Confusable Learning for Large-class Few-Shot Classification[⋆]

Bingcong Li[1,2], Bo Han[2], Zhuowei Wang[3], Jing Jiang[3], and Guodong Long[3]

[1] School of Automation, Guangdong University of Technology, China
[2] Department of Computer Science, Hong Kong Baptist University, HKSAR, China
[3] Australian Artificial Intelligence Institute, University of Technology Sydney, Australia

Abstract. Few-shot image classification is challenging due to the lack of ample samples in each class. Such a challenge becomes even tougher when the number of classes is very large, i.e., the large-class few-shot scenario. In this novel scenario, existing approaches do not perform well because they ignore confusable classes, namely similar classes that are difficult to distinguish from each other. These classes carry more information. In this paper, we propose a biased learning paradigm called *Confusable Learning*, which focuses more on confusable classes. Our method can be applied to mainstream meta-learning algorithms. Specifically, our method maintains a dynamically updating confusion matrix, which analyzes confusable classes in the dataset. Such a confusion matrix helps meta learners to emphasize on confusable classes. Comprehensive experiments on *Omniglot*, *Fungi*, and *ImageNet* demonstrate the efficacy of our method over state-of-the-art baselines.

Keywords: Large-Class Few-Shot Classification · Meta-Learning · Confusion Matrix.

1 Introduction

Deep Learning has made significant progress in many areas recently, but it relies on numerous labeled instances. Without enough labeled instances, deep models usually suffer from severe over-fitting, while a human can easily learn patterns from a few instances. By incorporating this ability, meta-learning based few-shot learning has become a hot topic [1, 2]. Mainstream meta-learning methods obtain meta-knowledge from a base dataset containing a large number of labeled instances, and employ meta-knowledge to classify an meta-testing dataset.

Recent progress in few-shot learning focuses on small-class few-shot scenario [3]. These methods consist of three directions: model initialization based methods, metric learning methods and data augmentation. In particular, model

[⋆] Preliminary work was done during an internship at Hong Kong Baptist University (HKBU). This research was partially funded by the Australian Government through the Australian Research Council (ARC) under grants LP180100654, HKBU Tier-1 Start-up Grant and HKBU CSD Start-up Grant.

initialization based methods include learning a good model initialization [1]. Metric learning methods assume that there exists an embedding for any given dataset, where the representation of instances drawn from the same class is close to each other [4, 2]. The predictions of these methods are conditioned on distance or metric to few labeled instances during meta-training. Data augmentation generates new instances based on existing "seed" instances [5].

Table 1. Difficulties of different learning scenarios.

	few-shot	many-shot
small-class	hard [1]	easy [24]
large-class	**hardest** [11]	medium [12]

Most existing methods are evaluated on tasks with less than 50 classes [6–9]. However, in practice, we are often asked to classify thousands of classes, which naturally brings large-class few-shot scenario [10, 11]. As shown in Table 1, this scenario is challenging to conquer. In this kind of scenario, some classes are more difficult for the model to classify. Performance of the model on these classes will suffer from a relatively low accuracy. Meanwhile, experiments in large-class many-shot scenario also provides the same conclusion [12].

To tackle the large-class few-shot scenario, we propose a biased learning paradigm called *Confusable Learning*. Our key idea is to focus on confusable classes in meta-training dataset, which can improve model robustness in meta-testing dataset. In each iteration, we uniformly sample a few classes and denote them as *target classes*. For each target class, our paradigm selects several similar classes, which the model has difficulty in distinguishing from their target class. We call these classes *distractors*[1]. The model is then trained by a meta-learning algorithm to recognize instances of target class from those of distractors. Note that distractors are dynamically changing: when the model fits the distractors in each iteration, they become less confusable; while other classes become relatively more confusable and have higher chance to be selected as distractors. In this way, the model goes through every class in meta-training dataset dynamically. We briefly show how *Confusable Learning* works in Figure 1, where *Confusable Learning* is presented as a framework agnostic to different meta-learners. In the experiment, we build our method on the top of several state-of-the-art meta-learning methods, including Prototypical Network, Matching Network, Prototypical Matching Network and Ridge Regression Differentiable Discriminator [4, 2, 13, 14]. *Confusable Learning* is a training framework applied only in meta-training stage. In meta-test stage, these models are evaluated in the same way their authors originally did [4, 2, 13, 14]. We evaluate our method on datasets that have more than a thousand classes, including *Omniglot* [15], *Fungi*[2], and *ImageNet* [16]. The empirical results show that the models with *Confusable Learning* has better generalization in

[1] Distractors defined in our paper are different from those defined by Ren [17].

[2] `https://github.com/visipedia/fgvcx_fungi_comp`

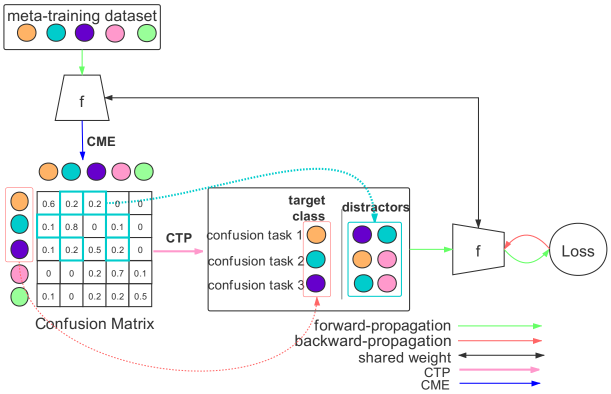


Fig. 1. Demonstration of one episode of *Confusable Learning* as specified in Algorithm 1. In this example, there are five classes in total in the meta-training dataset, which are represented by five circles with different colors. In the Confusion Matrix, the entry $\mathbf{C}_{i,j}$ (i≠j) is the average probability for instances of class i to be misclassified as class j. f denotes the meta-learning algorithm, where *Confusable Learning* is applied. Specifically, we use *Confusion Task Processor* in Algorithm 2 to sample three classes (red box) as our *target classes* and then locate two *distractors* for each target class (grey box). For example, for the orange class, we choose the purple and blue classes in the grey box as its *distractors*, because they have a higher probability in the Confusion Matrix. Second, *Confusion Matrix Estimation (CME)* in Algorithm 3, represented by the blue arrow, is applied to update the Confusion Matrix using probabilities calculated by the meta-learner. Confusable classes change as the model updates.

unseen meta-testing datasets than the models without *Confusable Learning* on large-class few-shot tasks.

2 Related Literature

In the following, we discuss representative few-shot learning algorithms organized into three main categories: model initialization and learnable optimizer based methods, metric learning methods, and data augmentation. The first category is learning an optimizer or a specified initialization. For example, Wichrowska *et al.* [18] used an RNN network to replace the gradient descent optimizer. Finn *et al.* [1] proposed a gradient-based method to backward-propagate through the learning process of a task, which finds a good initialization for the meta-learner. This method takes a very illuminating perspective on learning which is very effective. However, as reported by Chen *et al.* [5], this method can be easily hindered by large shifts between training and testing domain.

Metric learning methods assume that there exists an embedding for any given dataset, where the representation of instances drawn from the same class is close to each other. For example, Koch *et al.* [19] addressed one-shot problem by comparing distances from embedding of query instances to a single support instance. Vinyals *et al.* [4] proposed to focus attention on features that help solve a particular task. Instead of comparing distances to the embedding of labeled instances, Snell *et al.* [2] compared the distance to the prototype of each class, which is computed by averaging the embedding of support instances belonging to that class. In contrast with previous methods using handcraft metrics like Euclidean distance and cosine similarity, Garcia and Bruna [20] used a Graph Neural Network to learn the metric specific to a given dataset. Liu *et al.* [21, 22] proposed a novel graph structure to tackle the similar problem.

Data augmentation methods address few-shot problem by generating new instances based on existing "seed" instances. Chen *et al.* [5] showed that applying traditional data augmentation methods like crop, flip, and color jitter in traditional many-shot scenario can produce a competitive result. To capture more realistic variation for generating new instances, Hariharan and Girshick [10] learned to transfer the variation of the meta-training dataset to the meta-testing dataset. Instead of concerning the authenticity of those generated instances, Wang *et al.* [13] proposed to use a hallucinator to produce additional training instances. Data augmentation methods are orthogonal with other few-shot methods and can be considered as the pre-processing procedure for other few-shot methods. Thus, we do not consider these methods in this work.

On the other hand, researchers have studied large-class problem under many-shot and few-shot setting. Deng *et al.* [12] studied the ability of deep models to classify ImageNet subsets with more than 10000 classes of images. They found that only a small portions of classes are truly confusable and suggested to focus on these confusable classes to improve performance. Gupta *et al.* [23] treated confusable classes as the noise of lower priority and prevented the model from updating classes that are consistently misclassified. However, the assumption of ignoring confusable classes will cause performance degradation. Li *et al.* [11] proposed a novel large-class few-shot learning model by learning transferable visual features with the class hierarchy, which encodes the semantic relations between source and target classes. These works admit the difficulty brought by large-class setting. Deng *et al.* [12] motivates us to focus on, instead of ignoring, confusable classes in large-class setting to improve classification performance. Meanwhile, it is necessary to point out that Deng *et al.* [12] did not propose a method to deal with confusable classes, neither did they study how the learning of confusable classes of meta-training dataset influences the model performance on meta-testing dataset.

3 Algorithm

Here we clarify the notations that will be used later. Given a meta-training dataset with K classes called $\mathbb{D} = \{(\mathbf{x}^1, y^1), ..., (\mathbf{x}^N, y^N)\}$, we denote $\mathbb{D}_k$ as a set containing all instances within $\mathbb{D}$ that belong to the kth class. For conventional meta-learning methods, a few classes, represented by their indices in our work, are drawn from K classes in each episode. A support set and a held-out query set are sampled from instances of these classes in $\mathbb{D}$.

Algorithm 1: Confusable Learning

```
Input: Training set $\mathbb{D} = \{(\mathbf{x}^1, y^1), ..., (\mathbf{x}^N, y^N)\}$ (denote $\mathbb{D}_k$ as a set containing
       all instances of $\mathbb{D}$ that belong to $k$th class); Learnable weights $\Theta$;
       Number of times performing CME $M$.
Parameter : $\rho$
Initialize confusion matrix $\mathbf{E}$ as a matrix of shape $(K, K)$, with each entry
  initialized as $1/K$;
while Model does not converge do
    $\mathbf{C} \leftarrow \mathbf{E}$;
    // Algorithm 2
    $\Theta \leftarrow$ ConfusionTaskprocessor($\mathbb{D}$, $\Theta$, $\boldsymbol{C}$);
    foreach $i$ in $\{1, ..., M\}$ do
        // Algorithm 3
        $\mathbf{E} \leftarrow$ ConfusionMatrixEstimation($\mathbb{D}$, $\Theta$, $\boldsymbol{E}$);
    end
end
```

To focus on confusable classes, we propose our method called *Confusable Learning*. Specifically, *Confusable Learning* utilizes the confusion matrix $\mathbf{C}$, which is designed as a square matrix of K rows and K columns for a meta-training dataset with K classes. This matrix is calculated by training a model and then setting the entry $\mathbf{C}_{i,j}$ as the count of the test instances of class i that are misclassified as class j [26], as formally shown below:

$$\mathbf{C}_{i,j} = \sum_{(\mathbf{x},y)\in\mathbb{Q}^i} \mathbf{1}_{j=\arg\max_k P(\hat{y}=k|\mathbf{x})}, \tag{1}$$

in which $\mathbb{Q}$ denotes a query set, and $P(\hat{y} = k|\mathbf{x})$ denotes the prediction given by a meta-learning model.

The confusion matrix is often used to describe the performance of a classification model on a test dataset with true labels. We make use of the confusion matrix to find out the confusable classes and then learn from these confusable classes to update parameters in the model. We need to calculate this matrix in each episode because when parameters in the model change, the corresponding confusion matrix changes as well.

In the case where classes are imbalanced, it is useful to normalize the confusion matrix by dividing each entry of the confusion matrix by the sum of the entries of the corresponding row [28]. The normalized confusion matrix $\mathbf{C}^{\mathrm{n}}$ is formally given by:

$$\mathbf{C}^{\mathrm{n}}_{i,j} = \frac{\mathbf{C}_{i,j}}{\sum_{k=1}^{K} \mathbf{C}_{i,k}}. \tag{2}$$

As a result, the sum of entries in each row of $\mathbf{C}^{\mathrm{n}}$ is equal to 1.

To learn more about the confusable classes, we propose the definition of soft confusion matrix, which is slightly different from the traditional one. Instead of

Algorithm 2: ConfusionTaskProcessor($\mathbb{D}$, Θ,$\mathbf{C}$)

```
Input: Confusion Matrix $\mathbf{C}$; Training set $\mathbb{D}$; Learnable weights $\Theta$.
Parameter : Number of support instances for each class $N_S$; Number of query
            instances for each class $N_Q$; Number of distractors for each
            target class $N_D$; Number of confusable tasks $N_T^c$.
$\bar{v}^c$ ←RandomSample($\{1,...,K\}, N_T^c$) ;          // Sample target classes
foreach $k$ in $\{\bar{v}_1^c, ..., \bar{v}_{N_T^c}^c\}$ do
    Sample distractors $\bar{w}^k$ with Eq. (4);
    foreach $l$ in $\{\bar{w}_1^k, ..., \bar{w}_{N_D}^k\}$ do
        // Sample instances for distractors
        $\mathbb{S}_{k,l}^{\text{distractor}}$ ←RandomSample($\mathbb{D}_l, N_S$);
    end
    $\mathbb{S}_k^{\text{distractor}} \leftarrow \bigcup_{l=\{\bar{w}_1^k,...,\bar{w}_{N_D}^k\}} \mathbb{S}_{k,l}^{\text{distractor}}$;
    $\mathbb{S}_k^{\text{target}}$ ←RandomSample($\mathbb{D}_k, N_S$);     // Sample instances for target
      class
    $\mathbb{Q}_k$ ←RandomSample($\mathbb{D}_k, N_Q$) ;     // Sample instances for query set
end
$J \leftarrow 0$;
foreach $k$ in $\{\bar{v}_1^c, ..., \bar{v}_{N_T^c}^c\}$ do
    foreach $(\boldsymbol{x}, y)$ in $\mathbb{Q}_k$ do
        Calulate loss for any base meta-learning algorithm using the support set
          given by $\mathbb{S}_k^{\text{distractor}} \cup \mathbb{S}_k^{\text{target}}$;
        $J \leftarrow J + \frac{1}{N_T^c N_Q} \log \mathrm{P}(\hat{y} = k|\mathbf{x})$;
    end
end
Update $\Theta$ by back-propagating on $J$;
```

counting the times of classifying class i as class j, we utilize the probability of classifying instances of class i as class j:

$$\mathbf{C}_{i,j}^{\mathrm{p}} = \frac{1}{|\mathbb{Q}^i|} \sum_{(\mathbf{x},y)\in\mathbb{Q}^i} P(\hat{y} = j|\mathbf{x}). \tag{3}$$

It is easy to observe that $\sum_{j=1}^{K} \mathbf{C}_{i,j}^{\mathrm{p}}$ is always equal to 1. Thus, the soft confusion matrix defined by Eq. (3) is normalized itself. We will show in the experiment section that by using such a definition *Confusable Learning* exploit more detailed information about the error the model made for each instance.

We show the framework of our method in Algorithm 1. Algorithm 1 shows that *Confusable Learning* consists of 2 steps: Algorithm 2 and Algorithm 3, which will be introduced in detail in Section 3.1 and Section 3.2 respectively.

3.1 Confusion Task Processor

Before delving into the calculation of confusion matrix, we assume that we have a confusion matrix at the start of each episode. Next, *Confusion Learning* constructs several tasks called confusion tasks according to the confusion matrix. Then the meta-learning method is trained on these tasks.

As shown in Algorithm 2, to construct the confusion tasks, we first uniformly sample N_T^{c} target classes $\bar{v}^{\mathrm{c}} = \{\bar{v}_1^{\mathrm{c}}, ..., \bar{v}_{N_T^{\mathrm{c}}}^{\mathrm{c}}\}$. For target class $k \in \bar{v}^{\mathrm{c}}$, we use a multinomial distribution to sample N_D classes as distractors $\bar{w}^k = \{\bar{w}_1^k, ..., \bar{w}_{N_D}^k\}$. That is:

$$\bar{w}^k \sim \mathrm{Multinomial}(N_D, \mathbf{p}_1^k, ..., \mathbf{p}_{k-1}^k, \mathbf{p}_{k+1}^k, ..., \mathbf{p}_K^k), \tag{4}$$

in which:

$$\mathbf{p}_j^k = \frac{\mathbf{C}_{k,j}}{\sum_{h=\{1,2,...,k-1,k+1,...,K\}} \mathbf{C}_{k,h}}. \tag{5}$$

$\mathbf{C}$ is the confusion matrix given by either Eq. (2) or Eq. (3). We can see here that the classes with higher confusion are more likely to be sampled. By exposing the target class and the coupling distractors to the model, *Confusion Learning* enables the model to learn better.

Combining all target classes and their corresponding distractors, we have N_T^{c} pairs of target classes and distractors $\{(\bar{v}_1^{\mathrm{c}}, \bar{w}^1), ..., (\bar{v}_{N_T^{\mathrm{c}}}^{\mathrm{c}}, \bar{w}^{N_T^{\mathrm{c}}})\}$. For the k-th pair $(\bar{v}_k^{\mathrm{c}}, \bar{w}^k)$, our learning paradigm samples N_S instances for each distractor and target class, generating a support set $\mathbb{S}$ with $N_S \times (N_D + 1)$ instances, and samples N_Q instances of the target class $\bar{v}_k^{\mathrm{c}}$, generating a query set $\mathbb{Q}$ with N_Q instances. Combining $\mathbb{S}$ and $\mathbb{Q}$, we have a confusion task. Predictions $\mathrm{P}(\hat{y} = k|\mathbf{x} \in \mathbb{Q})$ are computed by the meta-learning algorithm using N_T^{c} confusion tasks. By maximizing $\mathrm{P}(\hat{y} = k|\mathbf{x})$ and applying stochastic gradient descent, we can optimize parameters in the model to distinguish target classes better in the next episode until they become less confusable, after which the model switches its attention to relatively more confusable classes as the confusion matrix updates dynamically. The pseudo-code is demonstrated in Algorithm 2.

3.2 Confusion Matrix Estimation

The traditional way of the confusion matrix calculation requires inferences of instances drawn from K classes, which can be extremely slow and require high RAM usage when K is large. To bypass this hinder, we propose a novel iterative way called Confusion Matrix Estimation (CME) to estimate the confusion matrix, the difficulty of which is much smaller than the traditional one.

Given a trained model, we regard the confusion matrix $\mathbf{C}$ calculated by evaluating the model on K classes using the traditional method as the ideal confusion matrix. The purpose of CME is to estimate $\mathbf{C}$ in an incremental yet less computation-consuming way. CME initializes a matrix $\mathbf{E}$ of size $K \times K$, which is the same as the ideal confusion matrix. Each entry of $\mathbf{E}$ is initialized with a positive constant. Since the summary of each row of $\mathbf{C}$ is equal to 1, $1/K$ is usually good for the initialization of $\mathbf{E}$. The purpose of CME is to update $\mathbf{E}$ in multiple steps to make it closer to the ideal confusion matrix $\mathbf{C}$. In each step, N_T^{e} classes are uniformly sampled from meta-training dataset. Let us mark their indices among the K meta-training classes as $\bar{v}^{\mathrm{e}} = \{\bar{v}_1^{\mathrm{e}}, ..., \bar{v}_{N_T^{\mathrm{e}}}^{\mathrm{e}}\}$. By performing inference on $\bar{v}^{\mathrm{e}}$, we are able to obtain a smaller confusion matrix named $\mathbf{E}'$ with the size of $N_T^{\mathrm{e}} \times N_T^{\mathrm{e}}$. Clearly, $\mathbf{E}'$ contains the information on how a class is confused with other classes among $\bar{v}^{\mathrm{e}}$. $\mathbf{E}'$ is somehow like an observation of $\mathbf{C}$ through a small "window". To incorporate the observation into $\mathbf{E}$, in each step, CME updates $\mathbf{E}$ by:

$$\mathbf{E}_{\bar{v}_i^{\mathrm{e}},\bar{v}_j^{\mathrm{e}}} = \rho \mathbf{E}_{\bar{v}_i^{\mathrm{e}},\bar{v}_j^{\mathrm{e}}} + (1-\rho)\mathbf{E}'_{i,j} Z, \tag{6}$$

in which ρ is a hyperparameter between 0 and 1, and Z is used to scale the observation to make sure the summary of the current observation is constant with the result of previous observations:

$$Z = \sum_{k=1,\dots,N^{\mathrm{e}}} \mathbf{E}_{\bar{v}_i^{\mathrm{e}},\bar{v}_k^{\mathrm{e}}}. \tag{7}$$

To combine CME with previously introduced *Confusable Learning* framework, we can initialize $\mathbf{E}$ and perform multi-steps CME at each *Confusable Learning* episode to get a reliable estimation of the confusion matrix. However, it is not necessary to make a fresh start in each episode. Denoting the ideal confusion matrix of the model at episode i as $\mathbf{C}_i$, since the ability of the model will not change a lot between successive episodes, $\mathbf{C}_{i+1}$ should be close to $\mathbf{C}_i$. Intuitively, we do not need to initialize a new $\mathbf{E}$ in each episode. Instead, to calculate the estimation $\mathbf{E}_{i+1}$ of episode $i+1$, we can perform CME update based on the estimation $\mathbf{E}_i$ of episode i. As such, $\mathbf{E}_0$ is initialized only once when the meta-learning model is initialized. Then at each episode, CME updates for M steps. In our experience, by setting M to 1, the performance of CME is good enough. Algorithm 3 shows how the confusion matrix is updated in an episode.

To demonstrate the training dynamic of our method using Algorithm 1, Figure 2 shows that *Confusable Learning* first spreads its attention to many classes, and then turns to the classes that are more difficult to distinguish. In the early stage (top 100 rows) of meta-training, attention is dynamically spread to many classes. Later, most of these classes are well fitted by the model and get less attention. Meanwhile, other classes get more attention because they are inherently intractable to distinguish. We found that these intractable classes are visually similar (Figure 2(c)).

4 Experiment

4.1 Experiment Setup

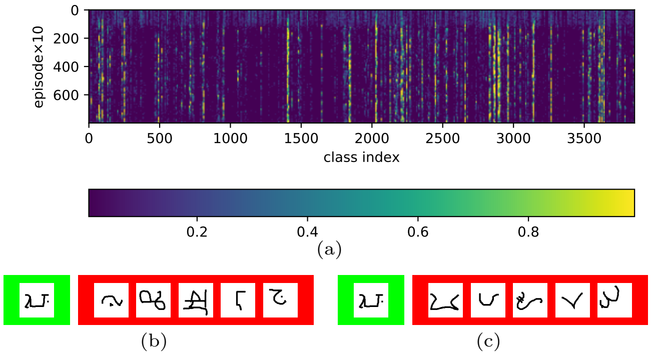
To empirically prove the efficacy of our method, we conduct experiments on three real-world datasets: *Omniglot*, *Fungi*, and *ImageNet*. We choose these datasets because they contain more than 1000 classes, unlike prior works on meta-learning which experiment with smaller images and fewer classes.

Confusable Learning can be easily applied to various mainstream meta-learning algorithms. To demonstrate the efficacy of our method, we choose four

```
Algorithm 3: ConfusionMatrixEstimation($\mathbb{D}$, $\Theta$, $\mathbf{E}$)
Input: Training set $\mathbb{D}$; Learnable weights $\Theta$; Estimation of confusion matrix $\mathbf{E}$.
Parameter : Number of support instances for each class $N_S$; number of query
            instances for each class $N_Q$; number of classes to use in each
            CME step $N_T^{\mathrm{e}}$.
Initialize $\mathbf{E}'$ as a zero matrix of shape $(N_T^{\mathrm{e}}, N_T^{\mathrm{e}})$, with each entry initialized as 0;
$\bar{v}^{\mathrm{e}} \leftarrow$RandomSample($\{1, ..., K\}, N_T^e$);
foreach $k$ in $\{\bar{v}_1^e, ..., \bar{v}_{N_T^e}^e\}$ do
    $\mathbb{S}_k \leftarrow$RandomSample($\mathbb{D}_k, N_S$);
    $\mathbb{Q}_k \leftarrow$RandomSample($\mathbb{D}_k, N_Q$);
end
foreach $m$ in $\{1, ..., N_T^e\}$ do
    foreach $(\boldsymbol{x}, y)$ in $\mathbb{Q}_{\bar{v}_m^e}$ do
        foreach $n$ in $\{1, ..., N_T^e\}$ do
            Calculate $\mathrm{P}(\hat{y} = \bar{v}_n^{\mathrm{e}} | \mathbf{x})$ with any base meta-learning algorithm using
              the support set given by $\bigcup_{k=\{\bar{v}_1^{\mathrm{e}}, ..., \bar{v}_{N_T^{\mathrm{e}}}^{\mathrm{e}}\}} \mathbb{S}_k$;
            $\mathbf{E}'_{m,n} \leftarrow \mathbf{E}'_{m,n} + \mathrm{P}(\hat{y} = \bar{v}_n^{\mathrm{e}} | \mathbf{x})$;
        end
    end
end
$\mathbf{E}' \leftarrow \frac{\mathbf{E}'}{N_Q}$;
Update $\mathbf{E}$ using Eq. (6);
return $\mathbf{E}$;
```

state-of-the-art meta-learning algorithms as our base meta-learning methods in our experiments: Prototypical Network (PN) [2], Matching Network (MN) [4], Prototype Matching Network (PMN) [13] and Ridge Regression Differentiable Discriminator (R2D2) [14]. We will show that by applying *Confusable Learning* to these methods, we can easily improve their performance in large-class few-shot learning setting. For notation, we denote our method by attaching a "w/CL" behind each of them. For example, PN w/CL means prototype networks with *Confusable Learning.*

In few-shot learning, N-shot K-way classification tasks consist of N labeled instances for each K classes. As stated in PN [2], it can be extremely beneficial to train meta-learning models with a higher way than that will be used in meta-testing dataset. Particularly, for PN, to train a model for 5/20 ways tasks, the author used training tasks with 60 classes [2]. The same setting is also mentioned in other mainstream few-shot methods [14]. However, for the large-class few-shot problem which has a large number of classes, it becomes impractical to build even larger support sets due to the limitation of memory and the exponentially growing load of computation. Therefore, in our experiment, the models are all trained in meta-training tasks with fewer ways than the meta-testing task. We evaluate the models using query sets and support sets constructed in the same way as their authors originally did [4, 2, 13, 14].



(a)

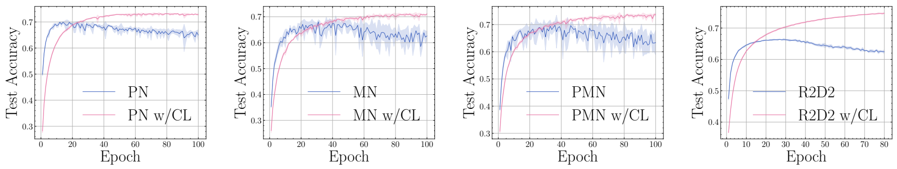
(b)

(c)

Fig. 2. (a) A Prototype Network is trained with *Confusable Learning* on *Omniglot*. Here, each pixel of the image denotes the frequency for a class to be focused, i.e., to be selected as a distractor in 10 episodes. For better visual effect, dilation has been applied to the image. (b) Target class (green) and distractors that get no attention (red) in the last 100 episodes. (c) Target class and distractors that get the most attention in the last 100 episodes.

Omniglot. *Omniglot* contains 1623 handwritten characters. As what Vinyals [4] has done, we resize the image to 28 × 28 and augment the dataset by rotating each image by 90, 180, and 270 degrees. For a more challenging meta-testing environment, we employ the split introduced by Lake [15], constructing our meta-training dataset with 3856 classes and meta-testing dataset with 2636 classes. In meta-testing stage, we use all 2636 classes in every single meta-testing task.

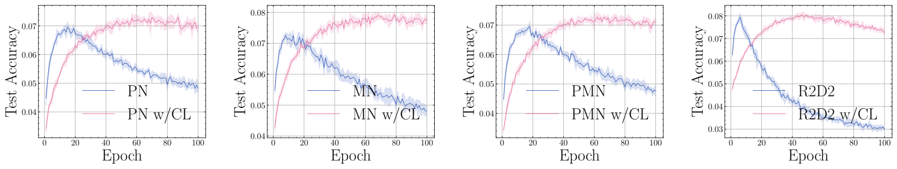
For PN, MN and PMN, learning rate is set to 1e-4. For R2D2, learning rate is set to 5e-5. For our method, We set N_D to 40, N_T^e to 500. In our experiments of all datasets, N_T^c is always set to $(N_T^e \times 2)/(N_D + 2)$, which is 23 here. ρ is set to 0.9. M is set to 1.



(a) PN vs. PN w/CL (b) MN vs. MN w/CL (c) PMN vs. PMN w/CL (d) R2D2 vs. R2D2 w/CL

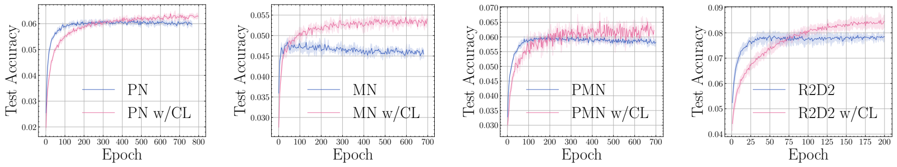
Fig. 3. Test accuracy vs. number of epoch of 4 kinds of meta-learning method without vs. with *Confusable Learning* in *Omniglot*.

Fungi. *Fungi* is originally introduced by the 2018 FGVCx Fungi Classification Challenge. We randomly sample 632 classes to construct meta-training dataset,



(a) PN vs. PN w/CL (b) MN vs. MN w/CL (c) PMN vs. PMN w/CL (d) R2D2 vs. R2D2 w/CL

Fig. 4. Test accuracy vs. number of epoch of 4 kinds of meta-learning method without vs. with *Confusable Learning* in *fungi*.



(a) PN vs. PN w/CL (b) MN vs. MN w/CL (c) PMN vs. PMN w/CL (d) R2D2 vs. R2D2 w/CL

Fig. 5. Test accuracy vs. number of epoch of 4 kinds of meta-learning methods without vs. with *Confusable Learning* in *Imagenet*.

674 classes to construct meta-testing dataset and 88 classes to construct validation dataset. All 674 meta-testing classes are used in every meta-testing task.

Learning rate is set to 5e-4 for all methods. For our method, We set N_D to 5. N_T^{e} is set to 70, and thus N_T^{c} is set to 20. ρ is set to 0.9. M is set to 1. For R2D2 w/CL, the temperature of the softmax is set to 0.1 in CME.

ImageNet64x64. We also conduct experiments on *ImageNet64x64* dataset [27], which is a downsampled version of the original *ImageNet* used in ILSVRC with images resized to 64×64 pixels. The reason why we do not perform experiments on regular few-shot datasets like *mini*ImageNet and *tiered*ImageNet is that neither of them holds enough number of classes for our large-class setting. Although *ImageNet64x64* is not regularly used to evaluate few-shot learning methods, it contains 1000 classes in total. Here we merge its original training dataset and original testing dataset together and then split it by class again into a new meta-training dataset, a validation dataset, and a new meta-testing dataset according to the category of the classes. All classes that belong to the category **Living Thing** are used for meta-training stage and all classes that belong to the category **Artifact** are used for meta-testing stage. This gives us a meta-training dataset with 522 classes and a meta-testing dataset with 410 classes. The rest 68 classes belong to the validation dataset.

For MN and MN w/CL, we set the learning rate to 1e-3. For the other settings, the learning rate is set to 1e-4. In our experiment, We set N_D to 5. N_T^{e} is set to 90, and thus N_T^{c} is set to 25. ρ is set to 0.9. M is set to 1.

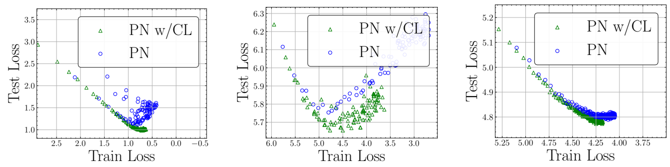
4.2 Result

We perform five repeated experiments with different random seeds. In Table 2, the averaged accuracy for each method is shown. It can be seen all meta learners coupled with our method outperform original ones in all three datasets.

To delve into the training dynamic, Figures 3, 4 and 5 show how testing accuracy changes with respect to the number of epochs, with the standard deviation shown with the shaded area. The meta-learning algorithm with *Confusable Learning* has a better generation than those without it. As shown in Figure 4, it is interesting to find out that *Confusable Learning* shows a great ability to resist over-fitting than corresponding original methods in *Fungi*, which contains very similar mushroom classes and thus leads to over-fitting easily.

Algorithm	PN	PN w/CL	MN	MN w/CL	PMN	PMN w/CL	R2D2	R2D2 w/CL
Omniglot	69.90%	**73.21%**	68.04%	**71.07%**	69.73%	**73.68%**	66.12%	**74.72%**
Fungi	6.96%	**7.29%**	7.26%	**7.92%**	6.96%	**7.28%**	7.92%	**8.04%**
ImageNet	6.02%	**6.27%**	4.79%	**5.41%**	5.90%	**6.41%**	7.78%	**8.46%**

Table 2. 5-shot classification test accuracies of PN, PN w/CL, MN, MNw/CL, PMN, PMNw/CL and R2D2, R2D2 w/CL in *Omniglot* (2636-way), *Fungi* (674-way) and *ImageNet64x64* (410-way).



(a) *Omniglot* (b) *Fungi* (c) *ImageNet64x64*

Fig. 6. (a) Training loss vs. testing loss in *Omniglot* dataset. The triangles or circles on the right have smaller training loss, and thus usually represent the performance of the meta-learning algorithm in the late stage of training (b) Training loss vs. testing loss in *Fungi* dataset. (b) Training loss vs. testing loss in *ImageNet64x64* dataset.

To further demonstrate the ability of *Confusable Learning* to resist over-fitting, we visualize the co-relationship between the training loss and the testing loss of PN and PN w/CL. Note that we estimate the training loss of the model using query sets and support sets constructed in the same way as Snell [2], but including all classes in the meta-training dataset in each task. To be specific, we estimate training losses in *Omniglot* (3856-way 5-shot), *Fungi* (632-way 5-shot), and *ImageNet64x64* (522-way 5-shot). Figure 6 shows that under the same training loss, *Confusable Learning* has smaller testing loss. We believe that PN over-fits the regular patterns and ignores confusable patterns, while *Confusable Learning* focuses on these confusable patterns.

4.3 Ablation Study

In this section, we will discuss the influence of proposed soft confusion matrix and Confusion Matrix Estimation in *Confusable Learning.*

Algorithm	PN	MN	PMN	R2D2
baseline	69.90±0.16%	68.04±0.23%	69.73±0.21%	66.12±0.16%
w/CLN	**73.23±0.10%**	70.63±0.09%	73.24±0.09%	74.65±0.14%
w/CL	73.21±0.04%	**71.07±0.19%**	**73.68±0.21%**	**74.72±0.52%**

Table 3. 5-shot classification test accuracies on *Omniglot* for *Confusable Learning* using the traditional confusion matrix and the proposed soft confusion matrix.

Influence of calculating confusion matrix with probability. *Confusable Learning* adopts a novel definition of confusion matrix called soft confusion matrix as shown in Eq. (3). To demonstrate its benefit, we implement *Confusable Learning* using the traditional definition of confusion matrix given in Eq. (2). To implement such a setting, we simply replace $\mathrm{P}(\hat{y} = \bar{v}_n^{\mathrm{e}}|\mathbf{x})$ in line 11 of Algorithm 3 with $\mathbf{1}_{\bar{v}_n^{\mathrm{e}}=\arg\max_k P(\hat{y}=k|(\mathbf{x},y))}$. We denote this setting by attaching a “w/CLN” behind the name of each meta-learning algorithm. Results of *Omniglot* and *Fungi* are shown in Table 3 and Table 4 respectively. It can be seen that the accuracy of the model using soft confusion matrix, marked with “w/CL”, is higher than the model marked as “w/CLN” on *Fungi.* However, on *Omniglot* they are almost equal. This is because the model is more confident about its prediction on *Omniglot*, making $P(\hat{y}|(\mathbf{x}, y))$ close to either 0 or 1. In this case, the proposed soft confusion matrix is equivalent to the traditional one.

Algorithm	PN	MN	PMN	R2D2
baseline	6.96±0.07%	7.26±0.09%	6.96±0.12%	7.92±0.13%
w/CLN	7.09±0.06%	7.89±0.02%	7.14±0.05%	8.04±0.14%
w/CL	**7.29±0.09%**	**7.92±0.04%**	**7.28±0.11%**	8.04±0.12%

Table 4. 5-shot classification test accuracies on *Fungi* for *Confusable Learning* using the traditional confusion matrix and the proposed soft confusion matrix.

Influence of Confusion Matrix Estimation (CME). To demonstrate the performance and the efficiency of CME, we implement *Confusable Learning* with the traditional way of confusion matrix calculation, which is performing the meta-learning algorithm on a K-way task and calculating confusion matrix with Eq.(3). In fact, the traditional confusion matrix calculation can be seen as a special case of CME, in which ρ is set to 0 and N_T^{e} is set to K. We denote this setting by attaching a "w/CLT" behind the name of meta-learning algorithm.

	PN w/CL-1	PN w/CL-2	PN w/CL-4	PN w/CL-8	PN w/CLT
Accuracy	73.21%	73.58%	73.46%	73.59%	73.60%
Elapsed Time	0.073s	0.146s	0.289s	0.570s	1.432s
GPU Memory	2463MiB	2479MiB	2511MiB	2567MiB	12457MiB

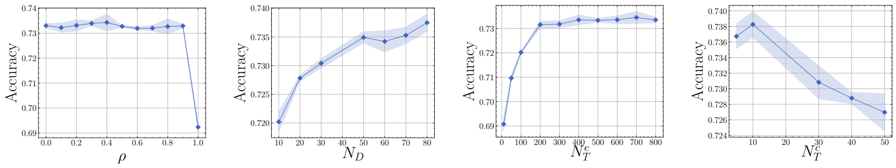
Table 5. 5-shot classification test accuracies, averaged elapsed time in each iteration and memory in need for confusion matrix calculation in *Omniglot* (2636-way).

In Table 5, we compare the results of *Confusable Learning* with the traditional confusion matrix calculation and the result of *Confusable Learning* with our proposed CME. Here, we denote *Confusable Learning* with M steps of CME by attaching a "w/CL-M". The experiment is conducted on a machine with a Tesla P100 GPU. It can be concluded that the CME settings achieve almost the same accuracy with the setting using the traditional confusion matrix calculation but largely decrease the elapsed time and memory requirement. It is noteworthy that *Omniglot* is a small dataset, so we are able to implement *Confusable Learning* with the traditional confusion matrix calculation easily. When training with a larger dataset like *fungi* and *Imagenet64x64*, traditional confusion matrix calculation will require much longer time and larger memory.

4.4 Parameter Sensitivity Analysis

Confusable Learning contains 5 parameters: N_D, N_T^{e}, N_T^{c}, M and ρ. Table 5 already shows that larger M yields a better result. In this section, we discuss how sensitive the other 4 parameters are.

Based on the parameters we use in *Omniglot* PN w/CL experiment in Section 4.1, we adjust each of the 4 parameters at a time. The results are shown in Figure 7. Performance of the model is very stable with ρ changing from 0 to 0.9. It is not surprising that the accuracy drops below 70% when ρ is set to 1. In such a case, the confusion matrix will not be updated and thus, *Confusable Learning* can not obtain any useful information from the confusion matrix. Increasing N_D and N_T^{e} always help improve accuracy but requires longer elapsed time and more memory. When increasing N_T^{c}, as shown in 7(d), accuracy firstly increases and then decreases. The optimal N_T^{c} is about 10. Regardless of this observation, the accuracy is over 73% within a large range of N_T^{c}, significantly higher than the accuracy of the PN model without $ConfusableLearning$ reported in Table 2. It can be concluded that *Confusable Learning* can yield a great performance without any elaborate tuning.



(a) ρ vs. accuracy (b) N_D vs. accuracy (c) N_T^{e} vs. accuracy (d) N_T^{c} vs. accuracy

Fig. 7. Accuracy of PN w/CL vs. parameters in *Omniglot*.

5 Conclusion

We have presented an approach to locate and learn from confusable classes in large-class few-shot classification problem. We show significant gains on top of multiple meta-learning methods, achieving state-of-the-art performance on three challenging datasets. Future work will involve constructing better confusion tasks to learn confusable classes better.

References

1. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: International Conference on Machine Learning (ICML), pp. 1126-1135. (2017)
2. Snell, J., Swersky, K., Zemel, R.: Prototypical networks for few-shot learning. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 4077-4087. (2017)
3. Wang, Y., Yao, Q.: Few-shot learning: A survey. arXiv preprint `arXiv:1904.05046`. (2019)
4. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D.: Matching networks for one shot learning. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 3630-3638. (2016)
5. Chen, W., Liu, Y., Zsolt, K., Wang, Y., Huang, J.: A Closer Look at Few-shot Classification. In: The International Conference on Learning Representations (ICLR),(2019)
6. Zhang, R., Che, T., Ghahramani, Z., Bengio, Y., Song, Y.: MetaGAN: An Adversarial Approach to Few-Shot Learning. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 2365-2374. (2018)
7. Oreshkin, B., Rodríguez López, P., Lacoste, A.: TADAM: Task dependent adaptive metric for improved few-shot learning. In:Advances in Neural Information Processing Systems (NeurIPS), pp. 721-731. (2018)
8. Liu, Y., Lee, J., Park, M., Kim, S., Yang, E., Hwang, S., Yang, Y.: Learning to propagate labels: transductive propagation network for few-shot learning. In: Conference on Computer Vision and Pattern Recognition (CVPR), (2018)
9. Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., Pontil, M.: Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In: International Conference on Machine Learning (ICML), (2018)
10. Hariharan, B., Girshick, Ross.: Low-shot visual recognition by shrinking and hallucinating features. In: International Conference on Computer Vision (ICCV), pp. 3018-3027. (2017)

11. Li, A., Luo, T., Lu, Z., Xiang, T., Wang, L.: Large-Scale Few-Shot Learning: Knowledge Transfer With Class Hierarchy. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7212-7220. (2019)
12. Deng, J., Berg, A., Li, K., Li, F.: What does classifying more than 10,000 image categories tell us? In European Conference on Computer Vision (ECCV), pp. 71-84. (2010)
13. Wang, Y., Girshick, R., Hebert, M., Hariharan, B.: Low-shot learning from imaginary data. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7278-7286. (2018)
14. Bertinetto, L., Henriques, J., Torr, P., Vedaldi, A.: Meta-learning with differentiable closed-form solvers. In: The International Conference on Learning Representations (ICLR), (2019)
15. Lake, B., Salakhutdinov, R., Tenenbaum, J.: Human-level concept learning through probabilistic program induction. Science 350(6266), 1332–1338 (2015)
16. Deng, J., Dong, W., Socher, R., Li, L., Li, Kai., Li, F.: ImageNet: A Large-Scale Hierarchical Image Database. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 248-255. (2009)
17. Ren, M., Ravi, S., Triantafillou, E., Snell, J., Swersky, K., Tenenbaum, J., Larochelle, H., Zemel, R.: Meta-Learning for Semi-Supervised Few-Shot Classification. In: The International Conference on Learning Representations (ICLR), (2018)
18. Wichrowska, O., Maheswaranathan, N., Hoffman, M., Colmenarejo, S., Denil, M., de Freitas, N., Sohl-Dickstein, J.: Learned optimizers that scale and generalize. In: International Conference on Machine Learning (ICML), pp. 3751-3760. (2017)
19. Koch, G., Zemel, R., Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. In: International Conference on Machine Learning Deep Learning Workshop (ICML), (2015)
20. Garcia, V., Bruna, E.: Few-Shot Learning with Graph Neural Networks. In: The International Conference on Learning Representations (ICLR), (2018)
21. Liu, L., Zhou, T., Long, G., Jiang. J., Yao, L., Zhang, C.: Prototype propagation networks (PPN) for weakly-supervised few-shot learning on category graph. In International Joint Conferences on Artificial Intelligence (IJCAI), (2019)
22. Liu, L., Zhou, T., Long, G., Jiang. J., Zhang, C.: Learning to propagate for graph meta-learning. In Advances in Neural Information Processing Systems (NeurIPS), pp.1037-1048. (2019)
23. Gupta, R., Bengio, S., Weston, J.: Training Highly Multiclass Classifiers. Journal of Machine Learning Research (JMLR) 15(1), 1461-1492 (2014)
24. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770-778. (2016)
25. Sun, Q., Liu, Y., Chua, T., Schiele, B.: Meta-transfer learning for few-shot learning. In Conference on Computer Vision and Pattern Recognition (CVPR), pp. 403-412. (2019)
26. Griffin, G., Perona, P.: Learning and using taxonomies for fast visual categorization. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1-8. (2008)
27. Chrabaszcz, P., Loshchilov, I., Hutter, F., A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets. arXiv preprint `arXiv:1707.08819`. (2017)
28. Giannakopoulos, T., Pikrakis, A.: Introduction to audio analysis: a MATLAB® approach. Academic Press. Academic Press. (2014)