Few-shot Learning; Self-supervised Learning

S. Gidaris et al., 2018, "Dynamic few-shot visual learning without forgetting"

This paper proposes a few-shot object recognition system that is capable of *dynamically learning novel categories* from only a few training data while at the same time does *not forget the base categories* on which it was trained.

To achieve that, the authors introduced the following two:
 1) Classifier of a ConvNet as a cosine similarity function between feature representations and classification vectors
2) Attention-based few-shot classification weight generator.

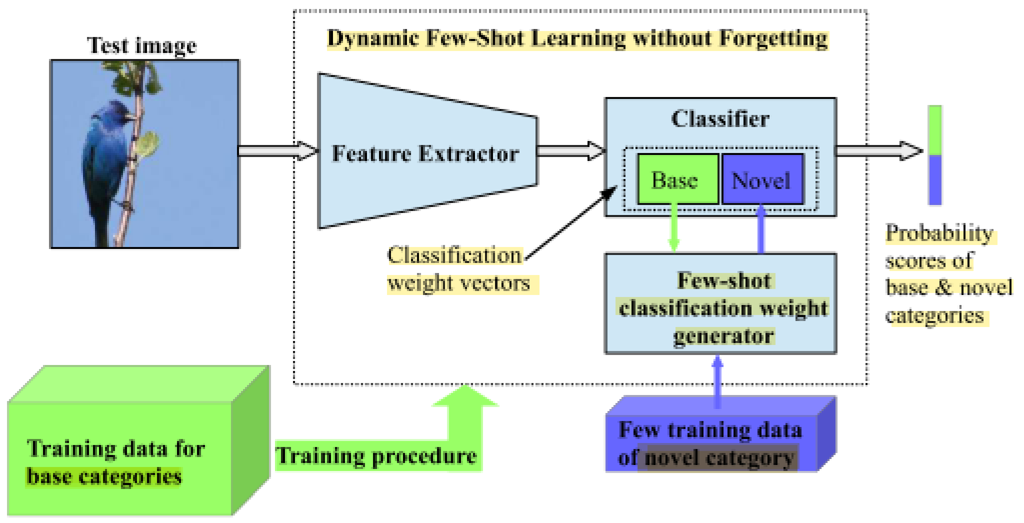

Test image
Dynamic Few-Shot Learning without Forgetting
Feature Extractor
Classifier
Base
Novel
Classification weight vectors
Few-shot classification weight generator
Probability scores of base & novel categories
Training data for base categories
Training procedure
Few training data of novel category

Figure 1: Overview of our system. It consists of: (a) a *ConvNet based recognition model* (that includes a feature extractor and a classifier) and (b) a *few-shot classification weight generator*. Both are trained on a set of base categories for which we have available a large set of training data. During test time, the weight generator gets as input a few training data of a novel category and the classification weight vectors of base categories (green rectangle inside the classifier box) and generates a classification weight vector for this novel category (blue rectangle inside the classifier box). This allows the ConvNet to recognize both base and novel categories.

Cosine-similarity based ConvNet recognition model

The standard setting for classification neural networks is, after having extracted the feature vector z

, to estimate the classification probability vector $p=C(z|W_*)$ by first computing the raw classification score s_k of each category $k\in[1,K_*]$ using the dot-product operator $s_k=z^T w_{*k}$ where w_k is the k-th classification weight vector in W_*, and then applying the softmax operator across all the K_* classification scores, i.e., $p_k=\mathrm{softmax}(s_j)$

.

In this paper, the classification weight vector w_{*k}

could come both from the base categories, i.e., $w_{*k}\in W_{base}$, and the novel categories, i.e., $w_{*k}\in W_{novel}$. However, the mechanisms involved during learning those classification weights are very different. Due to those differences, the weight values in those two cases can be completely different, and so the same applies to the raw classification scores computed with the dot-product operation, which can thus have totally different magnitudes depending on whether they come from the base of novel categories. To over this issue, the authors propose to modify the classifier $C(.|W_*)$

and compute the raw classification scores using the cosine similarity operator:

$s_k=\tau\cdot\cos(z,w_{*k})=\tau\cdot\bar{z}^T\bar{w}_{*k}$

where $\bar{z}=\frac{z}{||z||}$

and $\bar{w}_{*k}=\frac{w_{*k}}{||w_{*k}||}$ are the l_2-normalized vectors, and τ is a learnable scalar value. Since the cosine similarity can be implemented by first l_2-normalizing the feature vector z and the classification weight vector w_{*k}

and then applying the dot-product operator, the absolute magnitudes of the classification weight vectors can no longer affect the value of the raw classification score.

In addition to the above modification, the authors also choose to remove the ReLU non-linearity after the last hidden layer of the feature extractor, which allows the feature vector z

to take both positive and negative values.

The results from the paper show that the cosine-similarity-based classifier leads the feature extractor to learn features that generalize significantly better on novel categories than features learned with the dot-product-based classifier. A possible explanation for this is that, in order to minimize the classification loss of a cosine-similarity-based ConvNet model, the l_2

-normalized feature vector of an image must be very closely matched with the l_2-normalized classification weight vector of its ground truth category. As a consequence, the feature extractor is forced to (a) learn to encode those discriminative visual cues on its feature activations which the classification weight vectors of the ground truth categories learn to look for, and (b) learn to generate l_2

-normalized feature vectors with low intra-class variance, since all the feature vectors that belong to the same category must be very closely matched with the single classification weight vector of that category. Moreover, the cosine-similarity-based classification objective resembles the training objectives typically used by metric-learning approaches.

Few-shot classification-weight generator based on attention

The few-shot classification weight generator $G(.,.|\phi)$

receives the feature vectors $Z'=\{z'_i\}^{N'}_{i=1}$ of the N' training examples of a novel category and (optionally) the classification weight vectors of the base categories W_{base} as its input. Based on them, it infers a classification weight vector $w'=G(Z',W_{base}|\phi)$

for that novel category. Here, we explain how the above few-shot classification weight generator is constructed:

Feature averaging-based weight inference

An obvious choice to infer the classification weight vector w'

is by averaging the feature vectors of the training examples (after they've been l_2-normalized): $w'_{avg}=\frac{1}{N'}\sum^{N'}_{i=1}\bar{z}'_i$. The final classification weight vector if we only use the feature averaging mechanism is: $w'=\phi_{avg}\odot w'_{avg}$ where $\phi_{avg}\in R^d$

is a learnable weight vector. A similar strategy has been previously proposed by the *Prototypical Network*. However, it does not fully exploit the learned knowledge and the averaging cannot infer an accurate classification weight vector when there is only a single raining example for the novel category.

Attention-based weight inference

The authors enhance the above feature averaging mechanism with an attention-based mechanism that composes novel classification weight vectors by "looking" at a memory that contains the base classification weight vectors $W_{base}=\{w_b\}^{K_{base}}_{b=1}$

. More specifically, an extra attention-based classification weight vector w'_{att}

is computed as:

$$w'_{att}=\frac{1}{N'}\sum^{N'}_{i=1}\sum^{K_{base}}_{b=1}Att(\phi_q\bar{z}'_i,k_b)\cdot\bar{w}_b$$

where $\phi_q\in R^{d\times d}$

is a learnable weight matrix that transforms the feature vector $\bar{z}'_i$ to query vector used for querying the memory, $\{k_b\in R^d\}^{K_{base}}_{b}$ is a set of K_{base} learnable keys used for indexing the memory, and $Att(.,.)$ is an attention kernel implemented as a cosine similarity function followed by a softmax operation over the K_{base} base categories, and N' denotes a set of novel categories (typically, $N' <= 5$

). Note that the classification weight vector of a novel category can be composed as a linear combination of those base classification weight vectors that are most similar to the few training examples of that category. This allows our few-shot weight generator to explicitly explot the acquired knowledge.

The final classification weight vector is computed as: $w' = \phi_{avg} \odot w'_{avg} + \phi_{att} \odot w'_{att}$

where $\phi_{avg}, \phi_{att} \in R^d$

are learnable weight vectors.

H. Qi et al., 2017, "Low-shot learning with imprinted weights"

Weight imprinting is proposed in this paper. It directly sets the final layer weights for new classes from the embedding of training exemplars. Consider a single raining sample x_+

from a novel class, our method computes the embedding $\phi(x_+)$ and uses it to set a new column in the weight matrix for the new class, i.e., $w_+ = \phi(x_+)$

:

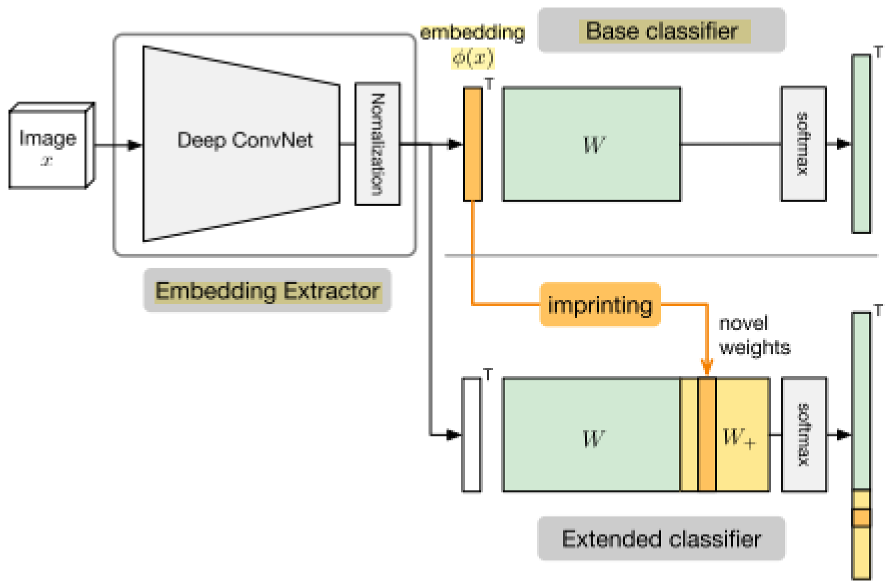


Figure 1. The overall architecture of imprinting. After a base classifier is trained, the embedding vectors of new low-shot examples are used to imprint weights for new classes in the extended classifier.

'T' denotes the transpose function.

Model Architecture

Different from standard ConvNet classifier architectures, the authors add an L_2

normalization layer at the end of the embedding extractor so that the output embedding has unit length, i.e. $||\phi(x)||_2=1$

.

Average embedding

If $n>1$

examples $\{x_{(i)+}\}_{n i=1}$ are available for a new class, we compute new weights by averaging the normalized embedding $\sim w_+ = {}_1 n \sum_{n i=1} \phi(x_{(i)+})$ and re-normalizing the resulting vector to unit length $w_+ = {}_{\sim w_+ || \sim w_+ ||}$

. In practice, the averaging operation can also be applied to the embedding computed from the randomly augmented versions of the original low-shot training examples.

W. Chen et al., 2020, "A closer look at few-shot classification"

This paper presents 1) a consistent comparative analysis of several representative few-shot classification algorithms, 2) a modified baseline method: Baseline++, which is like a cosine-similarity classifier, 3) a new experimental setting for evaluating the cross-domain generalization ability for few-shot classification algorithms.

One of the best findings is Baseline++, which is equal to the cosine-similarity classifier. Although it's a relatively much simpler model than the prior SOTAs such as MatchingNet, ProtoNet, MAML, and RelationNet, its performances are very competitive.

This paper also explains the overview of few-shot classification algorithms:

Overview of Few-shot Classification Algorithms

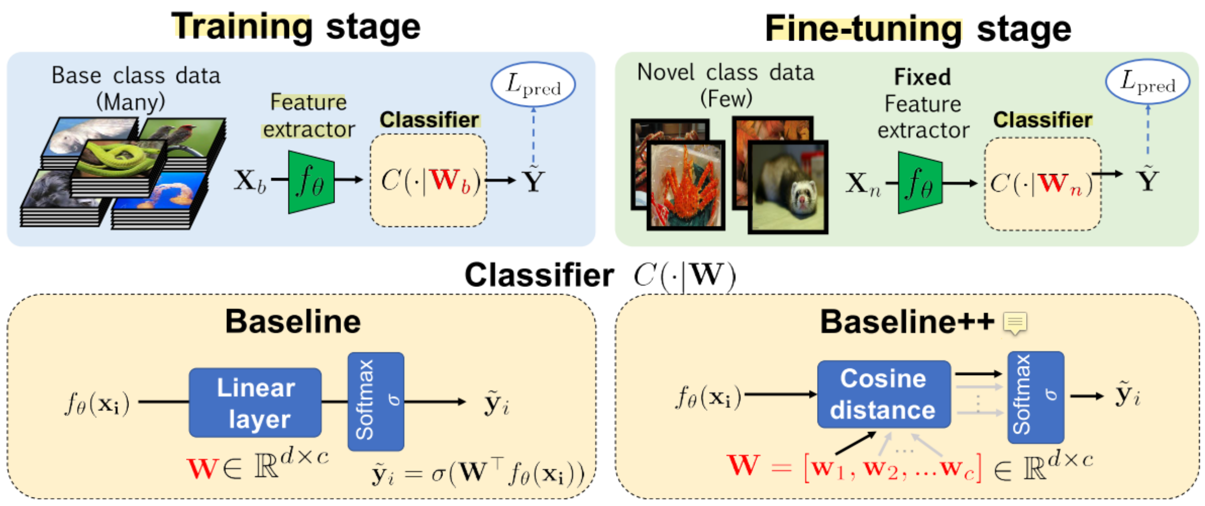


Figure 1: **Baseline and Baseline++ few-shot classification methods.** Both the baseline and baseline++ method train a feature extractor f_θ and classifier $C(.|\mathbf{W}_b)$ with base class data in the training stage In the fine-tuning stage, we fix the network parameters θ in the feature extractor f_θ and train a new classifier $C(.|\mathbf{W}_n)$ with the given labeled examples in novel classes. The baseline++ method differs from the baseline model in the use of cosine distances between the input feature and the weight vector for each class that aims to reduce intra-class variations.

Baseline

Training stage: We train a feature extractor f_θ

(parametrized by the network parameters θ) and the classifier $C(\cdot|W_b)$ (parametrized by the weight matrix $W_b \in R_{d\times c}$) from scratch by minimizing a standard cross-entropy classification loss L_{pred} using the training examples in the base classes $x_i \in W_b$. Here, we denote the dimension of the encoded feature as d and the number of output classes as c. The classifier $C(\cdot|W_b)$ consists of a linear layer $W_{Tb} f_\theta(x_i)$ followed by a softmax function σ

.

Fine-tuning stage: To adapt the model to recognize novel classes in the fine-tuning stage, we fix the pre-trained network parameters θ

in our feature extractor f_θ and train a new classifier $C(\cdot|W_n)$ (parametrized by the weight matrix W_n) by minimizing L_{pred} using the few labeled of examples (i.e., the support set) in the novel classes X_n

.

Baseline++

The baseline++ explicitly reduces intra-class variation among features during training.

The training procedure of Baseline++ is the same as the original Baseline model except for the classifier design. As shown in Figure 1, we still have a weight matrix $\mathbf{W}_b \in \mathbb{R}^{d\times c}$ of the classifier in the training stage and a $\mathbf{W}_n$ in the fine-tuning stage in Baseline++. The classifier design, however, is different from the linear classifier used in the Baseline. Take the weight matrix $\mathbf{W}_b$ as an example. We can write the weight matrix $\mathbf{W}_b$ as $[\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_c]$, where each class has a d-dimensional weight vector. In the training stage, for an input feature $f_\theta(\mathbf{x}_i)$ where $\mathbf{x}_i \in \mathbf{X}_b$, we compute its cosine similarity to each weight vector $[\mathbf{w}_1, \cdots, \mathbf{w}_c]$ and obtain the similarity scores $[s_{i,1}, s_{i,2}, \cdots, s_{i,c}]$ for all classes, where $s_{i,j} = f_\theta(\mathbf{x}_i)^\top \mathbf{w}_j / \|f_\theta(\mathbf{x}_i)\| \|\mathbf{w}_j\|$. We can then obtain the prediction probability for each class by normalizing these similarity scores with a softmax function. Here, the classifier makes a prediction based on the cosine distance between the input feature and the learned weight vectors representing each class. Consequently, training the model with this distance-based classifier explicitly reduce intra-class variations. Intuitively, the learned weight vectors $[\mathbf{w}_1, \cdots, \mathbf{w}_c]$ can be interpreted as prototypes (similar to Snell et al. (2017); Vinyals et al. (2016)) for each class and the classification is based on the distance of the input feature to these learned prototypes. The softmax function prevents the learned weight vectors collapsing to zeros.

Meta-learning Algorithms

Relation between few-shot learning and meta-learning: To do the few-shot classification (learning), the meta-learning algorithm is used. Hence, the few-shot learning is a higher-level concept.

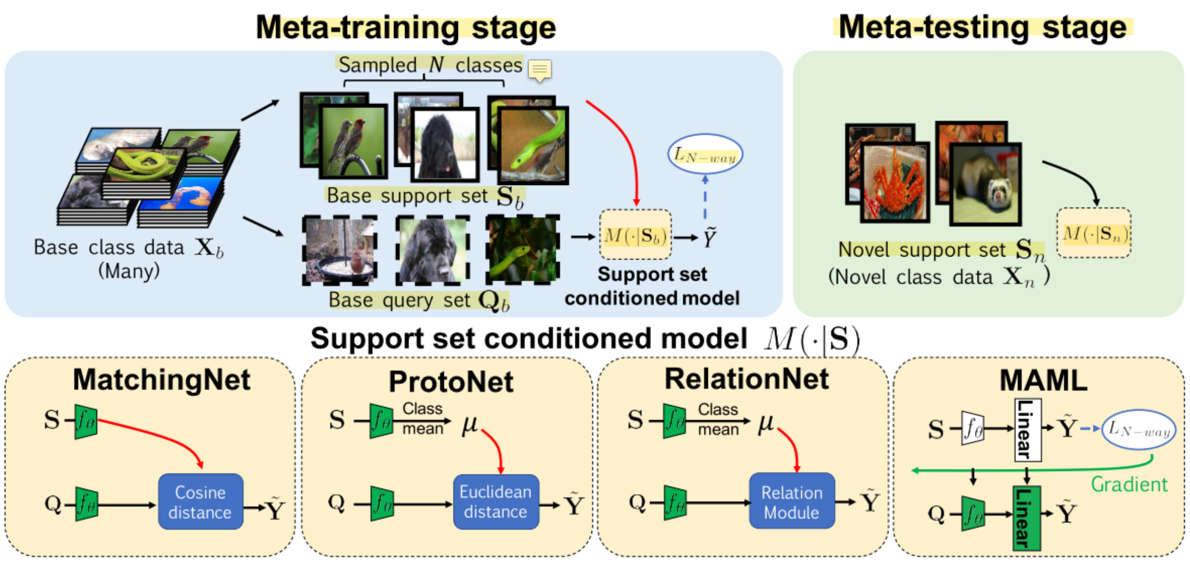
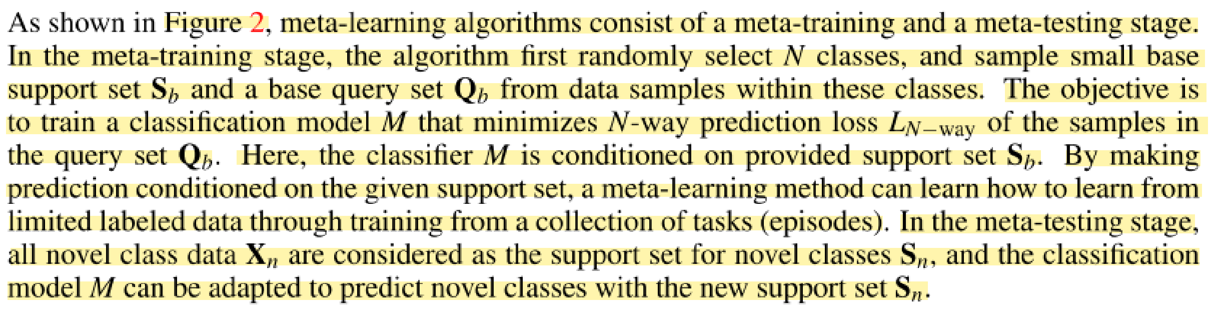


Figure 2: **Meta-learning few-shot classification algorithms.** The meta-learning classifier $M(\cdot|\mathbf{S})$ is conditioned on the support set $\mathbf{S}$. (*Top*) In the meta-train stage, the support set $\mathbf{S}_b$ and the query set $\mathbf{Q}_b$ are first sampled from random N classes, and then train the parameters in $M(.|\mathbf{S}_b)$ to minimize the N-way prediction loss $L_{\text{N-way}}$. In the meta-testing stage, the adapted classifier $M(.|\mathbf{S}_n)$ can predict novel classes with the support set in the novel classes $\mathbf{S}_n$. (*Bottom*) The design of $M(\cdot|\mathbf{S})$ in different meta-learning algorithms.

In this paper, MatchingNet, ProtoNet, RelationNet, and MAML are considered

The meta-leanring process shown in the above figure is explained below:


As shown in Figure 2, meta-learning algorithms consist of a meta-training and a meta-testing stage. In the meta-training stage, the algorithm first randomly select N classes, and sample small base support set $\mathbf{S}_b$ and a base query set $\mathbf{Q}_b$ from data samples within these classes. The objective is to train a classification model M that minimizes N-way prediction loss $L_{N-\text{way}}$ of the samples in the query set $\mathbf{Q}_b$. Here, the classifier M is conditioned on provided support set $\mathbf{S}_b$. By making prediction conditioned on the given support set, a meta-learning method can learn how to learn from limited labeled data through training from a collection of tasks (episodes). In the meta-testing stage, all novel class data $\mathbf{X}_n$ are considered as the support set for novel classes $\mathbf{S}_n$, and the classification model M can be adapted to predict novel classes with the new support set $\mathbf{S}_n$.

Experimental Results

Experimental Setup

Scenarios (such as classification and cross-domain scenario) and datasets are determined.

Results

Table 2: **Few-shot classification results for both the *mini*-ImageNet and *CUB* datasets.** The Baseline++ consistently improves the Baseline model by a large margin and is competitive with the state-of-the-art meta-learning methods. All experiments are from 5-way classification with a Conv-4 backbone and data augmentation.

	CUB		*mini*-ImageNet	
Method	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Baseline	47.12 ± 0.74	64.16 ± 0.71	42.11 ± 0.71	62.53 ±0.69
Baseline++	60.53 ± 0.83	79.34 ± 0.61	48.24 ± 0.75	66.43 ±0.63
MatchingNet Vinyals et al. (2016)	60.52 ± 0.88	75.29 ± 0.75	48.14 ± 0.78	63.48 ±0.66
ProtoNet Snell et al. (2017)	50.46 ± 0.88	76.39 ± 0.64	44.42 ± 0.84	64.24 ±0.72
MAML Finn et al. (2017)	54.73 ± 0.97	75.75 ± 0.76	46.47 ± 0.82	62.71 ±0.71
RelationNet Sung et al. (2018)	62.34 ± 0.94	77.84 ± 0.68	49.31 ± 0.85	66.60 ±0.69

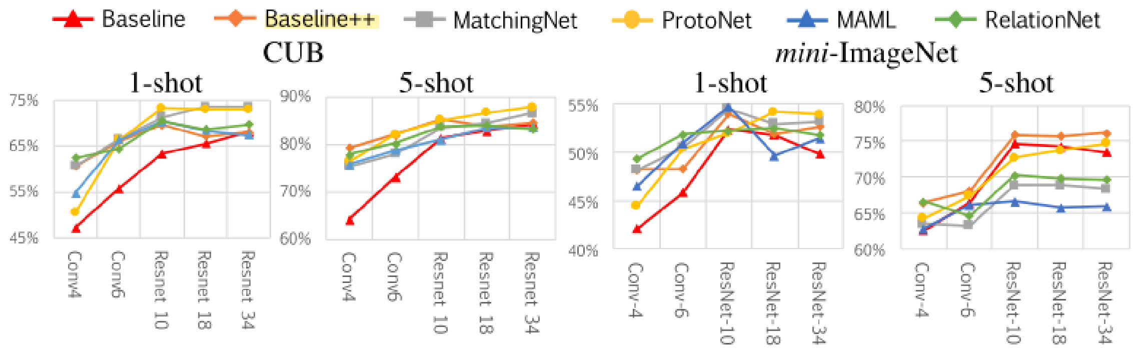


Figure 3: **Few-shot classification accuracy vs. backbone depth**. In the CUB dataset, gaps among different methods diminish as the backbone gets deeper. In *mini*-ImageNet 5-shot, some meta-learning methods are even beaten by Baseline with a deeper backbone. (Please refer to Figure A3 and Table A5 for larger figure and detailed statistics.)

	mini-ImageNet →CUB
Baseline	**65.57±0.70**
Baseline++	62.04±0.76
MatchingNet	53.07±0.74
ProtoNet	62.02±0.70
MAML	51.34±0.72
RelationNet	57.71±0.73

Table 3: **5-shot accuracy under the cross-domain scenario with a ResNet-18 backbone.** Baseline outperforms all other methods under this scenario.

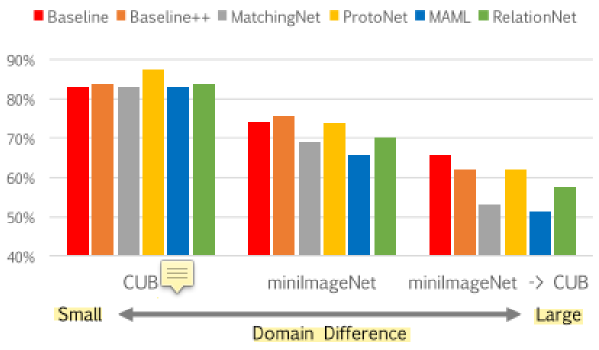


Figure 4: **5-shot accuracy in different scenarios with a ResNet-18 backbone.** The Baseline model performs relative well with larger domain differences.

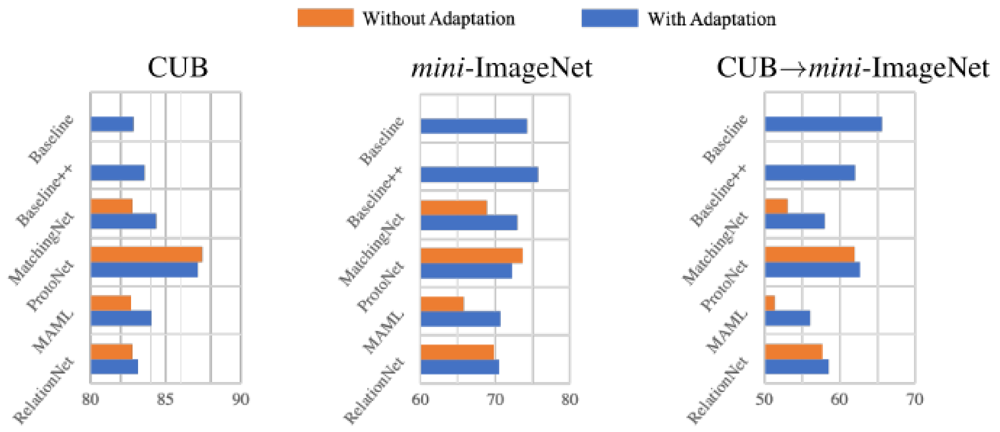


Figure 5: **Meta-learning methods with further adaptation steps.** Further adaptation improves MatchingNet and MAML, but has less improvement to RelationNet, and could instead harm ProtoNet under the scenarios with little domain differences.All statistics are for 5-shot accuracy with ResNet-18 backbone. Note that different methods use different further adaptation strategies.