained with a self-attention architecture called Transformer [31, 51], widely used in natural language processing and more recently in computer vision [37]. The self-attention mechanism enables learning the embedding transformation by considering the relationship among the labeled instances in the target task as well as considering their relationship with the test instances. (More detailed discussions are in Section 4.)

Our contribution consists of three parts. In Section 4, we describe a meta-learning flavored general framework for adapting representation to specific tasks – the representation can be learned with any existing approaches. We then evaluate the proposed approach on the standard few-shot learning benchmark datasets (*e.g.* *MiniImageNet* and *CUB200*) and demonstrate superior few-shot learning performances (Section 5.2). Finally, we consider four extended few-shot learning tasks (such as cross-domain, transductive, generalized few-shot learning, and large-scale low-shot learning), which are also important real-world use cases and show that our approach can outperform baseline algorithms under these settings too.

2. Related Work

Learning with limited annotation has been drawing a lot of interests in challenging application scenarios where traditional supervised learning approaches demand a large amount of labeled data. A standard approach is to pre-train models on datasets with ample labels and fine tune the models with a small amount of labeled data from the target task. However, the few-shot learning setting we investigate in this paper has more meager amount of labels (such as 1 or 5 instances per visual concept). Thus, the pre-train strategy often fails, by either significantly overfitting on the target task or updating the pre-trained models very little (and not being able to incorporate inductive bias from the target task).

Zero-shot learning (ZSL) is closely related to few-shot learning [1, 5, 28]. Similarly, ZSL distinguishes SEEN and UNSEEN classes. The main difference is that ZSL does not provide visual exemplars for the UNSEEN classes. Instead, the target tasks provide the semantic connections among the visual concepts in both SEEN and UNSEEN ones. Through those semantic relations, visual classifiers optimized on the SEEN classes are transferred to the UNSEEN ones.

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, 14, 29, 35, 39, 44], or the classifier itself is directly meta-predicted by the new task data [38, 56].

Another line of approach has focused on learning generalizable instance embeddings [1, 5, 6, 19, 24, 34, 45, 50, 53] 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 [24] first validated the importance of embedding in one-shot learning, whilst [53] 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 prototype [11, 47], decision ranking [50], and similarity comparison [49]. Most recently, [12] utilizes the graph convolution network [23] 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

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.

3. Learning Embedding for Task-agnostic Few-Shot Learning

In the standard formulation of few-shot learning (FSL) [10, 53], 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 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 the 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 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. Each sampling gives rise to a task to classify $\mathbf{x}^{\mathcal{S}}_{\mathbf{test}}$ into one of the N SEEN classes, by learning $f(\cdot)$ from $\mathcal{D}^{\mathcal{S}}_{\mathbf{train}}$ composed of the labeled instances on the same set of seen classes. This is similar to the meta-learning framework [10, 53]. Formally, the function $f(\cdot)$ is learnt to minimize the averaged error over those sampled tasks

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

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

We consider the approach based on learning embeddings for FSL [47, 53] (see Figure 2 (a) for a 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 is to apply the nearest neighbor classifiers in this space:

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

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*.

Algorithm 1 Training strategy of embedding adaptation

Require: Seen class set $\mathcal{S}$

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

4. Adapting Embedding for Task-specific Few-Shot Learning

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

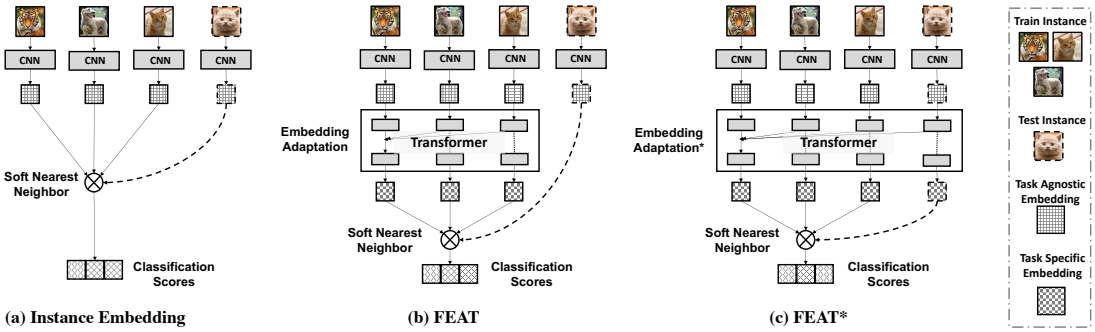
To this end, we introduce an adaption step where the embedding function $\phi_{\mathbf{x}}$ (more precisely, its values on instances) is transformed. This transformation is a set-to-set function as the instances are bags, or sets without orders, requiring the function to output the set of refined instance embeddings while being *permutation-invariant*. Concretely, we aim to learn

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

where $\mathcal{X}_{\mathbf{train}}$ is a set of all the instances in the training set $\mathcal{D}_{\mathbf{train}}$ for the target task. With *adapted* embedding $\psi_{\mathbf{x}}$, the test instance $\mathbf{x}_{\mathbf{test}}$ can be classified by computing nearest neighbors w.r.t. $\mathcal{D}_{\mathbf{train}}$:

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

Our approach is generally applicable to different types of task-agnostic embedding function $\mathbf{E}$ and similarity measure



(a) Instance Embedding (b) FEAT (c) FEAT*

Figure 2: Illustration of the proposed **F**ew-Shot **E**mbedding **A**daptation **T**ransformer (FEAT). Existing methods usually use the same embedding function **E** for all tasks. In our approach, our proposed architecture adapts the embeddings to each target few-shot learning task. Moreover, FEAT* performs joint embedding adaptation on both support instances and the test instance.

$\mathbf{sim}(\cdot,\cdot)$, e.g., the (normalized) cosine similarity used in matching network [53] (MatchNet) or the negative distance used in prototypical network [47] (ProtoNet).

Both the embedding function **E** and the set transformation function **T** are optimized over synthesized FSL tasks sampled from $\mathcal{D}^{\mathcal{S}}$, sketched in Alg. 1. Please note the key difference from conventional embedding-based FSL in *line 4 to line 8* where the embeddings are transformed.

4.2. Transformer as a Set Function for Adaptation

In this section, we describe the details about our choice of the set transformation function **T** and explain how it is implemented and optimized. We denote it as **F**ew-Shot **E**mbedding **A**daptation **T**ransformer (FEAT).

We use the *Transformer* architecture [51] to implement **T**. In particular, we employ self-attention mechanism [31, 51] to transform each instance embedding with consideration to its contextual instances. Note that it naturally satisfies the desired properties of **T** because it outputs refined instance embeddings and is permutation invariant.

Transformer is a store of triplets in the form of (query, key, and value). It computes what is the right value for a query point — the query is first matched against a list of keys where each key has a value. The final value is then returned as the sum of all the values *weighted* by the proximity of the key to the query point. In our implementation, the query point is an instance whose embeddings we would like to find. The keys are a set of instances whose embeddings are already known and are encoded in the values. Specifically, following the original definitions in [51], $\mathcal{Q}$ denotes the set of query points with $\mathcal{K}$ for keys and $\mathcal{V}$ for values – we describe later how we define them.

To compute proximity and return values, those points are first linearly mapped into some space:

$$\begin{aligned} Q &= W_Q^\top \left[\, \phi_{\mathbf{x}_q};\quad \forall \mathbf{x}_q \in \mathcal{Q} \,\right] \in \mathbb{R}^{d\times|\mathcal{Q}|} \\ K &= W_K^\top \left[\, \phi_{\mathbf{x}_k};\quad \forall \mathbf{x}_k \in \mathcal{K} \,\right] \in \mathbb{R}^{d\times|\mathcal{K}|} \\ V &= W_V^\top \left[\, \phi_{\mathbf{x}_v};\quad \forall \mathbf{x}_v \in \mathcal{V} \,\right] \in \mathbb{R}^{d\times|\mathcal{V}|} \end{aligned} \tag{5}$$

A query point $\mathbf{x}_q \in \mathcal{Q}$'s similarity to the keys $\mathcal{K}$ is then computed as "attention":

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

These attentions are then used as weights (after normalized to sum to 1) to compute the final embedding for $\mathbf{x}_q$:

$$\psi_{\mathbf{x}_q} = \phi_{\mathbf{x}_q} + \sum_k \alpha_{qk} V_{:,k} \tag{6}$$

$V_{:,k}$ is the k-th column of V.

Choices of the key and value sets. In the standard FSL setup, we have $\mathcal{Q} = \mathcal{K} = \mathcal{V} = \mathcal{X}_{\mathbf{train}}$. Note that each of them has a different projection matrix. We refer this as FEAT. We also consider incorporating the test instances from the task to exploit its relation to the training examples:

$$\mathcal{Q} = \mathcal{K} = \mathcal{V} = \mathcal{X}_{\mathbf{train}} \cup \mathbf{x}_{\mathbf{test}} \tag{7}$$

Note that the embedding of the training set is adapted specifically for each test instance, which we refer as FEAT*. Similar setups can extend FEAT to deal with more unlabeled test instances (transductive FSL) or the instances from both SEEN and UNSEEN classes (generalized FSL). We refer audience to Section 5.3 for details.

4.3. Contrastive Learning of Set Functions.

To facilitate the learning of embedding adaptation transformer, we apply a contrastive objective in addition to the general objective. 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 $\mathcal{D}^{\mathcal{S}}_{\mathbf{train}} \cup \mathcal{D}^{\mathcal{S}}_{\mathbf{train}}$, 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:

$$\mathcal{L}(\hat{\mathbf{y}}_{\mathbf{test}}, \mathbf{y}_{\mathbf{test}}) = \ell(\hat{\mathbf{y}}_{\mathbf{test}}, \mathbf{y}_{\mathbf{test}}) + \lambda \cdot \ell\left(\mathbf{softmax}\left(\mathbf{sim}(\psi'_{\mathbf{x}_{\mathbf{test}}}, \mathbf{c}_n)\right), \mathbf{y}_{\mathbf{test}}\right) \tag{8}$$

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 two backbone convolutional networks as instance embedding function $\mathbf{E}$: 1) A four-layer convolution network (ConvNet) [47, 50, 53] and 2) A residual network (ResNet) [36, 38, 44], as instance embedding functions $\mathbf{E}$. As suggested by [36, 38, 44], we apply an additional pre-training stage for the backbone convolutional networks. We find such pre-training on the SEEN classes of each dataset is effective and can significantly accelerate the time until convergence for each method. Such pre-training is applied to both baselines and our approaches. We follow the architecture as presented in [51] to build our FEAT model. The hidden dimensions for the linear transformations in the FEAT model are set to be 64 for ConvNet and 640 for ResNet. We empirically observed that the shallow transformer (with one set of projection and one stacked layer) gives the best overall performance (see supplementary material for ablation study of this). During the training, stochastic gradient descent with Adam [22] optimizer is employed, with the initial learning rate set to be 1e-3. The learning rate for the pre-trained parameters is scaled by 0.1. We refer readers to supplementary material for complete details. Our implementation is made publicly available at https://github.com/Sha-Lab/FEAT.

5. Experiments

We validate the effectiveness of our proposed approach through applying it to a variety of tasks across datasets, including *standard few-shot image classification* (Section 5.2), *few-shot domain generalization* (Section 5.3.1), *transductive few-shot learning* (Section 5.3.2), and large-scale *generalized & low-shot image classification* (Section 5.3.3). The proposed approach consistently outperforms previous approaches and baseline models (refer to Table 5 and Table 3 for a brief overview).

5.1. Experimental Setups

Datasets. The *mini*ImageNet dataset [53] is a subset of the ImageNet [43] that includes a total number of 100 classes and 600 examples per class. We follow the setup provided by [39], and use 64 classes as SEEN categories, 16 and 20 as two sets of UNSEEN categories for model validation and evaluation respectively. In addition to this, we investigate the OfficeHome [52] 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 supplementary material for more details.

Evaluation protocols. Previous approaches [10, 47, 50] usually follow the original setting of [53] and evaluate the models on 600 sampled target tasks (15 test instances per class). In a later study [44], it was suggested that such an evaluation process can potentially introduce high variances. Therefore, we follow the new and more trustworthy evaluation setting [44] 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.

Baselines. We re-implement two task agnostic embedding methods — matching network [53] (**MatchNet**) and prototypical network [47] (**ProtoNet**) as the baseline model. As suggested by [36], we tune the scalar temperature carefully to scale logits of both approaches in our re-implementation. We empirically observe that ProtoNet outperforms MatchNet in almost all cases, especially on the 5-shot learning scenario (see Table 5). Therefore use ProtoNet for all our models to learn embeddings.

Now we describe *four variants* under our generic framework of learning embedding adaptation functions. They mainly differ in the choice of set functions.

- **BILSTM** uses bidirectional LSTM [18] to approximate the set transformation function.
- **BILSTM*** extends BILSTM by additionally inputing the test instance for joint embedding adaptation.
- **DEEPSETS** uses a permutation-invariant DeepSets [58] as the set transformation function. It is worth noting that DEEPSETS aggregates the training instances into a holistic *set vector*. We learn a multi-layer perception (MLP) that takes both the current instance and *set vector* together to compute the output instance embeddings.
- **DEEPSETS*** extends DEEPSETS in a similar manner, which also leverages the test instance to compute the set embedding (shown as Figure 2 (c)).

Please refer to supplementary material for complete architecture details to the above baseline approaches.

Table 1: Few-shot classification accuracy on *Mini*ImageNet. The superscript "⋆" indicates joint embedding adaptation of test instance and training instances. ★ Note that LEO [44]'s results uses Wide ResNet (WRN) and comparison to other approach might be unfair. FEAT and FEAT⋆ can achieve **65.10**% and **66.69**% with WRN backbone in the 1-shot 5-way setup.

Setups →	1-Shot 5-Way		5-Shot 5-Way	
Backbone →	ConvNet	ResNet	ConvNet	ResNet
MatchNet [53]	43.40	-	51.09	-
MAML [10]	48.70	-	63.11	-
ProtoNet [47]	49.42	-	68.20	-
RelationNet [49]	51.38	-	67.07	-
PFA [38]	54.53	59.60	67.87	73.74
TADAM [36]	-	58.50	-	76.70
LEO [44] ★	-	61.76	-	77.59
Instance embedding				
MatchNet	52.87	60.66	67.49	75.05
ProtoNet	52.61	61.40	71.33	76.56
Embedding adaptation				
BILSTM	52.13	55.73	69.15	69.81
BILSTM⋆	54.10	55.73	69.39	69.93
DEEPSETS	54.41	60.02	70.96	77.30
DEEPSETS⋆	53.02	60.06	70.25	77.95
Ours: FEAT	55.15	**62.96**	71.61	**78.49**
Ours: FEAT⋆	**55.75**	62.60	**72.17**	78.06

5.2. Standard Few-Shot Image Classification

We compare our proposed FEAT method with baselines as well as previous methods on the standard *Mini*ImageNet [53] benchmark, and then perform detailed analysis on the ablated models. We have also experimented with CUB [54] dataset for few-shot classification, which shows similar observation. Due to the space limit, we refer readers to the supplementary material for CUB results.

5.2.1 Main Results

Results on *Mini*ImageNet (Table 5) show that FEAT outperforms the baselines and previous state-of-the-art methods. Note that the comparisons with early works, *i.e.*, MatchNet [53], MAML [10] and ProtoNet [47] might be unfair as they do not incorporate the pre-training stage. More recent works such as PFA [38], LEO [44], and TADAM [36] all rely on such strategy. To make fair comparisons, we re-implement MatchNet and ProtoNet with pre-training, and both re-implementations achieve better results than previously reported ones. Note that in the Table 5 the results of LEO [44] is based on Wide ResNet (WRN) and comparison to other approach might be unfair. FEAT and FEAT⋆ can achieve **65.10**% and **66.69**% with WRN backbone in the 1-shot 5-way setup. Please refer to supplementary for a complete study on this.

Table 2: After embedding adaptation, FEAT improves w.r.t. the pre-adaptation embeddings a lot for Few-shot classification.

	1-Shot 5-Way	5-Shot 5-Way
Pre-Adaptation	51.60	70.40
Post-Adaptation	55.15	71.61

Besides, we observe that both BILSTM and DEEPSETS do not achieve consistently performance improvement over baseline ProtoNet. In contrast, FEAT and its variants FEAT⋆ consistently improve ProtoNet in all cases, outperforming all previous approaches. This might due to the fact that Transformer naturally implements the permutation invariance set-to-set adaptation function. Note that with the joint embedding adaptation of test and support instances, FEAT⋆ slightly improves over FEAT on ConvNet.

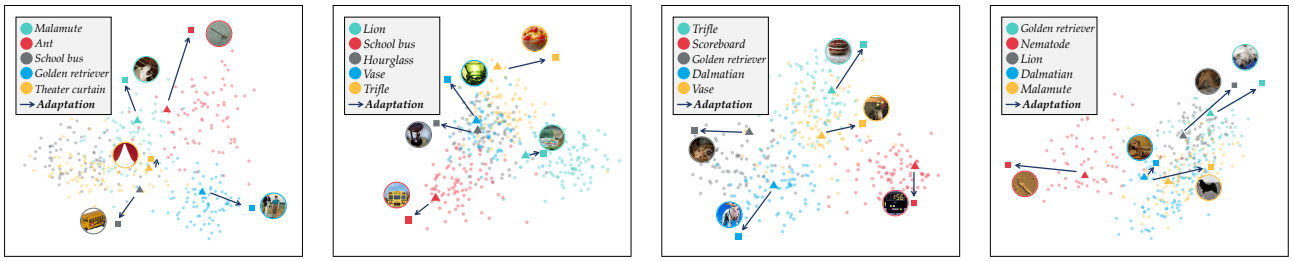
5.2.2 Ablation Studies and Analysis

We perform further analysis for FEAT and its ablated variants on the *Mini*ImageNet dataset with ConvNet backbone.

Analysis #1. We first investigate **whether embedding adaptation in FEAT actually improves the task-agnostic embeddings**. To this purpose, we report few-shot classification results by using pre-adaptation embeddings of support data, against those using post-adapted embeddings. Results in Table 2 show that with the task-specific embedding adaptation, both 1-shot and 5-shot classifications improve.

Analysis #2. We next perform analysis to **compare different implementations of set-to-set functions**. We measure the capability of interpolation and extrapolation for the set function. To do so, we train each variant of embedding adaptation functions with either 1-shot 20-way or 1-shot 5-way tasks, and measure the performance change as a function to the number of categories. We report the mean accuracies evaluated on few-shot classification with $N = \{5, 10, 15, 20\}$ classes, and show it as Figure 4. Surprisingly, we observe that FEAT *achieves almost the same numerical performances in both extrapolation and interpolation scenarios*, which demonstrates its capability of learning 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. BILSTM performs the worst in both cases, as it is by design not permutation invariant and might fit the arbitrary dependency between instances.

Analysis #3. We further look into the question **"How is the embedding adaptation qualitatively affects the embeddings"**. To achieve this goal, we sample four few-



(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 3: Qualitative visualization of embedding adaptation procedure on test tasks. Each figure shows the locations of supports' embeddings before and after adaptation. Values below the plots are the 1-shot 5-way classification accuracy before and after the the adaptation in FEAT . 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 fits the test data of its categories. Refer to text for more details. (Best view in colors!)

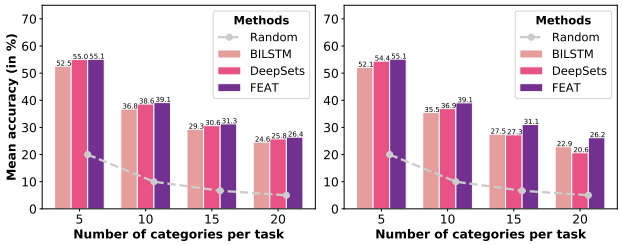
shot learning tasks and learn a principal component analysis (PCA) (that projects embeddings into 2-D space) using the instance embeddings of test data. We then apply this learned PCA projection to both the support set's embeddings of both pre-adaptation and post-adaptation. The results are shown in Fig. 3. In three out of four examples, post-adaptation embeddings of FEAT improves over 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 degenerate the performances, we observe that the embedding adaptation step has pushed two support embeddings "Golden Retriever" and "Lion" too close to each other. This visualization has qualitatively shown that the adaptation is crucial in obtaining superior performances and helps to contrast against task-agnostic embeddings.

Analysis #4. We study ablated model of FEAT to answer **whether it is beneficial to have more complicated Transformer**. We performed ablation study to test FEAT with more heads and layers in the transformer, which increases the complexity of $\mathbf{T}$ [51]. Empirically, we found that it results in similar performance when having more heads. Moreover, we observe that stacking more layers makes FEAT overfitting severely. Due to space limit, please refer to supplementary material for concrete numerical results.

5.3. Extended Few-Shot Learning Tasks

In this section, we evaluate FEAT on four different few-shot learning tasks. Specifically, they are cross-domain FSL, transductive FSL [32, 41], generalized FSL [7] and low shot learning [15, 55]. We begin with a brief overview.

Cross-Domain FSL assumes that examples in UNSEEN support and test set can come from the different domains, e.g., sampled from different distributions [9, 21]. The example of this task can be found in Figure 5. This task requires a model to recognize the intrinsic property rather than appearance of objects, and requires analogical recognition.



(*a*) **Task Interpolation** (*b*) **Task Extrapolation**

Figure 4: Interpolation and **Extrapolation** of few-shot tasks. We train different embedding adaptation models on 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.

Transductive FSL. The key difference between standard and transductive scenario [32, 41] is whether test instances arrive one at a time or all simultaneously. The later 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 from UNSEEN categories.

Generalized FSL & Low-shot Learning. 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 [7, 16, 40, 55]. 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. *low-shot learning* scales up the number of test classes based on the *generalized FSL*.

5.3.1 Few-Shot Domain Generalization

We show that FEAT learns to adapt *the intrinsic structure of tasks*, and **generalize across domains**, *i.e.*, predicting test

	C → C	C → R
Supervised	34.38±0.16	29.49±0.16
ProtoNet	35.51±0.16	29.47±0.16
FEAT	**36.83**±0.17	**30.89**±0.17

(a) Few-Shot Domain Generalization

	1-Shot	5-Shot
SemiProto [41]	50.41 ± 0.31	64.39 ± 0.24
TPN [32]	54.72 ± 0.84	69.25 ± 0.67
FEAT	**56.49** ± 0.21	**72.65** ± 0.16

(b) Transductive Few-shot Learning

	SEEN	UNSEEN	COMBINED
Random	1.56 ±0.00	20.00 ±0.00	1.45 ±0.00
ProtoNet	41.73±0.03	48.64±0.20	35.69±0.03
FEAT	**43.94**±0.03	**49.72**±0.20	**40.50**±0.03

(c) Generalized Few-shot Learning

Table 3: 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.

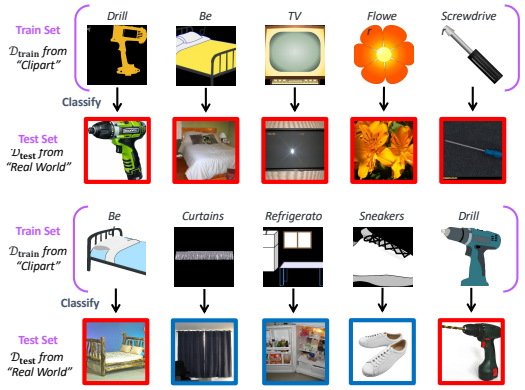


Figure 5: Qualitative results of **few-shot domain-generalization** for FEAT. Correctly classified examples are shown in **red** box and Incorrectly classified examples are shown in **blue** box. We visualize one task that FEAT succeeds (top) and one that fails (bottom).

instances even when the visual appearance is changed.

Setups. We train a 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 trained on tasks from the "Clipart" domain of OfficeHome dataset [52], 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 5 gives an overview of data).

Results. Table 3 (a) gives the quantitative results and Figure 5 qualitatively examines it. Here, the "supervised" refers to a model trained with standard classification and then is used for 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".

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 (details in Sec. 4.2), and then embedding adaptation takes relationship of all test instances into consideration. We evaluate this setting on the transductive protocol of *mini*ImageNet [41]. Please refer to supp. for details.

Results. We compare with two previous approaches, SemiProtoNet [41] and TPN [32]. The results are shown in Table 3 (b). We observe that FEAT improves its standard FSL performance (cf. Table 5) 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, and can effectively scale up to large scale low-shot learning.

Setups. In this scenario, we evaluate not only on classifying test instances from a N-way M-shot task from UNSEEN 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 3 (c). We observe that again FEAT outperforms baseline ProtoNet. To calibrate the prediction score on SEEN and UNSEEN classes [7, 55], 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.

Large-Scale Low-Shot Learning [13, 16, 55] is another generalized few-shot classification on the full ImageNet [42] dataset. There are in total 389 SEEN classes and 611 UNSEEN classes [16]. We follow the setting of prior work [55] and use features extracted based on the pre-trained ResNet50 [17]. Calibrated top-5 classification accuracy on $\mathcal{S} \cup \mathcal{U}$ with various number of shots are listed in Table 4. We observe that FEAT achieves better results

Table 4: The top-5 low-shot learning accuracy over all classes on the large scale ImageNet [42] dataset (w/ ResNet50).

All w/ Prior	1-Shot	2-Shot	5-Shot	10-Shot	20-Shot
ProtoNet	62.9	70.5	77.1	79.5	80.8
PMN [55]	63.4	70.8	77.9	80.9	82.7
FEAT	**63.8**	**71.2**	**78.1**	**81.3**	**83.4**

than others, which further validates FEAT's superiority in generalized classification setup, a large scale learning setup. Please refer supplementary for complete results and details.

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 **F**ew-shot **E**mbedding **A**daptation with **T**ransformer (FEAT), which customizes a task-specific embedding spaces via a self-attention architecture. The adapted embedding spaces leverage both the relationship between target task training instances and the relationship with test instances, which leads to discriminative instance representations. FEAT demonstrates the state-of-the-art performance, and its superiority can generalize to cross-domain, transductive and generalized few-shot classifications. Extension of FEAT with multiple modalities could be future work.

Acknowledgments We appreciate Liyu Chen, Bowen Zhang for their constructive feedbacks to an earlier version of this paper. This work is partially supported by DARPA# FA8750-18-2-0117, NSF IIS-1065243, 1451412, 1513966/1632803/1833137, 1208500, CCF-1139148, a Google Research Award, an Alfred P. Sloan Research Fellowship, gifts from Facebook and Netflix, ARO# W911NF-12-1-0241 and W911NF-15-1-0484, and China Scholarship Council.

References

[1] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. Label-embedding for attribute-based classification. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 819–826. IEEE, 2013.

[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 *Advances in Neural Information Processing Systems 29*, pages 3981–3989. Curran Associates, Inc., 2016.

[3] A. Antoniou, H. Edwards, and A. J. Storkey. How to train your MAML. *CoRR*, abs/1810.09502, 2018.

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

[5] S. Changpinyo, W.-L. Chao, B. Gong, and F. Sha. Synthesized classifiers for zero-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5327–5336, 2016.

[6] S. Changpinyo, W.-L. Chao, and F. Sha. Predicting visual exemplars of unseen classes for zero-shot learning. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3496–3505. IEEE, 2017.

[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 *Proceedings of the 14th European Conference on Computer Vision*, pages 52–68, Amsterdam, The Netherlands, 2016.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[9] N. Dong and E. P. Xing. Domain adaption in one-shot learning. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 573–588, 2018.

[10] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1126–1135, Sydney, Australia, 2017.

[11] H. Gao, Z. Shou, A. Zareian, H. Zhang, and S.-F. Chang. Low-shot learning via covariance-preserving adversarial augmentation networks. In *Advances in Neural Information Processing Systems 31*, pages 983–993. 2018.

[12] V. Garcia and J. Bruna. Few-shot learning with graph neural networks. *CoRR*, abs/1711.04043, 2017.

[13] S. Gidaris and N. Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, Salt Lake City, UT., 2018.

[14] L.-Y. Gui, Y.-X. Wang, D. Ramanan, and J. M. F. Moura. Few-shot human motion prediction via meta-learning. In *Proceedings of the 15th European Conference on Computer Vision*, pages 441–459, 2018.

[15] B. Hariharan and R. B. Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *IEEE International Conference on Computer Vision*, pages 3037–3046, Venice, Italy, 2017.

[16] B. Hariharan and R. B. Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *IEEE International Conference on Computer Vision*, pages 3037–3046, Venice, Italy, 2017.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of*

the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

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

[19] K. Hsu, S. Levine, and C. Finn. Unsupervised learning via meta-learning. *CoRR*, abs/1810.02334, 2018.

[20] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, Lille, France, 2015.

[21] B. Kang and J. Feng. Transferable meta learning across domains. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence*, pages 177–187, 2018.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[23] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

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

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[26] B. M. Lake, R. Salakhutdinov, J. Gross, and J. B. Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the 33th Annual Meeting of the Cognitive Science Society*, Boston, MA., 2011.

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

[28] C. H. Lampert, H. Nickisch, and S. Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465, 2014.

[29] Y. Lee and S. Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2933–2942, Stockholm, Sweden, 2018.

[30] F. Li, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

[31] 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.

[32] Y. Liu, J. Lee, M. Park, S. Kim, and Y. Yang. Transductive propagation network for few-shot learning. *CoRR*, abs/1805.10002, 2018.

[33] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten. Exploring the limits of weakly supervised pretraining. *ECCV*, 2018.

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

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

[36] B. N. Oreshkin, P. Rodríguez, and A. Lacoste. TADAM: task dependent adaptive metric for improved few-shot learning. *CoRR*, abs/1805.10123, 2018.

[37] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran. Image transformer. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4052–4061, Stockholm, Sweden, 2018.

[38] S. Qiao, C. Liu, W. Shen, and A. L. Yuille. Few-shot image recognition by predicting parameters from activations. *CoRR*, abs/1706.03466, 2017.

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

[40] M. Ren, R. Liao, E. Fetaya, and R. S. Zemel. Incremental few-shot learning with attention attractor networks. *CoRR*, abs/1810.07218, 2018.

[41] 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. *CoRR*, abs/1803.00676, 2018.

[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[43] 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. Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[44] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. *CoRR*, abs/1807.05960, 2018.

[45] T. R. Scott, K. Ridgeway, and M. C. Mozer. Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning. In *Advances in Neural Information Processing Systems 31*, pages 76–85. 2018.

[46] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[47] J. Snell, K. Swersky, and R. S. Zemel. Prototypical

networks for few-shot learning. In *Advances in Neural Information Processing Systems 30*, pages 4080–4090. Curran Associates, Inc., 2017.

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

[49] 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. *CoRR*, abs/1711.06025, 2017.

[50] E. Triantafillou, R. S. Zemel, and R. Urtasun. Few-shot learning through an information retrieval lens. In *Advances in Neural Information Processing Systems 30*, pages 2252–2262. Curran Associates, Inc., 2017.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 6000–6010. Curran Associates, Inc., 2017.

[52] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan. Deep hashing network for unsupervised domain adaptation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*, pages 5385–5394, 2017.

[53] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29*, pages 3630–3638. Curran Associates, Inc., 2016.

[54] 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.

[55] Y. Wang, R. B. Girshick, M. Hebert, and B. Hariharan. Low-shot learning from imaginary data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7278–7286, Salt Lake City, UT., 2018.

[56] 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. *CoRR*, abs/1805.04288, 2018.

[57] S. Zagoruyko and N. Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016*, York, UK, 2016.

[58] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems 30*, pages 3394–3404. Curran Associates, Inc., 2017.

[59] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Supplementary Material

A. Details of Baseline Methods

In this section, we describe two important embedding learning baselines *i.e*. Matching Network (MatchNet) [53] and Prototypical Network (ProtoNet) [47], to implement the prediction function $f(\mathbf{x_{test}}; \mathcal{D}_{\mathbf{train}})$ 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 $\mathcal{D}_{\mathbf{train}} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{NM}$ of a target M-shot N-way classification task, we can obtain the embedding of each training instance based on the function $\mathbf{E}$:

$$\phi(\mathbf{x}_i) = \mathbf{E}(\mathbf{x}_i),\ \forall \mathbf{x}_i \in \mathcal{X}_{\mathbf{train}} \tag{9}$$

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

$$\begin{aligned}\hat{\mathbf{y}}_{\mathbf{test}} &\propto \exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_i})\right) \cdot \mathbf{y}_i \\ &= \frac{\exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_i})\right)}{\sum_{\mathbf{x}_{i'} \in \mathcal{X}_{\mathbf{train}}} \exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_{i'}})\right)} \cdot \mathbf{y}_i \\ &\text{where } \forall (\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}_{\mathbf{train}}\end{aligned} \tag{10}$$

Here, MatchNet finds the most similar training instance to the test one, and assign the label of this 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 [36]. During the experiments, we tune this temperature value carefully, ranging from the reciprocal of $\{0.1, 1, 16, 32, 64, 128\}$.

The ProtoNet has two key differences comparing with the MatchNet. First, when $M > 1$ in the target task, ProtoNet uses the mean of the embeddings for all training instances as the class center (prototype) of each class to compute the nearest neighbor. In addition, it uses negative distance rather than cosine similarity as the distance metric to find the nearest neighbor:

$$\mathbf{c}_n = \frac{1}{M} \sum_{\mathbf{y}_i = n} \phi(\mathbf{x}_i),\ \forall n = 1, \ldots, N \tag{11}$$

$$\hat{\mathbf{y}}_{\mathbf{test}} = \frac{\exp\left(-\gamma \|\phi_{\mathbf{x_{test}}} - \mathbf{c}_n\|_2^2\right)}{\sum_{n=1}^{N} \exp\left(-\gamma \|\phi_{\mathbf{x_{test}}} - \mathbf{c}_n\|_2^2\right)} \mathbf{y}_n \tag{12}$$

Similar to the aforementioned scalar temperature for MatchNet, in Eq. 12 we also consider the scale γ. Here we abuse the notation by using $\mathbf{y}_i = n$ to enumerate the instances with label n.

B. Details of the Transformer and DeepSets

In this section, we provide details about two implementations of the embedding adaptation function, *i.e*. the DeepSets and the Transformer. The later one is the key component in our **F**ew-shot **E**mbedding **A**daptation **T**ransformer (FEAT) approach. Then we will introduce the setup for the transformer to extend to the instance-specific, transductive, and generalized Few-Shot Learning (FSL).

B.1. DeepSets for standard FSL

Deep sets [58] 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 besides the Transformer can be one that involves two component, 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:

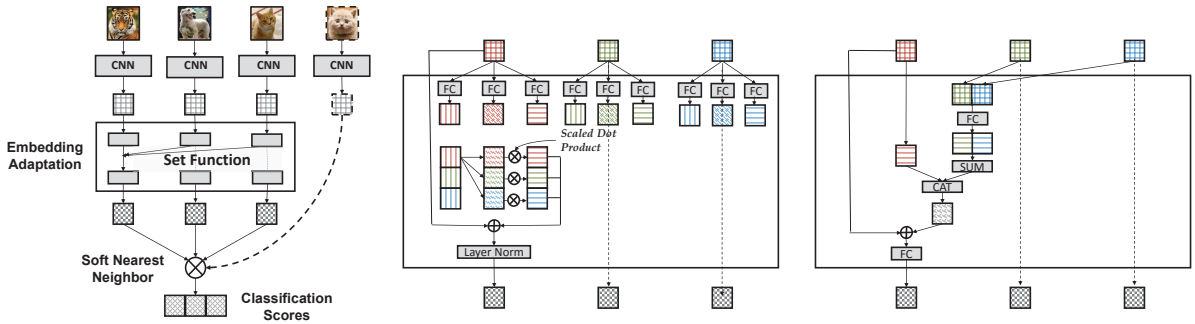
$$\psi(\mathbf{x}) = \phi(\mathbf{x}) + g([\phi(\mathbf{x}); \sum_{\mathbf{x}_{i'} \in \mathbf{x}^{\complement}} h(\phi(\mathbf{x}_{i'}))]) \tag{13}$$

In Eq. 13, 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 is first combined into a 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 each 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. 12. An illustration of the DeepSets notation in the embedding adaptation can be found in figure 6 (c).

B.2. Transformer for standard FSL

In this section, we describe in details about our **F**ew-Shot **E**mbedding **A**daptation **T**ransformer (FEAT) approach, specifically how to use the Transformer architecture [51] to implement the set-to-set function $\mathbf{T}$, where self-attention mechanism facilitates the instance embedding adaptation with the contextual embeddings consideration.

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



(a) FEAT (b) Transformer as the Set Function (c) DeepSets as the Set Function

Figure 6: Illustration of two embedding adpatation methods considered in the paper. (a) shows the main flow of **F**ew-Shot **E**mbedding **A**daptation **T**ransformer (FEAT), while (b) and (c) demonstrate the workflow of Transformer and DeepSets respectively.

the corresponding values of each key. The transformed input is served as a residual value which will be added over the input.

Basic Transformer. Following the definitions in [51], 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[1]. The points in sets are first projected by the corresponding projections

$$\begin{aligned} Q &= W_Q^\top \left[\ \phi_{\mathbf{x}_q};\quad \forall \mathbf{x}_q \in \mathcal{Q}\ \right] \in \mathbb{R}^{d'\times|\mathcal{Q}|} \\ K &= W_K^\top \left[\ \phi_{\mathbf{x}_k};\quad \forall \mathbf{x}_k \in \mathcal{K}\ \right] \in \mathbb{R}^{d'\times|\mathcal{K}|} \\ V &= W_V^\top \left[\ \phi_{\mathbf{x}_v};\quad \forall \mathbf{x}_v \in \mathcal{V}\ \right] \in \mathbb{R}^{d'\times|\mathcal{V}|} \end{aligned} \tag{14}$$

$|\mathcal{Q}|$, $|\mathcal{K}|$, 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":

$$\alpha_{qk} \propto \exp\{\big(\frac{\phi_{\mathbf{x}_q}^\top W_Q \cdot K}{\sqrt{d}}\big)\}; \forall \mathbf{x}_k \in \mathcal{K} \tag{15}$$

$$\alpha_{q,:} = \mathbf{softmax}\big(\frac{\phi_{\mathbf{x}_q}^\top W_Q \cdot K}{\sqrt{d}}\big) \in \mathbb{R}^{|\mathcal{K}|} \tag{16}$$

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 $\mathbf{x}_q$:

$$\tilde{\psi}_{\mathbf{x}_q} = \sum_k \alpha_{qk} V_{:,k} \tag{17}$$

$$\psi_{\mathbf{x}_q} = \tau\big(\phi_{\mathbf{x}_q} + W_{\mathbf{FC}}^\top \tilde{\psi}_{\mathbf{x}_q}\big) \tag{18}$$

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

$V_{:,k}$ 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 [48] and layer_norm [4]. The whole flow of transformer in our FEAT approach can be found in Fig. 6 (b). With the help of transformer, the embeddings of all training set instances are adapted (we denote this approach as FEAT). The test instance in a task can also be incorporated in this adaptation, where we augment the query, key, and value set with the specific test instance (we denote the test instance-specific embedding adaptation way as FEAT*).

Multi-Head Multi-Layer Transformer. Following [51], 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 the Section D.

B.3. Extension to transductive FSL

Facilitated by the flexible set transformer in Eq. 18, our adaptation approach can naturally be extended to both the transductive and generalized settings.

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

$$\mathcal{Q} = \mathcal{K} = \mathcal{V} = \mathcal{X}_{\mathbf{train}} \cup \mathcal{X}_{\mathbf{test}} \tag{19}$$

In this manner, the embedding adaptation procedure would also consider the structure among unlabeled test instances.

Contrastive attention learning (CAL). When the test instance is incorporated in the learning process, we leverage a contrastive loss that the two instances should be mapped close to each other if they are from the same class, which facilitates the learning of the linear mapping W_Q, W_K and W_V for the Transformer architecture. For $\mathbf{x}_q$ and $\mathbf{x}_k$ being mapped close to each other, the attention coefficients $\alpha_{qk'}$ should peak at the $k' = k$. Viewing the attention coefficients as a vector of probabilities, we thus reduce its KL divergence from the "ideal" proximity vector which is a binary vector having one at kth location. We add this term to the loss function

$$\mathcal{L}(\hat{\mathbf{y}}_{\mathbf{test}}, \mathbf{y}_{\mathbf{test}}) = \ell(\hat{\mathbf{y}}_{\mathbf{test}}, \mathbf{y}_{\mathbf{test}}) + \lambda \sum_{\mathbf{x}_q \in \mathcal{Q}} \sum_{\mathbf{y}_k = \mathbf{y}_q} \alpha_{qk} \log \alpha_{qk} \quad (20)$$

The influence of this regularizer can be found in the experiments (Section D.2).

C. Implementation Details

Backbone architecture. We consider three backbone convolutional networks as suggested in the literature as instance embedding functions $\mathbf{E}$ for the purpose of fair comparisons.

- **ConvNet.** Four-layer convolution network [47, 50, 53]. It contains 4 repeated convolutional blocks. In each block, there is a convolutional layer with 3×3 kernel, a Batch Normalization layer [20], a **ReLU**, and a Max pooling with size 2. We resize the input image to $84 \times 84 \times 3$, and 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 dimensionality 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.** Residual network [17, 36, 38]. We implement this large residual network exactly the same as the released code of [38].[2] Following the literature, we resize the input image to $80 \times 80 \times 3$. Three residual blocks are used after an initial convolutional layer (with stride 1 and padding 1) over the image, which have channels $160/320/640$, stride 2, and padding 2. After a global average pooling layer, it leads to a 640 dimensional embedding.
- **WRN.** Wide residual network [44, 57]. We also consider the vanilla Wide Residual Network as the backbone. We use the WRN-28-10 structure in [44, 57], which sets the depth to 28 and width to 10. After a global average pooling in the last layer of the backbone, we get a 640 dimensional embedding for further prediction. we resize the input image to $84 \times 84 \times 3$. Other details are the same as those with the ResNet backbone.

[2]The source code is publicly available on https://github.com/joe-siyuan-qiao/FewShot-CVPR

Datasets. Three datasets, *Mini*ImageNet [53], Caltech-UCSD Birds (CUB) 200-2011 [54], and OfficeHome [52] 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, 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 [50]. For all images in the CUB dataset, we use the provided bounding box to crop the images as a pre-processing [50]. Before input into the backbone network, all images in the dataset are resized based on the requirement of the network.

Pre-trained strategy. As mentioned before, we apply an additional pre-training strategy as suggested in [38, 44]. The backbone network, appended with a **softmax** layer, is trained to classify all classes in the training split (*e.g.* 64 classes in the *Mini*ImageNet). The pre-trained weights are then used to initialize the embedding function $\mathbf{E}$ in the few-shot learning. Specifically, based on [38]'s work, we pre-train the backbone in the following way. We learn a classifier on the training split with the cross-entropy loss. For example, a 64-way classifier is learned first over *Mini*ImageNet. The last embedding layer (the layer before softmax) is used over the instances from the model validation split (*e.g.* 16 classes in the *Mini*ImageNet), and then the few-shot classification performance of the embedding layer is evaluated to select the best pre-trained model.

Transformer Hyper-parameters. We follow the architecture as presented in [51] to build our FEAT and FEAT* models. The hidden dimension d' for the linear transformation in our FEAT model is set to be 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 Section D.2).

Optimization. Following the literature, different optimizers are used for these two backbones during the training.

For the ConvNet backbone, stochastic gradient descent with Adam [22] optimizer is employed, with the initial learning rate set to be 1e-3. As the backbone network has been pre-trained, we scale the learning rate for those set of parameters by 0.1. For the ResNet and WRN backbones, vanilla stochastic gradient descent with Nesterov acceleration is used with an initial rate 0.0001. We will decrease the learning rate 10 times smaller after 10 epochs.

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

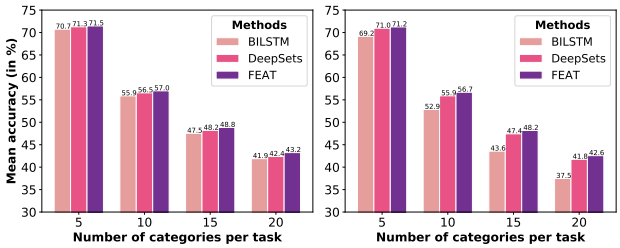
The full results of all methods on the *Mini*ImageNet can be found in Table 5. The results of MAML [10] optimized over the pre-trained embedding network is also included. We re-implement the ConvNet backbone of MAML and cite the MAML results over the WRN-28-10 backbone from [44], which utilizes the same way of initialization. We also investigate the Wide ResNet (WRN) backbone, which is also the popular one used in LEO [44]. We re-implement ProtoNet and our FEAT variants with WRN. With this stronger backbone, all methods achieve better results. It is notable that in this case, our FEAT variants achieve *much higher* promising results than the current state-of-the-art approaches.

Table 6 shows the 5-way 1-shot and 5-shot classification results on the CUB dataset. 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 and FEAT* get the top tier results.

D.2. Ablation Studies

In this section, we perform further analyses for our proposed FEAT and its ablated variant based on the ProtoNet, on the *Mini*ImageNet dataset, using the the ConvNet as the backbone network.

Can FEAT possesses the characteristic of the set function? We test three set transformation implementations, namely the BILSTM, the DEEPSET 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). Similarity, for interpolation, the model is trained with



(*a*) **Task Interpolation** (*b*) **Task Extrapolation**

Figure 7: 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.

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 7 (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.

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 [51], the transformer can be equipped with complex components using multiple heads and deep stacked layers. We evaluate this augmented structure, with attention heads increases to 2, 4, 8, as well as with the number of layers increases to 2 and 3. As in Table 7 and Table 8, 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 our approach). It might require some careful regularization to prevent over-fitting, which we leave for future work.

The effectiveness of contrastive loss. Table 9 and Table 10 show the few-shot classification results when the weight (λ) of the contrastive loss term for FEAT and FEAT*, respectively. From the results, we can find that the balance of both contrastive terms in the learning objective can influence the final results. Empirically, we set $\lambda = 0.1$ for FEAT and $\lambda = 10$ for FEAT* in our experiments.

Two implementations of the 5-Shot embedding adaptation based on the ProtoNet. ProtoNet classifies a test in-

Table 5: Few-shot classification accuracy$\pm$ 95% confidence interval on *mini*ImageNet. Our implementation methods are measured over 10,000 test trials. The super-script "$\star$" indicates the incorporation of test instance in the embedding adaptation process.

Setups →	1-Shot 5-Way			5-Shot 5-Way		
Backbone →	ConvNet	ResNet	WRN	ConvNet	ResNet	WRN
MatchNet [53]	43.40 ± 0.78	-	-	51.09 ± 0.71	-	-
MAML [10, 44]	48.70 ± 1.84	-	58.05 ± 0.10	63.11 ± 0.92	-	72.41 ± 0.20
ProtoNet [47]	49.42 ± 0.78	-	-	68.20 ± 0.66	-	-
RelationNet [49]	51.38 ± 0.82	-	-	67.07 ± 0.69	-	-
PFA [38]	54.53 ± 0.40	59.60 ± 0.41	-	67.87 ± 0.20	73.74 ± 0.19	-
TADAM [36]	-	58.50 ± 0.30	-	-	76.70 ± 0.30	-
LEO [44]	-	-	61.76 ± 0.08	-	-	77.59 ± 0.12
Instance embedding						
MatchNet	52.87 ± 0.20	60.66 ± 0.20	64.03 ± 0.20	67.49 ± 0.17	75.05 ± 0.16	76.32 ± 0.16
ProtoNet	52.61 ± 0.20	61.40 ± 0.20	62.60 ± 0.20	71.33 ± 0.16	76.56 ± 0.16	79.97 ± 0.14
Embedding adaptation						
BILSTM	52.13 ± 0.20	55.73 ± 0.21	60.47 ± 0.20	69.15 ± 0.16	69.81 ± 0.15	78.15 ± 0.15
BILSTM*	54.10 ± 0.20	55.73 ± 0.21	62.91 ± 0.20	69.39 ± 0.16	69.93 ± 0.15	80.25 ± 0.14
DEEPSETS	54.41 ± 0.20	60.02 ± 0.20	64.24 ± 0.20	70.96 ± 0.16	77.30 ± 0.15	79.49 ± 0.15
DEEPSETS*	53.02 ± 0.20	60.06 ± 0.20	63.03 ± 0.20	70.25 ± 0.16	77.95 ± 0.15	80.38 ± 0.14
Ours: FEAT	55.15 ± 0.20	**62.96 ± 0.20**	65.10 ± 0.20	71.61 ± 0.16	**78.49 ± 0.15**	81.11 ± 0.14
Ours: FEAT*	**55.75 ± 0.20**	62.60 ± 0.20	**66.69 ± 0.20**	**72.17 ± 0.16**	78.06 ± 0.15	**81.80 ± 0.15**

Table 6: 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 [53]	61.16 ± 0.89	72.86 ± 0.70
MAML [10]	55.92 ± 0.95	72.09 ± 0.76
ProtoNet [47]	51.31 ± 0.91	70.77 ± 0.69
RelationNet [49]	62.45 ± 0.98	76.11 ± 0.69
Instance embedding		
MatchNet	67.73 ± 0.23	79.00 ± 0.16
ProtoNet	63.72 ± 0.22	81.50 ± 0.15
Embedding adaptation		
BILSTM	62.05 ± 0.23	73.51 ± 0.19
BILSTM*	66.98 ± 0.23	80.08 ± 0.16
DEEPSETS	67.22 ± 0.23	79.65 ± 0.16
DEEPSETS*	68.12 ± 0.23	80.24 ± 0.16
Ours: FEAT	**68.87 ± 0.22**	82.90 ± 0.15
Ours: FEAT*	68.65 ± 0.22	**83.03 ± 0.15**

Table 7: Ablation studies on the number of heads in the Transformer of FEAT and FEAT* (with number of layers fixes to one).

Setups →	1-Shot 5-Way		5-Shot 5-Way	
# of heads	FEAT	FEAT*	FEAT	FEAT*
1	55.15 ± 0.20	55.75 ± 0.20	71.57 ± 0.16	72.17 ± 0.16
2	54.91 ± 0.20	55.14 ± 0.20	71.44 ± 0.16	71.51 ± 0.16
4	55.05 ± 0.20	55.04 ± 0.20	71.63 ± 0.16	71.57 ± 0.16
8	55.22 ± 0.20	54.42 ± 0.20	71.39 ± 0.16	71.21 ± 0.16

Table 8: Ablation studies on the number of layers in the Transformer of FEAT and FEAT* (with number of heads fixes to one).

Setups →	1-Shot 5-Way		5-Shot 5-Way	
# of layers	FEAT	FEAT*	FEAT	FEAT*
1	55.15 ± 0.20	55.75 ± 0.20	71.57 ± 0.16	72.17 ± 0.16
2	55.42 ± 0.20	55.08 ± 0.20	71.44 ± 0.16	71.70 ± 0.16
3	54.96 ± 0.20	54.93 ± 0.20	71.63 ± 0.16	71.87 ± 0.16

stance based on the label of the nearest class center. There are two embedding adaptation variants in this implementation when there is more than one instance in each class, *i.e.*, we can compute the class center *before* or *after* the embedding adaptation. In other words, the class prototype in Eq. 12 can be computed based on $\phi(\mathbf{x})$ or $\psi(\mathbf{x})$. In our experiments, we use the former strategy for 5-shot tasks to get the class centers. From empirical results, we find there does not exist an obvious difference between these two strategies. For example, for FEAT* with the later computation strategy, it achieves 72.08 ± 0.20 on 5-shot 5-way tasks.

The influence of the prediction strategy. We investigate two embedding-based prediction ways for few-shot classifi-

Table 9: Ablation studies on effects of contrastive learning of the set function on FEAT.

Setups →	1-Shot 5-Way	5-Shot 5-Way
$\lambda = 10$	53.92 ± 0.20	70.41 ± 0.16
$\lambda = 1$	54.84 ± 0.20	71.00 ± 0.16
$\lambda = 0.1$	**55.15 ± 0.20**	**71.61 ± 0.16**
$\lambda = 0.01$	54.67 ± 0.20	71.26 ± 0.16

Table 10: Ablation studies on effects of the contrastive attention learning (CAL) term on FEAT*.

Setups →	1-Shot 5-Way	5-Shot 5-Way
$\lambda = 0$	54.65 ± 0.20	71.47 ± 0.16
$\lambda = 1$	55.19 ± 0.20	71.77 ± 0.16
$\lambda = 10$	**55.75 ± 0.20**	**72.17 ± 0.16**
$\lambda = 100$	54.92 ± 0.20	71.83 ± 0.16

Table 11: Ablation studies on the prediction strategy (MatchNet or ProtoNet) of FEAT and FEAT*.

Setups →	1-Shot 5-Way		5-Shot 5-Way	
Backbone →	ConvNet	ResNet	ConvNet	ResNet
MatchNet-based prediction				
FEAT	54.64± 0.20	61.26± 0.20	71.72± 0.16	77.83± 0.15
FEAT*	55.15± 0.20	60.08± 0.20	70.49± 0.16	77.87± 0.15
ProtoNet-based prediction				
FEAT	55.15± 0.20	62.96± 0.20	71.61± 0.16	78.49± 0.15
FEAT*	55.75± 0.20	61.60± 0.20	72.17± 0.16	78.06± 0.15

Table 12: Cross-Domain 1-shot 5-way classification results of the FEAT approach.

	C → C	C → R	R → R
Supervised	34.38±0.16	29.49±0.16	37.43±0.16
ProtoNet	35.51±0.16	29.47±0.16	37.24±0.16
FEAT	**36.83±0.17**	**30.89±0.17**	**38.49±0.16**

cation, *i.e*. based on the MatchNet and ProtoNet, which use the cosine similarity and distance to measure the relationship between objects respectively. The comparison results are shown in Table 11. During the optimization, we tune the scale parameter for both these methods. We find that the ProtoNet achieve better performance than the MatchNet.

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.

Setups. We train a 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 trained on tasks from the "Clipart" domain of Office-Home dataset [52], 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 12 gives the quantitative results. Here, the "supervised" refers to a model trained with standard classification and then is used for 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 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.

Table 13: Results of models for transductive FSL with ConvNet backbone on *mini*ImageNet.

Setups →	1-Shot 5-Way	5-Shot 5-Way
Standard		
ProtoNet	52.61 ± 0.20	71.33 ± 0.16
FEAT	55.15 ± 0.20	71.61 ± 0.16
FEAT*	55.75 ± 0.20	72.17 ± 0.16
Transductive		
Semi-ProtoNet [41]	50.41 ± 0.31	64.39 ± 0.24
TPN [32]	54.72 ± 0.84	69.25 ± 0.67
FEAT (Transductive)	**56.49 ± 0.21**	**72.65 ± 0.16**

D.4. More Transductive FSL Discussions

We list the result of transductive few-shot classification in Table 13, where the unlabeled test instances arrive simultaneously when classifying a test instance. We compare with two previous approaches, Semi-ProtoNet [41] and TPN [32]. Semi-ProtoNet utilizes the unlabeled instances in the prototypical network determination, while TPN meta learns a label propagation way to take unlabeled instances relationship into consideration. We cite their results over *mini*ImageNet directly as a reference.

In this setting, our model leverages the unlabeled test instances to augment the transformer as discussed in Sec. B.2 and the embedding adaptation takes the relationship of all test instances into consideration. By utilizing only one test instance, the few-shot classification performance of FEAT* is higher than FEAT, this is also the case in other experiments. By using more unlabeled test instances in the trans-

Table 14: 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±0.03	48.64±0.20	35.69±0.03
FEAT	**43.94**±0.03	**49.72**±0.20	**40.50**±0.03
5-shot learning			
ProtoNet	41.06±0.03	64.94±0.17	38.04±0.02
FEAT	**44.94**±0.03	**65.33**±0.16	**41.68**±0.03
Random Chance	1.56	20.00	1.45

ductive environment, FEAT achieves further performance improvement compared with standard setting. The performance gain induced by the transductive FEAT is more significant in the one-shot learning setting comparing to 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 14, 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.

E. Large-Scale Low-Shot Learning

Similar to the generalized few-shot learning, the large-scale low-shot learning [13, 16, 55] considers the few-shot classification ability on both SEEN and UNSEEN classes on the full ImageNet [42] dataset. There are in total 389 SEEN classes and 611 UNSEEN classes [16]. We follow the setting (including the splits) of the prior work [16] and use features extracted based on the pre-trained ResNet50 [17]. 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 15. We observe that FEAT achieves better results than others, which further validates FEAT's superiority in generalized classification setup, a large scale learning setup.

Table 15: The top-5 low-shot learning accuracy over all classes on the large scale ImageNet [42] dataset (w/ ResNet50).

UNSEEN	1-Shot	2-Shot	5-Shot	10-Shot	20-Shot
ProtoNet [47]	49.6	64.0	74.4	78.1	80.0
PMN [55]	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 [47]	61.4	71.4	78.0	80.0	81.1
PMN [55]	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 [47]	62.9	70.5	77.1	79.5	80.8
PMN [55]	63.4	70.8	77.9	80.9	82.7
FEAT	**63.8**	**71.2**	**78.1**	**81.3**	**83.4**