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

Supplementary Material

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

Contents

1. Details of Baseline Methods

In this section, we describe two important embedding learning baselines *i.e.*, Matching Network (MatchNet) [26] and Prototypical Network (ProtoNet) [20], 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 an M-shot N-way classification task, we can obtain the embedding of each training instance based on the function $\mathbf{E}$:[1]

$$\phi(\mathbf{x}_i) = \mathbf{E}(\mathbf{x}_i),\ \forall \mathbf{x}_i \in \mathcal{X}_{\mathbf{train}} \tag{1}$$

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

$$\begin{aligned}\hat{\mathbf{y}}_{\mathbf{test}} &\propto \exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_i})\right) \cdot \mathbf{y}_i \\ &= \frac{\exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_i})\right)}{\sum_{\mathbf{x}_{i'} \in \mathcal{X}_{\mathbf{train}}} \exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_{i'}})\right)} \cdot \mathbf{y}_i \\ &= \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}_{\mathbf{train}}} \frac{\exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_i})\right)}{\sum_{\mathbf{x}_{i'} \in \mathcal{X}_{\mathbf{train}}} \exp\left(\gamma \cdot \mathbf{sim}(\phi_{\mathbf{x_{test}}}, \phi_{\mathbf{x}_{i'}})\right)} \cdot \mathbf{y}_i\end{aligned} \tag{2}$$

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 [13]. 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 than the cosine similarity as the similarity metric:

$$\mathbf{c}_n = \frac{1}{M} \sum_{\mathbf{y}_i = n} \phi(\mathbf{x}_i),\ \forall n = 1, \ldots, N \tag{3}$$

$$\begin{aligned}\hat{\mathbf{y}}_{\mathbf{test}} &\propto \exp\left(\gamma \cdot \|\phi_{\mathbf{x_{test}}} - \mathbf{c}_n\|_2^2\right) \cdot \mathbf{y}_n \\ &= \sum_{n=1}^{N} \frac{\exp\left(-\gamma \|\phi_{\mathbf{x_{test}}} - \mathbf{c}_n\|_2^2\right)}{\sum_{n'=1}^{N} \exp\left(-\gamma \|\phi_{\mathbf{x_{test}}} - \mathbf{c}_{n'}\|_2^2\right)} \mathbf{y}_n\end{aligned} \tag{4}$$

*Work mostly done when the author was a visiting scholar at USC.

†On leave from USC

[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.

Similar to the aforementioned scalar temperature for MatchNet, in Eq. 4 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. 4 outputs the probability to classify $\mathbf{x}_{\mathbf{test}}$ 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. 3 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.

2. 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).

2.1. BiLSTM as the Set-to-Set Transformation

Bidirectional LSTM (BILSTM) [7, 26] 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

$$\{\overleftarrow{\phi}(\mathbf{x}), \vec{\phi}(\mathbf{x})\} = \mathbf{BILSTM}(\{\phi(\mathbf{x})\}); \quad \forall \mathbf{x} \in \mathcal{X}_{\mathbf{train}} \tag{5}$$

Where $\overleftarrow{\phi}(\mathbf{x})$ and $\vec{\phi}(\mathbf{x})$ 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

$$\psi(\mathbf{x}) = \phi(\mathbf{x}) + \overleftarrow{\phi}(\mathbf{x}) + \vec{\phi}(\mathbf{x}) \tag{6}$$

It is notable that the output of the BILSTM suppose to depend on the order of the input set. Vinyals *et al.* [26] propose to use the Fully Conditional Embedding to encode the context of both the test instance and the support set instances based on BILSTM and LSTM w/ Attention module. Different from [26], we apply the set-to-set embedding adaptation only over the support set, which leads to a fully inductive learning setting.

2.2. DeepSets as the Set-to-Set Transformation

Deep sets [32] 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:

$$\psi(\mathbf{x}) = \phi(\mathbf{x}) + g([\phi(\mathbf{x}); \sum_{\mathbf{x}_{i'} \in \mathbf{x}^{\complement}} h(\phi(\mathbf{x}_{i'}))]) \tag{7}$$

In Eq. 7, 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. 4. An illustration of the DeepSets notation in the embedding adaptation can be found in Figure 1 (c). The summation operator in Eq. 7 could also be replaced as the maximum operator, and we find the maximum operator works better than summation operator in our experiments.

2.3. GCN as the Set-to-Set Transformation

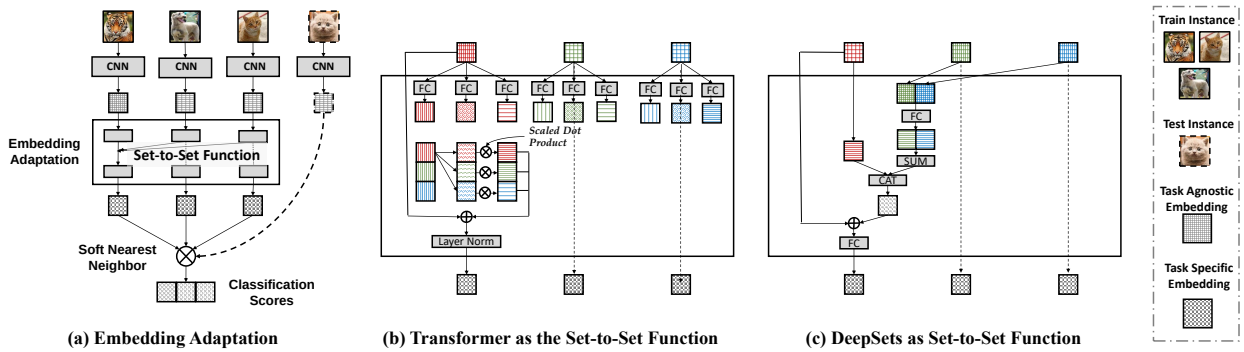
Graph Convolutional Networks (GCN) [10, 19] propagate the relationship between instances in the set. We first construct a degree matrix $A \in \mathbb{R}^{NK \times NK}$ to represent the similarity between instances in a set. If two instances $\mathbf{x}_i$ and $\mathbf{x}_j$ come from the same class, 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.*,

$$\Phi^{t+1} = \mathbf{ReLU}(S\Phi^t W) \ , \ t = 0, 1, \ldots, T-1 \tag{8}$$

W is a learned a projection matrix for feature transformation. In GCN, the embedding in the set is transformed based on Eq. 8 multiple times (we propagate the embedding set two times during the experiments), and the final propagated embedding set Φ^T gives rise to the $\psi_{\mathbf{x}}$.

2.4. 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 [24] to



(a) Embedding Adaptation (b) Transformer as the Set-to-Set Function (c) DeepSets as Set-to-Set Function

Figure 1: Illustration of two embedding adpatation 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.

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 [24], 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

$$\begin{aligned} Q &= W_Q^\top \left[\, \phi_{\mathbf{x}_q};\quad \forall \mathbf{x}_q \in \mathcal{Q} \,\right] \in \mathbb{R}^{d'\times|\mathcal{Q}|} \\ K &= W_K^\top \left[\, \phi_{\mathbf{x}_k};\quad \forall \mathbf{x}_k \in \mathcal{K} \,\right] \in \mathbb{R}^{d'\times|\mathcal{K}|} \\ V &= W_V^\top \left[\, \phi_{\mathbf{x}_v};\quad \forall \mathbf{x}_v \in \mathcal{V} \,\right] \in \mathbb{R}^{d'\times|\mathcal{V}|} \end{aligned} \tag{9}$$

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

$$\alpha_{qk} \propto \exp\left(\frac{\phi_{\mathbf{x}_q}^\top W_Q \cdot K}{\sqrt{d}}\right);\ \forall \mathbf{x}_k \in \mathcal{K} \tag{10}$$

$$\alpha_{q,:} = \mathbf{softmax}\left(\frac{\phi_{\mathbf{x}_q}^\top W_Q \cdot K}{\sqrt{d}}\right) \in \mathbb{R}^{|\mathcal{K}|} \tag{11}$$

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

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 $\mathbf{x}_q$:

$$\tilde{\psi}_{\mathbf{x}_q} = \sum_k \alpha_{qk} V_{:,k} \tag{12}$$

$$\psi_{\mathbf{x}_q} = \tau\big(\phi_{\mathbf{x}_q} + W_{\mathbf{FC}}^\top \tilde{\psi}_{\mathbf{x}_q}\big) \tag{13}$$

$V_{:,k}$ 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 [21] and layer normalization [1]. The whole flow of transformer in our FEAT approach can be found in Figure 1 (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 [24], 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 § 4.

2.5. Extension to transductive FSL

Facilitated by the flexible set-to-set transformer in Eq. 13, 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

$$\mathcal{Q} = \mathcal{K} = \mathcal{V} = \mathcal{X}_{\mathbf{train}} \cup \mathcal{X}_{\mathbf{test}} \tag{14}$$

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.

3. 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 [20, 23, 26] contains 4 repeated blocks. In each block, there is a convolutional layer with 3×3 kernel, a Batch Normalization layer [8], 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 [11].[4] The DropBlock [3] is used in this ResNet architecture to avoid over-fitting. A bit different from the ResNet-12 in [11], 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 [18, 31]. We use the WRN-28-10 structure as in [15, 18], 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.

Datasets. Four datasets, *Mini*ImageNet [26], *Tiered*ImageNet [16], Caltech-UCSD Birds (CUB) 200-2011 [27], and OfficeHome [25] 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 [23]. For all images in the CUB dataset, we use the provided bounding box to crop the images as a pre-processing [23]. 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 [15, 18]. 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 $\mathbf{E}$, 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 [24] 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 § 4.2).

Optimization. Following the literature, different optimizers are used for the backbones during the model training. For the ConvNet backbone, stochastic gradient descent with Adam [9] 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

[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 [15] in the old version of our paper [30], whose source code of ResNet is publicly available on https://github.com/joe-siyuan-qiao/FewShot-CVPR. Empirically we find the ResNet-12 [11] works better than our old ResNet architecture.

Table 1: 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 [26]	43.40± 0.78	-	51.09± 0.71	-
MAML [2]	48.70± 1.84	-	63.11± 0.92	-
ProtoNet [20]	49.42± 0.78	-	68.20± 0.66	-
RelationNet [22]	51.38± 0.82	-	67.07± 0.69	-
PFA [15]	54.53± 0.40	-	67.87± 0.20	-
TADAM [13]	-	58.50± 0.30	-	76.70± 0.30
MetaOptNet [11]	-	62.64± 0.61	-	78.63± 0.46
Baselines				
MAML	49.24± 0.21	58.05± 0.10	67.92± 0.17	72.41± 0.20
MatchNet	52.87± 0.20	65.64± 0.20	67.49± 0.17	78.72± 0.15
ProtoNet	52.61± 0.20	62.39± 0.21	71.33± 0.16	80.53± 0.14
Embedding Adaptation				
BILSTM	52.13± 0.20	63.90± 0.21	69.15± 0.16	80.63± 0.14
DEEPSETS	54.41± 0.20	64.14± 0.22	70.96± 0.16	80.93± 0.14
GCN	53.25± 0.20	64.50± 0.20	70.59± 0.16	81.65± 0.14
Ours: FEAT	**55.15**± **0.20**	**66.78**± **0.20**	**71.61**± **0.16**	**82.05**± **0.14**

Table 2: 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 [15]	59.60± 0.41	73.74± 0.19
LEO [18]	61.76± 0.08	77.59± 0.12
SimpleShot [28]	63.50± 0.20	80.33± 0.14
ProtoNet (Ours)	62.60± 0.20	79.97± 0.14
Ours: FEAT	**65.10** ± **0.20**	**81.11** ± **0.14**

Table 3: 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 [18]	66.33± 0.05	81.44± 0.09
SimpleShot [28]	69.75± 0.20	**85.31**± **0.15**
Ours: FEAT	**70.41** ± **0.23**	84.38 ± 0.16

pre-trained weights, we scale the learning rate for those parameters by 0.1.

Table 4: 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.

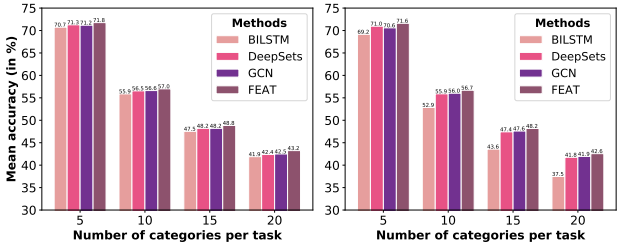
Setups →	1-Shot 5-Way	5-Shot 5-Way
MatchNet [26]	61.16 ± 0.89	72.86 ± 0.70
MAML [2]	55.92 ± 0.95	72.09 ± 0.76
ProtoNet [20]	51.31 ± 0.91	70.77 ± 0.69
RelationNet [22]	62.45 ± 0.98	76.11 ± 0.69
Instance Embedding		
MatchNet	67.73 ± 0.23	79.00 ± 0.16
ProtoNet	63.72 ± 0.22	81.50 ± 0.15
Embedding Adaptation		
BILSTM	62.05 ± 0.23	73.51 ± 0.19
DEEPSETS	67.22 ± 0.23	79.65 ± 0.16
GCN	67.83 ± 0.23	80.26 ± 0.15
Ours: FEAT	**68.87** ± **0.22**	**82.90** ± **0.15**

4. 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.

4.1. Main Results

The full results of all methods on the *Mini*ImageNet can be found in Table 1. The results of MAML [2] optimized over the pre-trained embedding network are also included.



(*a*) **Way Interpolation** (*b*) **Way Extrapolation**

Figure 2: Interpolation and **Extrapolation** of few-shot tasks from the "way" perspective. First, We train various embedding adaptation models on 5-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.

Table 5: 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

We re-implement the ConvNet backbone of MAML and cite the MAML results over the ResNet backbone from [18]. 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 [15, 18]. SimpleShot [28] 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 [15], LEO [18], and SimpleShot [28] from their papers. The results can be found in Table 2. 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 3 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 4 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.

Table 6: 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 $\rightarrow$	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 7: Ablation studies on the number of heads in the Transformer of FEAT (with number of layers fixes to one).

Setups $\rightarrow$	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 8: Ablation studies on the number of layers in the Transformer of FEAT (with number of heads fixes to one).

Setups $\rightarrow$	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

4.2. Ablation Studies

In this section, we perform further analyses for our proposed FEAT and its ablated variants classifying in the ProtoNet 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 using the pre-adapted embeddings of support data (*i.e.*, the embedding before adaptation), against those using adapted embeddings, for constructing classifiers. Table 5 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 four set-to-set transformation implementations, namely the BILSTM, the DEEPSETS, the GCN, and the Transformer (FEAT), w.r.t. two important properties of the set function, *i.e.*, way interpolation and way 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 2 (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. 3. 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 6, 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 [24], 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 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 regularizations to prevent over-fitting, which we leave for future work.

The effectiveness of contrastive loss. Table 9 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.

Table 9: 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 ± 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 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± 0.20	66.26± 0.20	71.72± 0.16	81.83± 0.15
Euclidean Distance-based Prediction				
FEAT	55.15± 0.20	66.78± 0.20	71.61± 0.16	82.05± 0.14

Table 11: 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**

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 10. Two strategies in Table 10 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.

Table 12: Results of models for transductive FSL with ConvNet backbone on *mini*ImageNet. We cite the results of Semi-ProtoNet and TPN from [16] and [14] respectively. For TEAM [14], 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 ± 0.20	71.33 ± 0.16
FEAT	55.15 ± 0.20	71.61 ± 0.16
Transductive		
Semi-ProtoNet [16]	50.41 ± 0.31	64.39 ± 0.24
TPN [12]	55.51 ± 0.84	69.86 ± 0.67
TEAM [14]	56.57 ± 0.00	72.04 ± 0.00
Semi-ProtoNet (Ours)	55.50 ± 0.10	71.76 ± 0.08
FEAT†	56.49 ± 0.16	72.65 ± 0.20
FEAT‡	**57.04 ± 0.16**	**72.89 ± 0.20**

4.3. 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 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 [25], 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 11 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.

4.4. Additional Discussions on Transductive FSL

We list the results of the transductive few-shot classification in Table 12, 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 [16], TPN [12], and TEAM [14]. 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 [16], and cite the results of TPN and TEAM from [14]. 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 § 2.4 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 FEAT†, 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 FEAT‡).

By using more unlabeled test instances in the transductive environment, FEAT† 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, FEAT‡ 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.

4.5. More Generalized FSL Results

Here we show the full results of FEAT in the generalized few-shot learning setting in Table 13, 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.

4.6. Large-Scale Low-Shot Learning

Similar to the generalized few-shot learning, the large-scale low-shot learning [4, 5, 29] considers the few-shot classification ability on both SEEN and UNSEEN classes on

Table 13: 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

Table 14: The top-5 low-shot learning accuracy over all classes on the large scale ImageNet [17] dataset (w/ ResNet-50).

UNSEEN	1-Shot	2-Shot	5-Shot	10-Shot	20-Shot
ProtoNet [20]	49.6	64.0	74.4	78.1	80.0
PMN [29]	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 [20]	61.4	71.4	78.0	80.0	81.1
PMN [29]	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 [20]	62.9	70.5	77.1	79.5	80.8
PMN [29]	63.4	70.8	77.9	80.9	82.7
FEAT	**63.8**	**71.2**	**78.1**	**81.3**	**83.4**

the full ImageNet [17] dataset. There are in total 389 SEEN classes and 611 UNSEEN classes [5]. We follow the setting (including the splits) of the prior work [5] and use features extracted based on the pre-trained ResNet-50 [6]. 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 14. We observe that FEAT achieves better results than others, which further validates FEAT's superiority in generalized classification setup, a large scale learning setup.

References

[1] L. J. Ba, R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 3

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

[3] G. Ghiasi, T.-Y. Lin, and Q. V. Le. Dropblock: A regularization method for convolutional networks. In *NeurIPS*, pages 10750–10760. 2018. 4

[4] S. Gidaris and N. Komodakis. Dynamic few-shot visual learning without forgetting. In *CVPR*, pages 4367–4375, 2018. 8

[5] B. Hariharan and R. B. Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *ICCV*, pages 3037–3046, 2017. 8, 9

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 9

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

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. 4

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 4

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

[11] K. Lee, S. Maji, A. Ravichandran, and S. Soatto. Meta-learning with differentiable convex optimization. In *CVPR*, pages 10657–10665, 2019. 4, 5

[12] 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. 8

[13] 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. 1, 5

[14] 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

[15] 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. 4, 5, 6

[16] 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. 4, 8

[17] 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. *International Journal of Computer Vision*, 115(3):211–252, 2015. 9

[18] 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. 4, 5, 6

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

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

networks for few-shot learning. In *NeurIPS*, pages 4080–4090. 2017. 1, 4, 5, 9

[21] 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. 3

[22] 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. 5

[23] E. Triantafillou, R. S. Zemel, and R. Urtasun. Few-shot learning through an information retrieval lens. In *NeurIPS*, pages 2252–2262. 2017. 4

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, pages 6000–6010. 2017. 2, 3, 4, 7

[25] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan. Deep hashing network for unsupervised domain adaptation. In *CVPR*, pages 5385–5394, 2017. 4, 8

[26] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *NeurIPS*, pages 3630–3638. 2016. 1, 2, 4, 5

[27] 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. 4

[28] 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. 5, 6

[29] Y.-X. Wang, R. B. Girshick, M. Hebert, and B. Hariharan. Low-shot learning from imaginary data. In *CVPR*, pages 7278–7286, 2018. 8, 9

[30] H.-J. Ye, H. Hu, D.-C. Zhan, and F. Sha. Learning embedding adaptation for few-shot learning. *CoRR*, abs/1812.03664, 2018. 4

[31] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016. 4

[32] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *NeurIPS*, pages 3394–3404. 2017. 2