Cross-domain Few-shot Learning with Task-specific Adapters

Cross-domain Few-shot Learning with Task-specific Adapters

Wei-Hong Li, Xialei Liu*, and Hakan Bilen
VICO Group, University of Edinburgh, United Kingdom
github.com/VICO-UoE/URL

Abstract

In this paper, we look at the problem of cross-domain few-shot classification that aims to learn a classifier from previously unseen classes and domains with few labeled samples. Recent approaches broadly solve this problem by parameterizing their few-shot classifiers with task-agnostic and task-specific weights where the former is typically learned on a large training set and the latter is dynamically predicted through an auxiliary network conditioned on a small support set. In this work, we focus on the estimation of the latter, and propose to learn task-specific weights from scratch directly on a small support set, in contrast to dynamically estimating them. In particular, through systematic analysis, we show that task-specific weights through parametric adapters in matrix form with residual connections to multiple intermediate layers of a backbone network significantly improves the performance of the state-of-the-art models in the Meta-Dataset benchmark with minor additional cost.

1. Introduction

Deep learning methods have seen remarkable progress in various fields where large quantities of data and compute power are available. However, the ability of deep networks to learn new concepts from small data remains limited. Few-shot classification [23,33] is inspired from this limitation and aims at learning a model that can be efficiently adapted to recognize unseen classes from few samples. In particular, the standard setting for learning few-shot classifiers involves two stages: (i) learning a model, typically from a large training set, (ii) adapting this model to learn new classes from a given small support set. These two stages are called meta-training and meta-testing respectively. The adapted model is finally evaluated on a query set where the task is to assign each query sample to one of the classes in the support set.

Early methods [16, 35, 37, 43, 45, 49] pose the few-shot classification problem in a learning-to-learn formulation by training a deep network over a distribution of related tasks, which are sampled from the training set, and transfer this experience to improve its performance for learning new classes. Concretely, Vinyals *et al*. [49] learn a feature encoder that is conditioned on the support set in meta-training and does not require any further training in meta-test thanks to its non-parametric classifier. Ravi and Larochelle [37] take the idea of learning a feature encoder in meta-train further by also learning an update rule through an LSTM that produces the updates for a classifier in meta-test. Finn *et al*. [16] pose the task as a meta-learning problem and learn the parameters of a deep network in meta-training such that a network initialized with the learned parameters can be efficiently finetuned on a new task. We refer to [18, 51] for comprehensive review of early works.

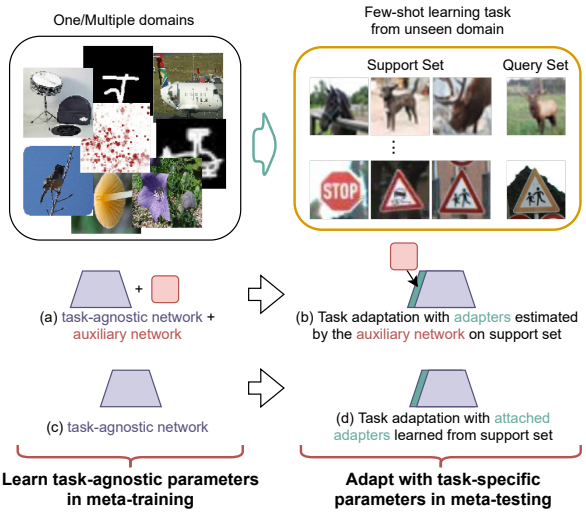

Figure 1. **Cross-domain Few-shot Learning** considers to learn a model from one or multiple domains to generalize to unseen domains with few samples. Prior works often learn a task-agnostic model with an auxiliary network during meta-training (a) and a set of adapters are generated by the auxiliary network to adapt to the given support set (b). While in this work, we propose to attach adapters directly to a pretrained task-agnostic model (c), which can be estimated from scratch during meta-testing (d). We also propose different architecture topologies of adapters and their efficient approximations.

*Xialei Liu is the corresponding author.

Despite the significant progress, the scope of the early methods has been limited to a restrictive setting where training and test samples come from a single domain (or data distribution) such as Omniglot [24], miniImageNet [49] and tieredImageNet [40]. They perform poorly in the more challenging cross-domain few-shot tasks, where test data is sampled from an unknown or previously unseen domain [48]. This setting poses an additional learning challenge, not only requires leveraging the limited information from the small support set for learning the target task but also *selectively transferring relevant knowledge* from previously seen domains to the target task.

Broadly, recent approaches address this challenge by parameterizing deep networks with a large set of task-agnostic and a small set of task-specific weights that encode generic representations valid for multiple tasks and private representations are specific to the target task respectively. While the task-agnostic weights are learned over multiple tasks, typically, from a large dataset in meta-training, the task-specific weights are estimated from a given small support set (*e.g.* 5 images per category) [3, 15, 26, 27, 29, 41, 47]. In the literature, the task-agnostic weights are used to parameterize a single network that is trained on large data from one domain [3, 14, 41] or on multiple domains [27], or to be distributed over multiple networks, each trained on a different domain [15, 29, 47] [1]. The task-specific weights are utilized to parameterize a linear classifier [26], a pre-classifier feature mapping [27] and an ensemble of classifiers at each layer of a deep neural network [1].

Recently, inspired from [36], *task-specific adapters* [3, 41], small capacity transformations that are applied to multiple layers of a deep network, have been successfully used to steer the few-shot classifiers to new tasks and domains. Their weights are often estimated dynamically through an auxiliary network conditioned on the support set [3, 29, 41, 47] (see Fig. 1.(a,b)), in a similar spirit to [4, 20]. As the auxiliary network is trained on multiple tasks in meta-training, the premise of estimating the task-specific adapter weights with it is based on the principle of transfer learning such that it can transfer the knowledge from the previous tasks to better estimate them for unseen tasks. However, learning an accurate auxiliary network is a challenging task due to two reasons. First, it has to generalize to previously unseen tasks and especially to significantly different unseen domains. Second, learning to predict high-dimensional weights where each corresponds to a dimension of a highly nonlinear feature space is a difficult learning problem too.

Motivated by this shortcoming, as shown in Fig. 1, we propose to employ a set of light-weight task-specific adapters along with the task-agnostic weights for adapting the few-shot classifier to the tasks from unseen domains. Unlike the prior work, we learn the weights of these adapters from scratch by directly optimizing them on a small support set (see Fig. 1.(c,d)). Moreover, we systematically study various combinations of several design choices for task-specific adaptation, which have not been explored before, including adapter connection types (serial or residual), parameterizations (matrix and its decomposed variations, channelwise operations) and estimation of task-specific parameters. Extensive experiments demonstrate that attaching parameteric adapters in matrix form to convolutional layers with residual connections significantly boosts the state-of-the-art performance in most domains, especially resulting in superior performance in unseen domains on Meta-Dataset with negligible increase in computations.

More related work. Here we provide more detailed discussion of the most related work. Both CNAPS [41] and Simple CNAPS [3] employ task-specific adapters via FiLM layers (which uses a channelwise affine transformation and connected to the backbone in a serial way) [36] to adapt their feature extractors to the target task and estimate them via an auxiliary network. Compared to them, we propose learning residual adapters in matrix form directly on the support set. SUR [15] and URT [29] learn an attention mechanism to select/fuse features from multiple domain-specific models in meta-train respectively. As we build on a single multi-domain feature extractor, our method does not require such attention but we attach task-specific adapters to the feature extractor to adapt the features to unseen tasks. URL [27] learns a pre-classifier feature mapping to adapt the feature from a single task-agnostic model learned from multiple domains for unseen tasks. While we build on their feature extractor and pre-classifier alignment, the pre-classifier alignment provides very limited capacity for task adaptation, which we address by adapting the feature extractor with adapters at multiple layers. FLUTE [47] follows a hybrid three step approach that first learns the parameters of domain-specific FiLM layers so called templates, employs an auxiliary network to initialize the parameters of a new FiLM layer for unseen task by combining the templates and finetunes them on the small support set. Different from FLUTE, our method learns such adaptation in a single step by learning residual adapters in meta-test.

There are also methods (*e.g.* [14, 44]) that do not fit into task-agnostic and task-specific parameterization grouping. BOHB [44] proposes to use multi-domain data as validation objective for hyper-parameter optimization such that the feature learned on ImageNet with the optimized hyper-parameter generalizes well to multi-domain. CTX [14] proposes to learn spatial correspondences from ImageNet and evaluates on the remaining (unseen) domains. We also compare our method to them in the setting where we use a standard single domain learning network learned from ImageNet and adapt its representations through residual adapters.

[1] Note that the task-agnostic weights can also be finetuned on the target task (*e.g.* [8, 13]).

2. Method

Few-shot classification aims at learning to classify samples of new categories efficiently from few samples only. Each few-shot learning task consists of a support set $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{|\mathcal{S}|}$ with $|\mathcal{S}|$ sample and label pairs respectively and a query set $\mathcal{Q} = \{(\boldsymbol{x}_j)\}_{j=1}^{|\mathcal{Q}|}$ with $|\mathcal{Q}|$ samples to be classified. The goal is to learn a classifier on $\mathcal{S}$ that accurately predicts the labels of $\mathcal{Q}$. Note that this paper focuses on few-shot image classification problem, *i.e.* $\boldsymbol{x}$ and y denote an image and its label.

As in [15, 27, 29], we solve this problem in two steps involving i) representation learning where we learn a task-agnostic feature extractor f from a large dataset $\mathcal{D}_b$, ii) task adaptation where we adapt the task-agnostic representations through various task-specific weights to the target tasks $(\mathcal{S}, \mathcal{Q})$ that are sampled from another large dataset $\mathcal{D}_t$ by taking the subsets of the dataset to build $\mathcal{S}$ and $\mathcal{Q}$. Note that $\mathcal{D}_b$ and $\mathcal{D}_t$ contain mutually exclusive classes.

2.1. Task-agnostic representation learning

Learning task-agnostic or universal representations [5] has been key to the success of cross-domain generalization. Representations learned from a large diverse dataset such as ImageNet [12] can be considered as universal and successfully transferred to tasks in different domains with minor adaptations [15, 29, 38]. We denote this setting as single domain learning (SDL).

More powerful and diverse representations can be obtained by training a single network over multiple domains. Let $\mathcal{D}_b = \{\mathcal{D}_k\}_{k=1}^{K}$ consists of K subdatasets, each sampled from a different domain. The vanilla multi-domain learning (MDL) strategy jointly optimizes network parameters over the images from all K subdatasets:

$$\min_{\phi, \psi_k} \sum_{k=1}^{K} \frac{1}{|\mathcal{D}_k|} \sum_{\boldsymbol{x}, y \in \mathcal{D}_k} \ell(g_{\psi_k} \circ f_\phi(\boldsymbol{x}), y), \tag{1}$$

where ℓ is cross-entropy loss, f is feature extractor that takes an image as input and outputs a d dimensional feature. f is parameterized by ϕ which is shared across K domains. g_{ψ_k} is the classifier for domain k and parameterized by ψ_k which is discarded in meta-test. We denote this setting as MDL. The challenge in MDL is to allow efficiently sharing the knowledge across the domains while preventing negative transfer between them and also carefully balancing the individual loss functions ([9]). URL [27], a variant of MDL, mitigates these challenges by first training individual domain-specific networks offline and then distilling their knowledge into a *single* multi-domain network. We refer to [27] for more details.

Another way of obtaining multi-domain representations is to employ multiple domain-specific feature extractors, one for each domain, and adaptively "fuse" their features for each task [15, 30, 47]. While these methods are effective, they require computing features for each image through multiple feature extractors and are thus computationally expensive. Due to its simplicity and effectiveness, we conduct experiments with the feature extractor of URL [27] along with the SDL one.

2.2. Task-specific weight learning

A good task-agnostic feature extractor f_ϕ is expected to produce representations that generalize to many previously unseen tasks and domains. However this gets more challenging when there is a large domain gap between the training set $\mathcal{D}_b$ and test set $\mathcal{D}_t$ which requires further adaptation to the target task. In this work, we propose to incorporate additional capacity to the task-agnostic feature extractor by adding task-specific weights to adapt the representations to the target task by using the support set. Specifically, we directly attach task-specific weights to a learned task-agnostic model, and estimate them from scratch given the support set. We denote the task-specific weights with ϑ and task-adapted classifier with $p_{(\phi,\vartheta)}$ that outputs a softmax probability vector whose dimensionality equals to the number of categories in the support set $\mathcal{S}$.

To obtain the task-specific weights, we freeze the task-agnostic weights ϕ and minimize the cross-entropy loss ℓ over the support samples in meta-test w.r.t. the task-specific weights ϑ [15, 27, 46]:

$$\min_{\vartheta} \frac{1}{|\mathcal{S}|} \sum_{(\boldsymbol{x}, y) \in \mathcal{S}} \ell(p_{(\phi,\vartheta)}(\boldsymbol{x}), y), \tag{2}$$

where $\mathcal{S}$ is sampled from the test set $\mathcal{D}_t$. Most previous works freeze the task-agnostic weights but estimate the task-specific weights through an auxiliary network (or a task encoder) [3, 27, 41, 47], where inaccurate prediction of parameters can lead to noisy adaptation and wrong prediction.

2.3. Task-specific adapter parameterization (ϑ)

Task adaptation techniques can be broadly grouped into two categories that aims to adapt the feature extractor or classifier to a given target task. We use α and β to denote task-specific weights for adapting the feature extractor and classifier respectively where $\vartheta = \{\alpha, \beta\}$.

Feature extractor adaptation. A simple method to adapt f_ϕ is finetuning its parameters on the support set [8, 13]. However, this strategy tends to suffer from the unproportionate optimization, *i.e.* updating very high-dimensional weights from a small number of support samples. In this paper, we propose to attach task-specific adapters directly to the existing task-agnostic model, *e.g.* we attach the adapters to each module of a ResNet backbone in Fig. 2 (a), and the adapters can be efficiently learned/estimated from few samples. Concretely, let f_{ϕ_l} denote the l-th layer of the feature

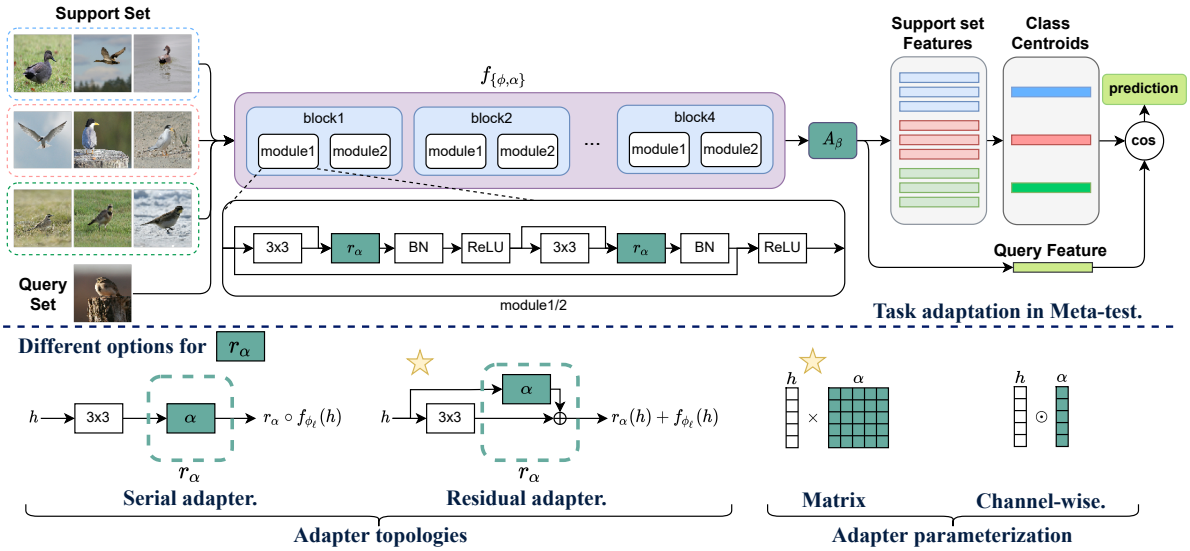

Figure 2. Illustration of our task adaptation for cross-domain few-shot learning. In meta-test stage (a), our method first attaches a parametric transformation r_α to each layer, where α can be constructed by (b) a serial or (c) a residual topology. They can be parameterized with matrix multiplication (d) or channel-wise scaling (e). We found that (c) is the best configuration with matrix parameterization which is further improved by attaching a linear transformation A_β to the end of the network. We adapt the network for a given task by optimizing α and A_β on a few labeled images from the support set, then map query images to the task-specific space and assign them to the nearest class center.

extractor f_ϕ (*i.e.* a convolutional layer) with the weights ϕ_l. Given a support set $\mathcal{S}$, the task-specific adapters r_α parameterized by α, can be incorporated to the output of the layer f_{ϕ_l} as

$$f_{\{\phi_l,\alpha\}}(\boldsymbol{h}) = r_\alpha(f_{\phi_l}(\boldsymbol{h}), \boldsymbol{h}) \tag{3}$$

where $\boldsymbol{h} \in \mathrm{R}^{W\times H\times C}$ is the input tensor, f_{ϕ_l} is a convolutional layer in f_ϕ. Importantly, the number of the task-specific adaptation parameters α are significantly smaller than the task-agnostic ones. The adapters can be designed in different ways.

Next we propose two connection types for incorporating r_α to f_{ϕ_l}: i) serial connection by subsequently applying it to the output of layer $f_{\phi_l}(\boldsymbol{h})$ as

$$f_{\{\phi_l,\alpha\}}(\boldsymbol{h}) = r_\alpha \circ f_{\phi_l}(\boldsymbol{h})$$

which is illustrated in Fig. 2(b), and ii) parallel connection by a residual addition as in [39]

$$f_{\{\phi_l,\alpha\}}(\boldsymbol{h}) = r_\alpha(\boldsymbol{h}) + f_{\phi_l}(\boldsymbol{h})$$

which is illustrated in Fig. 2(c). In our experiments, we found the parallel setting performing the best when α is learned on a support set during meta-test (illustrated in Fig. 2(c)) which we discuss in Sec. 3.

For the parameterization of r_α, we consider two options. Matrix multiplication (illustrated in Fig. 2(d)) with $\alpha \in \mathrm{R}^{C\times C}$:

$$r_\alpha(\boldsymbol{h}) = \boldsymbol{h} * \alpha,$$

where $*$ denotes a convolution, $\alpha \in \mathbb{R}^{C\times C}$ and the transformation is implemented as a convolutional operation with 1×1 kernels in our code. And channelwise scaling (illustrated in Fig. 2(e)):

$$r_\alpha(\boldsymbol{h}) = \boldsymbol{h} \odot \alpha,$$

where $\odot$ is a Hadamard product and $\alpha \in \mathbb{R}^C$. Note that one can also use an additive bias weight in both settings, however, this has not resulted in any significant gains in our experiments. While the matrix multiplication is more powerful than the scaling operation, it also requires more parameters to be estimated or learned. Note that, in a deep neural network, the number of input C_{in} and output channels C_{out} for a layer can be different. In that case, one can still use a non-square matrix: $\alpha \in \mathrm{R}^{C_{out}\times C_{in}}$, however, it is not possible to use a scaling operator in the parallel setting. In our experiments, we use ResNet architecture [17] where most input and output channels are the same. r_α connected in parallel with matrix multiplication form, when its parameters α are learned on the support set, is known as residual adapter [39] and r_α connected serial in channelwise is known as FiLM [36].

An alternative to reduce the dimensionality of α in case of matrix multiplication is matrix decomposition: $\alpha = V\gamma^\top$, where $V \in \mathrm{R}^{C\times B}$ and $\gamma \in \mathrm{R}^{C\times B}$, $B \ll C$. Using a bottleneck, *i.e.* setting $B < C/2$, reduces the number of parameters in the multiplication. In this work, we set $B = [C/N]$ and evaluate the performance for various N in Sec. 3.

Classifier learning. Finally, the adapted feature extractor $f_{(\phi,\alpha)}$ can be combined with a task-specific classifier c_β, parameterized by β to obtain the final model, *i.e.* $c \circ f_{(\phi,\alpha)}$.

Test Dataset	CNAPS [41]	Simple CNAPS [3]	TransductiveCNAPS [2]	SUR [15]	URT [29]	FLUTE [47]	tri-M [30]	URL [27]	Ours
ImageNet	50.8 ± 1.1	58.4 ± 1.1	57.9 ± 1.1	56.2 ± 1.0	56.8 ± 1.1	58.6 ± 1.0	51.8 ± 1.1	58.8 ± 1.1	$\mathbf{59.5 \pm 1.0}$
Omniglot	91.7 ± 0.5	91.6 ± 0.6	94.3 ± 0.4	94.1 ± 0.4	94.2 ± 0.4	92.0 ± 0.6	93.2 ± 0.5	94.5 ± 0.4	$\mathbf{94.9 \pm 0.4}$
Aircraft	83.7 ± 0.6	82.0 ± 0.7	84.7 ± 0.5	85.5 ± 0.5	85.8 ± 0.5	82.8 ± 0.7	87.2 ± 0.5	89.4 ± 0.4	$\mathbf{89.9 \pm 0.4}$
Birds	73.6 ± 0.9	74.8 ± 0.9	78.8 ± 0.7	71.0 ± 1.0	76.2 ± 0.8	75.3 ± 0.8	79.2 ± 0.8	80.7 ± 0.8	$\mathbf{81.1 \pm 0.8}$
Textures	59.5 ± 0.7	68.8 ± 0.9	66.2 ± 0.8	71.0 ± 0.8	71.6 ± 0.7	71.2 ± 0.8	68.8 ± 0.8	77.2 ± 0.7	$\mathbf{77.5 \pm 0.7}$
Quick Draw	74.7 ± 0.8	76.5 ± 0.8	77.9 ± 0.6	81.8 ± 0.6	82.4 ± 0.6	77.3 ± 0.7	79.5 ± 0.7	$\mathbf{82.5 \pm 0.6}$	81.7 ± 0.6
Fungi	50.2 ± 1.1	46.6 ± 1.0	48.9 ± 1.2	64.3 ± 0.9	64.0 ± 1.0	48.5 ± 1.0	58.1 ± 1.1	$\mathbf{68.1 \pm 0.9}$	66.3 ± 0.8
VGG Flower	88.9 ± 0.5	90.5 ± 0.5	$\mathbf{92.3 \pm 0.4}$	82.9 ± 0.8	87.9 ± 0.6	90.5 ± 0.5	91.6 ± 0.6	92.0 ± 0.5	92.2 ± 0.5
Traffic Sign	56.5 ± 1.1	57.2 ± 1.0	59.7 ± 1.1	51.0 ± 1.1	48.2 ± 1.1	63.0 ± 1.0	58.4 ± 1.1	63.3 ± 1.1	$\mathbf{82.8 \pm 1.0}$
MSCOCO	39.4 ± 1.0	48.9 ± 1.1	42.5 ± 1.1	52.0 ± 1.1	51.5 ± 1.1	52.8 ± 1.1	50.0 ± 1.0	57.3 ± 1.0	$\mathbf{57.6 \pm 1.0}$
MNIST	-	94.6 ± 0.4	94.7 ± 0.3	94.3 ± 0.4	90.6 ± 0.5	96.2 ± 0.3	95.6 ± 0.5	94.7 ± 0.4	$\mathbf{96.7 \pm 0.4}$
CIFAR-10	-	74.9 ± 0.7	73.6 ± 0.7	66.5 ± 0.9	67.0 ± 0.8	75.4 ± 0.8	78.6 ± 0.7	74.2 ± 0.8	$\mathbf{82.9 \pm 0.7}$
CIFAR-100	-	61.3 ± 1.1	61.8 ± 1.0	56.9 ± 1.1	57.3 ± 1.0	62.0 ± 1.0	67.1 ± 1.0	63.5 ± 1.0	$\mathbf{70.4 \pm 0.9}$
Average Seen	71.6	73.7	75.1	75.9	77.4	74.5	76.2	**80.4**	**80.4**
Average Unseen	-	67.4	66.5	64.1	62.9	69.9	69.9	70.6	**78.1**
Average All	-	71.2	71.8	71.4	71.8	72.7	73.8	76.6	**79.5**
Average Rank	-	6.1	5.5	5.6	5.5	4.8	4.4	2.5	**1.6**

Table 1. Comparison state-of-the-art methods on Meta-Dataset (using a multi-domain feature extractor of [27]). Mean accuracy, 95% confidence interval are reported. The first eight datasets are seen during training and the last five datasets are unseen and used for test only.

Based on the recent works, we investigate use of various linear classifiers in [8, 13, 26, 41], also nonparameteric ones including nearest centroid classifier (NCC) [32, 45] and their variants based on Mahalanobis distance (MD) [3]. Recently, it was shown in [27] that nonparametric classifiers can be successfully combined with a pre-classifier transformation. Concretely, the transformation in [27] that takes in the features computed from the network $f_{\{\phi,\alpha\}} \in \mathrm{R}^d$ and apply an affine transformation $A_\beta : \mathrm{R}^d \rightarrow \mathrm{R}^d$ parameterized by $\beta \in \mathrm{R}^{d \times d}$ to obtain the network embedding that is fed into the classifier, *i.e.* $p_{\phi,\vartheta} = c \circ A_\beta \circ f_{\{\phi,\alpha\}}$. Note that in the case of non-parametric classifier, c is not parameterized by β and we use β to denote the transformation parameters.

In our experiments, the best performing setting uses parallel adapters, whose parameters are in the matrix form, to adapt the feature extractor and followed by the pre-classifier transformation and NCC.

3. Experiments

Here we start with experimental setup, and then we compare our method to the state-of-the-art methods and rigorously evaluate various design decisions. We finally provide further analysis.

3.1. Experimental setup

Dataset. We use the Meta-Dataset [48] which is the standard benchmark for few-shot classification. It contains images from 13 diverse datasets and we follow the standard protocol in [48] (more details in the supplementary).

Implementation details. As in [3, 15, 27], we build our method on ResNet-18 [17] backbone, which is trained over eight training subdatasets by following [27] with the same hyperparameters in our experiments, unless stated otherwise. Once learned, we freeze its parameters and use them as the task-agnostic weights. For learning task-specific weights (ϑ), including the pre-classifier transformation β and the adapter parameters, we directly attach them to the task-agnostic weights and learn them on the support samples in meta-test by using Adadelta optimizer [52].

In the study of various task adaptation strategies in Section 3.3, we consider to only estimate the adapter parameters and learn the auxiliary network parameters by using Adam optimizer as in [3, 41] in meta-train. Note that estimation of pre-classifier and classifier weights via the auxiliary network leads to noisy and poor results and we do not report them. Similarly, we found that the auxiliary network fails to estimate very high-dimensional weights. Hence we only use it to estimate adapter weights that are parameterized with a vector for channelwise multiplication but not with a matrix.

3.2. Comparison to state-of-the-art methods

We evaluate our method in two settings, with multi-domain or single-domain feature extractor and compare our method to existing state-of-the-art methods. We also evaluate our method incorporated with different feature extractors, *i.e.* SDL, MDL, and URL in the supplementary.

Multi-domain feature extractor. Here we incorporate the proposed residual adapters in matrix form to the multi-domain feature extractor of [27] and compare its performance with the the state-of-the-art methods (CNAPS [41], SUR [15], URT [29], Simple CNAPS [3], Transductive CNAPS [2], FLUTE [47], tri-M [30], and URL [27]) in Tab. 1. To better analyze the results, we divide the table into two blocks that show the few-shot classification accuracy in previously seen domains and unseen domains along with their average accuracy. We also report average accuracy over all domains and the average rank as in [27, 47].[2] Simple CNAPS improves over CNAPS by adopting a simple Mahalanobis distance in stead of learning adapted linear classifier.

[2] As mentioned in https://github.com/google-research/meta-dataset/issues/54, we further update the evaluation protocol and report the updated results of all methods in the supplementary.

Test Dataset	ResNet-18							ResNet-34		
	Finetune [48]	ProtoNet [48]	fo-Proto-MAML [48]	ALFA+fo-Proto-MAML [48]	BOHB [44]	FLUTE [47]	Ours	ProtoNet [14]	CTX [14]	Ours
ImageNet	45.8 ± 1.1	50.5 ± 1.1	49.5 ± 1.1	52.8 ± 1.1	51.9 ± 1.1	46.9 ± 1.1	$\mathbf{59.5 \pm 1.1}$	53.7 ± 1.1	62.8 ± 1.0	$\mathbf{63.7 \pm 1.0}$
Omniglot	60.9 ± 1.6	60.0 ± 1.4	63.4 ± 1.3	61.9 ± 1.5	67.6 ± 1.2	61.6 ± 1.4	$\mathbf{78.2 \pm 1.2}$	68.5 ± 1.3	82.2 ± 1.0	$\mathbf{82.6 \pm 1.1}$
Aircraft	68.7 ± 1.3	53.1 ± 1.0	56.0 ± 1.0	63.4 ± 1.1	54.1 ± 0.9	48.5 ± 1.0	$\mathbf{72.2 \pm 1.0}$	58.0 ± 1.0	79.5 ± 0.9	$\mathbf{80.1 \pm 1.0}$
Birds	57.3 ± 1.3	68.8 ± 1.0	68.7 ± 1.0	69.8 ± 1.1	70.7 ± 0.9	47.9 ± 1.0	$\mathbf{74.9 \pm 0.9}$	74.1 ± 0.9	80.6 ± 0.9	$\mathbf{83.4 \pm 0.8}$
Textures	69.0 ± 0.9	66.6 ± 0.8	66.5 ± 0.8	70.8 ± 0.9	68.3 ± 0.8	63.8 ± 0.8	$\mathbf{77.3 \pm 0.7}$	68.8 ± 0.8	75.6 ± 0.6	$\mathbf{79.6 \pm 0.7}$
Quick Draw	42.6 ± 1.2	49.0 ± 1.1	51.5 ± 1.0	59.2 ± 1.2	50.3 ± 1.0	57.5 ± 1.0	$\mathbf{67.6 \pm 0.9}$	53.3 ± 1.1	$\mathbf{72.7 \pm 0.8}$	71.0 ± 0.8
Fungi	38.2 ± 1.0	39.7 ± 1.1	40.0 ± 1.1	41.5 ± 1.2	41.4 ± 1.1	31.8 ± 1.0	$\mathbf{44.7 \pm 1.0}$	40.7 ± 1.1	$\mathbf{51.6 \pm 1.1}$	51.4 ± 1.2
VGG Flower	85.5 ± 0.7	85.3 ± 0.8	87.2 ± 0.7	86.0 ± 0.8	87.3 ± 0.6	80.1 ± 0.9	$\mathbf{90.9 \pm 0.6}$	87.0 ± 0.7	$\mathbf{95.3 \pm 0.4}$	94.0 ± 0.5
Traffic Sign	66.8 ± 1.3	47.1 ± 1.1	48.8 ± 1.1	60.8 ± 1.3	51.8 ± 1.0	46.5 ± 1.1	$\mathbf{82.5 \pm 0.8}$	58.1 ± 1.1	$\mathbf{82.7 \pm 0.8}$	81.7 ± 0.9
MSCOCO	34.9 ± 1.0	41.0 ± 1.1	43.7 ± 1.1	48.1 ± 1.1	48.0 ± 1.0	41.4 ± 1.0	$\mathbf{59.0 \pm 1.0}$	41.7 ± 1.1	59.9 ± 1.0	$\mathbf{61.7 \pm 0.9}$
MNIST	-	-	-	-	-	80.8 ± 0.8	$\mathbf{93.9 \pm 0.6}$	-	-	$\mathbf{94.6 \pm 0.5}$
CIFAR-10	-	-	-	-	-	65.4 ± 0.8	$\mathbf{82.1 \pm 0.7}$	-	-	$\mathbf{86.0 \pm 0.6}$
CIFAR-100	-	-	-	-	-	52.7 ± 1.1	$\mathbf{70.7 \pm 0.9}$	-	-	$\mathbf{78.3 \pm 0.8}$
Average Seen	45.8	50.5	49.5	52.8	51.9	46.9	**59.5**	53.7	62.8	**63.7**
Average Unseen	58.2	56.7	58.4	62.4	60.0	53.2	**71.9**	61.1	75.6	**76.2**
Average All	57.0	56.1	57.5	61.4	59.2	52.6	**70.7**	60.4	74.3	**74.9**
Average Rank	7.9	8.3	7.0	5.3	6.0	8.9	**2.8**	5.5	1.8	**1.5**

Table 2. Comparison to state-of-the-art methods on Meta-Dataset (using a single-domain feature extractor which is trained only on ImageNet). Mean accuracy, 95% confidence interval are reported. Only ImageNet is seen during training and the rest datasets are unseen for test only.

Transductive CNAPS further improves by using unlabelled test images. SUR and URT fuse multi-domain features to get better performance. FLUTE improves URT by fusing FiLM parameters as initialization which is further finetuned on the support set in meta-test. tri-M adopts the same strategy of learning modulation parameters as CNAPS, where the parameters are further divided into the domain-specific set and the domain-cooperative set to explore the intra-domain information and inter-domain correlations, respectively. URL surpasses previous methods by learning a universal representation with distillation from multiple domains.

From the results, our method outperforms other methods on most domains (10 out of 13), especially obtaining significant improvement on 5 unseen datasets than the second best method, *i.e*. Average Unseen (+7.5). More specifically, our method obtains significant better results than the second best approach on Traffic Sign (+19.5), CIFAR-10 (+8.7), and CIFAR-100 (+6.8). Achieving improvement on unseen domains is more challenging due to the large gap between seen and unseen domain and the scarcity of labeled samples for the unseen task. We address this problem by attaching light-weight adapters to the feature extractor residually and learn the attached adapters on support set from scratch. This allows the model to learn more accurate and effective task-specific parameters (adapters) from the support set to efficiently steer the task-agnostic features for the unseen task, compared with predicting task-specific parameters by an auxiliary network learned in meta-train, *e.g*. Simple CNAPS, tri-M, or fusing representations from multiple feature extractors *e.g*. SUR, URT. Though FLUTE uses a hybrid approach which uses auxiliary networks learned from meta-train to initialize the FiLM parameters for further fine-tuning, their results are not better than URL, which achieves very competitive results as it learns a good universal representation that generalizes well to seen domains and can be further improved with the adaptation strategy proposed in this work, especially significant improvements on unseen domains.

Single-domain feature extractor. We also evaluate our method with a single-domain feature extractor trained on ImageNet only on ResNet-18 as in [48] or ResNet-34 as in [14]. This setting is more challenging than the multi-domain one, as the model is trained only on one domain and tested on both test split of ImageNet but also of other domains. We report the results of our method and state-of-the-art methods (BOHB [44], FLUTE [47], Finetune [48], ProtoNet [48], fo-Proto-MAML [48], and ALFA+fo-Proto-MAML [48], CTX [14]) in Tab. 2. ALFA+fo-Proto-MAML achieves the prior best performance by combining the complementary strengths of Prototypical Networks and MAML (fo-Proto-MAML), with extra meta-learning of per-step hyperparameters: learning rate and weight decay coefficients. FLUTE fails to surpass it with one training source domain, probably due to the lack of FiLM parameters from multiple domains. Our method, when using ResNet18 backbone, outperforms other methods on all domains, especially obtaining significant improvement, *i.e*. Average Unseen (+9.5), on 12 unseen datasets than the second best method. We compare our method to CTX and ProtoNet, which use ResNet-34 backbone. [3] CTX is very competitive by learning coarse spatial correspondence between the query and the support images with an attention mechanism. Ours is orthogonal to CTX and both CTX and our method can potentially be complementary, but we leave this as future work due to high computational cost of CTX. Specifically, we see that our method obtains the best average rank and outperforms CTX on most domains (6 out of 10) while our method being more

[3]Note that CTX also uses augmentation strategies such as AutoAugment [11] and other ones from SimClr [7]. We expect applying the same augmentation strategies to our method would yield further improvements, but we leave this for future work.

Test Dataset	classifier	Aux-Net or Ad	serial or residual	M or CW	β	#params	Image -Net	Omni -glot	Air- craft	Birds	Tex- tures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MS- COCO	MNIST	CIFAR -10	CIFAR -100
NCC	NCC	-	-	-	✗	-	57.0	94.4	88.0	80.3	74.6	81.8	66.2	91.5	49.8	54.1	91.1	70.6	59.1
MD	MD	-	-	-	✗	-	53.9	93.8	87.6	78.3	73.7	80.9	57.7	89.7	62.2	48.5	95.1	68.9	60.0
LR	LR	-	-	-	✗	-	56.0	93.7	88.3	79.7	74.7	80.0	62.1	91.1	59.7	51.2	93.5	73.1	60.1
SVM	SVM	-	-	-	✗	-	54.5	94.3	87.7	78.1	73.8	80.0	58.5	91.4	65.7	50.5	95.4	72.0	60.5
Finetune	NCC	-	-	-	✗	-	55.9	94.0	87.3	77.8	76.8	75.3	57.6	91.5	**86.1**	53.1	**96.8**	80.9	65.9
Aux-S-CW	NCC	Aux-Net	serial	CW	✗	76.98%	54.6	93.5	86.6	78.6	71.5	79.3	66.0	87.6	43.3	49.1	87.9	62.8	51.5
Aux-R-CW	NCC	Aux-Net	residual	CW	✗	76.98%	56.1	94.2	88.4	80.6	74.9	82.0	66.4	91.6	48.5	53.5	90.8	70.2	59.7
Aux-S-CW	MD	Aux-Net	serial	CW	✗	76.98%	55.1	93.8	86.8	77.4	73.2	79.9	57.4	88.1	58.4	50.1	92.7	66.5	55.7
Aux-R-CW	MD	Aux-Net	residual	CW	✗	76.98%	54.8	93.8	87.4	78.2	73.4	81.1	58.8	90.1	63.6	48.5	94.8	69.6	60.6
Ad-S-CW	NCC	Ad	serial	CW	✗	0.06%	56.8	94.8	89.3	80.7	74.5	81.6	65.8	91.3	73.9	53.6	95.7	78.4	64.3
Ad-R-CW	NCC	Ad	residual	CW	✗	1.57%	57.6	94.7	89.0	81.2	75.2	81.5	65.4	91.8	79.2	54.7	96.4	79.5	67.4
Ad-S-M	NCC	Ad	serial	M	✗	12.50%	56.2	94.4	89.1	80.6	75.8	81.6	**67.1**	92.1	67.6	54.8	95.9	78.9	66.6
Ad-R-M	NCC	Ad	residual	M	✗	10.93%	57.3	94.9	88.9	81.0	76.7	80.6	65.4	91.4	82.6	55.0	96.6	82.1	66.4
Ad-R-CW-PA	NCC	Ad	residual	CW	✓	3.91%	58.6	94.5	**90.0**	80.5	**77.6**	**81.9**	67.0	**92.2**	80.2	57.2	96.1	81.5	**71.4**
Ad-R-M-PA	NCC	Ad	residual	M	✓	13.27%	**59.5**	**94.9**	89.9	**81.1**	77.5	81.7	66.3	**92.2**	82.8	**57.6**	96.7	**82.9**	70.4

Table 3. Comparisons to methods that learn classifiers and model adaptation methods during meta-test stage based on URL model. NCC, MD, LR, SVM denote nearest centroid classifier, Mahalanobis distance, logistic regression, support vector machines respectively. 'Aux-Net or Ad' indicates using Auxiliary Network to predict α or attaching adapter α directly. 'M or CW' means using matrix multiplication or channel-wise scaling adapters. 'S' and 'R' denote serial adapter and residual adapter, respectively. 'β' indicates using the pre-classifier adaptation. The standard deviation results can be found in the supplementary. The first eight datasets are seen during training and the last five datasets are unseen and used for test only.

efficient (We train our model on one single Nvidia GPU for around 33 hours while CTX requires 8 Nvidia V100 GPUs and 7 days for training. Please refer to the supplementary for more details).

3.3. Analysis of task-specific parameterizations

Classifier learning. First we study the adaptation strategies for learning only a task-specific classifier on the pre-trained feature extractor of [27]. We evaluate non-parametric classifiers including nearest cetroid classifier (*NCC*) and NCC Mahalanobis Distance (*MD*) and parametric classifiers including logistic regression (*LR*), support vector machine *SVM* whose parameters are learned on support samples. We also include another baseline with NCC that finetunes all the feature extractor parameters, and report the results in Tab. 3. We observe that NCC obtains the best results for the seen domains and its performance is further improved by MD, while SVM achieves the best for the unseen domains among other classifiers. Finetuning baseline provides competitive results especially for the unseen domains. However, it performs poor in most seen domains.

Feature extractor adaptation. Next we analyze various design decisions for the feature extractor adaptation including connection types (serial, residual), *i.e*. Fig. 2(b), (c), its parameterization including channelwise modulation (*CW*) when they are estimated by an auxiliary network (*Aux-Net*), which has around 77% capacity of the feature extractor. We use with each combination with two nonparameteric classifier, either NCC or MD. *While the adaptation strategies using residual connections performs better than the serial one in almost all cases, the gains are more substantial when generalizing to unseen domains.* Learning adapter weights from few samples only can be very noisy. With residual addition, it is not necessary to change all connections for passing the information forward, which can improve the robustness of useful features and reduce learning burdens for new task, hence increase the generalization ability. While the serial connections may damage the previous learned structures. We also observe that NCC and MD obtain comparable performances. Note that Aux-S-CW with MD corresponds to our implementation of Simple CNAPS [3] with the more powerful feature extractor. We show that replacing its serial connection with a residual one leads to a strong performance boost.

Next we look at the adaptation strategy that learns the task-specific weights directly on the support set as in Eq. (2). We evaluate serial and residual connection types with channelwise and matrix parameterizations by using NCC. We denote this setting as Ad in Tab. 3. Note that we omit MD here, as it produces similar results to NCC. First we observe that learning the weights on the support set outperforms the strategy of estimating them through an auxiliary network almost in all cases. In addition, the learnable weights requires less number of parameters per task, while the capacity of auxiliary network is fixed. We again observe that the residual connections are more effective, especially when used with the matrix parameterization (Ad-R-M). However,the channelwise ones provide a good performance/computation tradeoff. Finally using the pre-classifier alignment (Ad-R-CW-PA and Ad-R-M-PA) further boosts the performance of the best models and we use our best model Ad-R-M-PA to compare against the state-of-the-art.

3.4. Further results

Varying-way Five-shot. After evaluating our method over a broad range of varying shots (*e.g*. up to 100 shots), we fol-

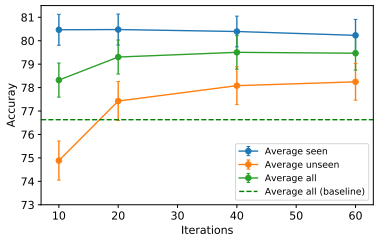

Figure 3. Sensitivity of performance to number of iterations.

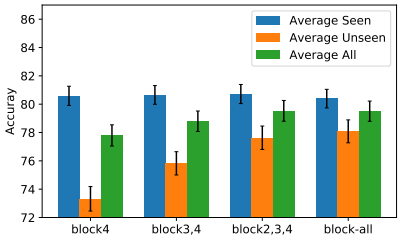

Figure 4. Block (layer) analysis for adapters.

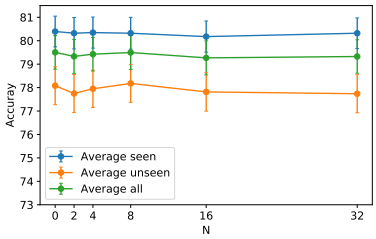

Figure 5. Decomposed residual adapters on block-3,4.

low [14, 27] to further analyze our method in 5-shot setting of varying number of categories. In this setting, we sample a varying number of ways with a fixed number of shots to form balanced support and query sets. As shown in Table 4, overall performance for all methods decreases in most datasets compared to results in Table 1 indicating that this is a more challenging setting. It is due to that five-shot setting samples much less support images per class than the standard setting. The top-2 methods remain the same and ours still outperforms the state-of-the-art URL when the number of support images per class is fewer, especially on unseen domains (Average Unseen +6.2).

Five-way One-shot. The similar conclusion can be drawn from this challenging case. Note that there are extremely few samples available for training in this case. As we can see, Ours achieves similar results with URL on seen domains but much better performance on unseen domains due to the learning of attached residual adapters is less over-fitting.

	Varying-Way Five-Shot					Five-Way One-Shot				
Test Dataset	Simple CNAPS [3]	SUR [15]	URT [29]	URL [27]	Ours	Simple CNAPS [3]	SUR [15]	URT [29]	URL [27]	Ours
Average Seen	69.0	71.2	73.8	76.6	**76.7**	65.0	64.0	70.6	73.4	**73.5**
Average Unseen	62.6	56.0	59.6	65.2	**71.4**	57.7	49.6	57.5	62.4	**63.4**
Average All	66.5	65.4	68.3	72.2	**74.6**	62.2	58.5	65.5	69.2	**69.6**
Average Rank	4.1	3.9	3.4	2.1	**1.5**	3.8	4.5	3.3	**1.7**	**1.7**

Table 4. Results of Varying-Way Five-Shot and Five-Way One-Shot scenarios. Mean accuracies are reported and more detailed results can be found in the supplementary.

3.5. Further ablation study

Here, we conduct ablation study for the sensitivity analysis for number of iterations, layer analysis for adapters, and decomposed residual adapters. We summarize results in figures and refer to supplementary for more detailed results.

Sensitivity analysis for number of iterations. In our method, we optimize the attached parameters (α, β) with 40 iterations. Figure 3 reports the results with 10, 20, 40, 60 iterations and indicates that our method (solid green) converges to a stable solution after 20 iterations and achieves better average performance on all domains than the baseline URL (dash green).

Layer analysis for adapters. Here we investigate whether it is sufficient to attach the adapters only to the later layers. We evaluate this on ResNet18 which is composed of four blocks and attach the adapters to only later blocks (block4, block3,4, block2,3,4 and block-all, see Fig. 2). Figure 4 shows that applying our adapters to only the last block (block4) obtains around 78% average accuracy on all domains which outperforms the URL. With attaching residual adapters to more layers, the performance on unseen domains is improved significantly while the one on seen domains remains stable.

Decomposing residual adapters. Here we investigate whether one can reduce the number of parameters in the adapters while retaining its performance by using matrix decomposition (see Sec. 2). As in deep neural network, the adapters in earlier layers are relatively small, we then decompose the adapters in the last two blocks only where the adapter dimensionality goes up to 512×512. Figure 5 shows that our method can achieve good performance with less parameters by decomposing large residual adapters, (*e.g.* when $N = 32$ where the number of additional parameters equal to around 4% vs 13%, the performance is still comparable to the original form of residual adapters, *i.e.* N=0). We refer to supplementary for more details.

4. Conclusion and Limitations

In this work, we investigate various strategies for adapting deep networks to few-shot classification tasks and show that light-weight adapters connected to a deep network with residual connections achieves strong adaptation to new tasks and domains only from few samples and obtains state-of-the-art performance while being efficient in the challenging Meta-Dataset benchmark. We demonstrate that the proposed solution can be incorporated to various feature extractors with a negligible increase in number of parameters.

Our method has limitations too. We build our method on existing backbones such ResNet-18 and ResNet-34, employ fixed adapter parameterizations and connection types which may not be optimal for every layer and task in multi-domain few-shot learning. Thus it would be desirable to have more flexible adapter structures that can be altered and tuned based on the target task.

Acknowledgments. HB is supported by the EPSRC programme grant Visual AI EP/T028572/1.

References

[1] Thomas Adler, Johannes Brandstetter, Michael Widrich, Andreas Mayr, David Kreil, Michael Kopp, Günter Klambauer, and Sepp Hochreiter. Cross-domain few-shot learning by representation fusion. *arXiv preprint arXiv:2010.06498*, 2020. 2

[2] Peyman Bateni, Jarred Barber, Jan-Willem van de Meent, and Frank Wood. Enhancing few-shot image classification with unlabelled examples. *arXiv preprint arXiv:2006.12245*, 2020. 5, 14

[3] Peyman Bateni, Raghav Goyal, Vaden Masrani, Frank Wood, and Leonid Sigal. Improved few-shot visual classification. In *CVPR*, pages 14493–14502, 2020. 2, 3, 5, 7, 8, 12, 13, 14, 17, 18, 19

[4] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. In *Advances in neural information processing systems*, pages 523–531, 2016. 2

[5] Hakan Bilen and Andrea Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv preprint arXiv:1701.07275*, 2017. 3

[6] Schroeder Brigit and Cui Yin. Fgvcx fungi classification challenge. *online*, 2018. 12

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020. 6

[8] Yinbo Chen, Xiaolong Wang, Zhuang Liu, Huijuan Xu, and Trevor Darrell. A new meta-baseline for few-shot learning. *arXiv preprint arXiv:2003.04390*, 2020. 2, 3, 5

[9] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *ICML*, pages 794–803. PMLR, 2018. 3

[10] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *CVPR*, pages 3606–3613, 2014. 12

[11] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, pages 113–123, 2019. 6

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. Ieee, 2009. 3

[13] Guneet S Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. In *ICLR*, 2020. 2, 3, 5

[14] Carl Doersch, Ankush Gupta, and Andrew Zisserman. Crosstransformers: spatially-aware few-shot transfer. In *NeurIPS*, 2020. 2, 6, 8, 12

[15] Nikita Dvornik, Cordelia Schmid, and Julien Mairal. Selecting relevant features from a multi-domain representation for few-shot classification. In *ECCV*, pages 769–786, 2020. 2, 3, 5, 8, 12, 14, 17, 18, 19

[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICLR*, pages 1126–1135, 2017. 1

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 4, 5, 12

[18] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020. 1

[19] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *IJCNN*, pages 1–8. Ieee, 2013. 12

[20] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. *Advances in neural information processing systems*, 29:667–675, 2016. 2

[21] Jonas Jongejan, Rowley Henry, Kawashima Takashi, Kim Jongmin, and Fox-Gieg Nick. The quick, draw! a.i. experiment. *online*, 2016. 12

[22] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009. 12

[23] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the annual meeting of the cognitive science society*, volume 33, 2011. 1

[24] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 2, 12

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 12

[26] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *CVPR*, pages 10657–10665, 2019. 2, 5

[27] Wei-Hong Li, Xialei Liu, and Hakan Bilen. Universal representation learning from multiple domains for few-shot classification. *ICCV*, 2021. 2, 3, 5, 7, 8, 12, 13, 14, 17, 18, 19

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014. 12

[29] Lu Liu, William Hamilton, Guodong Long, Jing Jiang, and Hugo Larochelle. A universal representation transformer layer for few-shot image classification. In *ICLR*, 2021. 2, 3, 5, 8, 14, 17, 18, 19

[30] Yanbin Liu, Juho Lee, Linchao Zhu, Ling Chen, Humphrey Shi, and Yi Yang. A multi-mode modulator for multi-domain few-shot classification. In *ICCV*, pages 8453–8462, 2021. 3, 5, 14

[31] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 12

[32] Thomas Mensink, Jakob Verbeek, Florent Perronnin, and Gabriela Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *TPAMI*, 35(11):2624–2637, 2013. 5

[33] Erik G Miller, Nicholas E Matsakis, and Paul A Viola. Learning from one example through shared densities on transforms. In *CVPR*, volume 1, pages 464–471. IEEE, 2000. 1

[34] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008. 12

[35] Boris N Oreshkin, Pau Rodriguez, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, 2018. 1

[36] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. 2, 4

[37] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016. 1

[38] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *NeurIPS*, 2017. 3

[39] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *CVPR*, pages 8119–8127, 2018. 4

[40] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. In *ICLR*, 2018. 2

[41] James Requeima, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E Turner. Fast and flexible multi-task classification using conditional neural adaptive processes. In *NeurIPS*, 2019. 2, 3, 5, 14

[42] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 12

[43] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *ICLR*, 2020. 1

[44] Tonmoy Saikia, Thomas Brox, and Cordelia Schmid. Optimized generic feature learning for few-shot classification across domains. *arXiv preprint arXiv:2001.07926*, 2020. 2, 6

[45] Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, 2017. 1, 5

[46] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? In *ECCV*, 2020. 3

[47] Eleni Triantafillou, Hugo Larochelle, Richard Zemel, and Vincent Dumoulin. Learning a universal template for few-shot dataset generalization. In *ICML*, 2021. 2, 3, 5, 6, 14

[48] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. In *ICLR*, 2020. 2, 5, 6, 12

[49] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NeurIPS*, 2016. 1, 2

[50] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. *California Institute of Technology*, 2011. 12

[51] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53(3):1–34, 2020. 1

[52] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 5, 12

A. Dataset

Meta-Dataset [48] is a few-shot classification benchmark that initially consists of ten datasets: ILSVRC_2012 [42] (ImageNet), Omniglot [24], FGVC-Aircraft [31] (Aircraft), CUB-200-2011 [50] (Birds), Describable Textures [10] (DTD), QuickDraw [21], FGVCx Fungi [6] (Fungi), VGG Flower [34] (Flower), Traffic Signs [19] and MSCOCO [28] then further expands with MNIST [25], CIFAR-10 [22] and CIFAR-100 [22]. We follow the standard procedure in [48] and consider both the 'Training on all datasets' (multi-domain learning) and 'Training on ImageNet only' (single-domain learning) settings. In 'Training on all datasets' setting, we follow the standard procedure and use the first eight datasets for meta-training, in which each dataset is further divided into train, validation and test set with disjoint classes. While the evaluation within these datasets is used to measure the generalization ability in the seen domains, the remaining five datasets are reserved as unseen domains in meta-test for measuring the cross-domain generalization ability. In 'Training on ImageNet only' setting, we follow the standard procedure and only use train split of ImageNet for meta-training. The evaluation of models is in the test split of ImageNet and the rest 12 datasets which are reserved as unseen domains in meta-test. As in [48], we evaluate our method on 600 randomly sampled tasks for each dataset with varying number of ways and shots, and report average accuracy and 95% confidence score in all experiments.

B. Implementation details

In this section, we explain the details of task-agnostic (feature extractor) learning and then task-specific (adapter) learning.

B.1. Task-agnostic learning

Here we consider learning the parameters of the feature extractor from either multiple or single domains.

Multi-domain learning. When we learn the feature extractor from multiple domains, we consider two cases. In the first case, which we call vanilla multiple domain learning (or MDL), we design a deep network where we share all the layers across all domains and have domain-specific classifiers. This setting corresponds to Eq (1) in the main text. Second we consider a variant of MDL, URL [27] which also involves learning a single network with shared and domain-specific layers as such, however, it is learned by distilling information from multiple domain-specific networks as described [27]. In these two settings, as in [3, 15, 27], we build MDL and URL on the ResNet-18 [17] backbone and use 84×84 image size.

For optimization of both MDL and URL, we follow the same protocol in [27], use SGD optimizer and cosine annealing with a weight decay of 7×10^{-4} for learning 240,000 iterations. The learning rate is 0.03 and the annealing frequency is 48,000. As in [27], the batch size for ImageNet is 64×7 and is 64 for the other 7 datasets. We refer readers to [27] for more details.

Single domain learning (SDL). We also evaluate our method on a feature extractor that is learned on single domain which we call SDL. Here we evaluate our method on two backbones, ResNet-18 (SDL-ResNet-18) and ResNet34 (SDL-ResNet-34).

Backbone	learning rate	batch size	annealing freq.	max. iter.
SDL-ResNet-18	3×10^{-2}	64	48,000	480,000
SDL-ResNet-34	3×10^{-2}	128	48,000	480,000

Table 5. Training hyper-parameters of single domain learning.

SDL-ResNet-18. Following [15,27,48], we train a ResNet-18 on the train split of ImageNet and use 84×84 image size, which is denoted as SDL-ResNet-18. For optimization, we follow the training protocol in [15, 27]. Specifically, we use SGD optimizer and cosine annealing for all experiments with a momentum of 0.9 and a weight decay of 7×10^{-4}. Some other hyperparameters are shown in Tab. 5 as in [15, 27]. To regularize training, we also use the exact same data augmentations as in [15, 27], *e.g.* random crops and random color augmentations.

SDL-ResNet-34. We also apply our method to the single domain learning model with ResNet-34 backbone learned on ImageNet only as in [14]. We follow [14] and use higher-resolution (224×224) images for meta-training and meta-testing. For optimization, we follow the training protocol as in [15, 27]. Specifically, we use SGD optimizer and cosine annealing with a momentum of 0.9, a weight decay of 1×10^{-4} with a batch size of 128. Other hyperparameters are the same as in SDL-ResNet-18 and are shown in Tab. 5. To regularize training, we also use the exact same data augmentations as in [15, 27], *e.g.* random crops and random color augmentations with an additional stage that randomly downsamples and upsamples images as in [14].

B.2. Task-specific learning

Attaching and learning adapters. For the optimization of the adaptation parameters α which is attached directly and learned on support set and the pre-classifier adaptation β, we follow the optimization strategy in [27], initialize β as an identity matrix and optimize both α and β for 40 iterations using Adadelta [52] as optimizer. The learning rate of β is 0.1 for first eight datasets and 1 for the last five datasets as in [27] and we set the learning rate of α as half of the learning rate of β, *i.e.* 0.05 for the first eight datasets and 0.5 for the last five datasets. Note that, we learn α and β on a per-task basis using the task's support set during meta-test. That is, α and β are not re-used across the test tasks drawn from $\mathcal{D}_t$.

Test Dataset	ImageNet	Omniglot	Aircraft	Birds	Textures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MSCOCO	MNIST	CIFAR-10	CIFAR-100
MDL	53.4 ± 1.1	93.8 ± 0.4	86.6 ± 0.5	78.6 ± 0.8	71.4 ± 0.7	81.5 ± 0.6	61.9 ± 1.0	88.7 ± 0.6	51.0 ± 1.0	49.7 ± 1.1	94.4 ± 0.3	66.7 ± 0.8	53.6 ± 1.0
Ours (MDL)	$\mathbf{55.6 \pm 1.0}$	$\mathbf{94.3 \pm 0.4}$	$\mathbf{86.7 \pm 0.5}$	$\mathbf{79.4 \pm 0.8}$	$\mathbf{73.2 \pm 0.8}$	$\mathbf{81.7 \pm 0.6}$	$\mathbf{64.0 \pm 0.9}$	$\mathbf{90.9 \pm 0.5}$	$\mathbf{81.1 \pm 0.9}$	$\mathbf{51.4 \pm 1.1}$	$\mathbf{96.9 \pm 0.3}$	$\mathbf{78.5 \pm 0.8}$	$\mathbf{64.3 \pm 1.1}$
URL [27]	58.8 ± 1.1	94.5 ± 0.4	89.4 ± 0.4	80.7 ± 0.8	77.2 ± 0.7	$\mathbf{82.5 \pm 0.6}$	$\mathbf{68.1 \pm 0.9}$	92.0 ± 0.5	63.3 ± 1.2	57.3 ± 1.0	94.7 ± 0.4	74.2 ± 0.8	63.6 ± 1.0
Ours (URL)	$\mathbf{59.5 \pm 1.0}$	$\mathbf{94.9 \pm 0.4}$	$\mathbf{89.9 \pm 0.4}$	$\mathbf{81.1 \pm 0.8}$	$\mathbf{77.5 \pm 0.7}$	81.7 ± 0.6	66.3 ± 0.9	$\mathbf{92.2 \pm 0.5}$	$\mathbf{82.8 \pm 1.0}$	$\mathbf{57.6 \pm 1.0}$	$\mathbf{96.7 \pm 0.4}$	$\mathbf{82.9 \pm 0.7}$	$\mathbf{70.4 \pm 1.0}$
SDL-ResNet-18	55.8 ± 1.0	67.4 ± 1.2	49.5 ± 0.9	71.2 ± 0.9	73.0 ± 0.6	53.9 ± 1.0	41.6 ± 1.0	87.0 ± 0.6	47.4 ± 1.1	53.5 ± 1.0	78.1 ± 0.7	67.3 ± 0.8	56.6 ± 0.9
Ours (SDL-ResNet-18)	$\mathbf{59.5 \pm 1.1}$	$\mathbf{78.2 \pm 1.2}$	$\mathbf{72.2 \pm 1.0}$	$\mathbf{74.9 \pm 0.9}$	$\mathbf{77.3 \pm 0.7}$	$\mathbf{67.6 \pm 0.9}$	$\mathbf{44.7 \pm 1.0}$	$\mathbf{90.9 \pm 0.6}$	$\mathbf{82.5 \pm 0.8}$	$\mathbf{59.0 \pm 1.0}$	$\mathbf{93.9 \pm 0.6}$	$\mathbf{82.1 \pm 0.7}$	$\mathbf{70.7 \pm 0.9}$
SDL-ResNet-34	62.2 ± 1.1	72.8 ± 1.1	62.9 ± 0.9	79.6 ± 0.8	75.6 ± 0.6	64.5 ± 0.8	47.4 ± 1.1	90.4 ± 0.6	54.8 ± 1.0	56.1 ± 1.0	79.3 ± 0.6	83.0 ± 0.6	74.8 ± 0.8
Ours (SDL-ResNet-34)	$\mathbf{63.7 \pm 1.0}$	$\mathbf{82.6 \pm 1.1}$	$\mathbf{80.1 \pm 1.0}$	$\mathbf{83.4 \pm 0.8}$	$\mathbf{79.6 \pm 0.7}$	$\mathbf{71.0 \pm 0.8}$	$\mathbf{51.4 \pm 1.2}$	$\mathbf{94.0 \pm 0.5}$	$\mathbf{81.7 \pm 0.9}$	$\mathbf{61.7 \pm 0.9}$	$\mathbf{94.6 \pm 0.5}$	$\mathbf{86.0 \pm 0.6}$	$\mathbf{78.3 \pm 0.8}$

Table 6. Results of attaching residual adapters to different baselines. 'SDL-ResNet-18' is the single domain model with ResNet-18 backbone pretrained on ImageNet. 'SDL-ResNet-34' is the single domain model with ResNet-34 backbone pretrained on ImageNet. 'MDL' is a vanilla Multi-Domain Learning (MDL) model trained on eight seen datasets jointly.

Test Dataset	classifier	Aux-Net or Ad	serial or parallel	M or CW	β	#params	ImageNet	Omniglot	Aircraft	Birds	Textures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MSCOCO	MNIST	CIFAR-10	CIFAR-100
NCC	NCC	-	-	-	✗	-	57.0 ± 1.1	94.4 ± 0.4	88.0 ± 0.5	80.3 ± 0.7	74.6 ± 0.7	81.8 ± 0.6	66.2 ± 0.9	91.5 ± 0.5	49.8 ± 1.1	54.1 ± 1.0	91.1 ± 0.4	70.6 ± 0.7	59.1 ± 1.0
MD	MD	-	-	-	✗	-	53.9 ± 1.0	93.8 ± 0.5	87.6 ± 0.5	78.3 ± 0.7	73.7 ± 0.7	80.9 ± 0.7	57.7 ± 0.9	89.7 ± 0.6	62.2 ± 1.1	48.5 ± 1.0	95.1 ± 0.4	68.9 ± 0.8	60.0 ± 0.9
LR	LR	-	-	-	✗	-	56.0 ± 1.1	93.7 ± 0.5	88.3 ± 0.6	79.7 ± 0.8	74.7 ± 0.7	80.0 ± 0.7	62.1 ± 0.8	91.1 ± 0.5	59.7 ± 1.1	51.2 ± 1.1	93.5 ± 0.5	73.1 ± 0.8	60.1 ± 1.1
SVM	SVM	-	-	-	✗	-	54.5 ± 1.1	94.3 ± 0.5	87.7 ± 0.5	78.1 ± 0.8	73.8 ± 0.8	80.0 ± 0.6	58.5 ± 0.9	91.4 ± 0.6	65.7 ± 1.2	50.5 ± 1.0	95.4 ± 0.4	72.0 ± 0.8	60.5 ± 1.1
Softmax	Softmax	-	-	-	✗	-	42.2 ± 1.0	85.3 ± 0.7	71.9 ± 0.8	59.6 ± 1.0	62.0 ± 0.8	61.2 ± 1.0	37.3 ± 0.9	66.7 ± 1.0	51.4 ± 1.1	48.2 ± 1.1	93.5 ± 0.5	70.4 ± 0.8	59.3 ± 1.0
KNN	KNN	-	-	-	✗	-	48.1 ± 1.1	94.1 ± 0.4	84.5 ± 0.6	70.7 ± 0.8	65.9 ± 0.8	74.8 ± 0.7	53.5 ± 0.9	86.0 ± 0.6	56.9 ± 1.2	44.7 ± 1.1	91.4 ± 0.5	60.3 ± 0.8	49.4 ± 1.0
PA	NCC	-	-	-	✓	-	58.8 ± 1.1	94.5 ± 0.4	89.4 ± 0.4	80.7 ± 0.8	77.2 ± 0.7	$\mathbf{82.5 \pm 0.6}$	$\mathbf{68.1 \pm 0.9}$	92.0 ± 0.5	63.3 ± 1.1	57.3 ± 1.0	94.7 ± 0.4	74.2 ± 0.8	63.5 ± 1.0
PA	Softmax	-	-	-	✓	-	53.4 ± 1.2	92.7 ± 0.5	85.7 ± 0.6	76.1 ± 0.9	73.9 ± 0.8	76.5 ± 0.8	51.1 ± 0.9	86.9 ± 0.7	52.5 ± 1.1	48.2 ± 1.1	94.3 ± 0.4	69.7 ± 0.8	60.4 ± 1.0
Finetune	NCC	-	-	-	✗	-	55.9 ± 1.2	94.0 ± 0.5	87.3 ± 0.6	77.8 ± 0.9	76.8 ± 0.8	75.3 ± 0.9	57.6 ± 1.1	91.5 ± 0.6	$\mathbf{86.1 \pm 0.9}$	53.1 ± 1.2	$\mathbf{96.8 \pm 0.4}$	80.9 ± 0.8	65.9 ± 1.1
Finetune	Softmax	-	-	-	✗	-	48.4 ± 1.2	92.2 ± 0.6	81.6 ± 0.9	70.3 ± 1.3	72.0 ± 0.9	73.5 ± 1.0	44.2 ± 1.1	90.3 ± 0.7	65.5 ± 1.4	41.0 ± 1.3	96.3 ± 0.4	71.6 ± 1.0	53.8 ± 1.4
Aux-S-CW	NCC	Aux-Net	serial	CW	✗	-	54.6 ± 1.1	93.5 ± 0.5	86.6 ± 0.5	78.6 ± 0.8	71.5 ± 0.7	79.3 ± 0.6	66.0 ± 0.9	87.6 ± 0.6	43.3 ± 0.9	49.1 ± 1.0	87.9 ± 0.5	62.8 ± 0.8	51.5 ± 1.0
Aux-R-CW	NCC	Aux-Net	residual	CW	✗	-	56.1 ± 1.1	94.2 ± 0.4	88.4 ± 0.5	80.6 ± 0.7	74.9 ± 0.6	82.0 ± 0.6	66.4 ± 0.9	91.6 ± 0.5	48.5 ± 1.0	53.5 ± 1.0	90.8 ± 0.5	70.2 ± 0.8	59.7 ± 1.0
Aux-S-CW	MD	Aux-Net	serial	CW	✗	-	55.1 ± 1.1	93.8 ± 0.5	86.8 ± 0.5	77.4 ± 0.8	73.2 ± 0.8	79.9 ± 0.7	57.4 ± 0.9	88.1 ± 0.7	58.4 ± 1.1	50.1 ± 1.1	92.7 ± 0.5	66.5 ± 0.8	55.7 ± 1.1
Aux-R-CW	MD	Aux-Net	residual	CW	✗	-	54.8 ± 1.1	93.8 ± 0.5	87.4 ± 0.5	78.2 ± 0.7	73.4 ± 0.7	81.1 ± 0.7	58.8 ± 0.9	90.1 ± 0.5	63.6 ± 1.2	48.5 ± 1.1	94.8 ± 0.4	69.6 ± 0.8	60.6 ± 0.9
Ad-S-CW	NCC	Ad	serial	CW	✗	0.06%	56.8 ± 1.1	94.8 ± 0.4	89.3 ± 0.5	80.7 ± 0.7	74.5 ± 0.7	81.6 ± 0.6	65.8 ± 0.9	91.3 ± 0.5	73.9 ± 1.1	53.6 ± 1.1	95.7 ± 0.4	78.4 ± 0.7	64.3 ± 1.0
Ad-R-CW	NCC	Ad	residual	CW	✗	1.57%	57.6 ± 1.1	94.7 ± 0.4	89.0 ± 0.4	81.2 ± 0.8	75.2 ± 0.7	81.5 ± 0.6	65.4 ± 0.8	91.8 ± 0.5	79.2 ± 1.1	54.7 ± 1.1	96.4 ± 0.4	79.5 ± 0.8	67.4 ± 1.0
Ad-S-M	NCC	Ad	serial	M	✗	12.50%	56.2 ± 1.1	94.4 ± 0.4	89.1 ± 0.5	80.6 ± 0.7	75.8 ± 0.7	81.6 ± 0.6	67.1 ± 0.9	92.1 ± 0.4	67.6 ± 1.2	54.8 ± 1.1	95.9 ± 0.4	78.9 ± 0.7	66.6 ± 1.1
Ad-R-M	NCC	Ad	residual	M	✗	10.93%	57.3 ± 1.1	94.9 ± 0.4	88.9 ± 0.5	81.0 ± 0.7	76.7 ± 0.7	80.6 ± 0.6	65.4 ± 0.9	91.4 ± 0.5	82.6 ± 1.0	55.0 ± 1.1	96.6 ± 0.4	82.1 ± 0.7	66.4 ± 1.1
Ad-R-CW-PA	NCC	Ad	residual	CW	✓	3.91%	58.6 ± 1.1	94.5 ± 0.4	$\mathbf{90.0 \pm 0.4}$	80.5 ± 0.8	$\mathbf{77.6 \pm 0.7}$	81.9 ± 0.6	67.0 ± 0.9	92.2 ± 0.5	80.2 ± 0.9	57.2 ± 1.0	96.1 ± 0.4	81.5 ± 0.8	$\mathbf{71.4 \pm 0.9}$
Ad-R-M-PA	NCC	Ad	residual	M	✓	13.27%	$\mathbf{59.5 \pm 1.0}$	$\mathbf{94.9 \pm 0.4}$	89.9 ± 0.4	$\mathbf{81.1 \pm 0.8}$	77.5 ± 0.7	81.7 ± 0.6	66.3 ± 0.9	$\mathbf{92.2 \pm 0.5}$	82.8 ± 1.0	$\mathbf{57.6 \pm 1.0}$	96.7 ± 0.4	$\mathbf{82.9 \pm 0.7}$	70.4 ± 1.0

Table 7. Comparisons to methods that learn classifiers and model adaptation methods during meta-test stage based on URL model. NCC, MD, LR, SVM, Softmax, KNN denote nearest centroid classifier, Mahalanobis distance, logistic regression, support vector machines, softmax classifier and k-nearest neighbors classifier respectively. PA indicates pre-classifier alignment. 'Aux-Net or Ad' indicates using Auxiliary Network to predict α or attaching adapter α directly. 'M or CW' means using matrix multiplication or channel-wise scaling adapters. 'S' and 'R' denote serial adapter and residual adapter, respectively. 'β' indicates using the pre-classifier adaptation. Mean accuracy, 95% confidence interval are reported. The first eight datasets are seen during training and the last five datasets are unseen and used for test only.

Predicting r_α. In case of modulating α with the auxiliary network, we follow the auxiliary training protocols in [3]. We train for 10K episodes to optimize the task encoder using Adam with a learning rate of 1×10^{-5} on eight training domains in meta-train. We validate every 5K iterations to save the best model for test.

C. More results

C.1. Our method with different feature extractors

Tab. 6 shows the results of our method (the proposed residual adapters in matrix form) when incorporated to different feature extractors, single domain model with ResNet-18 backbone (SDL-ResNet-18) pre-trained on ImageNet, single domain model with ResNet-34 (SDL-ResNet-34) pre-trained on ImageNet, vanilla multi-domain learning (MDL) and URL [27]. We see that attaching and learning residual adapters can significantly improve the performance on all domains over SDL-ResNet-18, SDL-ResNet-34 and MDL and obtain better performance on most domains over URL (11 out of 13 domains). This strongly indicates that our method can efficiently adapt the model for unseen categories and domains with few support samples while being agnostic to the feature extractor with different backbone and resolution of images.

C.2. Task-specific parameterizations

In Tab. 7, we report additional 95% confidence interval of each dataset to the main paper for the comparison of different r_α choices based on the URL model. The first eight datasets are seen during training and the last five datasets are unseen and used for test only. We can see that the confidence intervals for different methods have marginal differences.

C.3. Varying-way 5-shot and 5-way-1-shot

In the main paper, we only report the average accuracy of Varying-Way Five-Shot and Five-Way One-Shot scenarios due to limited space, and detailed results are depicted in Tab. 8. In the table, we report the Mean accuracy, 95% confidence interval of each dataset. The first eight datasets are seen during training and the last five datasets are unseen and used for test only. URT and URL are two strong baselines surpassing both Simple CNAPS and SUR, while Ours outperforms them on most datasets, especially on unseen

Test Dataset	Varying-Way Five-Shot					Five-Way One-Shot				
	Simple CNAPS [3]	SUR [15]	URT [29]	URL [27]	Ours	Simple CNAPS [3]	SUR [15]	URT [29]	URL [27]	Ours
ImageNet	47.2 ± 1.0	46.7 ± 1.0	48.6 ± 1.0	$\mathbf{49.4 \pm 1.0}$	48.3 ± 1.0	42.6 ± 0.9	40.7 ± 1.0	47.4 ± 1.0	$\mathbf{49.6 \pm 1.1}$	48.0 ± 1.0
Omniglot	95.1 ± 0.3	95.8 ± 0.3	96.0 ± 0.3	96.0 ± 0.3	$\mathbf{96.8 \pm 0.3}$	93.1 ± 0.5	93.0 ± 0.7	95.6 ± 0.5	95.8 ± 0.5	$\mathbf{96.3 \pm 0.4}$
Aircraft	74.6 ± 0.6	82.1 ± 0.6	81.2 ± 0.6	84.8 ± 0.5	$\mathbf{85.5 \pm 0.5}$	65.8 ± 0.9	67.1 ± 1.4	77.9 ± 0.9	$\mathbf{79.6 \pm 0.9}$	$\mathbf{79.6 \pm 0.9}$
Birds	69.6 ± 0.7	62.8 ± 0.9	71.2 ± 0.7	76.0 ± 0.6	$\mathbf{76.6 \pm 0.6}$	67.9 ± 0.9	59.2 ± 1.0	70.9 ± 0.9	$\mathbf{74.9 \pm 0.9}$	74.5 ± 0.9
Textures	57.5 ± 0.7	60.2 ± 0.7	65.2 ± 0.7	$\mathbf{69.1 \pm 0.6}$	68.3 ± 0.7	42.2 ± 0.8	42.5 ± 0.8	49.4 ± 0.9	53.6 ± 0.9	$\mathbf{54.5 \pm 0.9}$
Quick Draw	70.9 ± 0.6	79.0 ± 0.5	$\mathbf{79.2 \pm 0.5}$	78.2 ± 0.5	77.9 ± 0.6	70.5 ± 0.9	$\mathbf{79.8 \pm 0.9}$	79.6 ± 0.9	79.0 ± 0.8	79.3 ± 0.9
Fungi	50.3 ± 1.0	66.5 ± 0.8	66.9 ± 0.9	70.0 ± 0.8	$\mathbf{70.4 \pm 0.8}$	58.3 ± 1.1	64.8 ± 1.1	71.0 ± 1.0	75.2 ± 1.0	$\mathbf{75.3 \pm 1.0}$
VGG Flower	86.5 ± 0.4	76.9 ± 0.6	82.4 ± 0.5	89.3 ± 0.4	$\mathbf{89.5 \pm 0.4}$	79.9 ± 0.7	65.0 ± 1.0	72.7 ± 0.0	79.9 ± 0.8	$\mathbf{80.3 \pm 0.8}$
Traffic Sign	55.2 ± 0.8	44.9 ± 0.9	45.1 ± 0.9	57.5 ± 0.8	$\mathbf{72.3 \pm 0.6}$	55.3 ± 0.9	44.6 ± 0.9	52.7 ± 0.9	$\mathbf{57.9 \pm 0.9}$	57.2 ± 1.0
MSCOCO	49.2 ± 0.8	48.1 ± 0.9	52.3 ± 0.9	$\mathbf{56.1 \pm 0.8}$	56.0 ± 0.8	48.8 ± 0.9	47.8 ± 1.1	56.9 ± 1.1	59.2 ± 1.0	$\mathbf{59.9 \pm 1.0}$
MNIST	88.9 ± 0.4	90.1 ± 0.4	86.5 ± 0.5	89.7 ± 0.4	$\mathbf{92.5 \pm 0.4}$	$\mathbf{80.1 \pm 0.9}$	77.1 ± 0.9	75.6 ± 0.9	78.7 ± 0.9	$\mathbf{80.1 \pm 0.9}$
CIFAR-10	66.1 ± 0.7	50.3 ± 1.0	61.4 ± 0.7	66.0 ± 0.7	$\mathbf{72.0 \pm 0.7}$	50.3 ± 0.9	35.8 ± 0.8	47.3 ± 0.9	54.7 ± 0.9	$\mathbf{55.8 \pm 0.9}$
CIFAR-100	53.8 ± 0.9	46.4 ± 0.9	52.5 ± 0.9	57.0 ± 0.9	$\mathbf{64.1 \pm 0.8}$	53.8 ± 0.9	42.9 ± 1.0	54.9 ± 1.1	61.8 ± 1.0	$\mathbf{63.7 \pm 1.0}$
Average Seen	69.0	71.2	73.8	76.6	**76.7**	65.0	64.0	70.6	73.4	**73.5**
Average Unseen	62.6	56.0	59.6	65.2	**71.4**	57.7	49.6	57.5	62.4	**63.4**
Average All	66.5	65.4	68.3	72.2	**74.6**	62.2	58.5	65.5	69.2	**69.6**
Average Rank	4.1	3.9	3.4	2.1	**1.5**	3.8	4.5	3.3	**1.7**	**1.7**

Table 8. Results of Varying-Way Five-Shot and Five-Way One-Shot scenarios. Mean accuracy, 95% confidence interval are reported.

Test Dataset	CNAPS [41]	Simple CNAPS [3]	TransductiveCNAPS [2]	SUR [15]	URT [29]	FLUTE [47]	tri-M [30]	URL [27]	Ours
ImageNet	50.8 ± 1.1	56.5 ± 1.1	57.9 ± 1.1	54.5 ± 1.1	55.0 ± 1.1	51.8 ± 1.1	$\mathbf{58.6 \pm 1.0}$	57.5 ± 1.1	57.4 ± 1.1
Omniglot	91.7 ± 0.5	91.9 ± 0.6	94.3 ± 0.4	93.0 ± 0.5	93.3 ± 0.5	93.2 ± 0.5	92.0 ± 0.6	94.5 ± 0.4	$\mathbf{95.0 \pm 0.4}$
Aircraft	83.7 ± 0.6	83.8 ± 0.6	84.7 ± 0.5	84.3 ± 0.5	84.5 ± 0.6	87.2 ± 0.5	82.8 ± 0.7	88.6 ± 0.5	$\mathbf{89.3 \pm 0.4}$
Birds	73.6 ± 0.9	76.1 ± 0.9	78.8 ± 0.7	70.4 ± 1.1	75.8 ± 0.8	79.2 ± 0.8	75.3 ± 0.8	80.5 ± 0.7	$\mathbf{81.4 \pm 0.7}$
Textures	59.5 ± 0.7	70.0 ± 0.8	66.2 ± 0.8	70.5 ± 0.7	70.6 ± 0.7	68.8 ± 0.8	71.2 ± 0.8	76.2 ± 0.7	$\mathbf{76.7 \pm 0.7}$
Quick Draw	74.7 ± 0.8	78.3 ± 0.7	77.9 ± 0.6	81.6 ± 0.6	$\mathbf{82.1 \pm 0.6}$	79.5 ± 0.7	77.3 ± 0.7	81.9 ± 0.6	82.0 ± 0.6
Fungi	50.2 ± 1.1	49.1 ± 1.2	48.9 ± 1.2	65.0 ± 1.0	63.7 ± 1.0	58.1 ± 1.1	48.5 ± 1.0	$\mathbf{68.8 \pm 0.9}$	67.4 ± 1.0
VGG Flower	88.9 ± 0.5	91.3 ± 0.6	$\mathbf{92.3 \pm 0.4}$	82.2 ± 0.8	88.3 ± 0.6	91.6 ± 0.6	90.5 ± 0.5	92.1 ± 0.5	92.2 ± 0.5
Traffic Sign	56.5 ± 1.1	59.2 ± 1.0	59.7 ± 1.1	49.8 ± 1.1	50.1 ± 1.1	58.4 ± 1.1	63.0 ± 1.0	63.3 ± 1.2	$\mathbf{83.5 \pm 0.9}$
MSCOCO	39.4 ± 1.0	42.4 ± 1.1	42.5 ± 1.1	49.4 ± 1.1	48.9 ± 1.1	50.0 ± 1.0	52.8 ± 1.1	54.0 ± 1.0	$\mathbf{55.8 \pm 1.1}$
MNIST	-	94.3 ± 0.4	94.7 ± 0.3	94.9 ± 0.4	90.5 ± 0.4	95.6 ± 0.5	96.2 ± 0.3	94.5 ± 0.5	$\mathbf{96.7 \pm 0.4}$
CIFAR-10	-	72.0 ± 0.8	73.6 ± 0.7	64.2 ± 0.9	65.1 ± 0.8	78.6 ± 0.7	75.4 ± 0.8	71.9 ± 0.7	$\mathbf{80.6 \pm 0.8}$
CIFAR-100	-	60.9 ± 1.1	61.8 ± 1.0	57.1 ± 1.1	57.2 ± 1.0	67.1 ± 1.0	62.0 ± 1.0	62.6 ± 1.0	$\mathbf{69.6 \pm 1.0}$
Average Seen	71.6	74.6	75.1	75.2	76.7	76.2	74.5	80.0	80.2
Average Unseen	-	65.8	66.5	63.1	62.4	69.9	69.9	69.3	77.2
Average All	-	71.2	71.8	70.5	71.2	73.8	72.7	75.9	79.0
Average Rank	-	6.3	4.9	5.8	5.7	4.3	4.8	2.7	1.5

Table 9. Comparison state-of-the-art methods on Meta-Dataset (using a multi-domain feature extractor of [27]). Mean accuracy, 95% confidence interval are reported. The first eight datasets are seen during training and the last five datasets are unseen and used for test only.

domains.

C.4. Results evaluated with updated evaluation protocol.

As the code from Meta-dataset has been updated, we evaluate all methods with the updated evaluation protocol from the Meta-dataset [4] and report the results [5] in Tab. 9. As shown in Tab. 9, the update does not affect much on the results and our method rank 1.5 in average and the state-of-the-art method URL rank 2.7. Our method outperforms other methods on most domains (9 out of 13), especially obtaining significant improvement on 5 unseen datasets than the second best method, *i.e*. Average Unseen (+7.9). More specifically, our method obtains significant better results than the second best approach (URL) on Traffic Sign (+20.2), CIFAR-10 (+8.7), and CIFAR-100 (+7.0).

C.5. Ablation study

Here, we conduct ablation study of our method with the URL model, unless stated otherwise.

Sensitivity analysis for number of iterations. In our method, we optimize the attached parameters (α, β) with 40 iterations. Figure 6 and Figure 7 report the results with 10, 20, 40, 60 iterations and indicates that our method (solid green) converges to a stable solution after 20 iterations and achieves better average performance on all domains than the baseline URL (dash green). The mean accuracy with 95%

[4] As mentioned in https://github.com/google-research/meta-dataset/issues/54, we also set the shuffle_buffer_size as 1000 to evaluate all methods and report the results in Tab. 9. This change does not affect much on the results as the datasets we used were shuffled using the latest data convert code from Meta-Dataset.

[5] The results of Simple CNAPS [3] and Transductive CNAPS [2] are reproduced by the authors and reported at https://github.com/peymanbateni/simple-cnaps. Results of FLUTE [47] and tri-M [30] are from their papers. We reproduce the results of SUR [15] and URT [29] with the updated evaluation protocol for fair comparison.

Test Dataset	ImageNet	Omniglot	Aircraft	Birds	Textures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MSCOCO	MNIST	CIFAR-10	CIFAR-100
10 iterations	55.5 ± 1.1	93.9 ± 0.5	86.4 ± 0.5	78.6 ± 0.7	73.3 ± 0.7	81.9 ± 0.6	63.1 ± 0.9	90.3 ± 0.5	77.6 ± 1.0	50.6 ± 1.1	96.9 ± 0.3	77.0 ± 0.8	62.6 ± 1.1
20 iterations	56.2 ± 1.1	94.7 ± 0.4	86.3 ± 0.5	78.3 ± 0.8	73.9 ± 0.7	81.6 ± 0.6	63.4 ± 0.9	90.1 ± 0.6	79.4 ± 1.0	52.8 ± 1.1	97.2 ± 0.3	78.6 ± 0.8	65.9 ± 1.1
40 iterations	55.6 ± 1.0	94.3 ± 0.4	86.7 ± 0.5	79.4 ± 0.8	73.2 ± 0.8	81.7 ± 0.6	64.0 ± 0.9	90.9 ± 0.5	81.1 ± 0.9	51.4 ± 1.1	96.9 ± 0.3	78.5 ± 0.8	64.3 ± 1.1
60 iterations	55.9 ± 1.1	95.1 ± 0.4	85.9 ± 0.6	77.5 ± 0.8	74.7 ± 0.7	80.9 ± 0.6	62.1 ± 0.9	90.7 ± 0.6	82.2 ± 0.9	52.2 ± 1.1	97.0 ± 0.4	78.4 ± 0.8	64.4 ± 1.1

Table 10. Sensitivity of performance to number of iterations based on MDL model.

Test Dataset	ImageNet	Omniglot	Aircraft	Birds	Textures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MSCOCO	MNIST	CIFAR-10	CIFAR-100
10 iterations	58.4 ± 1.1	94.8 ± 0.4	89.9 ± 0.4	81.3 ± 0.7	76.6 ± 0.7	81.8 ± 0.6	68.4 ± 0.9	92.5 ± 0.5	76.5 ± 1.1	55.6 ± 1.1	96.4 ± 0.4	79.0 ± 0.7	66.9 ± 1.0
20 iterations	58.2 ± 1.1	94.8 ± 0.4	89.9 ± 0.4	81.1 ± 0.7	77.5 ± 0.8	81.9 ± 0.6	68.0 ± 0.9	92.4 ± 0.5	81.8 ± 1.0	57.8 ± 1.1	96.7 ± 0.4	81.7 ± 0.8	69.1 ± 0.9
40 iterations	59.5 ± 1.0	94.9 ± 0.4	89.9 ± 0.4	81.1 ± 0.8	77.5 ± 0.7	81.7 ± 0.6	66.3 ± 0.9	92.2 ± 0.5	82.8 ± 1.0	57.6 ± 1.0	96.7 ± 0.4	82.9 ± 0.7	70.4 ± 1.0
60 iterations	58.7 ± 1.1	94.9 ± 0.4	89.5 ± 0.5	80.8 ± 0.7	77.4 ± 0.8	81.8 ± 0.6	66.2 ± 0.9	92.5 ± 0.5	83.7 ± 0.9	56.9 ± 1.0	96.6 ± 0.3	82.0 ± 0.8	72.0 ± 0.9

Table 11. Sensitivity of performance to number of iterations based on URL model.

confidence interval are reported in Tabs. 10 and 11

Influence of α and β. We evaluate different components of our method and report the results in Tab. 12. The results show that both residual adapters α and the linear transformation β help adapt features to unseen classes while residual adapters significantly improve the performance on unseen domains. The best results are achieved by using both α and β.

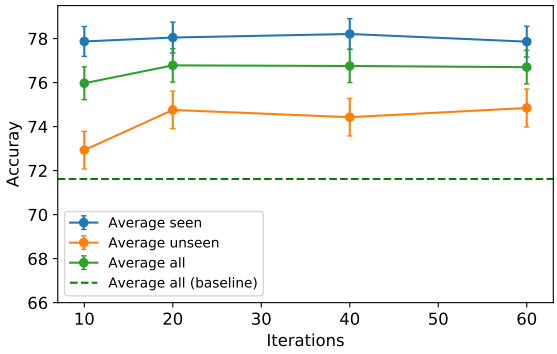

Figure 6. Sensitivity of performance to number of iterations based on MDL model.

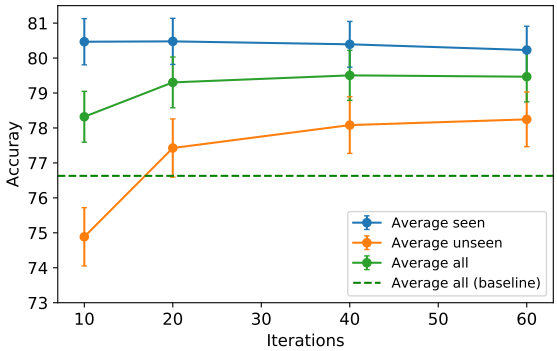

Figure 7. Sensitivity of performance to number of iterations based on URL model.

Initialization analysis for adapters. Here, we investigate using different initialization strategies for adapters: i) Identity initialization: in this work we initialize each residual adapter as an identity matrix scaled by a scalar δ and we set $\delta = 1e-4$; ii) randomly initialization: alternatively, we can randomly initialize each residual adapter. The results of different initialization are summarized in Fig. 8. We can see that our methods with different initialization strategies obtain similar results, which indicates that our method works also with randomly initialization and again verifies the stability of our method. Detailed results of each datasets are shown in Tab. 13.

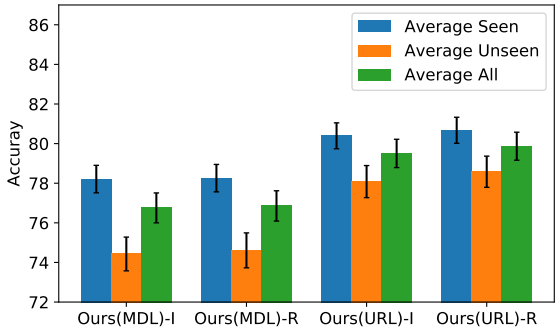

Figure 8. Initialization analysis for adapters. '-I' indicates identity initialization and '-R' is randomly initialization.

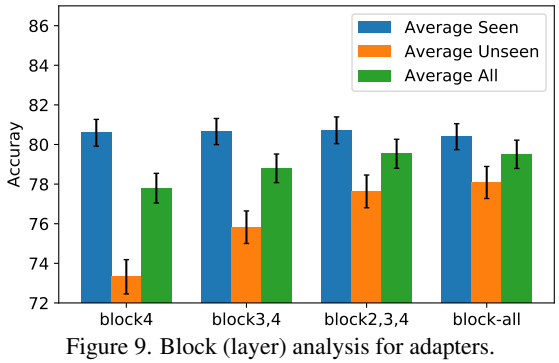

Figure 9. Block (layer) analysis for adapters.

Layer analysis for adapters. Here we investigate whether it is sufficient to attach the adapters only to the later layers. We evaluate this on ResNet18 which is composed of four blocks and attach the adapters to only later blocks (block4,

Test Dataset	ImageNet	Omniglot	Aircraft	Birds	Textures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MSCOCO	MNIST	CIFAR-10	CIFAR-100
Ours w/o α & β	57.0 ± 1.1	94.4 ± 0.4	88.0 ± 0.5	80.3 ± 0.7	74.6 ± 0.7	81.8 ± 0.6	66.2 ± 0.9	91.5 ± 0.5	49.8 ± 1.1	54.1 ± 1.0	91.1 ± 0.4	70.6 ± 0.7	59.1 ± 1.0
Ours w/o β	57.3 ± 1.1	$\mathbf{94.9 \pm 0.4}$	88.9 ± 0.5	81.0 ± 0.7	76.7 ± 0.7	80.6 ± 0.6	65.4 ± 0.9	91.4 ± 0.5	82.6 ± 1.0	55.0 ± 1.1	96.6 ± 0.4	82.1 ± 0.7	66.4 ± 1.1
Ours w/o α	58.8 ± 1.1	94.5 ± 0.4	89.4 ± 0.4	80.7 ± 0.8	77.2 ± 0.7	$\mathbf{82.5 \pm 0.6}$	$\mathbf{68.1 \pm 0.9}$	92.0 ± 0.5	63.3 ± 1.2	57.3 ± 1.0	94.7 ± 0.4	74.2 ± 0.8	63.6 ± 1.0
Ours	$\mathbf{59.5 \pm 1.0}$	$\mathbf{94.9 \pm 0.4}$	$\mathbf{89.9 \pm 0.4}$	$\mathbf{81.1 \pm 0.8}$	$\mathbf{77.5 \pm 0.7}$	81.7 ± 0.6	66.3 ± 0.9	$\mathbf{92.2 \pm 0.5}$	$\mathbf{82.8 \pm 1.0}$	$\mathbf{57.6 \pm 1.0}$	$\mathbf{96.7 \pm 0.4}$	$\mathbf{82.9 \pm 0.7}$	$\mathbf{70.4 \pm 1.0}$

Table 12. Effect of each component. We build our method on the URL model and 'Ours w/o α & β' means we remove both residual adapters α and the pre-classifier adaptation layer β in our method.

Test Dataset	ImageNet	Omniglot	Aircraft	Birds	Textures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MSCOCO	MNIST	CIFAR-10	CIFAR-100
Ours(SDL-ResNet-18)-I	59.5 ± 1.1	78.2 ± 1.2	72.2 ± 1.0	74.9 ± 0.9	77.3 ± 0.7	67.6 ± 0.9	44.7 ± 1.0	90.9 ± 0.6	82.5 ± 0.8	59.0 ± 1.0	93.9 ± 0.6	82.1 ± 0.7	70.7 ± 0.9
Ours(SDL-ResNet-18)-R	58.2 ± 1.0	78.4 ± 1.2	71.1 ± 1.1	74.4 ± 1.0	77.1 ± 0.7	67.2 ± 1.0	45.9 ± 1.0	90.7 ± 0.6	81.9 ± 1.0	57.7 ± 1.1	94.1 ± 0.5	81.9 ± 0.7	70.5 ± 0.9
Ours(MDL)-I	55.6 ± 1.0	94.3 ± 0.4	86.7 ± 0.5	79.4 ± 0.8	73.2 ± 0.8	81.7 ± 0.6	64.0 ± 0.9	90.9 ± 0.5	81.1 ± 0.9	51.4 ± 1.1	96.9 ± 0.3	78.5 ± 0.8	64.3 ± 1.1
Ours(MDL)-R	56.0 ± 1.1	94.1 ± 0.4	87.1 ± 0.5	79.7 ± 0.8	74.0 ± 0.7	82.0 ± 0.6	62.6 ± 0.9	90.6 ± 0.6	80.9 ± 0.9	51.7 ± 1.1	96.9 ± 0.4	77.7 ± 0.9	65.8 ± 1.1
Ours(URL)-I	59.5 ± 1.0	94.9 ± 0.4	89.9 ± 0.4	81.1 ± 0.8	77.5 ± 0.7	81.7 ± 0.6	66.3 ± 0.9	92.2 ± 0.5	82.8 ± 1.0	57.6 ± 1.0	96.7 ± 0.4	82.9 ± 0.7	70.4 ± 1.0
Ours(URL)-R	58.8 ± 1.1	94.9 ± 0.4	90.5 ± 0.4	81.8 ± 0.6	77.7 ± 0.7	82.3 ± 0.6	66.8 ± 0.9	92.6 ± 0.5	83.7 ± 0.8	57.7 ± 1.1	96.9 ± 0.4	82.5 ± 0.7	72.0 ± 0.9

Table 13. Initialization analysis of adapters. 'Ours(URL)-I' indicates our method using URL as the pretrained model and initializing residual adapters as identity matrix (scaled by $\delta = 0.0001$) while 'Ours(URL)-R' means our method initialize residual adapters randomly.

Test Dataset	ImageNet	Omniglot	Aircraft	Birds	Textures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MSCOCO	MNIST	CIFAR-10	CIFAR-100
Ours (block4)	59.0 ± 1.1	95.0 ± 0.4	90.0 ± 0.4	80.6 ± 0.8	77.8 ± 0.7	82.3 ± 0.6	68.2 ± 0.9	91.8 ± 0.6	70.6 ± 1.1	57.1 ± 1.1	95.9 ± 0.4	77.2 ± 0.8	65.9 ± 1.0
Ours (block3,4)	60.4 ± 1.1	94.7 ± 0.4	90.0 ± 0.5	80.4 ± 0.7	77.8 ± 0.7	82.2 ± 0.6	67.2 ± 0.8	92.5 ± 0.5	77.2 ± 1.0	57.9 ± 1.0	96.7 ± 0.3	78.8 ± 0.9	68.6 ± 0.9
Ours (block2,3,4)	59.6 ± 1.1	94.9 ± 0.4	89.9 ± 0.5	81.0 ± 0.8	78.2 ± 0.7	82.4 ± 0.6	67.6 ± 0.9	92.3 ± 0.5	81.5 ± 1.0	57.9 ± 1.0	96.6 ± 0.4	81.5 ± 0.8	70.6 ± 1.0
Ours (block-all)	59.5 ± 1.0	94.9 ± 0.4	89.9 ± 0.4	81.1 ± 0.8	77.5 ± 0.7	81.7 ± 0.6	66.3 ± 0.9	92.2 ± 0.5	82.8 ± 1.0	57.6 ± 1.0	96.7 ± 0.4	82.9 ± 0.7	70.4 ± 1.0

Table 14. Block (layer) analysis for adapters based on URL model.

block3,4, block2,3,4 and block-all. Figure 9 shows that applying our adapters to only the last block (block4) obtains around 78% average accuracy on all domains which outperforms the URL. With attaching residual adapters to more layers, the performance on unseen domains is improved significantly while the one on seen domains remains stable. The mean accuracy with 95% confidence interval for layer analysis are shown in Tab. 14.

Decomposing residual adapters. Here we investigate whether one can reduce the number of parameters in the adapters while retaining its performance by using matrix decomposition. As in deep neural network, the adapters in earlier layers are relatively small, we then decompose the adapters in the last two blocks only where the adapter dimensionality goes up to 512×512. Figure 10 shows that our method can achieve good performance with less parameters by decomposing large residual adapters, (*e.g.* when $N = 32$ where the number of additional parameters equal to around 4% vs 13%, the performance is still comparable to the original form of residual adapters, *i.e.* N=0). Results of each datasets in Tab. 15, also show that, by decomposing large residual adapters, the performance of our method is still comparable to the original form of residual adapters (*i.e.* Ours) with less parameters.

The similar conclusion can be drawn from results (shown in Fig. 11) of our method using decomposed residual adapters in all layers. When N increases, *i.e.*, smaller residual adapters, the average accuracy on all domains is still comparable to the original form of residual adapters (*i.e.* N=0) with less parameters though the average accuracy on unseen domains drops slightly. From the results depicted in Tab. 16, we can see that when N increases, the performance

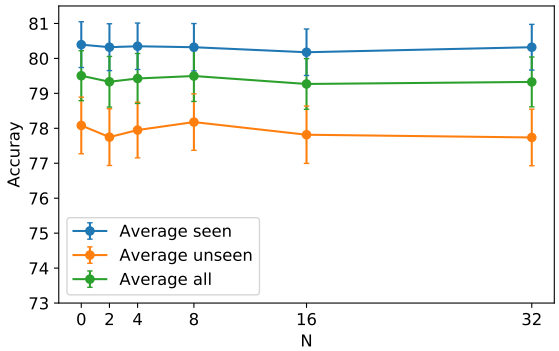

Figure 10. Decomposed residual adapters on block-3,4.

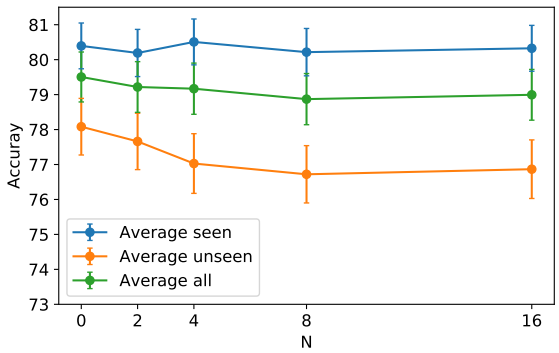

Figure 11. Decomposed residual adapters on all layers.

of most domains are still comparable to the original form of residual adapters (*i.e.* Ours) while the performance on Traffic Sign drops slightly as the adapters in earlier layers are small and when N is larger the decomposed residual adapters might be too small to tranform the features. In overall, our

Test Dataset	ImageNet	Omniglot	Aircraft	Birds	Textures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MSCOCO	MNIST	CIFAR-10	CIFAR-100
Ours	59.5 ± 1.0	94.9 ± 0.4	89.9 ± 0.4	81.1 ± 0.8	77.5 ± 0.7	81.7 ± 0.6	66.3 ± 0.9	92.2 ± 0.5	82.8 ± 1.0	57.6 ± 1.0	96.7 ± 0.4	82.9 ± 0.7	70.4 ± 1.0
Ours(N=2)	58.9 ± 1.1	95.2 ± 0.4	89.7 ± 0.5	80.9 ± 0.7	76.7 ± 0.7	81.4 ± 0.6	67.7 ± 0.9	92.2 ± 0.5	82.4 ± 1.0	57.1 ± 1.0	96.5 ± 0.4	82.4 ± 0.7	70.3 ± 1.0
Ours(N=4)	58.7 ± 1.1	94.9 ± 0.4	89.7 ± 0.5	80.3 ± 0.7	77.0 ± 0.7	82.5 ± 0.6	67.2 ± 0.9	92.5 ± 0.5	82.6 ± 1.0	57.5 ± 1.1	96.5 ± 0.4	82.5 ± 0.7	70.8 ± 0.9
Ours(N=8)	59.1 ± 1.1	95.0 ± 0.4	89.8 ± 0.5	80.2 ± 0.8	77.2 ± 0.7	82.1 ± 0.6	67.0 ± 0.9	92.2 ± 0.5	82.5 ± 1.0	57.2 ± 1.1	96.8 ± 0.4	82.6 ± 0.7	71.8 ± 0.9
Ours(N=16)	58.2 ± 1.1	94.7 ± 0.4	90.1 ± 0.4	80.3 ± 0.8	76.9 ± 0.7	81.7 ± 0.6	67.6 ± 0.9	92.0 ± 0.5	81.8 ± 1.0	58.1 ± 1.1	96.4 ± 0.4	81.8 ± 0.7	71.1 ± 0.9
Ours(N=32)	59.2 ± 1.1	94.8 ± 0.4	89.6 ± 0.5	80.0 ± 0.8	77.3 ± 0.6	82.4 ± 0.6	67.2 ± 0.9	92.1 ± 0.5	82.1 ± 1.0	57.1 ± 1.0	96.7 ± 0.3	81.6 ± 0.8	71.1 ± 0.9

Table 15. Results of using decomposed RA on layer3,4.

Test Dataset	ImageNet	Omniglot	Aircraft	Birds	Textures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MSCOCO	MNIST	CIFAR-10	CIFAR-100
Ours	59.5 ± 1.0	94.9 ± 0.4	89.9 ± 0.4	81.1 ± 0.8	77.5 ± 0.7	81.7 ± 0.6	66.3 ± 0.9	92.2 ± 0.5	82.8 ± 1.0	57.6 ± 1.0	96.7 ± 0.4	82.9 ± 0.7	70.4 ± 1.0
Ours(N=2)	58.1 ± 1.1	94.8 ± 0.4	89.7 ± 0.5	80.2 ± 0.8	76.9 ± 0.7	82.1 ± 0.6	67.8 ± 0.9	92.0 ± 0.6	82.5 ± 0.9	56.9 ± 1.1	96.7 ± 0.3	82.0 ± 0.8	70.3 ± 1.0
Ours(N=4)	59.6 ± 1.1	94.8 ± 0.4	89.9 ± 0.5	80.3 ± 0.8	77.4 ± 0.7	82.6 ± 0.6	66.6 ± 0.9	92.9 ± 0.5	79.7 ± 1.1	57.6 ± 1.1	96.5 ± 0.4	80.9 ± 0.8	70.6 ± 1.0
Ours(N=8)	58.2 ± 1.1	94.6 ± 0.4	89.6 ± 0.5	81.2 ± 0.8	76.6 ± 0.7	82.7 ± 0.6	66.5 ± 0.9	92.3 ± 0.5	78.1 ± 1.1	57.3 ± 1.0	96.3 ± 0.3	81.0 ± 0.8	70.9 ± 0.9
Ours(N=16)	58.9 ± 1.1	94.6 ± 0.4	89.7 ± 0.5	80.1 ± 0.7	77.0 ± 0.7	82.1 ± 0.6	68.4 ± 0.9	91.9 ± 0.5	78.3 ± 1.0	57.8 ± 1.1	96.0 ± 0.4	82.0 ± 0.7	70.3 ± 1.0

Table 16. Results of using decomposed RA on all layers.

method can achieve good performance with less parameters by decomposing large residual adapters.

Training time. The training time (meta-train) of our method is equal to the one of URL (hence no additional cost), *i.e*. 48 hours in multi-domain setting, 6 hours for Resnet-18 and 33 hours for Resnet-34 in single-domain learning in one Nvidia V100 GPU. Whereas CTX meta-training requires 8 Nvidia V100 GPUs for 7 days and approximately 40 times more expensive than ours. During the meta-test stage, the model parameters are further trained using support set of each episode. Meta-test training cost is depicted in Tab. 12 for Meta-Dataset tasks. URL baseline only finetunes parameters of PA β. Finetune+NCC updates the entire backbone parameters. Ours learn RA and PA parameters. While URL is the fastest baseline, as it does not require backpropagating the error to early layers, ours is more efficient than finetuning all the backbone parameters.

Test Dataset	Image -Net	Omni -glot	Air- craft	Birds	Tex- tures	Quick Draw	Fungi	VGG Flower	Traffic Sign	MS- COCO	MNIST	CIFAR -10	CIFAR -100
URL	0.7	0.7	0.4	0.7	0.4	1.0	1.0	0.5	0.9	0.9	0.4	0.4	1.0
Finetune+NCC	7.7	2.5	7.4	7.0	5.8	9.3	8.7	6.6	9.1	9.0	6.5	6.7	9.3
Ours (URL+RA+PA)	7.2	2.4	6.1	6.8	4.8	8.9	7.4	5.2	8.8	8.3	6.0	6.2	8.6

Table 12. Computation cost (# second per task) during meta-test.

C.6. Qualitative results

We qualitatively analyze our method and compare it to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in Figs. 12 to 24 by illustrating the nearest neighbors in all test datasets given a query image as in [27]. It is clear that our method produces more correct neighbors than other methods. While other methods retrieve images with more similar colors, shapes and backgrounds, *e.g*. in Figs. 20, 21, 23 and 24, our method is able to retrieve semantically similar images. More specifically, as shown in Fig. 15, our method correctly produces neighbors of the bird in the query image while other methods pick images with similar appearances or similar background, *e.g*. images with twigs. In Fig. 20, other methods mainly retrieve the triangle sign while our method is able to retrieve the correct sign with illumination distortion. In Fig. 24, other methods including SUR, URT are distracted by the blue background but our method select the correct shark images. It again suggests that our method is able to quickly adapt the features for unseen few-shot tasks.

Figure 12. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in ImageNet. Green and red colors indicate correct and false predictions respectively.

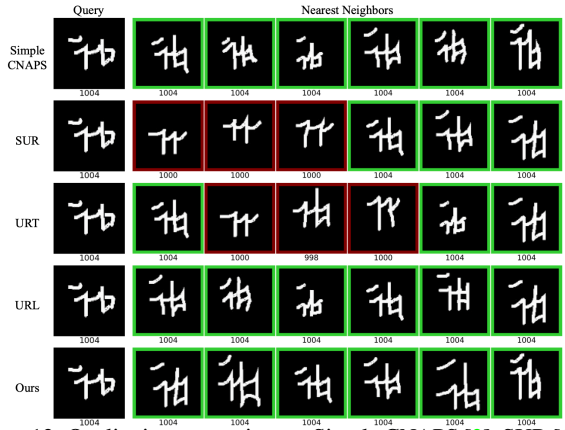

Figure 13. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in Omniglot. Green and red colors indicate correct and false predictions respectively.

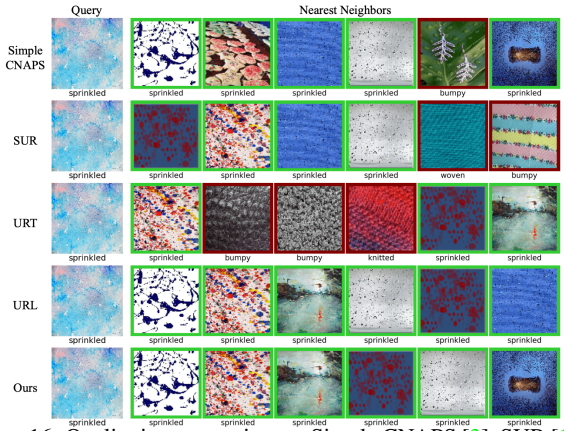

Figure 16. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in Textures. Green and red colors indicate correct and false predictions respectively.

Figure 14. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in Aircraft. Green and red colors indicate correct and false predictions respectively.

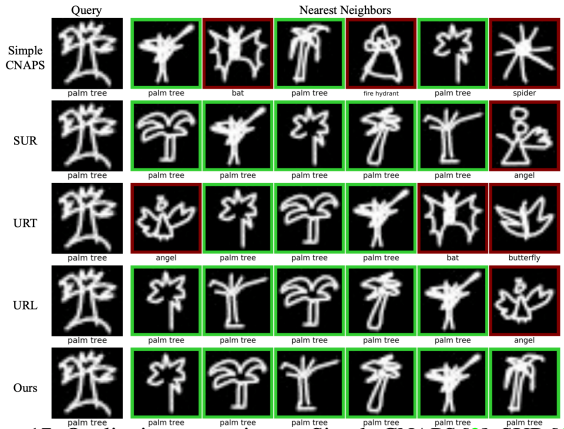

Figure 17. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in Quick Draw. Green and red colors indicate correct and false predictions respectively.

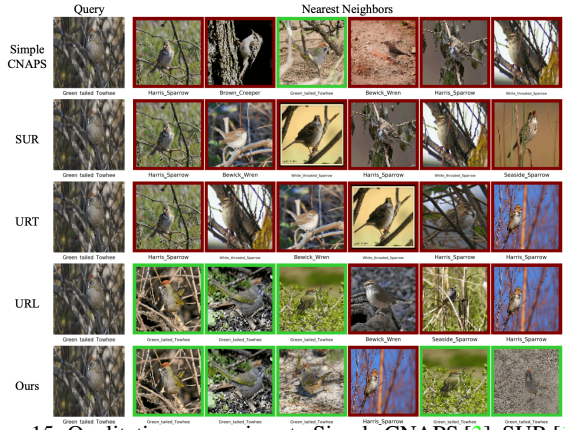

Figure 15. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in Birds. Green and red colors indicate correct and false predictions respectively.

Figure 18. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in Fungi. Green and red colors indicate correct and false predictions respectively.

Figure 19. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in VGG Flower. Green and red colors indicate correct and false predictions respectively.

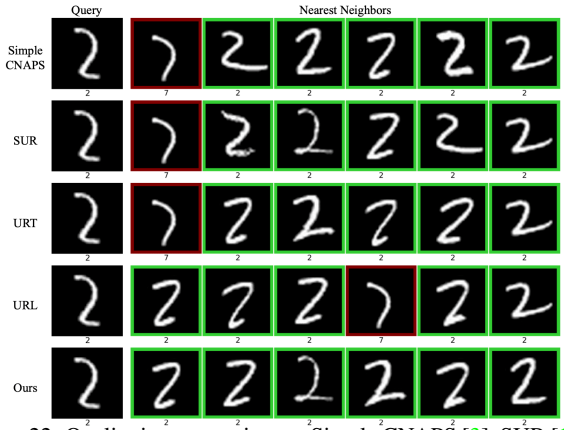

Figure 22. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in MNIST. Green and red colors indicate correct and false predictions respectively.

Figure 20. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in Traffic Sign. Green and red colors indicate correct and false predictions respectively.

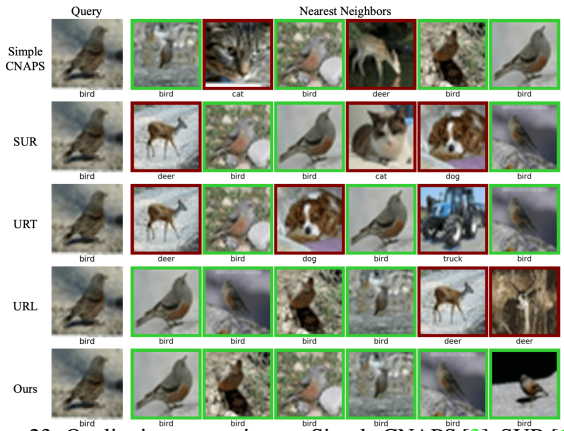

Figure 23. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in CIFAR-10. Green and red colors indicate correct and false predictions respectively.

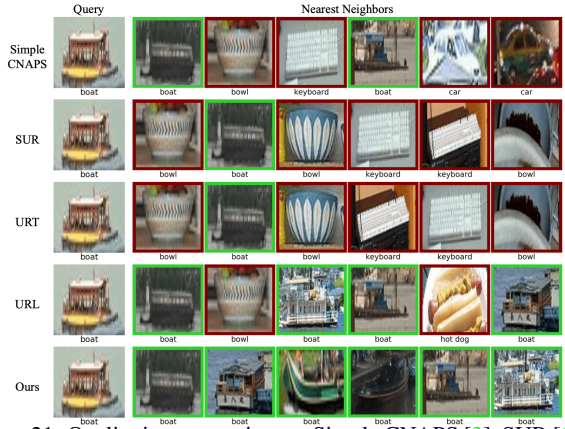

Figure 21. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in MSCOCO. Green and red colors indicate correct and false predictions respectively.

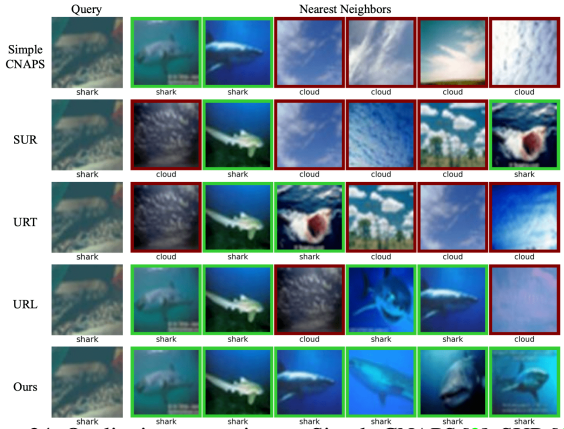

Figure 24. Qualitative comparison to Simple CNAPS [3], SUR [15], URT [29], and URL [27] in CIFAR-100. Green and red colors indicate correct and false predictions respectively.