LibFewShot: A Comprehensive Library for Few-shot Learning

Wenbin Li, Ziyi Wang, Xuesong Yang, Chuanqi Dong, Pinzhuo Tian, Tiexin Qin, Jing Huo, Yinghuan Shi, Lei Wang, *Senior Member, IEEE,* Yang Gao, and Jiebo Luo, *Fellow, IEEE*

Abstract—Few-shot learning, especially few-shot image classification, has received increasing attention and witnessed significant advances in recent years. Some recent studies implicitly show that many generic techniques or "tricks", such as data augmentation, pre-training, knowledge distillation, and self-supervision, may greatly boost the performance of a few-shot learning method. Moreover, different works may employ different software platforms, backbone architectures and input image sizes, making fair comparisons difficult and practitioners struggle with reproducibility. To address these situations, we propose a comprehensive library for few-shot learning (LibFewShot) by re-implementing eighteen state-of-the-art few-shot learning methods in a unified framework with the same single codebase in PyTorch. Furthermore, based on LibFewShot, we provide comprehensive evaluations on multiple benchmarks with various backbone architectures to evaluate common pitfalls and effects of different training tricks. In addition, with respect to the recent doubts on the necessity of meta- or episodic-training mechanism, our evaluation results confirm that such a mechanism is still necessary especially when combined with pre-training. We hope our work can not only lower the barriers for beginners to enter the area of few-shot learning but also elucidate the effects of nontrivial tricks to facilitate intrinsic research on few-shot learning. The source code is available from https://github.com/RL-VIG/LibFewShot.

Index Terms—Unified framework, Few-shot learning, Image classification, Fair comparison.

1 Introduction

Few-shot learning (FSL), especially few-shot image classification, has received considerable attention in recent years [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. It tries to learn an effective classification model from a few labeled training examples. A wide variety of advanced FSL methods has been proposed and significantly improved the classification performance on multiple benchmark datasets [11], [12], [13], [14], [15], [16], [17], [18].

Because of the extreme scarcity of training examples per class, it is almost impossible to only use the few available examples to learn an effective classifier to solve the few-shot classification problem. Therefore, the current FSL methods generally follow a paradigm of transfer learning, *i.e.*, using a large labeled but class-disjoint auxiliary set to learn transferable knowledge (or representations) to boost the target few-shot task. More importantly, different from standard transfer learning [19], the existing FSL methods normally adopt a meta-training [20] or episodic-training mechanism [2] to train a few-shot model by constructing massive few-shot tasks (episodes) from the auxiliary set to simulate the target few-shot task. Like the target few-shot task, each simulated task also consists of a labeled support (training) set and an unlabeled query (test) set.

In general, these typical FSL methods can be roughly divided into two types, *i.e.*, meta-learning based [20], [21] and metric-learning based [22], [23]. The former normally adopts a meta-learning or learning-to-learn paradigm [24], [25] to learn some kind of cross-task knowledge through an alternate optimization between the meta-learner and base-learner. In this way, it is able to make the model quickly generalize to new unseen tasks with a few training examples (*i.e.*, test time fine-tuning). In contrast, the latter employs a learning-to-compare paradigm [2] to learn representations that can be transferred between tasks without test time fine-tuning (test-tuning for short). This is implemented by directly comparing the relations between query images and support images in each training task through an episodic training mechanism [2]. Both types of methods have greatly advanced the development of few-shot learning.

However, the question *"Is meta- or episodic-training paradigm really crucial and optimal for the FSL problem?"* has been raised recently by the community. Some recent works [26], [27], [28], [29], [30] have attempted to answer this question. For example, [27] finds that a simple baseline method, *i.e.*, pre-training + test-tuning (called non-episodic based methods), can obtain competitive results when compared with the meta- or episodic-training paradigm based methods. Similarly, [30] proposes an improved baseline method by using logistic regression as the linear classifier in the test-tuning phase, which surprisingly achieves the state of the art. Therefore, *can we conclude that the meta- or episodic-training paradigm is indeed NOT necessary for FSL?*

In addition, we observe that the implementation and evaluation details of different FSL methods vary signifi-

- *Wenbin Li, Ziyi Wang, Xuesong Yang, Chuanqi Dong, Jing Huo, Yinghuan Shi and Yang Gao are with the State Key Laboratory for Novel Software Technology, Nanjing University, China, 210023 (e-mail: {liwenbin, huojing, syh, gaoy}@nju.edu.cn; {wangziyi, yangxuesong, dongchuanqi}@smail.nju.edu.cn).*
- *Pinzhuo Tian is with the School of Computer Engineering and Science, Shanghai University, China (e-mail: pinzhuo@shu.edu.cn).*
- *Tiexin Qin is with the Department of Electrical Engineering, City University of Hong Kong, China (e-mail: tiexinqin@gmail.com).*
- *Lei Wang is with the School of Computing and Information Technology, University of Wollongong, Australia (e-mail: leiw@uow.edu.au).*
- *Jiebo Luo is with the Department of Computer Science, University of Rochester, America (e-mail: jluo@cs.rochester.edu).*

cantly. This could make fair comparison difficult, or even worse, make some conclusions questionable. Specifically, certain key discrepancies of existing FSL methods can be summarized as follows: (1) different software platforms (*e.g.,* TensorFlow vs. PyTorch); (2) different backbones (*e.g.,* Wider ResNet12 vs. ResNet12); (3) classifiers with different parameters (*e.g.,* heavy-parametric classifier vs. non-parametric classifier); (4) different input image sizes (*e.g.,* 224×224 vs. 84×84); (5) different test time evaluations (*e.g.,* center crop evaluation vs. raw evaluation). Clearly, such wide discrepancies are not amenable to a fair comparison and therefore cannot truthfully reflect the actual progress of FSL. Moreover, some FSL methods may employ additional fancy deep learning tricks, such as stronger data augmentation, knowledge distillation, self-supervision, label smoothing, and DropBlock, in the training process. The effects of different deep learning tricks are worth studying in FSL. Also, the additional used tricks are another key type of discrepancy.

Our work. Therefore, to facilitate fair comparison and conveniently investigate the common issues in FSL, we develop *a comprehensive library for few-shot learning (LibFewShot)* by making most of the implementation details of different FSL methods consistent. To be specific, eighteen representative state-of-the-art FSL methods, including seven meta-learning based, six metric-learning based and five non-episodic based methods, are systematically re-implemented in a unified framework with the same single codebase in PyTorch. In LibFewShot, we try our best to ensure that all the methods use the same settings and the same bag of tricks, except for some specific tricks or specific neural architectures that are the main contributions of certain methods. In this way, we could take a true picture of the actual state-of-the-art results on FSL. More importantly, we are able to construct a large-scale study to analyze the impact of different tricks, such as pre-training, global classification, knowledge distillation and label smoothing, in a fair way.

Our contributions. The main contributions of this work are as follows:

- We develop a unified framework LibFewShot with eighteen re-implemented FSL methods for the first time in the literature. It can be used as a toolbox and a platform to help practitioners efficiently use and reproduce FSL methods.
- We provide comprehensive evaluations of the eighteen methods on multiple benchmark datasets with various embedding backbones, by controlling the implementation details. This can reveal the actual progress of FSL and can be conveniently referenced to perform comparative experiments.
- We conduct a large-scale study on multiple representative FSL methods in a fair way, revealing that (1) pre-training indeed can learn a good initial representation but is not necessarily an optimal representation; (2) meta- or episodic-training can further improve this initial representation; (3) ℓ_2 normalization of image feature vectors can significantly boost the final classification more than test-tuning in the test phase, especially in data-limited scenarios.
- We conduct comprehensive ablation studies for multiple deep learning tricks on the FSL problem, showing that many tricks could achieve significant algorithm-agnostic performance improvements and are universally applicable to different FSL methods.
- We have released LibFewShot as an open-source project on GitHub, and will continue to add new methods into this project. We welcome other researchers to contribute to this library to facilitate the community to conduct research on this important topic together.

2 Overview of Few-shot Learning Methods

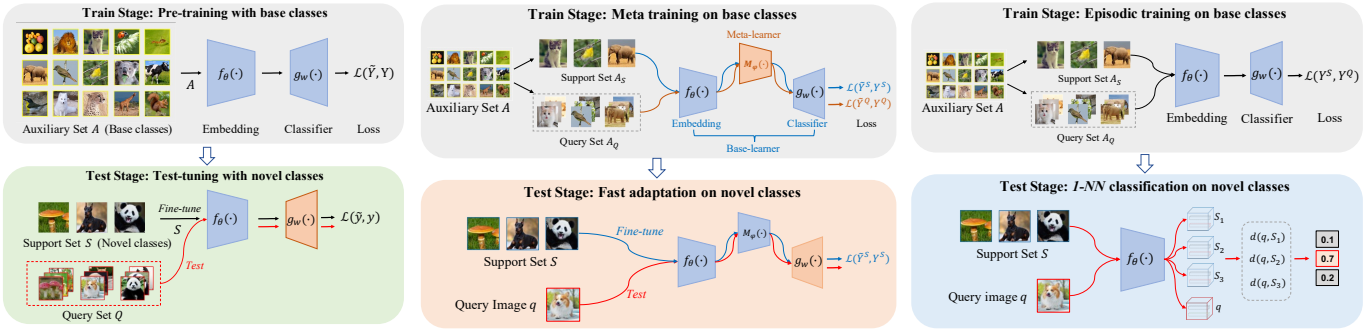
In this section, we will first introduce the problem formulation of FSL, and then review three kinds of FSL methods, *i.e.,* non-episodic based methods, meta-learning based methods and metric-learning based methods, where multiple representative methods are further reviewed in detail.

2.1 Problem Formulation

In few-shot setting, there are usually three sets of data, including a target labeled support set $\mathcal{S}$, a target unlabeled query set $\mathcal{Q}$ and a class-disjoint auxiliary set $\mathcal{A}$. In particular, $\mathcal{S}$ and $\mathcal{Q}$ share the same label space, which corresponds to the training and test sets in generic classification, respectively. The concept of "*few-shot*" in fact comes from $\mathcal{S}$, where there are $\mathcal{C}$ classes but each class only has $\mathcal{K}$ (*e.g.,* 1 or 5) labeled samples. We call this kind of classification task a $\mathcal{C}$-way $\mathcal{K}$-shot task. Clearly, such a few labeled samples in each class make it almost impossible to train an effective classification model, no matter using deep neural networks or traditional machine learning algorithms. Therefore, one solution of FSL becomes how to use $\mathcal{A}$ to boost the learning on the target task (*i.e.,* $\mathcal{S}$ and $\mathcal{Q}$). The good point is that $\mathcal{A}$ generally enjoys more classes and samples per class than $\mathcal{S}$, while the challenge is that $\mathcal{A}$ has a disjoint label space from $\mathcal{S}$ and even may have a large domain shift from $\mathcal{S}$.

Therefore, the current FSL methods mainly focus on how to effectively learn transferable knowledge from $\mathcal{A}$ for fast adaptation (*e.g.,* meta-learning based and non-episodic based methods) or for good generalization (*e.g.,* metric-learning based methods) on $\mathcal{S}$ with a few labeled support examples. Note that, in experiment at study, given a dataset $\mathcal{D}$, it will be divided into $\mathcal{D}_{train}$, $\mathcal{D}_{val}$ and $\mathcal{D}_{test}$ for training, validation and test, respectively. Typically, $\mathcal{D}_{train}$ will be taken as the auxiliary set $\mathcal{A}$, and multiple evaluation few-shot tasks $\mathcal{T} = \langle \mathcal{S}, \mathcal{Q} \rangle$ will be formed by randomly sampling from $\mathcal{D}_{val}$ and $\mathcal{D}_{test}$, respectively.

Notation. Following the literature, the auxiliary set, *i.e.,* the set of base classes, is denoted as $\mathcal{A} = \{X_i, y_i\}_{i=1}^{N}$, with the image $X_i \in \mathbb{R}^{H \times W \times 3}$ and the one-hot labeling vector $y_i \in Y = \{0, 1\}^{\mathcal{C}_{base}}$. For a $\mathcal{C}$-way $\mathcal{K}$-shot task with $\mathcal{C}$ novel classes, the support set and query set are represented as $\mathcal{S} = \{S_1, \cdots, S_{\mathcal{C}}\} = \{X_i, y_i\}_{i=1}^{\mathcal{CK}}$ and $\mathcal{Q} = \{Q_i, y_i\}_{i=1}^{\mathcal{CM}}$, respectively, where $S_c = \{X_i, y_c\}_{i=1}^{\mathcal{K}}$ contains $\mathcal{K}$ images and is the c-th class in $\mathcal{S}$. Let $f_\theta(\cdot)$ and $g_\omega(\cdot)$ denote the convolutional neural network based embedding backbone and classifier, respectively. Also, $g_\omega(\cdot)$ can be integrated with $f_\theta(\cdot)$ into a same network and trained in an end-to-end manner. For generic classification and the base



(a) Non-episodic based methods (b) Meta-learning based methods (c) Metric-learning based methods

Fig. 1. Illustrations of *Non-episodic based*, *Meta-learning based* and *Metric-learning based methods*, respectively. The first one uses a generic classification task as the proxy task, while the latter two use a meta- or episodic-training paradigm in the training stage. At the test stage, the first two will test-tune a new task-relevant classifier, while the last one simply uses a 1-NN classifier without any test-tuning.

learner in meta-learning, the cost function is represented as $\mathcal{L}\big(g_\omega(f_\theta(X)), y\big)$. As for the metric-learning based FSL, the cost function is represented as $\mathcal{L}\big(g_\omega(f_\theta(X)|\mathcal{S}), y\big)$.

Episodic-training and Meta-training. To learn an effective FSL model, meta-training [20] or episodic-training mechanism [2] is normally adopted at the training stage. Both meta- and episodic-training rely on a lot of simulation few-shot tasks, which are randomly constructed from the auxiliary set $\mathcal{A}$. Each simulated task $\mathcal{T}$ consists of two subsets, $\mathcal{A}_\mathcal{S}$ and $\mathcal{A}_\mathcal{Q}$, which are akin to $\mathcal{S}$ and $\mathcal{Q}$, respectively. Note that because the labels in each simulated task are randomly assigned according to the original real labels in the auxiliary set, we call such a kind of label as *local-label*. In contrast, if the real labels of the auxiliary set are directly used, we mean that *global-label* is used.

At each iteration, one simulated task (episode), *i.e.*, $\mathcal{T} = \langle \mathcal{A}_\mathcal{S}, \mathcal{A}_\mathcal{Q} \rangle$, is adopted to train the current model. Conceptually, tens of thousands of tasks, *i.e.*, $\{\mathcal{T}^i = \langle \mathcal{A}^i_\mathcal{S}, \mathcal{A}^i_\mathcal{Q} \rangle\}_{i=1}^{\mathcal{M}} \in \rho(\mathcal{T})$, will be randomly sampled from a task distribution $\rho(\mathcal{T})$ to train this model. The core principle is that the training condition (*i.e.*, the training task) must match the test condition (*i.e.*, the target task) [2]. From the perspective of meta-learning, as more tasks are observed, the model can use the accumulated meta-knowledge to adapt its own bias according to the characteristics of each task [25], [31].

2.2 Non-episodic based Methods

As illustrated in Figure 1(a), the *non-episodic based methods* [9], [27], [28], [30], [32], [33] generally follow the standard transfer learning procedure [19], consisting of two phases, *i.e.*, pre-training with base classes and test-tuning with novel classes.

Pre-training with base classes. In this phase, the whole auxiliary set $\mathcal{A}$ is used to train a $\mathcal{C}_{base}$-class classifier by using the standard cross-entropy loss as below,

$$\Gamma = \arg\min_{\theta,\omega} \sum_{i=1}^{N} \mathcal{L}^{\mathrm{CE}}\Big(g_\omega(f_\theta(X_i)), y\Big), \tag{1}$$

where $\mathcal{L}^{\mathrm{CE}}$ is the cross-entropy loss function.

Test-tuning with novel classes. Test-tuning is performed in the test phase. Specifically, for each specific novel task $\mathcal{T} = \langle \mathcal{S}, \mathcal{Q} \rangle$, a new $\mathcal{C}$-class classifier will be re-learned based on $\mathcal{S}$ every time. Basically, the pre-trained embedding parameter θ is fixed to avoid over-fitting, because there is limited labeled data in $\mathcal{S}$. Once the novel classifier is learned, the labels of $\mathcal{Q}$ can be predicted.

Representative methods include *Baseline* [27], *Baseline++* [27], RFS-simple [27], SKD-GEN0 [32], S2M2 [33] and Neg-Cosine [34], *etc.* The main difference between the first three methods is that they use different classifiers at the test-tuning stage: (1) Baseline [27] adopts a linear layer, *i.e.*, a fully-connected (FC) layer, as the new classifier; (2) Baseline++ [27] replaces the standard inner product (in the FC layer) with a cosine distance between the input feature and weight vector; (3) RFS-simple [27] employs logistic regression instead of the FC layer as the new classifier by first using ℓ_2 normalization for the feature vector.

SKD-GEN0 [32] also uses logistic regression as the classifier as RFS-simple [27], where the only difference is that additional rotation-based self-supervision [35] is further introduced into the pre-training stage. In addition, both [27] and [32] develop an extended version, respectively, by using knowledge distillation [36]. As for S2M2, more auxiliary tasks, such as Manifold Mixup [37], rotation [35] and Exemplar [38], are introduced into the pre-training stage to learn more powerful representations. Neg-Cosine [34] introduces a negative margin loss, *i.e.*, a negative-margin cosine softmax loss, at the pre-training stage, and shows that this will benefit the novel classes in the test-tuning phase.

Discussions. The above non-episodic based methods have achieved surprisingly good results with a much simpler methodology, shaking the foundation of the current pure meta- or episodic-training based methods. The question of "*Should we discard meta- or episodic-training in FSL?*" will be interesting to investigate. On the other hand, intuitively, the cross-entropy loss used in the pre-training stage may make the learned representations overfit the seen base classes, thus lacking generalization ability for unseen classes. Moreover, the non-episodic methods strictly follow the paradigm of standard transfer learning and heavily focus on improving the pre-training stage by utilizing the latest and popular deep learning tricks, which may somewhat overlook the intrinsic problems of FSL.

2.3 Meta-learning based Methods

As illustrated in Figure 1(b), *meta-learning based methods* [3], [12], [26], [39], [40], [41], [42] normally perform a meta-training paradigm on a family of few-shot tasks constructed from the base classes at the training stage, aiming to make the learned model able to quickly adapt to unseen novel tasks at the test stage. In particular, the meta-training procedure consists of a two-step optimization between the base-learner and meta-learner. Specifically, given a sampled task $\mathcal{T} = \langle \mathcal{A}_\mathcal{S}, \mathcal{A}_\mathcal{Q} \rangle$, Step-1 (*i.e.,* base-learning or inner loop) is to use $\mathcal{A}_\mathcal{S}$ (*i.e.,* training examples in each task) to learn the base-learner. Next, in Step-2 (*i.e.,* meta-tuning or outer loop), $\mathcal{A}_\mathcal{Q}$ (*i.e.,* test samples in each task) is employed to optimize the meta-learner. In this way, the meta-learner is expected to learn a kind of across-task meta-knowledge, which can be used for the fast adaptation on novel tasks.

Model-Agnostic Meta-Learning (MAML) is one representative method [3], whose core idea is to train a model's initial parameters by involving the second-order gradients, making this model able to rapidly adapt to a new task just with one or a few gradient steps. Specifically, in the base-learning phase (inner loop), given $\mathcal{T} = \langle \mathcal{A}_\mathcal{S}, \mathcal{A}_\mathcal{Q} \rangle$, the current model $F_\Theta = f_\theta \circ g_\omega$ and $\Theta = \Theta_0$, we can obtain the m-th inner loop gradient update as,

$$\Theta_m = \Theta_{m-1} - \alpha \nabla_{\Theta_{m-1}} \mathcal{L}_{\mathcal{A}_\mathcal{S}}(F_{\Theta_{m-1}}), \tag{2}$$

where $\Theta = \{\theta, \omega\}$, α is a step size hyper-parameter, and m is the total number of inner iterations. Next, in the meta-tuning phase (outer loop), the parameter of the model is truly updated over the previous parameter Θ rather than Θ_m by using the query set $\mathcal{A}_\mathcal{Q}$, *i.e.,*

$$\Theta = \Theta - \beta \nabla_\Theta \mathcal{L}_{\mathcal{A}_\mathcal{Q}}(F_{\Theta_m}), \tag{3}$$

where β is a meta step size hyper-parameter.

Ridge Regression Differentiable Discriminator (R2D2) is designed from another perspective [26], by adopting a standard machine learning algorithm such as ridge regression as the base-learner classifier $g_\omega(\cdot)$ in the inner loop. Note that the base-learner classifier in MAML is a standard FC layer. The advantage of R2D2 is that ridge regression enjoys a closed-form solution, which can make the base-learning phase more efficient. Specifically, the ridge regression with parameter matrix $\boldsymbol{W} \in \mathbb{R}^{d \times c}$ is formulated as,

$$\begin{aligned} \Gamma &= \arg\min_{\boldsymbol{W}} \|\boldsymbol{X}\boldsymbol{W} - \boldsymbol{Y}\|^2 + \lambda \|\boldsymbol{W}\|^2 \\ &= (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}, \end{aligned} \tag{4}$$

where $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{Y} \in \mathbb{R}^{n \times c}$ denote n input samples with d-dimensional features and the corresponding labels (*i.e.,* c classes), respectively, $\boldsymbol{I}$ is the identity matrix, and λ is a regularization hyper-parameter.

Specifically, suppose there are n support images in $\mathcal{A}_\mathcal{S}$, and $f_\theta(\cdot)$ can be used to obtain the feature embeddings, *i.e.,* $\boldsymbol{X} = f_\theta(\mathcal{A}_\mathcal{S}) \in \mathbb{R}^{n \times d}$. In the base-learning phase, the optimal parameter matrix $\boldsymbol{W}^*$ can be easily obtained according to Eq.(4). Next, in the meta-tuning phase, the predictions of $\boldsymbol{X}_\mathcal{Q} = f_\theta(\mathcal{A}_\mathcal{Q}) \in \mathbb{R}^{n \times d}$ can be achieved as,

$$\widehat{\boldsymbol{Y}} = \alpha \boldsymbol{X}_\mathcal{Q} \boldsymbol{W}^* + \beta, \tag{5}$$

where α and β are scale and bias, respectively, which can be learned by optimizing the meta-loss $\mathcal{L}(\widehat{\boldsymbol{Y}}, \boldsymbol{Y}_\mathcal{Q})$.

Other Representative Methods include *Latent Embedding Optimization (LEO)* [40], *Almost No Inner Loop (ANIL)* [41], *Body Only update in Inner Loop (BOIL)* [43], *MetaOptNet* [12] and *Versa* [39]. Specifically, LEO, ANIL and BOIL all follow the same optimization procedure as MAML. The core idea of LEO is to optimize the meta-learning process within a low-dimensional latent space, and to learn a generative distribution of model parameters, instead of directly learning the explicit high-dimensional model parameters in MAML. ANIL tries to remove the inner loop updates for the embedding backbone (*i.e.,* f_θ), but only applies the inner loop adaptation to the classifier (*i.e.,* g_ω), which means that only ω is updated in Eq.(2). In contrast with ANIL, BOIL updates only the embedding backbone f_θ but freezes the update of the classifier g_ω in the inner loop. Similar to R2D2, MetaOptNet also attempts to use a convex base learner, *i.e,* linear support vector machine (SVM), in the inner loop for FSL. Different from the above four methods, Versa is designed from a new perspective of Bayesian learning by introducing a versatile amortization network.

Discussions. We can see that there are mainly two development directions in the meta-learning based methods: (1) an implicit two-loop optimization direction following the procedure of MAML, such as LEO, ANIL and BOIL; (2) an explicit two-loop optimization direction following the procedure of R2D2, *e.g.* MetaOpeNet. The former follows a trend of designing a more efficient optimization-based meta-learning method, aiming to address the complicated optimization problem of meta-training. The latter aims to design a more effective base learner by introducing the traditional and classic machine learning algorithms into the paradigm of meta learning. On the other hand, note that the early meta-learning based methods mainly employ the pure meta-training paradigm to learn a model from scratch. Some methods, such as MTL [13] and LEO [40], have already introduced pre-training into the training process, because the pre-training technique can be easily leveraged as pre-processing. Therefore, the effect of pre-training is worth further investigating in meta-learning based FSL methods.

2.4 Metric-learning based Methods

As illustrated in Figure 1(c), different from the two-loop structure of meta-learning based methods, *metric-learning based methods* [4], [15], [22], [23], [44], [45], [46], [47] directly compare the similarities (or distances) between the query images and support classes (*i.e.,* learning-to-compare) through *one single feed-forward pass* through the episodic-training mechanism [2]. In other words, for each input query image, the entire support set $\mathcal{A}_\mathcal{S}$ is jointly encoded into the latent embedding space simultaneously, and their relationships (*i.e.,* outputs) are used to classify. In this way, *i.e.,* by conditioning on the support set, it is able to enable the model adapt to the characteristics of each task, and make the learned representations transferable between different tasks.

Prototypical Networks (ProtoNet) is a typical metric-learning based method [23], which takes the mean vector of each support class as its corresponding prototype representation, and then compares the relationships between the

query image and prototypes. Specifically, given a few-shot task $\mathcal{T} = \langle \mathcal{A}_\mathcal{S}, \mathcal{A}_\mathcal{Q} \rangle$, $\mathcal{A}_\mathcal{S} = \{S_1, S_2, \cdots, S_C\}$, the prototype $\boldsymbol{c}_i \in \mathbb{R}^d$ of each class S_i can be formulated as,

$$\boldsymbol{c}_i = \frac{1}{|S_i|} \sum_{j=1}^{\mathcal{K}} f_\theta(X_j), \tag{6}$$

where $X_j \in S_i$, $|S_i| = \mathcal{K}$ denotes there are $\mathcal{K}$ images (*i.e.,* $\mathcal{K}$-shot) in the i-th support class. Here, $f_\theta(X_j) \in \mathbb{R}^d$ means $f_\theta(\cdot)$ extracts d-dimensional global feature representation for each input image. Given a distance function $D(\cdot, \cdot)$, such as Euclidean distance, the predicted posterior probability distributions of a query image Q is,

$$\rho(y = i|Q) = \frac{\exp\big(-D(f_\theta(Q), \boldsymbol{c}_i)\big)}{\sum_{j=1}^{C} \exp\big(-D(f_\theta(Q), \boldsymbol{c}_j)\big)}. \tag{7}$$

Specifically, at the training stage, the standard cross-entropy loss can be employed to train the entire model. Also, during test, the nearest-neighbor classifier (1-NN) can be conveniently used for prediction.

Deep Nearest Neighbor Neural Network (DN4) is another representative method [15], which argues that performing pooling on local features into a compact global-level representation will lose considerable discriminative information. Instead, DN4 advocates to directly use the raw local features and employs a local descriptor based *image-to-class (I2C)* measure to learn transferable local features. Specifically, given an input image X, without the last pooling or FC layer of the embedding network, $f_\theta(X) \in \mathbb{R}^{d \times h \times w}$ will be a three-dimensional tensor, and can be reshaped as a set of d-dimensional local descriptors

$$f_\theta(X) = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}, \tag{8}$$

where $\boldsymbol{x}_i$ is the i-th local descriptor and $n = h \times w$ is the total number of local descriptors for image X. Note that, in ProtoNet, both query image and the images in the support classes are represented with a global feature vector, respectively. Especially, the class prototype is also an average-pooling of multiple global feature vectors (*e.g.,* the $\mathcal{K}$-shot setting). In contrast, in DN4, both query image and each support class are represented with a set of local descriptors without any pooling.

Suppose a query image Q and a support class S are represented as $f_\theta(Q) = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ and $f_\theta(S) = [f_\theta(X_1), \ldots, f_\theta(X_K)] \in \mathbb{R}^{d \times nK}$, respectively. The image-to-class measure will be calculated as

$$D_{\text{I2C}}(Q, S) = \sum_{i=1}^{n} \text{Topk}\Big(\frac{f_\theta(Q)^\top \cdot f_\theta(S)}{\|f_\theta(Q)\|_F \cdot \|f_\theta(S)\|_F}\Big), \tag{9}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, and $\text{Topk}(\cdot)$ means selecting the k largest elements in each row of the correlation matrix between Q and S.

Other Representative Methods include *Relation Network (RelationNet)* [4], *Covariance Metric Network (CovaMNet)* [14], *Cross Attention Network (CAN)* [48], *Deep Earth Mover's Distance (DeepEMD)* [45], *Few-shot Embedding Adaptation with Transformer (FEAT)* [16], and *Relational Embedding Network (RENet)* [49], *etc.* The core of RelationNet is to learn a non-linear metric through a deep convolutional neural network, instead of choosing a specific metric function, *e.g.,* Euclidean distance. Instead of using traditional first-order class representations, *e.g.,* mean vector, CovaMNet proposes a second-order local covariance representation to represent each class along with a new covariance metric. From the perspective of attention, CAN proposes to calculate the cross attention between each pair of class feature and query feature so as to learn more discriminative features. Similar to DN4, DeepEMD also uses the set of local descriptors as the representation for an image and employs the Earth Mover's Distance to calculate a structural distance between dense local representations of two images. FEAT proposes to take a set-to-set transformation via a transformer layer to make the global instance embedding of support set become task-specific for better adaptation. Similar to CAN, RENet proposes a self-correlational representation module and a cross-correlational attention module to learn relational patterns within and between images, respectively.

Discussions. From the recent advances, we can see that the main trends in metric-learning based methods are in two folds: (1) how to effectively represent each image and each support class; and (2) how to design a more powerful metric function. Specifically, the research trends show that using local descriptor representations may be a good choice and designing a task-adaptive metric function is important. In addition, we notice that because there are generally no data-dependent parameters in the classifier (*i.e.,* 1-NN classifier), the metric-learning based methods do not have the test-tuning procedure at the test stage. Therefore, is the paradigm employed by metric-learning based methods reasonable? In other words, is test-tuning really essential for the FSL problem?

3 Evaluation Settings and LibFewShot

Datasets. Our main experiments are conducted on two benchmark datasets, *i.e., miniImageNet* [2] and *tieredImageNet* [50]. Moreover, we also evaluate the cross-domain generalization ability of each FSL method, three fine-grained benchmark datasets, *i.e., Stanford Dogs* [51], *Stanford Cars* [52] and *CUB Birds-200-2011* [53]. Following the literature, each data set is split into training (auxiliary), validation and test sets, respectively. The details can be seen in Table 1. Note that all the images in the above datasets are resized to a resolution of 84×84.

Backbone Architectures. Following the literature [12], [23], [27], we adopt three different embedding backbones from shallow to deep, *i.e., Conv64F*, *ResNet12* and *ResNet18*. Specifically, Conv64F contains four convolutional blocks, each of which consists of a convolutional (Conv) layer, a batch-normalization (BN) layer, a ReLU/LeakyReLU layer and a max-pooling (MP) layer, where the numbers of filters of these blocks are $\{64, 64, 64, 64\}$. ResNet12 consists of four residual blocks, each of which further contains three convolutional blocks (each is built as Conv-BN-ReLU-MP) along with a skip connection layer, where the numbers of filters of these blocks are $\{64, 160, 320, 640\}$. ResNet18 is the standard architecture used in [54]. One important difference between ResNet12 and ResNet18 is that ResNet12 uses Dropblock [55] in each residual block, while ResNet18 does not. Note that, in Table 2, the numbers of filters of ResNet12† are $\{64, 96, 128, 256\}$; Conv64F† has five Conv blocks; and Conv64F‡ uses additional low-level features.

TABLE 1
Data splits used in each dataset. $\mathcal{C}_{\text{all}}/\mathcal{N}_{\text{train}}$ is the total number of classes/images. $\mathcal{C}_{\text{train}}/\mathcal{N}_{\text{train}}$, $\mathcal{C}_{\text{val}}/\mathcal{N}_{\text{val}}$ and $\mathcal{C}_{\text{test}}/\mathcal{N}_{\text{test}}$ indicate the number of classes/images in training (auxiliary), validation and test sets, respectively.

Dataset	*mini*ImageNet	*tiered*ImageNet	Stanford Dogs	Stanford Cars	CUB Birds-200-2011
$\mathcal{C}_{\text{all}}$	100	608	120	196	200
$\mathcal{C}_{\text{train}}$	64	351	70	130	130
$\mathcal{C}_{\text{val}}$	16	97	20	17	20
$\mathcal{C}_{\text{test}}$	20	160	30	49	50
$\mathcal{N}_{\text{all}}$	60000	779165	20580	16185	11788
$\mathcal{N}_{\text{train}}$	38400	448695	12165	10766	7648
$\mathcal{N}_{\text{val}}$	9600	124261	3312	1394	1182
$\mathcal{N}_{\text{test}}$	12000	206209	5103	4025	2958

TABLE 2
Reproduction results on *mini*ImageNet using the original paper settings. Results are reported with the mean accuracy over 3000 5-way 1-shot and 5-way 5-shot test tasks, respectively. Global-label indicates that the global labels of the auxiliary set are used for pre-training or additional global classification during training. Local-label means that only specific local labels are used in the episodic- or meta-training phase. Test-Tune means test-tuning of using the support set during test. Test-DA denote data augmentation during test. KD means knowledge distillation and SS means self-supervision. † and ‡ indicate that the standard backbones are different or slightly modified. ✓ with blue color indicates a given trick is not used in the original setting but used in our reproduction experiment. - means the result is not reported in the original paper.

Method	Embed.	Image Size		Training Tricks							5-way 1-shot		5-way 5-shot	
		84	224	Lr/Optimizer/Decay	Global-label	Local-label	Test-Tune	Test-DA	KD	SS	Reported	Ours	Reported	Ours
Baseline	Conv64F	✓		0.001/Adam/No	✓		✓				42.11	42.34	62.53	62.18
	ResNet18		✓	0.001/Adam/No	✓		✓				51.75	51.18	74.27	74.05
Baseline++	Conv64F	✓		0.001/Adam/No	✓		✓				48.24	46.21	66.43	65.18
	ResNet18		✓	0.001/Adam/No	✓		✓				51.87	53.60	75.68	73.63
RFS-simple	ResNet12	✓		0.05/SGD/Step	✓		✓	✓			62.02	62.80	79.64	79.57
RFS-distill	ResNet12	✓		0.05/SGD/Step	✓		✓	✓	✓		64.82	63.44	82.14	80.17
SKD-GEN0	ResNet12	✓		0.05/SGD/Step	✓		✓	✓		✓	65.93	66.45	83.15	83.43
SKD-GEN1	ResNet12	✓		0.05/SGD/Step	✓		✓	✓	✓	✓	67.04	67.09	83.54	83.67
Neg-Cosine	ResNet12	✓		0.003/Adam/Cosine	✓		✓				63.85	63.28	81.57	81.24
MAML	Conv32F	✓		0.001/Adam/Step		✓	✓				48.70	47.41	63.11	65.24
Versa	Conv64F†	✓		$1e^{-4}$/Adam/Step		✓	✓				53.40	51.92	67.37	66.26
R2D2	Conv64F	✓		0.001/Adam/Step		✓	✓				49.50	47.57	65.40	66.68
	Conv64F‡	✓		0.001/Adam/Step		✓	✓				51.80	55.53	68.40	70.79
ANIL	Conv32F	✓		0.001/Adam/Step		✓	✓				46.70	48.44	61.50	64.35
LEO	WRN-28-10	✓		$4e^{-4}$/Adam/No	✓	✓	✓				61.76	55.89	77.59	70.55
BOIL	Conv64F	✓		0.001/Adam/Step		✓	✓				49.61	48.00	66.45	64.39
	ResNet12	✓		0.001/Adam/Step		✓	✓				–	58.87	71.30	72.88
MTL	ResNet12	✓		0.1/SGD/Step 0.001/Adam/Step	✓	✓	✓				60.20	60.20	74.30	75.86
ProtoNet†	Conv64F	✓		0.001/Adam/Step		✓					46.14	46.30	65.77	66.24
RelationNet	Conv64F	✓		0.001/Adam/Step		✓					50.44	51.75	65.32	66.77
CovaMNet	Conv64F	✓		0.001/Adam/Step		✓					51.19	53.36	67.65	68.17
DN4	Conv64F	✓		0.001/Adam/Step		✓					51.24	51.95	71.02	71.42
	ResNet12†	✓		0.001/Adam/Step		✓					54.37	57.76	74.44	77.57
CAN	ResNet12	✓		0.1/SGD/Step	✓	✓					63.85	66.62	79.44	78.96
RENet	ResNet12	✓		0.1/SGD/MultiStep	✓	✓					67.60	66.83	82.58	82.13

TABLE 3
The overview picture of the state of the art on *mini*ImageNet and *tiered*ImageNet by controlling the most common implementation details except some special tricks, with our **LibFewShot**. The fifth column shows the total number of trainable parameters used by each method. Global-label indicates that the global labels of the auxiliary set are used for pre-training or global classification during training. Local-label means that only the specific local labels are used in the episodic- or meta-training phase. Test-tune means test-tuning of using the support set at the test stage. Note that SKD-GEN0 uses an additional self-supervision trick that is the core of this method.

Method	Venue	Embed.	Type	Para.	Tricks			*mini*ImageNet		*tiered*ImageNet	
					Global-label	Local-label	Test-tune	1-shot	5-shot	1-shot	5-shot
Baseline [27]	ICLR'19	Conv64F	Non-episodic	0.22M	✓		✓	44.90	63.96	48.20	68.96
Baseline++ [27]	ICML'19	Conv64F	Non-episodic	0.22M	✓		✓	**48.54**	65.47	49.73	70.14
RFS-simple [30]	ECCV'20	Conv64F	Non-episodic	0.22M	✓		✓	47.97	65.88	**52.21**	**71.82**
Neg-Cosine [34]	ECCV'20	Conv64F	Non-episodic	0.22M	✓		✓	47.34	65.97	51.21	71.57
SKD-GEN0 [32]	BMVC'21	Conv64F	Non-episodic	0.22M	✓		✓	48.14	**66.36**	51.78	70.65
MAML [3]	ICML'17	Conv64F	Meta	0.12M		✓	✓	49.55	64.92	50.98	67.12
Versa [39]	NeurIPS'18	Conv64F	Meta	1.18M		✓	✓	52.75	67.40	52.28	69.41
R2D2 [26]	ICLR'19	Conv64F	Meta	0.11M		✓	✓	51.19	67.29	52.18	69.19
LEO [40]	ICLR'19	Conv64F	Meta	1.20M	✓	✓	✓	**53.31**	**67.47**	**58.15**	**74.21**
MTL [13]	CVPR'19	Conv64F	Meta	1.80M	✓	✓	✓	46.70	64.79	49.11	69.13
ANIL [41]	ICLR'20	Conv64F	Meta	0.12M		✓	✓	48.01	63.88	49.05	66.32
BOIL [43]	ICLR'21	Conv64F	Meta	0.12M		✓	✓	47.92	64.39	50.04	65.51
ProtoNet [23]	NeurIPS'17	Conv64F	Metric	0.11M		✓		47.05	68.56	46.11	70.07
RelationNet [4]	CVPR'18	Conv64F	Metric	0.23M		✓		51.52	66.76	54.37	71.93
CovaMNet [14]	AAAI'19	Conv64F	Metric	0.11M		✓		51.59	67.65	51.92	69.76
DN4 [15]	CVPR'19	Conv64F	Metric	0.11M		✓		54.47	72.15	56.07	75.75
CAN [48]	NeurIPS'19	Conv64F	Metric	0.13M	✓	✓		55.88	70.98	55.96	70.52
RENet [49]	ICCV'21	Conv64F	Metric	0.20M	✓	✓		**57.62**	**74.14**	**61.62**	**76.74**
Baseline [27]	ICLR'19	ResNet12	Non-episodic	12.47M	✓		✓	56.39	76.18	65.54	83.46
Baseline++ [27]	ICML'19	ResNet12	Non-episodic	12.47M	✓		✓	58.79	75.31	66.32	83.05
RFS-simple [30]	ECCV'20	ResNet12	Non-episodic	12.47M	✓		✓	61.65	78.88	70.55	84.74
Neg-Cosine [34]	ECCV'20	ResNet12	Non-episodic	12.47M	✓		✓	60.60	78.80	70.15	84.94
SKD-GEN0 [32]	BMVC'21	ResNet12	Non-episodic	12.47M	✓		✓	**66.40**	**83.06**	**71.90**	**86.20**
Versa [39]	NeurIPS'18	ResNet12	Meta	13.25M		✓	✓	55.71	70.05	57.14	75.48
R2D2 [26]	ICLR'19	ResNet12	Meta	12.42M		✓	✓	59.52	74.61	65.07	83.04
LEO [40]	ICLR'19	ResNet12	Meta	12.60M	✓	✓	✓	56.62	69.99	64.75	81.42
MTL [13]	CVPR'19	ResNet12	Meta	13.13M	✓	✓	✓	**62.67**	**79.16**	**68.68**	**84.58**
ANIL [41]	ICLR'20	ResNet12	Meta	12.43M		✓	✓	52.77	68.11	55.65	73.52
BOIL [43]	ICLR'21	ResNet12	Meta	12.43M		✓	✓	58.87	72.88	64.66	80.38
ProtoNet [23]	NeurIPS'17	ResNet12	Metric	12.42M		✓		57.10	74.20	62.93	83.30
RelationNet [4]	CVPR'18	ResNet12	Metric	23.53M		✓		55.22	69.25	56.86	74.66
CovaMNet [14]	AAAI'19	ResNet12	Metric	12.42M		✓		54.69	70.72	56.03	75.21
DN4 [15]	CVPR'19	ResNet12	Metric	12.42M		✓		59.14	75.26	64.41	82.59
CAN [48]	NeurIPS'19	ResNet12	Metric	12.65M	✓	✓		62.68	78.36	**70.46**	**84.50**
RENet [49]	ICCV'21	ResNet12	Metric	12.67M	✓	✓		**64.81**	**79.90**	70.14	82.70
Baseline [27]	ICLR'19	ResNet18	Non-episodic	11.20M	✓		✓	54.11	74.44	64.65	82.73
Baseline++ [27]	ICML'19	ResNet18	Non-episodic	11.20M	✓		✓	52.70	75.36	65.85	83.33
RFS-simple [30]	ECCV'20	ResNet18	Non-episodic	11.20M	✓		✓	61.65	76.60	69.14	83.21
Neg-Cosine [34]	ECCV'20	ResNet18	Non-episodic	11.20M	✓		✓	60.99	76.40	68.36	83.77
SKD-GEN0 [32]	BMVC'21	ResNet18	Non-episodic	11.20M	✓		✓	**66.18**	**82.21**	**70.00**	**84.70**
Versa [39]	NeurIPS'18	ResNet18	Meta	11.96M		✓	✓	55.08	69.16	57.30	75.67
R2D2 [26]	ICLR'19	ResNet18	Meta	11.17M		✓	✓	58.36	75.69	64.73	**83.40**
LEO [40]	ICLR'19	ResNet18	Meta	11.31M	✓	✓	✓	57.51	69.33	64.02	78.89
MTL [13]	CVPR'19	ResNet18	Meta	11.88M	✓	✓	✓	**60.29**	**76.25**	**65.12**	79.99
ANIL [41]	ICLR'20	ResNet18	Meta	11.17M		✓	✓	52.96	65.88	55.81	73.53
BOIL [43]	ICLR'21	ResNet18	Meta	11.17M		✓	✓	57.85	71.88	62.26	77.94
ProtoNet [23]	NeurIPS'17	ResNet18	Metric	11.17M		✓		58.48	75.16	63.50	82.51
RelationNet [4]	CVPR'18	ResNet18	Metric	18.29M		✓		53.98	71.27	60.80	77.94
DN4 [15]	CVPR'19	ResNet18	Metric	11.17M		✓		57.30	74.23	64.83	82.77
CAN [48]	NeurIPS'19	ResNet18	Metric	11.35M	✓	✓		62.33	77.12	**71.70**	**84.61**
RENet [49]	ICCV'21	ResNet18	Metric	11.52M	✓	✓		**66.21**	**81.20**	71.53	84.55

Evaluation Protocols. Following the prior works [2], [4], [23], in this paper, we control the evaluation setting for all methods, evaluate them on 600 sampled tasks and repeat this process five times, *i.e.,* a total of 3000 tasks. The top-1 mean accuracy will be reported. In addition, early works [2], [4], [23] generally use raw evaluation, *i.e.,* directly resizing the test image into 84×84 for evaluation, while the recent work [9], [16], [27] has tried single center crop evaluation like the generic image classification [54]. To make the comparison more fair and can adapt to future development, we follow the latest setting to use the single center crop evaluation.

Bag of Tricks. Many works in other fields show that *trick matters*, especially in deep learning [56], [57], so as to FSL. For example, some recent FSL works have introduced such "tricks", such as knowledge distillation [30], [32], self-supervision [58], [59] and Mixup [33], into the FSL problem. Specifically, we empirically summarize some key training tricks in FSL as below: (1) using data augmentation at the training stage (Train-DA); (2) augmenting the support set multiple times at the test stage (Test-DA); (3) pre-training on the auxiliary set (Pre-train); (4) global classification of using the global labels of the auxiliary set (Global-label); (5) Larger Episode size, *i.e.,* increasing the number of tasks at each iteration; (6) higher way-number or higher shot-number during the episodic training; (7) using knowledge distillation or self-distillation (KD); (8) using self-supervision (SS); (9) using Dropblock; (10) using label smoothing; and (11) using more learnable parameters.

LibFewShot. In the literature, many implementation details or "tricks" are only briefly mentioned or even overlooked in many FSL works. However, these non-trivial tricks may lead to significant *algorithm-agnostic performance boost*, which will make the comparison somewhat unfair and make some conclusions untenable. In this sense, it will not only make beginners struggle with the reproduction of other comparison methods, but also hinder them from developing their own methods. On the other hand, there are many interesting issues worth studying under a unified framework, including (1) the doubts on the necessity of meta- or episodic-training mechanism posted by the recent non-episodic based methods, (2) the effects of different deep learning tricks on the FSL problem, (3) the actual progress of FSL in the case of restricting deep learning tricks, (4) the effect of transformers on FSL.

Therefore, to address the above issues, we develop *a comprehensive library for few-shot learning (LibFewShot)* by re-implementing the state-of-the-art methods into the same framework and applying the same training tricks to the maximum extent. So far, eighteen representative methods have been investigated, including five *non-episodic based methods*, seven *meta-learning based methods* and six *metric-learning based methods*, where the details can be seen in Table 2. The details of the architecture of LibFewShot are illustrated in the supplementary material.

4 Experimental Results and Discussions

4.1 Reproduction Results

To validate the correctness of our re-implementation, we adopt the original settings of these methods and re-implement them with LibFewShot. Our re-implemented results and their originally reported results on *mini*ImageNet are shown in Table 2. Importantly, the original implementation details and tricks are also listed in detail, which can intuitively show the differences between different FSL methods on the implementation.

From Table 2, we can observe that: (1) different FSL methods employ different backbones, especially the early meta- and metric-learning based methods only use a shallow backbone of Conv64F, making the comparison between different methods of using different backbones somewhat unfair; (2) non-episodic based methods, especially RFS [30] and SKD [32], indeed obtain the state-of-the-art results, but they employ a much deeper backbone and much more tricks than other methods; (3) both knowledge distillation (KD) and self-supervision (SS) can significantly boost the performance; (4) when using additional pre-training or global classification with global labels, MTL [13], CAN [48] and RENet [49] can achieve much better results. Notably, RENet consistently outperforms RFS-simple and RFS-distill no matter the officially reported results or our re-implemented results in both the 1-shot and 5-shot settings, *which shows great potential of the paradigm of pre-training + meta-training*.

4.2 Overview of the State of the Art

As seen in Table 2, the implementation details of different methods vary a lot, which cannot trustingly reflect the actual progress of FSL. To this end, we keep most of the common training tricks consistent for all the methods except some special tricks (*e.g.,* self-supervision (SS) for SKD-GEN0 [32]) to make a relatively fair comparison. To be specific, we use the exactly same embedding backbone, fixed input image size, *i.e.,* 84×84, the standard data augmentations (which consists of Resize, RandomCrop, RandomHorizonFlip and ColorJitter) during training, and center crop evaluation for all the methods. As for the optimizer and learning rate scheduler, only minor modifications are made according to their settings in the original papers. All methods are trained with 100 epochs, except for Versa with 200 epochs. Also, 2000 episodes are used for *mini*ImageNet per epoch, while 5000 episodes for *tiered*ImageNet. The results are reported in Table 3.

First, when using Conv64F as the embedding backbone, we can see that metric-learning based methods generally achieve relatively better results than meta-learning based and non-episodic based methods. This may be because when the embedding backbone is shallow, *i.e.,* the feature representation is weak, the design or selection of metric function will be important. As also can be seen, the best methods of the metric-learning based and meta-learning based methods are RENet and LEO, respectively. One common characteristic of these two methods is that they employ both global- and local-label during training, which shows both global and local labels will benefit few-shot learning. In Section 4.4, we will further demonstrate this point. Similarly, MTL and CAN also employ such a kind of training strategy, both of which can obtain competitive results too. It is worth mentioning that some methods use more trainable parameters, such as Versa, LEO and MTL, which will also somewhat affect the final results.

TABLE 4
The necessity of test-tuning in the test phase, where RFS-simple is taken as an example method by using different test settings. Test-tune means test-tuning of using the support set at the test stage. ℓ_2 means using ℓ_2 normalization. ($\mathcal{C}$, $\mathcal{K}$) means $\mathcal{C}$-way $\mathcal{K}$-shot tasks.

Method	Classifier	Test-tune	ℓ_2	(5,1)	(5,5)	(10,1)	(10,5)	(15,1)	(15,5)	(20,1)	(20,5)
RFS-simple	LR	✓	✗	58.69	77.72	43.87	64.86	36.25	57.08	31.24	51.70
	LR	✓	✓	**61.61**	78.80	**45.85**	65.18	**37.72**	57.01	**32.47**	51.25
RFS-NN	1-NN	✗	✗	56.74	78.27	41.64	65.73	33.87	58.06	29.19	52.70
	1-NN	✗	✓	60.64	**78.85**	44.89	**66.20**	36.86	**58.46**	31.70	**53.15**

Second, when using ResNet12 or ResNet18, a much deeper network, as the embedding backbone, we observe that the non-episodic based methods especially RFS-simple, Neg-Cosine and SKD-GEN0, perform significantly better than local-label based meta- and metric-learning methods, such as Versa, R2D2, ANIL, BOIL, ProtoNet, RelationNet, CovaMNet and DN4. However, when both global and local labels are used, we can see that MTL, CAN and RENet will be on par and even superior to RFS-simple and Neg-Cosine. Note that SKD-GEN0 utilizes an additional self-supervision trick, which essentially is not so fair for other methods. It is also worth noting that when using ResNet12 as the backbone most of these methods will perform much better than using ResNet18. This is because ResNet12 is much wider than ResNet18, *i.e.,* ResNet12 enjoys more parameters than ResNet18. In addition, Dropblock is also used in ResNet12, while ResNet18 does not.

4.3 The Necessity of Episodic- and Meta-training

One concern of our work is to investigate the necessity of episodic- and meta-training and try to answer the questions raised in Section 2. To this end, we employ RFS-simple [30] without Test-DA as the benchmark, and use its pre-trained ResNet12 on *mini*ImageNet as the embedding backbone.

Is test-tuning really important at the test stage in FSL? From Table 3, we can see that RFS-simple consistently outperforms Baseline and Baseline++ when using ResNet12 or ResNet18 as the embedding backbone. However, the only difference between them is that RFS adopts logistic regression (LR) with ℓ_2 normalization as the classifier in the test phase. That is to say, for each novel test task, a new LR classifier will be specially learned during test (*i.e.,* test-tuning). Instead of using LR, we directly employ ℓ_2 normalization based 1-NN for the final test classification, *i.e.,* a non-parametric classifier without requiring test-tuning, and name this new variant as RFS-NN.

As seen in Table 4, without using ℓ_2 normalization, the performance of RFS-simple will significantly degrade, especially in the 1-shot setting. In addition, using neither test-tuning nor ℓ_2 normalization, RFS-NN still performs better than RFS-simple on the 5-shot tasks. More importantly, when only using ℓ_2 normalization, RFS-NN could achieve very competitive results as RFS-simple on the 1-shot tasks and clearly outperforms RFS-simple on the 5-shot tasks especially in the higher way settings. This reveals that *LR or test-tuning is not so important in the limited-data regime, but a good embedding and the ℓ_2 normalization are!* One reason is that ℓ_2 normalization can align the distributions between the source and target domains (base and novel classes) to some extent to guarantee good transferability (evidence can be seen in [60]).

In summary, we argue that test-tuning is not so necessary at the test stage in FSL because of the limited-data regime. In another word, a few labeled examples normally cannot learn an effective classifier in the test phase. This also demonstrates that the paradigm of metric-learning based methods is reasonable.

Is episodic- or meta-training necessary at the training stage in FSL? To answer this question, we employ three methods, *i.e.,* ProtoNet, DN4 and R2D2, to episodic/meta training (fine-tuning) the same pre-trained feature embedding used in RFS-simple. That is to say, this is a kind of paradigm of using pre-training with global labels and meta fine-tuning with local labels. Note that for ProtoNet, we use a cosine distance instead of the original Euclidean distance, because the above analysis shows that ℓ_2 normalization is important. Specifically, for each method, we perform $\mathcal{C}$-way 1-shot episodic/meta training and the corresponding $\mathcal{C}'$-way 1-shot testing.

The results are shown in Figure 2, whose details can be found in Section 4.4. As seen, for ProtoNet, DN4 and R2D2, all the episodic- or meta- fine-tuning (*i.e.,* Two Stages) can further improve the performance over RFS-simple. Note that the data augmentation used in the fine-tuning phase (which only contains Resize and RandomResizedCrop) shall be somewhat different from the data augmentation used in the pre-training phase, which is one indetectable trick for this success overlooked in the literature. We also notice that if using somewhat different or more complicated metric functions in the meta fine-tuning phase will also significantly benefits the final performance. Overall, this simple experiment demonstrates that pre-training may obtain a good initial embedding, but it is not the optimal one. *Therefore, we argue that the episodic- or meta-training is worthy of further investigation at the training stage in FSL.*

4.4 Ablation Study on Non-trivial Tricks

In this section, we select multiple representative methods from the three kinds of FSL methods and conduct ablation studies on multiple non-trivial tricks. To be specific, *mini*ImageNet and ResNet12 are taken as the default benchmark dataset and embedding backbone, respectively. Also, both 5-way 1-shot and 5-shot tasks are considered. Typically, non-trivial tricks, including *global-label*, *local-label*, *strong data augmentation*, *knowledge distillation*, *label smoothing* and *self supervision*, are taken into consideration. In addition, RFS-simple, R2D2, ProtoNet, DN4, CAN-Local and RENet-Local will be selected as the representative FSL methods. Note that

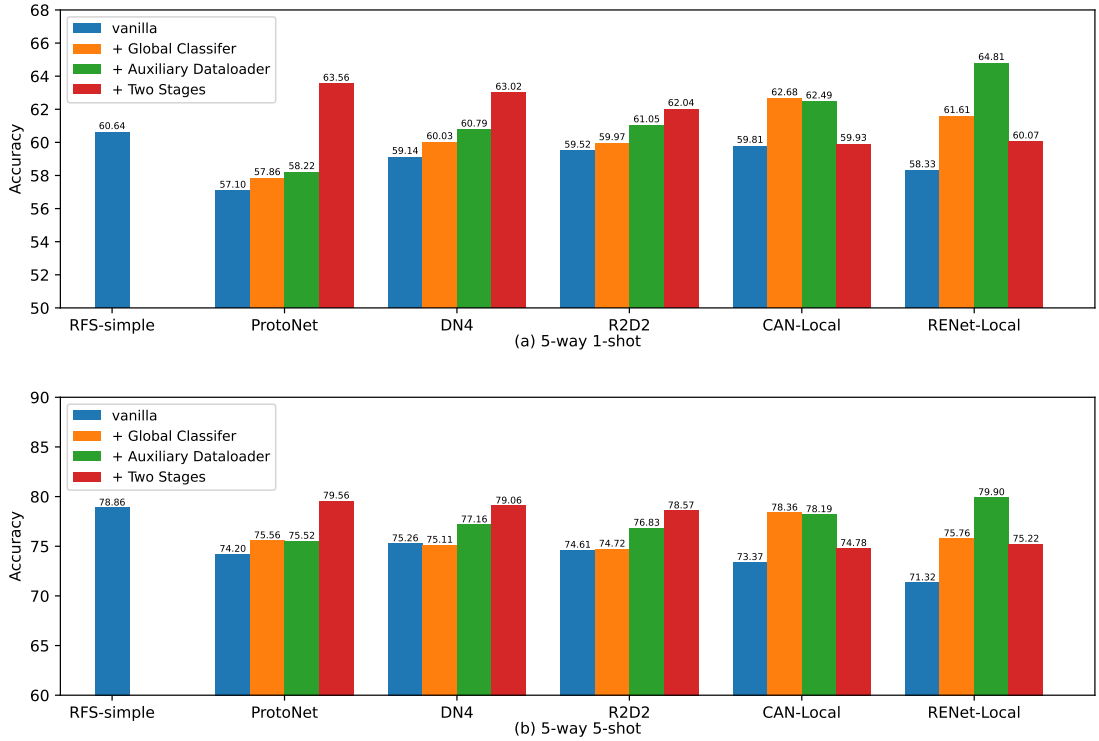


Fig. 2. **Effects of using both global and local labels** on *mini*ImageNet with a ResNet12 backbone under both 5-way 1-shot and 5-shot settings.

CAN-Local and RENet-Local are the variants of CAN and RENet of only using the local labels, respectively.

4.4.1 Effects of Global and Local Labels

To thoroughly investigate the effects of global and local labels, we consider three types of how to use both of them: (1) Pre-training with global labels at the first stage and episodic fine-tuning with local labels using a specific method at the second stage, we name this type as *Two Stages* for simplicity; (2) Global classifier with global labels + local classifier with local labels through episodic-training in a multi-task learning manner, we name this type as *Global Classifier* for short; (3) is similar to (2) but uses different data loaders for these two tasks, which is first adopted in RENet. We name the type of (3) as *Auxiliary Dataloader* for simplicity. Note that both (2) and (3) are one-stage methods, which use both global and local labels via a single training process. From the results in Fig. 2, we have the following observations.

(1) All three types of using both global and local labels work well, and can effectively improve the performance over the vanilla version of only using the local or global labels; For example, under the 5-way 1-shot setting, *ProtoNet + Two Stages* (using both global and local labels) can achieve an accuracy of 63.56%, obtaining 6.46% improvements over the vanilla version (using local labels only), which is significantly better than RFS-simple (60.64%). Similarly, *DN4 + Two Stages* (63.02%) and *R2D2 + Two Stages* (62.04%) gains 4.34% and 2.52% improvements over their vanilla versions, respectively.

(2) For the episodic-training based methods of only using local labels, such as ProtoNet, DN4 and R2D2, using extra global labels by pre-training (*i.e.*, a two-stage way) is the most effective way. In contrast, for the methods originally designed with both global and local labels, such as CAN and RENet, using the global labels with a multi-task manner (*i.e.*, a one-stage way) is generally the more effective way.

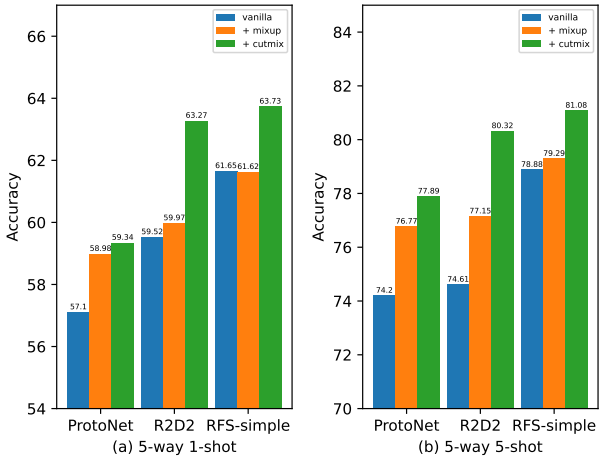


Fig. 3. **Effects of using strong data augmentation**, *i.e.*, mixup and cutmix, for ProtoNet, R2D2 and RFS-simple on *mini*ImageNet with a ResNet12 backbone.

4.4.2 Effect of Strong Data Augmentation

Data augmentation is one of the most general and effective tricks in the field of deep learning. Therefore, it will be interesting to investigate the effect of strong data augmentation techniques for FSL. To be specific, we take mixup [61] with alpha of 0.2 and cutmix [62] with alpha of 1.0 (alpha is the hyper-parameter of the beta distribution in these two tricks)

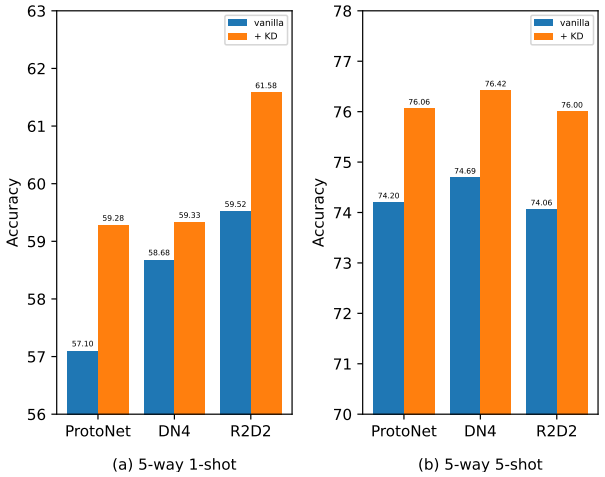


Fig. 4. **Effect of knowledge distillation (KD)** for ProtoNet, DN4 and R2D2 on *mini*ImageNet with a ResNet12 backbone.

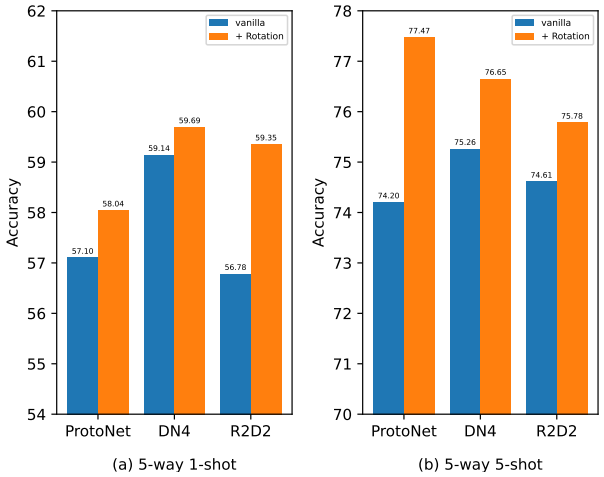


Fig. 5. **Effect of self supervision (SS)** for ProtoNet, DN4 and R2D2 on *mini*ImageNet with a ResNet12 backbone.

as the representative strong data augmentations and take ProtoNet, R2D2 and RFS-simple as the representative FSL methods.

From the results in Fig. 3, we can observe that: (1) cutmix can significantly improve the performance of all the three FSL methods. For example, in the 1-shot setting, cutmix obtains 2.24%, 3.75%, and 2.08% improvements over the vanilla versions of ProtoNet, R2D2 and RFS-simple, respectively. Similarly, in the 5-shot setting, cutmix obtains 3.69%, 5.71%, and 2.2% improvements over the three vanilla versions, respectively. (2) In most cases, mixup can effectively improve the performance of all the three FSL methods, especially for ProtoNet and R2D2. For example, in the 5-shot setting, mixup gains 2.57%, 2.54%, 0.41% improvements over the vanilla versions of ProtoNet, R2D2 and RFS-simple, respectively. (3) When using strong data augmentation, *e.g.*, cutmix, R2D2 is surprisingly competitive to RFS-simple. That is to say, by just using a simple data augmentation technique, *R2D2 + cutmix* achieves accuracies of 63.27% and 80.32% in the 1-shot and 5-shot settings, respectively, which are already very competitive in Table 3.

4.4.3 Effect of Knowledge Distillation

Knowledge distillation (KD) has been introduced for FSL by some recent FSL methods, such as RFS-distill and SKD-GEN1. To further verify its effect, we apply KD to three early FSL methods, *i.e.*, ProtoNet, DN4 and R2D2. Following RFS-distill and SKD-GEN1, we also adopt the self-distillation to distill the knowledge from a trained few-shot model to a new identical model initialized from scratch for once.

As seen in Fig. 4, in both settings of 1-shot and 5-shot, KD can consistently improve the performance of the vanilla versions of all the three FSL methods. For example, under the 5-shot setting, the gained improvements for ProtoNet, DN4 and R2D2 are 1.86%, 1.73%, 1.94%, respectively. Similarly, under the 1-shot setting, KD boosts the performance of ProtoNet, DN4 and R2D2 by 2.18%, 0.65%, 2.06% improvements, respectively. This shows that KD is indeed effective in FSL. On the other hand, we shall notice that KD is a general trick, which is applicable to other FSL methods.

4.4.4 Effect of Self Supervision

Self-supervision methods have been shown to be effective in many fields of deep learning with the advantage of requiring no additional annotation costs. In the field of FSL, Gudarus *et al.* [63] has used the additional rotation prediction self-supervision [64] as an auxiliary parallel task. Similarly, SKD [32] also adopts the rotation prediction as an auxiliary task but in the pre-training stage. These works have demonstrated that self-supervision, especially rotation prediction self-supervision, is effective for FSL.

To further investigate whether self supervision is a general trick for other FSL methods, we take the rotation prediction as an auxiliary task and employ ProtoNet, DN4 and R2D2 as three representative FSL methods. Specifically, all methods are trained from scratch through the episodic-training mechanism by using an additional rotation classifier in a multi-task manner. The results are shown in Fig. 5. We can see that no matter in the 5-way 1-shot setting nor 5-way 5-shot setting, the rotation prediction self supervision can consistently boost the performance of all three FSL methods. For example, in the 1-shot setting, the rotation self-supervision can boost ProtoNet, DN4 and R2D2 for 0.94%, 0.55%, 2.57% improvements, respectively. Similarly, in the 5-shot setting, the gained performance improvements of ProtoNet, DN4 and R2D2 are 3.27%, 1.39%, 1.17%, respectively. This reveals that the self-supervision task, especially the rotation prediction task, is indeed effective for FSL and can be a general trick for different FSL methods.

4.4.5 Effect of Label Smoothing

Label smoothing (LS) [65] attempts to prevent the model to be over-confident by softening the ground-truth labels, which is a common trick in the field of deep learning and representation learning. Therefore, it is also interesting to investigate its effect on the problem of FSL. To be specific, we apply LS to three FSL methods, including ProtoNet, R2D2 and RFS-simple. In addition, for each method of using LS, we adopt two different hyper-parameters for LS, *i.e.*, 0.1 and 0.2, to balance the weight between the original ground-truth and the uniform distribution.

TABLE 5
Using Transformer in FSL on both *mini*ImageNet and *tiered*ImageNet. All model are trained from scratch without pre-training. The fourth column shows the total number of trainable parameters used by each model and the fifth column denotes the details of optimization during training.

Method	Embedding	Image Size	Parameters	Optimizer / Lr / Decay	*mini*ImageNet		*tiered*ImageNet	
					(5,1)	(5,5)	(5,1)	(5,5)
ProtoNet	ResNet12	84×84	12.42M	Adam/1e-3/Step	57.10	74.20	62.93	83.30
	Swin-T	84×84	26.59M	AdamW/5e-4/Cosine	57.23	74.67	62.33	81.83
DN4	ResNet12	84×84	12.42M	Adam/1e-3/Step	59.14	75.26	64.41	82.59
	Swin-T	84×84	26.59M	AdamW/1e-3/Cosine	54.29	67.46	64.94	79.36
R2D2	ResNet12	84×84	12.42M	Adam/1e-3/Step	59.52	74.61	65.07	83.04
	Swin-T	84×84	26.59M	AdamW/5e-5/Cosine	50.41	61.68	54.71	67.68
RFS	ResNet12	84×84	12.47M	Adam/1e-3/Step	61.65	78.88	70.55	84.74
	Swin-T	84×84	26.64M	AdamW/5e-4/Cosine	58.13	76.94	68.12	83.99

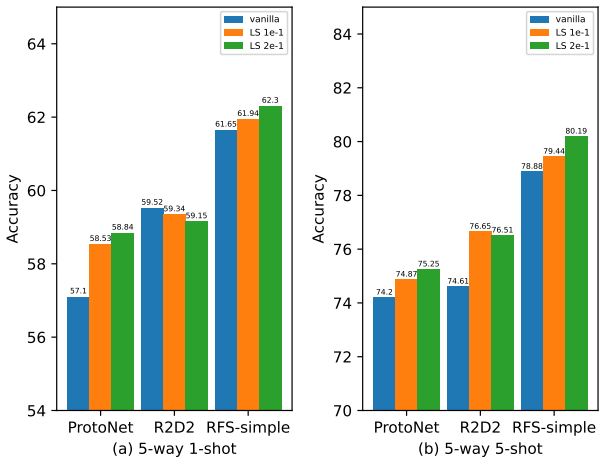


Fig. 6. **Effect of using label smoothing (LS)** for ProtoNet, R2D2 and RFS-simple on *mini*ImageNet with ResNet12.

The results are reported in Fig. 6. From the results, not surprisingly, we can see that in most cases, LS can further improve the performance of the vanilla versions of the three FSL methods, especially in the 5-shot setting. For example, in the 5-shot setting, LS (0.2) boosts the performance of ProtoNet, R2D2 and RFS-simple by 1.05%, 1.90%, 1.31% improvements, respectively. This reveals that label smoothing can also be a general trick for boosting the performance of FSL methods.

5 Using Transformer in FSL

In recent years, many new vision transformer based models have been proposed, such as ViT [66], DeiT [67], DETR [68], SETR [69], and Swin-Transformer [70], and shown promising results on a variety of vision tasks, compared with CNN architecture-based models. Therefore, it will be interesting to explore the effect of transformer in few-shot learning.

In particular, we implement four representative FSL methods, including ProtoNet, DN4, R2D2 and RFS, by using the tiny version of Swin Transformer (Swin-T for short) [70], a powerful transformer network, as the embedding backbone. Also, the versions using ResNet12 as the embedding backbone for these methods are taken as comparisons. Note that the common Swin-T is designed for the generic image classification with the image size of 224×224, while FSL normally uses an image size of 84×84. Therefore, for a fair comparison, we use an image size of 84×84 for Swin-T and change the down-sampling factors for each stage (linear embedding + transformer block) from $(4, 2, 2, 2)$ to $(3, 2, 2, 1)$ but keep all the other architecture hyper-parameters by default. In terms of data augmentation, RandomCrop, RandomHorizontalFlip, and ColorJitter, which are commonly used in few-shot learning, could not achieve good results when using Swin-T as the backbone. Therefore, we add RandAugment, mixup and cutmix into the data augmentation transformations for better results. In addition, we find that transformer is sensitive to the learning rate, so we follow the literature of generic classification, use AdamW as the optimizer, and carefully adjust each model's learning rate.

The results are reported in Table 5. From the results, we observe that: (1) In most cases, Swin-T shows worse results than ResNet12 on both *mini*ImageNet and *tiered*ImageNet. This is because transformers generally rely on a large-scale dataset for training to achieve excellent performance, because transformers lack locality and translation invariance properties that existed in the CNN architecture [71], [72]. (2) Swin-T can benefit from a large dataset, *i.e.*, *tiered*ImageNet. As seen, all methods especially DN4 and RFS, obtain much higher results on *tiered*ImageNet than the results on *mini*ImageNet under both 1-shot and 5-shot settings. (3) Interestingly, when using Swin-T, ProtoNet can achieve very competitive results with the results of using ResNet12. Also, DN4 with Swin-T achieves a higher result than DN4 with ResNet12 on *tiered*ImageNet under the 1-shot setting.

In summary, the above analyses reveal that transformers, as the latest popular and powerful architecture, are also promising and showing good potential in the field of few-shot learning. However, the key to achieving this expectation is how to effectively address the generic limitation, *i.e.*, the data-hungry property, of transformers.

TABLE 6
Cross-domain transferability. All methods are learned from the source domain (*e.g.*, *mini*ImageNet), and directly evaluated on the test set of the target domain (*i.e.*, Stanford Dogs, Stanford Cars and CUB Birds-200-2011) with a ResNet12 backbone.

Method	Type	Test-tune	Global-label	KD	SS	*mini*→Dogs		*mini*→Birds		*mini*→Cars		*tiered*→Dogs		*tiered*→Birds		*tiered*→Cars	
						1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
Baseline [27]	Non-episodic	✓	✓			51.21	69.54	47.81	68.79	33.07	**50.72**	76.82	92.18	68.32	87.34	34.88	50.72
Baseline++ [27]	Non-episodic	✓	✓			51.16	65.62	41.34	57.60	27.52	37.17	79.53	92.13	67.60	84.33	33.30	50.29
RFS-simple [30]	Non-episodic	✓	✓			56.15	71.90	47.88	65.53	32.18	44.80	81.46	92.85	70.96	85.89	34.17	48.58
RFS-distill [30]	Non-episodic	✓	✓	✓		**58.46**	**74.88**	50.12	69.19	33.20	47.83	81.44	93.01	**71.35**	**87.42**	**35.61**	**53.36**
Neg-Cosine [34]	Non-episodic	✓	✓			56.09	72.31	47.39	66.34	31.30	45.12	**81.87**	**93.25**	70.53	86.70	33.31	49.80
SKD-GEN0 [32]	Non-episodic	✓	✓		✓	56.29	74.51	**51.94**	72.71	33.04	48.66	73.55	89.18	70.32	87.06	35.03	50.87
SKD-GEN1 [32]	Non-episodic	✓	✓	✓	✓	57.38	75.20	52.57	**73.11**	**33.39**	49.30	74.28	71.24	71.24	86.96	34.81	50.22
Versa [39]	Meta	✓				42.35	57.83	40.98	58.10	25.98	31.63	51.28	76.17	49.71	66.85	25.09	35.38
R2D2 [26]	Meta	✓				52.16	68.72	44.87	62.47	29.37	45.42	65.58	87.62	59.81	82.29	30.99	50.58
LEO [40]	Meta	✓	✓			47.47	61.62	37.71	51.70	27.66	34.07	70.62	88.79	61.17	72.05	29.34	35.76
ANIL [41]	Meta	✓				31.41	50.31	35.16	49.78	27.56	32.94	43.08	69.22	41.67	63.98	28.78	38.67
BOIL [43]	Meta	✓				44.60	60.75	44.81	59.68	28.77	37.65	63.50	83.54	60.22	78.38	31.44	46.91
MTL [13]	Meta	✓	✓			**54.35**	**72.11**	**48.01**	**66.39**	**31.97**	**46.77**	**72.23**	**89.41**	**66.66**	**85.49**	**35.20**	**53.94**
ProtoNet [23]	Metric					40.62	65.80	44.17	**67.73**	27.82	41.53	57.21	86.70	56.15	82.47	29.93	48.51
RelationNet [4]	Metric					38.77	57.96	40.45	53.49	26.69	31.70	49.40	72.44	50.84	68.26	27.91	37.91
CovaMNet [14]	Metric					40.18	53.30	40.24	48.66	28.96	33.37	46.75	65.06	36.04	58.88	29.10	36.43
DN4 [15]	Metric					43.72	59.82	42.77	61.73	29.08	43.66	55.36	76.83	55.78	74.84	32.81	47.71
CAN [48]	Metric		✓			**56.03**	71.34	43.94	62.37	29.09	39.16	76.56	90.57	**69.74**	**85.95**	**32.99**	**48.73**
RENet [49]	Metric		✓			53.60	**71.44**	**48.69**	65.79	**31.09**	**44.45**	**77.67**	**90.68**	69.50	85.17	32.35	44.00

5.1 Cross-domain Transferability

Cross-domain few-shot tasks have been introduced in many FSL works in the literature. Therefore, it will also be interesting to further evaluate the cross-domain transfer ability of different FSL methods in LibFewShot, which are not specially designed for this purpose. To this end, following the literature [27], we conduct an experiment on six cross-domain scenarios, *e.g.*, *mini*ImageNet→Stanford Dogs (*mini*→Dogs for short). In this experiment, all the few-shot models are trained on the source domain, *e.g.*, *mini*ImageNet or *tiered*ImageNet, using a ResNet12 backbone with the same setting in Table 3 and directly tested on the target domain, *e.g.*, Standford Dogs or CUB Birds.

The results are reported in Table 6. From the results, we have the following observations: (1) The models trained on *mini*ImageNet and *tiered*ImageNet can easily generalize to Stanford Dogs and CUB Birds that enjoy a small domain-shift. However, their performance significantly drops when performing on the Stanford Cars with a large domain-shift. (2) In most cases, especially in the large domain-shift scenario, *i.e.*, *mini*→Cars, non-episodic based FSL methods perform somewhat better than metric-based or meta-based FSL methods. This means that pre-training is beneficial to cross-domain scenarios. Pre-training can easily bring prior strong representations which work well in natural image cross-domain scenarios. Especially, MTL, CAN and RENet use a pre-trained model or global labels, and they can also perform well on the *mini*→Cars task. (3) We can see that the classification accuracy on the cross-domain target datasets is significantly lower than that on the in-domain target datasets. This reveals that the current state-of-the-art FSL methods cannot handle the cross-domain scenarios well, which needs to be further investigated in the future.

6 Conclusions

In this paper, we present *a comprehensive library for few-shot learning* (LibFewShot) by re-implementing the state-of-the-art FSL methods in a unified framework. Through LibFewShot, first, we are able to make a relatively fair comparison between different methods to reflect the actual progress of FSL. Second, we emphasize and demonstrate the necessity of episodic- or meta-training. Third, we find that test-tuning is not very important at the test stage because of the limited-data setting in FSL, while a good embedding and ℓ_2 normalization are truly important. Finally, we verify that many deep learning tricks are indeed non-trivial but are universal for different FSL methods. Also, we show that transformers are promising for FSL but are still needed to be further investigated in the future. We hope our work will facilitate healthy research on few-shot learning.

Acknowledgments

This work is supported in part by the Science and Technology Innovation 2030 New Generation Artificial Intelligence Major Project (2021ZD0113303), National Natural Science Foundation of China (62106100, 62192783, 61806092), Jiangsu Natural Science Foundation (BK20221441), Collaborative Innovation Center of Novel Software Technology and Industrialization, and Jiangsu Provincial Double-Innovation Doctor Program (JSSCBS20210021).

REFERENCES

[1] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[2] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2016, pp. 3630–3638.

[3] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017, pp. 1126–1135.

[4] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 1199–1208.

[5] Q. Sun, Y. Liu, Z. Chen, T.-S. Chua, and B. Schiele, "Meta-transfer learning through hard tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[6] C. Simon, P. Koniusz, R. Nock, and M. Harandi, "Adaptive subspaces for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 4135–4144.

[7] W. Li, L. Wang, J. Huo, Y. Shi, Y. Gao, and J. Luo, "Asymmetric distribution measure for few-shot learning," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2020, pp. 2957–2963.

[8] H.-J. Ye, D.-C. Zhan, Y. Jiang, and Z.-H. Zhou, "Heterogeneous few-shot model rectification with semantic mapping," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[9] S. Yang, L. Liu, and M. Xu, "Free lunch for few-shot learning: Distribution calibration," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[10] M. Abbas, Q. Xiao, L. Chen, P.-Y. Chen, and T. Chen, "Sharp-maml: Sharpness-aware model-agnostic meta learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

[11] S. Gidaris and N. Komodakis, "Dynamic few-shot visual learning without forgetting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4367–4375.

[12] K. Lee, S. Maji, A. Ravichandran, and S. Soatto, "Meta-learning with differentiable convex optimization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 657–10 665.

[13] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele, "Meta-transfer learning for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 403–412.

[14] W. Li, J. Xu, J. Huo, L. Wang, Y. Gao, and J. Luo, "Distribution consistency based covariance metric networks for few-shot learning," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019, pp. 8642–8649.

[15] W. Li, L. Wang, J. Xu, J. Huo, Y. Gao, and J. Luo, "Revisiting local descriptor based image-to-class measure for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7260–7268.

[16] H.-J. Ye, H. Hu, D.-C. Zhan, and F. Sha, "Few-shot learning via embedding adaptation with set-to-set functions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 8808–8817.

[17] J. Xie, F. Long, J. Lv, Q. Wang, and P. Li, "Joint distribution matters: Deep brownian distance covariance for few-shot classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 7972–7981.

[18] A. Afrasiyabi, H. Larochelle, J.-F. Lalonde, and C. Gagné, "Matching feature sets for few-shot image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 9014–9024.

[19] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1345–1359, 2009.

[20] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 1842–1850.

[21] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[22] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Proceedings of the International Conference on Machine Learning (ICML) Deep Learning Workshop*, 2015.

[23] J. Snell, K. Swersky, R. Zemel, and R. Zemel, "Prototypical networks for few-shot learning," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2017, pp. 4077–4087.

[24] S. Thrun, "Lifelong learning algorithms," in *Learning to Learn*, 1998, pp. 181–209.

[25] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, pp. 77–95, 2002.

[26] L. Bertinetto, J. F. Henriques, P. H. S. Torr, and A. Vedaldi, "Meta-learning with differentiable closed-form solvers," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[27] W. Chen, Y. Liu, Z. Kira, Y. F. Wang, and J. Huang, "A closer look at few-shot classification," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[28] G. S. Dhillon, P. Chaudhari, A. Ravichandran, and S. Soatto, "A baseline for few-shot image classification," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[29] Y. Chen, X. Wang, Z. Liu, H. Xu, and T. Darrell, "A new meta-baseline for few-shot learning," *arXiv preprint arXiv:2003.04390*, 2020.

[30] Y. Tian, Y. Wang, D. Krishnan, J. B. Tenenbaum, and P. Isola, "Rethinking few-shot image classification: A good embedding is all you need?" in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020, pp. 266–282.

[31] S. Thrun and L. Pratt, "Learning to learn: Introduction and overview," in *Learning to Learn*, 1998, pp. 3–17.

[32] J. Rajasegaran, S. Khan, M. Hayat, F. S. Khan, and M. Shah, "Self-supervised knowledge distillation for few-shot learning," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2021.

[33] P. Mangla, M. Singh, A. Sinha, N. Kumari, V. N. Balasubramanian, and B. Krishnamurthy, "Charting the right manifold: Manifold mixup for few-shot learning," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020, pp. 2207–2216.

[34] B. Liu, Y. Cao, Y. Lin, Q. Li, Z. Zhang, M. Long, and H. Hu, "Negative margin matters: Understanding margin in few-shot classification," in *Proceedings of the European Conference on Computer Vision (ECCV)*, vol. 12349, 2020, pp. 438–455.

[35] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[36] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS) Deep Learning and Representation Learning Workshop*, 2015.

[37] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 97, 2019, pp. 6438–6447.

[38] A. Dosovitskiy, J. T. Springenberg, M. A. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with convolutional neural networks," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2014, pp. 766–774.

[39] J. Gordon, J. Bronskill, M. Bauer, S. Nowozin, and R. E. Turner, "Versa: Versatile and efficient few-shot learning," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2018, pp. 1–9.

[40] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, "Meta-learning with latent embedding optimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[41] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals, "Rapid learning or feature reuse? towards understanding the effectiveness of maml," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[42] W. Xu, H. Wang, Z. Tu *et al.*, "Attentional constellation nets for few-shot learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[43] J. Oh, H. Yoo, C. Kim, and S. Yun, "BOIL: towards representation change for few-shot learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[44] C. Doersch, A. Gupta, and A. Zisserman, "Crosstransformers: Spatially-aware few-shot transfer," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
[45] C. Zhang, Y. Cai, G. Lin, and C. Shen, "Deepemd: Few-shot image classification with differentiable earth mover's distance and structured classifiers," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 12 203–12 213.
[46] D. Wertheimer, L. Tang, and B. Hariharan, "Few-shot classification with feature map reconstruction networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 8012–8021.
[47] D. Kang, H. Kwon, J. Min, and M. Cho, "Relational embedding for few-shot classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.
[48] R. Hou, H. Chang, B. Ma, S. Shan, and X. Chen, "Cross attention network for few-shot classification," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2019.
[49] D. Kang, H. Kwon, J. Min, and M. Cho, "Relational embedding for few-shot classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.
[50] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel, "Meta-learning for semi-supervised few-shot classification," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
[51] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, "Novel dataset for fine-grained image categorization: Stanford dogs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshop*, 2011, p. 1.
[52] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshop*, 2013, pp. 554–561.
[53] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," 2011.
[54] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
[55] G. Ghiasi, T. Lin, and Q. V. Le, "Dropblock: A regularization method for convolutional networks," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2018, pp. 10 750–10 760.
[56] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 558–567.
[57] T. Pang, X. Yang, Y. Dong, H. Su, and J. Zhu, "Bag of tricks for adversarial training," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
[58] S. Gidaris, A. Bursuc, N. Komodakis, P. Pérez, and M. Cord, "Boosting few-shot visual learning with self-supervision," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 8059–8068.
[59] J. Su, S. Maji, and B. Hariharan, "When does self-supervision improve few-shot learning?" in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020, pp. 645–666.
[60] R. Xu, G. Li, J. Yang, and L. Lin, "Larger norm more transferable: An adaptive feature norm approach for unsupervised domain adaptation," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 1426–1435.
[61] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
[62] S. Yun, D. Han, S. Chun, S. J. Oh, Y. Yoo, and J. Choe, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 6022–6031.
[63] S. Gidaris, A. Bursuc, N. Komodakis, P. Pérez, and M. Cord, "Boosting few-shot visual learning with self-supervision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 8058–8067.
[64] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
[65] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.
[66] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
[67] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 139, 2021, pp. 10 347–10 357.
[68] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: deformable transformers for end-to-end object detection," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
[69] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. H. S. Torr, and L. Zhang, "Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 6881–6890.
[70] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9992–10 002.
[71] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, "Cvt: Introducing convolutions to vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 22–31.
[72] Y.-H. Cao, H. Yu, and J. Wu, "Training vision transformers with only 2040 images," *arXiv preprint arXiv:2201.10728*, 2022.
[73] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation policies from data," *arXiv preprint arXiv:1805.09501*, 2019.
[74] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.
[75] E. D. Cubuk, B. Zoph, J. Shlens, and Q. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

APPENDIX A
ARCHITECTURE

LibFewShot is built on PyTorch 1.5.0, and its architecture can be seen in Fig. 7. Because of the great differences between different FSL methods in terms of the network architecture, loss function and optimizer, it is difficult to directly integrate the existing FSL methods into the same framework. To address this issue, we disassemble each FSL method into multiple small common modules, aiming to integrate them into the same framework in a more flexible way. The details will be described in the following sections.

A.1 Model

The **Model** module belonging to the **Trainer** module is a key part of the whole framework, because all the network architectures of the FSL methods are implemented within this module. Specifically, **Model** consists of *Backbone*, *Classifier*, *Local Optimizer*, *Metric Function* and *Loss Function*. Also, we will briefly describe some of these core parts.

Backbone. The embedding backbone plays an important role in the field of deep learning. In order to support different requirements, LibFewShot provides options of the commonly used embedding modules, *e.g.*, Conv64F, ResNet12, ResNet18, Wide ResNet (WRN) and Vision Transformer (ViT). Moreover, because some methods may need to modify the backbones in some cases, such as feature flattening, global average pooling and using multi-level

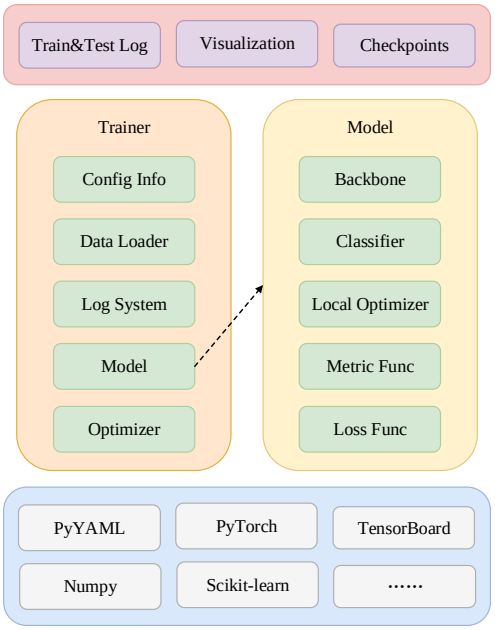


Fig. 7. Architecture of the proposed LibFewShot built on PyTorch.

features, LibFewShot can conveniently meet such kinds of requirements by simply modifying the configuration files.

Classifier. Despite the backbone part, the classifier may be the soul of one FSL method. However, we find that some FSL methods have inconsistent operations in the training phase and evaluation phase. To overcome this issue, we implement two functions in the *Classifier* module, *i.e.*, *set_forward_loss function* and *set_forward function*, which can be flexibly used for the training mode and evaluation mode, respectively. As mentioned in the main paper, we divide the FSL methods into three categories, *i.e.*, *non-episodic based methods*, *meta-learning based methods* and *metric-learning based methods*. To avoid the duplication of work, we provide a category-dependent function for each category, and they all inherit the same *abstract function*, in which the commonly used and model-agnostic hyper-parameters can be defined.

In addition, we notice that the official implementations of some FSL methods can only support single-task episodic training (*i.e.*, one task in each mini-batch), which may make the FSL models be sensitive to the hyper-parameters and initializations. In contrast, some other FSL methods have already supported multi-task episodic training (*i.e.*, multiple tasks in each mini-batch). To make a more fair comparison, we re-implement the architectures of the classifiers of the methods that can only support single-task episodic training to support multi-task episodic training. In this sense, users can realize this operation by simply modifying the parameter of *episode size* in the configuration file.

A.2 Dataloader

LibFewShot provides a special dataloader, which can fulfill the requirements in few-shot learning. Specifically, LibFewShot assumes that all datasets have a similar file structure. It means that each dataset should have an image folder containing all the images and three csv files (train, validation and test) to indicate the image path and its corresponding class label. Moreover, LibFewShot will provide the processed datasets or the corresponding conversion program, depending on the open source protocol of these datasets.

In FSL, the input data structure is generally different from that of the generic computer vision tasks. In another word, in FSL, the smallest data unit is not an image but a task, which contains $5 \times (5 + 15) = 100$ images in a 5-way 5-shot setting when there are 15 query images per class. For this reason, many open source codes for FSL only sample a whole task per thread, which severely limits the efficiency of data loading. Differently, LibFewShot deigns a *Categories-Sampler* to sample a task by sampling one image per thread. In this way, data loading will not be a bottleneck anymore, even though under the condition of a large number of batch (task) size, *i.e.*, multi-task episodic training.

Data augmentation is a nontrivial technique to boost the classification performance in FSL. Most of the existing FSL methods adopt the same data augmentation for both support set and query set at the training stage, but do not apply data augmentation at the test stage. However, recently, some works [30], [32] have introduced data augmentation into the test stage for the support set, which shows the effectiveness of such an operation. Therefore, both kinds of data augmentation strategies are supported in LibFewShot (*i.e.*, *collate function*).

Note that, *collate function* in LibFewShot can apply the same or different transformations on the support set and query set separately. In addition, LibFewShot also provides some latest data augmentation methods, such as AutoAugment [73], Cutout [74] and RandomAugment [75], which are not officially provided by PyTorch but are still useful, and allows users to conveniently define their own data transformation list.

A.3 Trainer and Tester

Trainer is the core of LibFewShot, and *Tester* is an enhanced test version of *Trainer*. In the training phase, *Trainer* prepares the training environment by using the configuration information. According to the configuration information, *Trainer* initializes the network parameters, creates the optimizer and assigns the GPU and so on. After that, it calls the training, evaluation, and test functions in a loop until the training is completed. The training information, *e.g.* the configuration information, training log, and checkpoints, is also dumped into the disk. In the test phase, *Tester* does similar things, but only calls the test function to calculate the final evaluation criteria.

A.4 Configs

LibFewShot obtains the configuration information from the YAML file, in which the network structure, episode size, data root, and training epochs are determined. In order to avoid missing some important parameters, we set a default configuration file, and the framework will read this file first. In addition, our framework can also support a user-defined configuration file, which will replace the same parameters in the default configuration file. If some parameters are not defined in the users' configuration file, the framework will use the default profile settings for training.

TABLE 7
Supported functions of LibFewShot compared with other official FSL codes. Methods using for-loops multi-episodes are marked with †, which can not use the characteristic of GPU paralleling and will lead to slower computation.

	Multi-episodes		Multi-GPUs		Different-ways & shots		Different Data Augmentations	
Method	Official	LibFewShot	Official	LibFewShot	Official	LibFewShot	Official	LibFewShot
Baseline				✓				✓
Baseline++				✓				✓
RFS			✓	✓			✓	✓
SKD			✓	✓			✓	✓
MAML	†	†		✓				✓
Versa	✓	✓		✓		✓		✓
R2D2	✓	✓		✓	✓	✓		✓
LEO		✓		✓				✓
MTL		†		✓				✓
ANIL		†		✓				✓
ProtoNet		✓		✓	✓	✓		✓
RelationNet		✓		✓		✓		✓
CovaMNet	†	✓	✓	✓	✓	✓		✓
DN4	†	✓	✓	✓		✓		✓
CAN		✓		✓		✓		✓

A.5 How to Run the LibFewShot?

The whole program can be stated by the *run_trainer* and *run_tester* scripts. When users have implemented their own methods and the corresponding configuration files, or just use our re-implemented methods and configuration files, they only need to modify the configuration file's path in the *run_trainer* and then run it. The *run_trainer* script will parse the configuration file first and overwrite some options in the default configuration, and then pass the configurations to *Trainer* to start the training stage. When the training stage is finished, the users can modify the checkpoints' path in the *run_tester* and overwrite some options to run a test. *Tester* will automatically use the configuration file in the checkpoints directory to set up a network. Note that, the parameters at the test stage can also be overwritten by a manually defined parameter list.

A.6 Other Supported Functions

Based on the above designs, LibFewShot can already support multiple advanced functions, including *multi-episodes*, *multi-GPUs*, *different-ways & shots* and *different data augmentations* for all re-implemented FSL methods. Multi-episodes and multi-GPUs mean that LibFewShot supports multi-task episodic training and multi-GPUs training for each method, respectively. Different-ways & shots indicate that LibFewShot supports different numbers of ways and shots in the training and evaluation phases. Different data augmentations mean that LibFewShot can support using more flexible data augmentation for the support set and query set. An overview of the comparison between LibFewShot and other FSL methods is shown in Table 7.

APPENDIX B
MULTI-GPUS

LibFewShot adopts DataParallel provided in PyTorch to provide the multi-GPUs processing ability. Moreover, LibFewShot not only supports the backbones to train in parallel, but also enables the classifiers to be processed in parallel. Notably, all the re-implemented FSL methods in LibFewShot can be parallelized.

TABLE 8
Average train/interface time for each task and memory for each GPU, respectively, when using DN4 in LibFewShot.

GPUs	1	2	4	8
Train	16.07ms	12.17ms	8.48ms	6.84ms
Inference	12.04ms	6.57ms	4.27ms	3.04ms
Memory$_{\text{mean}}$	15263MB	15425MB	15430MB	15691MB
Memory$_{\text{min}}$	15263MB	15207MB	15425MB	15407MB
Memory$_{\text{max}}$	15263MB	15633MB	15957MB	17355MB

In order to measure the efficiency of multi-GPUs training in LibFewShot, we randomly sample images from *mini*ImageNet and use these images to construct $160,000$ 5-Way 1-Shot tasks. Specifically, DN4 is selected to process these tasks, and we calculate the training/interface time and memory distributions. For fairness, we use different episode sizes when using different numbers of GPUs to make sure 16 tasks are in 1 GPU. The time and memory consumed by different GPUs during training are shown in Table 8.

As seen, when the number of GPUs increases, the average memory occupied by each task will also increase. This is because the communication between GPUs will also increase the time cost during the multi-GPUs training. However, correspondingly, because more GPUs can be used to train more tasks at the same time, the training speed will be significantly improved. When 8 GPUs are used, and each GPU has 16 few-shot tasks, the training speed can reach more than 2 times faster than only using 1 GPU.