Many-Class Few-Shot Learning on Multi-Granularity Class Hierarchy

Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang

Abstract—We study many-class few-shot (MCFS) problem in both supervised learning and meta-learning settings. Compared to the well-studied many-class many-shot and few-class few-shot problems, the MCFS problem commonly occurs in practical applications but has been rarely studied in previous literature. It brings new challenges of distinguishing between many classes given only a few training samples per class. In this article, we leverage the class hierarchy as a prior knowledge to train a coarse-to-fine classifier that can produce accurate predictions for MCFS problem in both settings. The propose model, "memory-augmented hierarchical-classification network (MahiNet)", performs coarse-to-fine classification where each coarse class can cover multiple fine classes. Since it is challenging to directly distinguish a variety of fine classes given few-shot data per class, MahiNet starts from learning a classifier over coarse-classes with more training data whose labels are much cheaper to obtain. The coarse classifier reduces the searching range over the fine classes and thus alleviates the challenges from "many classes". On architecture, MahiNet first deploys a convolutional neural network (CNN) to extract features. It then integrates a memory-augmented attention module and a multi-layer perceptron (MLP) together to produce the probabilities over coarse and fine classes. While the MLP extends the linear classifier, the attention module extends the KNN classifier, both together targeting the "few-shot" problem. We design several training strategies of MahiNet for supervised learning and meta-learning. In addition, we propose two novel benchmark datasets "*mcfs*ImageNet" (as a subset of ImageNet) and "*mcfs*Omniglot" (re-splitted Omniglot) specially designed for MCFS problem. In experiments, we show that MahiNet outperforms several state-of-the-art models (e.g., prototypical networks and relation networks) on MCFS problems in both supervised learning and meta-learning.

Index Terms—Deep learning, many-class few-shot classification, class hierarchy, meta-learning

1 Introduction

The representation power of deep neural networks (DNN) has significantly improved in recent years, as deeper, wider and more complicated DNN architectures have emerged to match the increasing computation power of new hardware [1], [2]. Although this brings hope to complex tasks that could be hardly solved by previous shallow models, more labeled data is usually required to train the deep models. The scarcity of annotated data has become a new bottleneck for training more powerful DNNs. It is quite common in practical applications such as image search, robot navigation and video surveillance. For example, in image classification, the number of candidate classes easily exceeds tens of thousands (i.e., many-class), but the training samples available for each class can be less than 100 (i.e., few-shot). Unfortunately, this scenario is beyond of the scope of current meta-learning methods for few-shot classification, which aims to address the data scarcity (few-shot data per class) but the number of classes in each task is usually less than 10. Additionally, in life-long learning, models are always updated once new training data becomes available, and those models are expected to quickly adapt to new classes with a few training samples available.

• *Lu Liu, Guodong Long, Jing Jiang, and Chengqi Zhang are with the Centre for Artificial Intelligence, FEIT, University of Technology Sydney, Ultimo, NSW 2007, Australia. E-mail: lu.liu-10@student.uts.edu.au, {guodong.long, jing.jiang, chengqi.zhang}@uts.edu.au.*
• *Tianyi Zhou is with the Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA 98195 USA. E-mail: tianyizh@uw.edu.*

Manuscript received 7 July 2019; revised 11 June 2020; accepted 14 June 2020. Date of publication 25 June 2020; date of current version 1 Apr. 2022. (Corresponding author: Guodong Long.) Recommended for acceptance by Benjamin C. M. Fung. Digital Object Identifier no. 10.1109/TKDE.2020.3004939

Although previous works of fully supervised learning have shown the remarkable power of DNN when "many-class many-shot" training data is available, their performance degrades dramatically when each class only has a few samples available for training. In practical applications, acquiring samples of rare species or personal data from edge devices is usually difficult, expensive, and forbidden due to privacy protection. In many-class cases, annotating even one additional sample per class can be very expensive and requires a lot of human efforts. Moreover, the training set cannot be fully balanced over all the classes in practice. In these few-shot learning scenarios, the capacity of a deep model cannot be fully utilized, and it becomes much harder to generalize the model to unseen data. Recently, several approaches have been proposed to address the few-shot learning problem. Most of them are based on the idea of "meta-learning", which trains a meta-learner over different few-shot tasks so it can generalize to new few-shot tasks with unseen classes. Thereby, the meta-learner aims to learn a stronger prior encoding the general knowledge achieved during learning various tasks, so it is capable to help a learner model quickly adapt to a new task with new classes of insufficient training samples. Meta-learning can be categorized into two types: methods based on "learning to optimize", and methods based on metric learning. The former type adaptively modifies the optimizer (or some parts of it) used for training the task-specific model. It includes methods

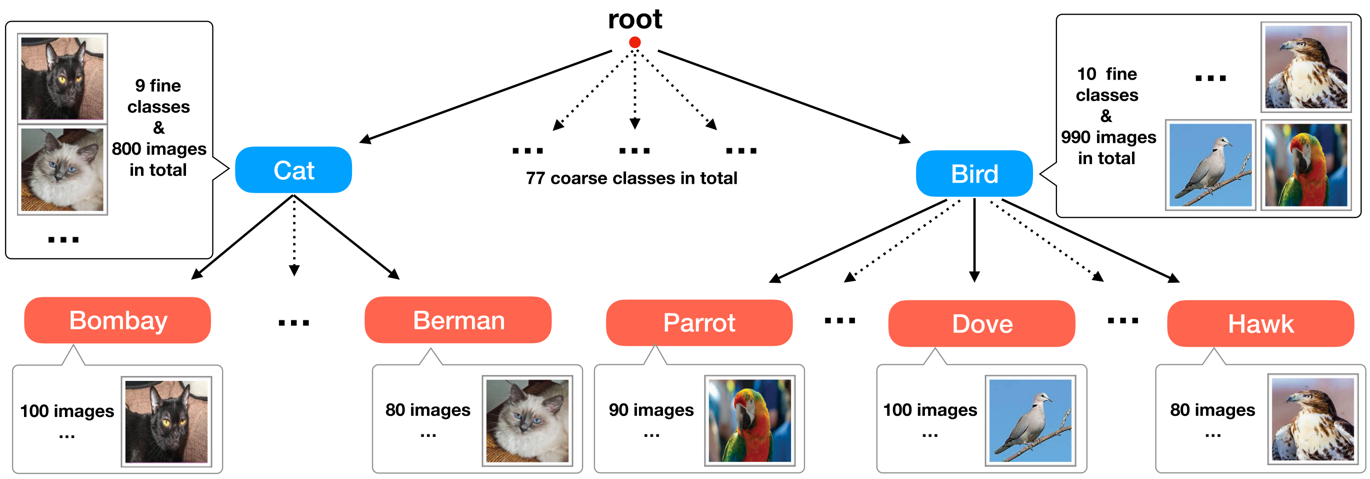

root
9 fine classes & 800 images in total
Cat
77 coarse classes in total
Bird
10 fine classes & 990 images in total
Bombay
Berman
Parrot
Dove
Hawk
100 images
80 images
90 images
100 images
80 images

Fig. 1. A many-class few-shot learning (MCFS) problem using multi-label class hierarchy information. There are a few coarse classes (blue) and each coarse class covers a large number of fine classes (red) so that the total number of fine classes is large. Only a few training samples are available for each fine class. The goal is to train a classifier to generate a prediction over all fine classes with the help of coarse prediction. We utilize meta-learning to solve the problem of many-class few-shot learning problem, where each task is an MCFS problem sampled from a certain distribution. The meta-learner's goal is to help train a classifier for any sampled task with better adaptation to few-shot data within the sampled task.

that incorporate an Recurrent Neural Networks (RNN) meta-learner [3], [4], [5], and model-agnostic meta-learning (MAML) methods aiming to learn a generally compelling initialization [6]. The second type learns a similarity/distance metric [7] or a model generating a support set of samples [8] that can be used to build K-Nearest Neighbors (KNN) classifiers from few-shot data in different tasks.

Instead of meta-learning methods, data augmentation based approaches, such as the hallucination method proposed in [9], address the few-shot learning problem by generating more artificial samples for each class. However, most existing few-shot learning approaches only focus on "few-class" cases (e.g., 5 or 10) per task, and their performance drastically collapses when the number of classes slightly grows to tens to hundreds. This is because the samples per class no longer provide enough information to distinguish them from other possible samples within a large number of other classes. And in real-world few-shot problems, a task is usually complicated involving many classes.

Fortunately, in practice, multi-outputs/labels information such as coarse-class labels in a class hierarchy is usually available or cheaper to obtain. In this case, the correlation between fine and coarse classes can be leveraged in solving the MCFS problem, e.g., by training a coarse-to-fine hierarchical prediction model. As shown in Fig. 1, coarse class labels might reveal the relationships among the targeted fine classes. Moreover, the samples per coarse class are sufficient to train a reliable coarse classifier, whose predictions are able to narrow down the candidates for the corresponding fine class. For example, a sheepdog with long hair could be easily mis-classified as a mop when training samples of sheepdog are insufficient. However, if we could train a reliable dog classifier, it would be much simpler to predict an image as a sheepdog than a mop given a correct prediction of the coarse class as "dog". Training coarse-class prediction models is much easier and less suffered from the "many class few shot"problem (since the coarse classes are fewer than the fine classes and the samples per coarse class are much more than that for each fine class). It provides helpful information to fine-class prediction due to the relationship between coarse classes and fine classes. Hence, class hierarchy might provide weakly supervised information to help solve the "many-class few-shot (MCFS)" problem in a framework of multi-output learning, which aims to predict the class label on each level of the class hierarchy.

In this paper, we address the MCFS problem in both traditional supervised learning and in meta-learning settings by exploring the multi-output information on multiple class hierarchy levels. We develop a neural network architecture "memory-augmented hierarchical-classification networks (MahiNet)" that can be applied to both learning settings. MahiNet uses a Convolutional Neural Network (CNN), i.e., ResNet [1], as a backbone network to first extract features from raw images. It then trains coarse-class and fine-class classifiers based on the features, and combines their outputs to produce the final probability prediction over the fine classes. In this way, both the coarse-class and the fine-class classifiers mutually help each other within MahiNet: the coarse classifier helps to narrow down the candidates for the fine classifier, while the fine classifier provides multiple attributes describing every coarse class and can regularize the coarse classifier. This design leverages the relationship between the fine classes as well as the relationship between the fine classes and the coarse classes, which mitigates the difficulty caused by the "many class" challenge. To the best of our knowledge, we are the first to successfully train a multi-output model to employ existing class hierarchy prior for improving both few-shot learning and supervised learning. It is different from those methods discussed in related works that use a latent clustering hierarchy instead of the explicit class hierarchy we use. In Table 1, we provide a brief comparison of MahiNet with other popular models on the learning scenarios they excel.

To address the "few-shot" problem, we apply two types of classifiers in MahiNet, i.e., MLP classifier and KNN classifier, which respectively have advantages in many-shot and few-shot situations. In principle, we always use MLP for coarse classification, and KNN for fine classification. Specially, with a sufficient amount of data in supervised learning, MLP is combined with KNN for fine classification;

TABLE 1
Targeted Problems of Different Methods

	many-class	few-class
few-shot	**MahiNet (ours)**	MAML, Matching Net, *etc.*
many-shot	ResNet, DenseNet, Inception, VGG, *etc.*	

MahiNet targets many-class few-shot problem, which is more challenging and practical than others.

and in meta-learning when less data is available, we also use KNN for coarse classification to assist MLP.

To make the KNN learnable and adaptive to new classes with few-shot data, we train an attention module to provide the similarity/distance metric used in KNN, and a re-writable memory of limited size to store and update the KNN support set during training. In supervised learning, it is necessary to maintain and update a relatively small memory (e.g., 7.2 percent of the dataset in our experiment) by selecting a few representative samples, because conducting a KNN search over all available training samples is too expensive in computation. In meta-learning, the attention module can be treated as a meta-learner that learns a universal similarity metric across different tasks.

Since the commonly used datasets in meta-learning do not have hierarchical multi-label annotations, we randomly extract a large subset of ImageNet [10] "*mcfs*ImageNet" as a benchmark dataset specifically designed for MCFS problem. Each sample in *mcfs*ImageNet has two labels: a coarse class label and a fine class label. It contains 139,346 images from 77 non-overlapping coarse classes composed of 754 randomly sampled fine classes, each has only ≈ 180 images, which might be further splitted for training, validation and test. The imbalance between different classes in the original ImageNet are preserved to reflect the imbalance in practical problems. Similarly, we further extract "*mcfs*Omniglot" from Omniglot [11] for the same purpose. We will make them publicly available later. In fully supervised learning experiments on these two datasets, MahiNet outperforms the widely used ResNet [1] (for fairness, MahiNet uses the same network (i.e., ResNet) as its backbone network). In meta-learning scenario where each test task covers many classes, it shows more promising performance than popular few-shot methods including prototypical networks [8] and relation networks [12], which are specifically designed for few-shot learning.

Our contributions can be concluded as: In the new version, we conclude our contributions at the end of introduction as follows: Our contributions can be concluded as: 1) We address a new problem setting called "many-class few-shot learning (MCFS)" that has been widely encountered in practice. Compared to the conventional "few-class few-shot (as in most few-shot learning methods)" or "many-class many-shot (as in most supervised learning methods)" settings, MCFS is more practical and challenging but has been rarely studied in the ML community. 2) To alleviate the challenge of "many-class few-shot", we propose to utilize the knowledge of a predefined class hierarchy to train a model that takes the relationship between classes into account when generating a classifier over a relatively large number of classes. 3) To empirically justify whether our model can improve the MCFS problem by using class hierarchy information, we extract two new datasets from existing benchmarks, each coupled with a class hierarchy on reasonably specified classes. In experiments, we show that our method outperforms the other baselines in MCFS setting.

2 Related Works

2.1 Few-Shot Learning

Generative models [13] were trained to provide a global prior knowledge for solving the one-shot learning problem. With the advent of deep learning techniques, some recent approaches [14], [15] use generative models to encode specific prior knowledge, such as strokes and patches. More recently, works in [9] and [16] have applied hallucinations to training images and to generate more training samples, which converts a few-shot problem to a many-shot problem.

Meta-learning has been widely studied to address the few-shot learning problems for fast adaptation to new tasks with only few-shot data. Meta-learning was first proposed in the last century [17], [18], and has recently brought significant improvements to few-shot learning, continual learning and online learning [19]. For example, the authors of [20] proposed a dataset of characters "Omniglot" for meta-learning while the work in [21] apply a Siamese network to this dataset. A more challenging dataset "*mini*ImageNet" [5], [7] was introduced later. *mini*ImageNet is a subset of ImageNet [10] and has more variety and higher recognition difficulty compared to Omniglot. Researchers have also studied a combination of RNN and attention based method to overcome the few-shot problem [5]. More recently, the method in [8] was proposed based on metric learning to learn a shared KNN [22] classifier for different tasks. In contrast, the authors of [6] developed their approach based on the second order optimization where the model can adapt quickly to new tasks or classes from an initialization shared by all tasks. The work in [23] addresses the few-shot image recognition problem by temporal convolution, which sequentially encodes the samples in a task. More recently, meta-learning strategy has been studied to solve other problems and tasks, such as using meta-learning training strategy to help design a more efficient unsupervised learning rule [24]; mitigating the low-resource problems in neural machine translation task [25]; alleviating the few annotations problem in object detection problem [26].

Our model is closely related to prototypical networks, in which every class is represented by a prototype that averages the embedding of all samples from that class, and the embedding is achieved by applying a shared encoder, i.e., the meta-learner. It can be modified to generate an adaptive number of prototypes [27] and to handle extra weakly-supervised labels [28], [29] on a category graph. Unlike these few-shot learning methods that produces a KNN classifier defined by the prototypes, our model produces multi-labels or multi-output predictions by combining the outputs of an MLP classifier and a KNN classifier, which are trained in either supervised learning or few-shot learning settings.

2.2 Multi-Label Classification

Multi-label classification aims to assign multiple class labels to every sample [30]. One solution is to use a chain of classifiers to turn multi-label problem into several binary classification problems [31], [32], [33]. Another solution treats multi-label classification as multi-class classification over all possible subsets of labels [34]. Other works learn an embedding or encoding space using metric learning approaches [35], feature-aware label space encoding, label propagation, etc. The work in [36] proposes a decision trees method for multi-label annotation problems. The efficiency can be improved by using an ensemble of a pruned set of labels [37]. The applications include multi-label classification for textual data [38], multi-label learning from crowds [39] and disease resistance prediction [40]. Our approach differs from these works in that the multi-output labels and predictions serve as auxiliary information to improve the many-class few-shot classification over fine classes.

2.3 Hierarchical Classification

Hierarchical classification is a special case of multi-label or multi-output problems [41], [42]. It has been applied to traditional supervised learning tasks [43], such as text classification [44], [45], community data [46], case-based reasoning [47], popularity prediction [48], supergraph search in graph databases [49], road networks [50], image annotation and robot navigation. However, to the best of our knowledge, our paper is the first work successfully leveraging class hierarchy information in few-shot learning and meta-learning tasks. Previous methods such as [51], have considered the class hierarchy information but failed to achieve improvement and thus did not integrate it in their method, whereas our method successfully leverage the hierarchical relationship between classes to improve the classification performance by using a memory-augmented model.

Similar idea of using hierarchy for few-shot learning has been studied in [52], [53], which learns to cluster semantic information and task representation, respectively. [28] utilizes coarsely labeled data in the hierarchy as weakly supervised data rather than multi-label annotations for few-shot learning. In computational biology, hierarchy information has also been found helpful in gene function prediction, where two main taxonomies are Gene Ontology and Functional Catalogue. [54] proposed a truth path rule as an ensemble method to govern both taxonomies. [55] shows that the key factors for the success of hierarchical ensemble methods are: (1)the integration and synergy among multi-label hierarchy, (2) data fusion, (3) cost-sensitive approaches, and (4) the strategy of selecting negative examples. [56] and [57] address the incomplete annotations of proteins using label hierarchy (as studied in this paper) by following the idea of few-shot learning. Specifically, they learn to predict the new gene ontology annotations by Bi-random walks on a hybrid graph [56] and downward random walks on a gene ontology [57], respectively.

3 Targeted Problem and Proposed Model

In this section, we first introduce the formulation of many-class few-shot problem in Section 3.1. Then we generally elaborate our network architecture in Section 3.2. Details for how to learn a similarity metric for a KNN classifier with attention module and how to update the memory (as a support set of the KNN classifier) are given in Sections 3.3 and 3.4 respectively.

3.1 Problem Formulation

We study supervised learning and meta-learning. Given a training set of n samples $\mathbb{D} = \{(\boldsymbol{x}_i, y_i, z_i)\}_{i=1}^n$, where each sample $\boldsymbol{x}_i \in \mathbb{X}$ is associated with multiple labels. For simplicity, we assume that each training sample is associated with two labels: a fine-class label $y_i \in \mathbb{Y}$ and a coarse-class label $z_i \in \mathbb{Z}$. The training data is sampled from a data set $\mathbb{D}$, i.e., $(\boldsymbol{x}_i, y_i, z_i) \sim \mathbb{D}$. Here, $\mathbb{X}$ denotes the set of samples; $\mathbb{Y}$ denotes the set of all the fine classes, and $\mathbb{Z}$ denotes the set of all the coarse classes. To define a class hierarchy for $\mathbb{Y}$ and $\mathbb{Z}$, we further assume that each coarse class $z \in \mathbb{Z}$ covers a subset of fine classes $\mathbb{Y}_z$, and that distinct coarse classes are associated with disjoint subsets of fine classes, i.e., for any $z_1,\ z_2 \in \mathbb{Z}$, we have $\mathbb{Y}_{z_1} \cap \mathbb{Y}_{z_2} = \emptyset$. Our goal is fine-class classification by using the class hierarchy information and the coarse labels of the training data. In particular, the supervised learning setting in this case can be formulated as:

$$\min_{\Theta} \mathbb{E}_{(\boldsymbol{x},y,z)\sim\mathcal{D}} - \log \Pr(y|\boldsymbol{x};\Theta), \tag{1}$$

where Θ is the model parameters and $\mathbb{E}$ refers to the expectation w.r.t. the data distribution. In practice, we solve the corresponding empirical risk minimization (ERM) during training, i.e.,

$$\min_{\Theta} \sum_{i=1}^{n} -\log \Pr(y_i|\boldsymbol{x}_i;\Theta). \tag{2}$$

In contrast, meta-learning aims to learn a meta-learner model that can be applied to different tasks. Its objective is to maximize the expectation of the prediction likelihood of a task drawn from a distribution of tasks. Specifically, we assume that each task is the classification over a subset of fine classes T sampled from a distribution $\mathcal{T}$ over all classes, and the problem is formulated as

$$\min_{\Theta} \mathbb{E}_{T\sim\mathcal{T}} \left[\mathbb{E}_{(\boldsymbol{x},y,z)\sim\mathbb{D}_T} - \log \Pr(y|\boldsymbol{x};\Theta)\right], \tag{3}$$

where $\mathbb{D}_T$ refers to the set of samples with label $y_i \in T$. The corresponding ERM is

$$\min_{\Theta} \sum_{T} \left[\sum_{i\in\mathbb{D}_T} -\log \Pr(y_i|\boldsymbol{x}_i;\Theta)\right], \tag{4}$$

where T represents a task (defined by a subset of fine classes) sampled from distribution $\mathcal{T}$, and $\mathbb{D}_T$ is a training set for task T sampled from $\mathbb{D}_T$.

To leverage the coarse class information of z, we write $\Pr(y|\boldsymbol{x};\Theta)$ in Eqs. (1) and (3) as

$$\Pr(y|\boldsymbol{x};\Theta) = \sum_{z\in\mathcal{Z}} \Pr(y|z,\boldsymbol{x};\Theta_f) \Pr(z|\boldsymbol{x};\Theta_c), \tag{5}$$

where Θ_f and Θ_c are the model parameters for fine classifier and coarse classifier, respectively.[1] Accordingly, given a specific sample $(\boldsymbol{x}_i, y_i, z_i)$ with its ground truth labels for coarse and fine classes, we can write $\Pr(y_i|\boldsymbol{x}_i;\Theta)$ in Eqs. (2) and (4) as follows.

1. For simplicity, we neglect model parameters θ^{CNN} for feature extraction here.

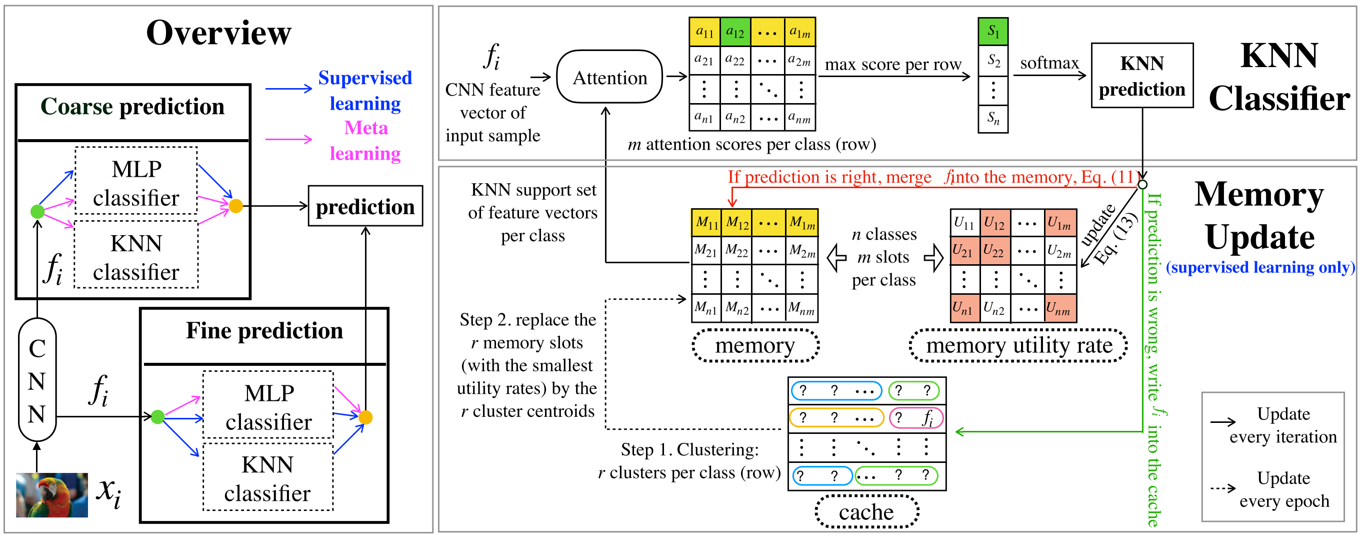

Overview
Coarse prediction
Supervised learning
Meta learning
MLP classifier
KNN classifier
prediction
Fine prediction
CNN
f_i
x_i
CNN feature vector of input sample
Attention
max score per row
softmax
KNN prediction
KNN Classifier
m attention scores per class (row)
If prediction is right, merge f_i into the memory, Eq. (11)
KNN support set of feature vectors per class
n classes m slots per class
memory
memory utility rate
update Eq. (13)
Memory Update (supervised learning only)
If prediction is wrong, write f_i into the cache
Step 2. replace the r memory slots (with the smallest utility rates) by the r cluster centroids
Step 1. Clustering: r clusters per class (row)
cache
Update every iteration
Update every epoch

Fig. 2. *Left:* MahiNet. The final fine-class prediction combines predictions based on multiple classes (both fine classes and coarse classes), each of which is produced by an MLP classifier or/and an attention-based KNN classifier. *Top right:* KNN classifier with learnable similarity metric and updatable support set. Attention provides a similarity metric $a_{j,k}$ between each input sample f_i and a small support set per class stored in memory $M_{j,k}$. The learning of KNN classifier aims to optimize 1) the similarity metric parameterized by the attention, detailed in Section 3.3; and 2) a small support set of feature vectors per class stored in memory, detailed in Section 3.4. *Bottom right:* The memory update mechanism. In meta-learning, the memory stores the features of all training samples of a task. In supervised learning, the memory is updated during training as follows: for each sample x_i within an epoch, if the KNN classifier produces correct prediction, f_i will be merged into the memory; otherwise, f_i will be written into a "cache". At the end of each epoch, we apply clustering to the samples per class stored in the cache, and use the resultant centroids to replace r slots of the memory with the smallest utility rate [58].

$$\Pr(y_i|\boldsymbol{x}_i;\Theta) = \Pr(y_i|z_i,\boldsymbol{x}_i;\Theta_f)\Pr(z_i|\boldsymbol{x}_i;\Theta_c). \tag{6}$$

Suppose that a DNN model already produces a logit a_y for each fine class y, and a logit b_z for each coarse class z, the two probabilities in the right hand side of Eq. (6) are computed by applying softmax function to the logit values in the following way.

$$\begin{aligned}\Pr(y_i|z_i,\boldsymbol{x}_i;\Theta_f) &= \frac{\exp(a_{y_i})}{\sum_{y\in\mathcal{Y}_{z_i}}\exp(a_y)},\\ \Pr(z_i|\boldsymbol{x}_i;\Theta_c) &= \frac{\exp(b_{z_i})}{\sum_{z\in\mathcal{Z}}\exp(a_z)}.\end{aligned} \tag{7}$$

Therefore, we integrate multiple labels (both the fine-class label and coarse-class label) in an ERM, whose goal is to maximize the likelihood of the ground truth fine-class label. Given a DNN that produces two vectors of logits a and b for fine class and coarse class respectively, we can train the DNN for supervised learning or meta-learning by solving the ERM problems in Eqs. (2) or (4) (with Eqs. (6) and (7) plugged in).

3.2 Network Architecture

To address MCFS problem in both supervised learning and meta-learning scenarios, we developed a universal model, MahiNet, as in Fig. 2. MahiNet uses a CNN to extract features from raw inputs, and then applies two modules to produce coarse-class prediction and fine-class prediction, respectively. Each module includes one or two classifiers: either an MLP or an attention-based KNN classifier or both. Intuitively, MLP performs better when data is sufficient, while the KNN classifier is more stable in few-shot scenario. Hence, we always apply MLP to coarse prediction and always apply KNN to fine prediction. In addition, we use KNN to assist MLP for the coarse module in meta-learning, and use MLP to assist KNN for the fine module in supervised learning. We develop two mechanisms to make the KNN classifier learnable and be able to quickly adapt to new tasks in the meta-learning scenario. In the attention-based KNN classifier, an attention module is trained to compute the similarity between two samples, and a re-writable memory is maintained with a highly representative support set for KNN prediction. The memory is updated during training.

Our method for learning a KNN classifier combines the ideas from two popular meta-learning methods, i.e., matching networks [7] that aim to learn a similarity metric, and prototypical networks [8] that aim to find a representative center per class for NN search. However, our method relies on an augmented memory rather than a bidirectional RNN for retrieving of NN in matching networks. In contrast to prototypical networks, which only has one prototype per class, we allow multiple prototypes as long as they can fit in the memory budget. Together these two mechanisms prevent the confusion caused by subtle differences between classes in "many-class" scenario. Notably, MahiNet can also be extended to "life-long learning" given this memory updating mechanism. We do not adopt the architecture used in [23] since it requires the representations of all historical data to be stored.

3.3 Learn a KNN Similarity Metric With an Attention

In MahiNet, we train an attention module to compute the similarity used in the KNN classifier. The attention module learns a distance metric between the feature vector f_i of a given sample x_i and any feature vector from the support set stored in the memory. Specifically, we use the dot product attention similar to the one adopted in [59] for supervised learning, and use an euclidean distance based attention for meta-learning, following the instruction from [8]. Given a sample $\boldsymbol{x}_i$, we compute a feature vector $\boldsymbol{f}_i \in \mathbb{R}^d$ by applying a backbone CNN to $\boldsymbol{x}_i$. In the memory, we maintain a support set of m feature vectors for each class, i.e., $\boldsymbol{M} \in \mathbb{R}^{C\times m\times d}$, where C is the number of classes. The KNN classifier produces the class probabilities of $\boldsymbol{x}_i$ by first calculating the attention scores between $\boldsymbol{f}_i$ and each feature vector in the memory, as follows.

$$a(\boldsymbol{f}_i, \boldsymbol{M}_{j,k}) = \frac{g(\boldsymbol{f}_i)\cdot h(\boldsymbol{M}_{j,k})}{\|g(\boldsymbol{f}_i)\|\|h(\boldsymbol{M}_{j,k})\|} \quad \text{or} \quad -\|g(\boldsymbol{f}_i) - h(\boldsymbol{M}_{j,k})\|_2, \quad \forall j \in [C],\ k \in [m], \tag{8}$$

where g and h are learnable transformations for $\boldsymbol{f}_i$ and the feature vectors in the memory.

We select the top K nearest neighbors, denoted by top_k, of $\boldsymbol{f}_i$ among the m feature vectors for each class j, and compute the sum of their similarity scores as the attention score of $\boldsymbol{f}_i$ to class j, i.e.,

$$s(\boldsymbol{f}_i, \boldsymbol{M}_j) = \sum_{k\in[m]} top_k(a(\boldsymbol{f}_i, \boldsymbol{M}_{j,k})), \quad \forall j \in [C]. \tag{9}$$

We usually find $K = 1$ is sufficient in practice. The predicted class probability is derived by applying a softmax function to the attention scores of $\boldsymbol{f}_i$ over all C classes, i.e.,

$$\Pr(y_i = j) \triangleq \frac{\exp(s(\boldsymbol{f}_i, \boldsymbol{M}_j))}{\sum_{j'=1}^{C}\exp(s(\boldsymbol{f}_i, \boldsymbol{M}_{j'}))}, \quad \forall j \in [C]. \tag{10}$$

3.4 Memory Mechanism for the Support Set of KNN

Ideally, the memory $\boldsymbol{M} \in \mathbb{R}^{C\times m\times d}$ can store all available training samples as the support set of the KNN classifier. In meta-learning, in each episode, we sample a task with C classes and m training samples per class, and store them in the memory. Due to the small amount of training data for each task, we can store all of them and thus do not need to update the memory. In supervised learning, we only focus on one task. This task usually has a large training set and it is inefficient, unnecessary and too computationally expensive to store all the training set in the memory. Hence, we set up a budget hyper-parameter m for each class. m is the maximal number of feature vectors to be stored in the memory for one class. Moreover, we develop a memory update mechanism to maintain a small memory with diverse and representative feature vectors per class (t-SNE visualization of the diversity and representability of the memory can be found in the experiment section.). Intuitively, it can choose to forget or merge feature vectors that are no longer representative, and select new important feature vectors into memory.

We will show later in experiments that a small memory can result in sufficient improvement, while the time cost of memory updating is negligible. Accordingly, the memory in meta-learning does not need to be updated according to a rule, while in supervised learning, we design the writing rule as follows. During training, for the data that can be correctly predicted by the KNN classifier, we merge its feature with corresponding slots in the memory by computing their convex combination, i.e.,

$$\boldsymbol{M}_{j,k} = \begin{cases} \gamma \times \boldsymbol{M}_{j,k} + (1-\gamma)\times \boldsymbol{f}_i, & \text{if } \hat{y}_i = y_i \\ \boldsymbol{M}_{j,k}, & \text{otherwise} \end{cases}, \tag{11}$$

where y_i is the ground truth label, and $\gamma = 0.95$ is a combination weight that works well in most of our empirical studies; for input feature vector that cannot be correctly predicted, we write it to a cache $\mathbb{C} = \{\mathbb{C}_1, \ldots, \mathbb{C}_C\}$ that stores the candidates written into the memory for the next epoch, i.e.,

$$\mathbb{C}_j = \begin{cases} \mathbb{C}_j, & \text{if } \hat{y}_i = y_i \\ \mathbb{C}_j \cup \{\boldsymbol{f}_i\}, & \text{otherwise} \end{cases}, \tag{12}$$

Concurrently, we record the utility rate of the feature vectors in the memory, i.e., how many times each feature vector being selected into the K nearest neighbor during the epoch. The rates are stored in a matrix $\boldsymbol{U} \in \mathbb{R}^{C\times m}$, and we update it as follows.

$$\boldsymbol{U}_{j,k} = \begin{cases} \boldsymbol{U}_{j,k} \times \mu, & \text{if } \hat{y}_i = y_i \\ \boldsymbol{U}_{j,k} \times \eta, & \text{otherwise} \end{cases}, \tag{13}$$

where $\mu \in (1, 2)$ and $\eta \in (0, 1)$ are hyper-parameters.

At the end of each epoch, we apply clustering to the feature vectors per class in the cache, and obtain r cluster centroids as the candidates for memory update in the next epoch. Then, for each class, we replace r feature vectors in the memory that have the smallest utility rate with the r cluster centroids.

Algorithm 1. Training MahiNet for Supervised Learning

```
Input: Training set $\mathbb{D} = \{(\boldsymbol{x}_i, y_i, z_i)\}_{i=1}^n$;
       Randomly initialized $\theta_f^{KNN}$, pre-trained $\theta^{CNN}$, $\theta_c^{MLP}$ and $\theta_f^{MLP}$;
       Hyper-parameters: memory update parameters $r$, $\gamma$, $\mu$ and $\eta$; learning rate and its scheduler;
 1: while no converge do
 2:   for mini-batch $\{(\boldsymbol{x}_i, y_i, z_i)\}_{i\in B}$ in $\mathbb{D}$ do
 3:     Compute fine-class logits $a$ and coarse-class logits $b$ from the outputs of MLP/KNN classifiers;
 4:     Apply one step of mini-batch SGD for ERM in Eq. (2) (with Eqs. (6) and (7) plugged in);
 5:     for sample in the mini-batch do
 6:       Update the memory $\boldsymbol{M}$ according to Eq. (11);
 7:       Update the utility rate $\boldsymbol{U}$ according to Eq. (13);
 8:       Expand the feature cache $\mathbb{C}$ according to Eq. (12);
 9:     end for
10:   end for
11:   for each fine class $j$ in $\mathcal{Y}$ do
12:     Find the indexes of the $r$ smallest values in $\boldsymbol{U}_j$, denoted as $\{k_1, k_2, \ldots, k_r\}$;
13:     Clustering of the feature vectors within cache $\mathbb{C}_j$ to $r$ clusters with centroids $\{c_1, c_2, \ldots, c_r\}$;
14:     Replace the $r$ memory slots by centroids: $\boldsymbol{M}_{j,k_i} = c_i$ for $i \in [r]$;
15:   end for
16: end while
```

4 Training Strategies

As shown in the network structure in Fig. 2, in supervised learning and meta-learning, we use different combinations of MLP and KNN to produce the multi-output predictions of fine-class and coarse-class. The classifiers are combined by summing up their output logits for each class, and a softmax function is applied to the combined logits to generate the class probabilities. Assume the MLP classifiers for the coarse classes and the fine classes are $\phi(\cdot;\theta_c^{MLP})$ and $\phi(\cdot;\theta_f^{MLP})$, the KNN classifiers for the coarse classes and the fine classes are $\phi(\cdot;\theta_c^{KNN})$ and $\phi(\cdot;\theta_f^{KNN})$. In supervised learning, the model parameters are θ^{CNN}, $\Theta_c = \theta_c^{MLP}$ and $\Theta_f = \{\theta_f^{MLP}, \theta_f^{KNN}\}$; in meta-learning setting, the model parameters are θ^{CNN}, $\Theta_c = \{\theta_c^{MLP}, \theta_c^{KNN}\}$ and $\Theta_f = \theta_f^{KNN}$.

According to Section 3.1, we train MahiNet for supervised learning by solving the ERM problem in Eq. (2). For meta-learning, we instead train MahiNet by solving Eq. (4). As previously mentioned, the multi-output logits (for either fine classes or coarse classes) used in those ERM problems are obtained by summing up the logits produced by the corresponding combination of the classifiers.

4.1 Training MahiNet for Supervised learning

In supervised learning, the memory update relies heavily on the clustering of the merged feature vectors in the cache. To achieve relatively high-quality feature vectors, we first pretrain the CNN+MLP model by using standard backpropagation, which minimizes the sum of cross entropy loss on both the coarse-classes and fine-classes. The loss is computed based on the logits of fine-classes and the logits of coarse-classes, which are the outputs of the corresponding MLP classifiers. Then, we fint-tune the whole model, which includes the fine-class KNN classifier, with memory update applied. Specifically, in every iteration, we first update the parameters of the model with mini-batch SGD based on the coarse-level and fine-level logits produced by the MLP and KNN classifiers. Then, the memory that stores a limited number of representative feature maps for every class is updated by the rules explained in Section 3.4, as well as the associated utility rates and feature cache. The details of the training procedure during the fine-tune stage is explained in Algorithm 1.

Algorithm 2. Training MahiNet for Meta-Learning

```
Input: Training set $\mathbb{D} = \{(x_i, y_i, z_i)\}_{i=1}^n$ and fine class set $\mathbb{Y}$;
       Parameters: randomly initialized $\theta^{CNN}$, $\theta_c^{MLP}$, $\theta_f^{MLP}$, and $\theta_f^{KNN}$;
       Hyper-parameters: learning rate, scheduler; for each class, number of queries $n_s$, support set size $n_S$;
1:  while not converge do
2:    Sample a task $T \sim \mathcal{T}$ as a subset of fine classes $T \subseteq \mathbb{Y}$.
3:    for class $j$ in $T$ do
4:      Randomly sample $n_s$ data points of class $j$ from $\mathbb{D}$ to be the support set $\mathbb{S}_j$ of class $j$.
5:      Randomly sample $n_q$ data points of class $j$ from $\mathbb{D}\backslash\mathbb{S}_j$ to be the query set $\mathbb{Q}_j$ of class $j$.
6:    end for
7:    for mini-batch from $\mathbb{Q}$ do
8:      Compute fine-class logits $a$ and coarse-class logits $b$ from the outputs of MLP/KNN classifiers;
9:      Apply one step of mini-batch SGD for ERM in Eq. (4) (with Eqs. (6) and (7) plugged in);
10:   end for
11: end while
```

4.2 Training MahiNet for Meta-Learning

When sampling the training/test tasks, we allow unseen fine classes that were not covered in any training task to appear in test tasks, but we fix the ground set of the coarse classes for both training and test tasks. Hence, every coarse class appearing in any test task has been seen during training, but the corresponding fine classes belonging to this coarse class in training and test tasks can be vary.

In the meta-learning setting, since the tasks in different iterations targets samples from different subsets of classes and thus ideally need different feature map representations [60], we do not maintain the memory during training so every class can have a fixed representation. Instead, the memory is only maintained within each iteration for a specific task and it stores feature map representations extracted from the support set of the task. The detailed training procedure can be found in Algorithm 2. In summary, we generate a training task by randomly sampling a subset of fine classes, and then we randomly sample a support set $\mathbb{S}$ and a query set $\mathbb{Q}$ from the associated samples belonging to these classes. We store the CNN feature vectors of $\mathbb{S}$ in the memory, and train MahiNet to produce correct predictions for the samples in $\mathbb{Q}$ with the help of coarse label predictions.

5 Experiments

In this section, we first introduce two new benchmark datasets specifically designed for MCFS problem, and compare them with other popular datasets in Section 5.1. Then, we show that MahiNet has better generality and can outperform the specially designed models in both supervised learning (in Section 5.2) and meta-learning (in Section 5.3) scenarios. We also present an ablation study of MahiNet to show the improvements brought by different components and how they interact with each other. In Section 5.4, we report the comparison with the variants of baseline models in order to show the broad applicability of our strategy of leveraging the hierarchy structures. In Section 5.6, we present an analysis of the computational costs caused by the augmented memory in MahiNet.

5.1 Two New Benchmarks for MCFS Problem: *mcfs*ImageNet & *mcfs*Omniglot

Since existing datasets for both supervised learning and meta-learning settings neither support multiple labels nor fulfill the requirement of "many-class few-shot" settings, we propose two novel benchmark datasets specifically for MCFS Problem, i.e., *mcfs*ImageNet & *mcfs*Omniglot.[2] The samples in them are annotated by multiple labels of different granularity. We compare them with several existing datasets in Table 2. Our following experimental study focuses on these two datasets.

ImageNet [10] is one of the most widely used large-scale benchmark dataset for image classification. Although it provides hierarchical information about class labels, it cannot be directly used to test the performance of MCFS learning methods. One main reason is that a fine class in ImageNet

2. More details about the class hierarchy split information for mcfsImageNet and mcfsOmniglot can be found at: https://github.com/liulu112601/MahiNet

TABLE 2
Comparison of the Statistics for *mcfs*ImageNet, *mcfs*Omniglot, and Previously Popular Datasets for Supervised Learning and Few-Shot Learning

	Meta-Learning						Supervised Learning				#image	image size
	Train		Val		Test		Train		Test			
	#c	#f	#c	#f	#c	#f	#c	#f	#c	#f		
ImageNet-1k	-	-	-	-	-	-	1	1000	1	1000	1.43M	224
*mini*ImageNet	1	64	1	16	1	20	-	-	-	-	0.06M	84
Omniglot	33	1028	5	172	13	423	-	-	-	-	0.03M	28
*mcfs***Omniglot**	50	973	50	244	50	1624	-	-	-	-	0.03M	28
*mcfs***ImageNet**	77	482	61	120	68	152	77	754	77	754	0.14M	112

We propose datasets mcfs*ImageNet,* mcfs*Omniglot, in which every image is annotated by multiple labels: a coarse label and a fine label. "#c" and "#f" denote the number of coarse classes and fine classes, respectively. "-" means "not applicable".*

may belong to multiple coarse classes, which introduces unnecessary noise in narrowing down the fine-class candidates and is outside the scope of MCFS problem, in which each sample has only one unique coarse-class label. In addition, ImageNet does not satisfy the criteria of "few-shot" per class: it has around 1,430 images per class on average. *mini*ImageNet [7] is a widely used benchmark dataset in meta-learning community to test the performance on few-shot learning tasks. *mini*ImageNet is a subset extracted from ImageNet; however, its data are collected from only 80 fine classes, which is much less than "many-class" usually refers to in practice. In addition, it does not provide an associated class hierarchy.

Hence, to develop a benchmark dataset specifically for the purpose of testing the performance of MCFS learning, we extracted a subset of images from ImageNet and created a dataset called "*mcfs*ImageNet". Table 2 compares the statistics of *mcfs*ImageNet with several benchmark datasets. Comparing to the original ImageNet, we avoided selecting the samples that belong to more than one coarse classes into *mcfs*ImageNet in order to meet the class hierarchy requirements of MCFS problem, i.e., each fine class only belongs to one coarse class. Compared to *mini*ImageNet, *mcfs*ImageNet is about $5\times$ larger, and covers 754 fine classes in total, which is much more than the 80 fine classes in *mini*ImageNet. Moreover, on average, each fine class only has ~ 185 images for training and test, which matches the typical MCFS scenarios in practice. Additionally, the number of coarse classes in *mcfs*ImageNet is 77, which is much less than 754 of the fine classes. This is consistent with the data properties found in many practical applications, where the coarse-class labels can only provide weak supervision, but each coarse class has sufficient training samples. Further, we avoided selecting coarse classes which are too broad or contain too many different fine classes. For example, the "Misc" class in ImageNet has 20400 sub-classes, and includes both animal (3998 sub-classes) and plant (4486 sub-classes). This kind of coarse label covers too many classes and cannot provide valuable information to distinguish different fine classes.

Omniglot [11] is a small hand-written character dataset with two-level class labels. However, in their original training/test splitting, the test set contains new coarse classes that are not covered by the training set, since this weakly labeled information is not supposed to be utilized during the training stage. This is inconsistent with the MCFS settings, in which all the coarse classes are exposed in training, but new fine classes can still emerge during test. Therefore, we re-split Omniglot to fulfill the MCFS problem requirement.

5.2 Supervised Learning Experiments

5.2.1 Setup

We use ResNet18 [1] as the backbone CNN. The transformation functions g and h in the attention module are two fully connected layers followed by group normalization [62] with a residual connection. We set the memory size to $m = 12$ and the number of clusters to $r = 3$, which can achieve a better trade-off between the memory cost and performance. Batch normalization [63] is applied after each convolution and before activation. During pre-training, we apply the cross entropy loss on the probability predictions in Eq. (7). During fine-tuning, we fix the θ^{CNN}, θ_c^{MLP}, and θ_f^{MLP} to ensure the fine-tuning process is stable. We use SGD with a mini-batch size of 128 and a cosine learning rate scheduler with an initial learning rate 0.1. $\mu = 1.05$, $\eta = 0.95$, a weight decay of 0.0001, and a momentum of 0.9 are used. We train the model for 100 epochs during pre-training and 90 epochs for the fine-tuning. All hyperparameters are tuned on 20 percent samples randomly sampled from the training set. we use 20 percent data randomly sampled from the training set to serve as the validation set to tune all the hyperparameters.

5.2.2 Experiments on mcfs*ImageNet*

Table 3 compares MahiNet with the supervised learning model (i.e., ResNet18) and meta-learning model (i.e., prototypical networks) in the setting of supervised learning. The results show that MahiNet outperforms the specialized models, such as ResNet18 in MCFS scenario.

We also show the number of parameters for every method, which indicates that our model only has 9 percent more parameters than the baselines. Even using the version without KNN (which has the same number of parameters as baselines) can still outperform them.

Prototypical networks have been specifically designed for few-shot learning. Although it can achieve promising performance on few-shot tasks each covering only a few classes and samples, it fails when directly used to classify many classes in the classical supervised learning setting, as shown in Table 3. A primary reason, as we discussed in Section 1, is that the resulted prototype suffers from high variance (since it is simply the average over an extremely few amount of samples per class) and cannot distinguish

TABLE 3
The Performance (Test Accuracy) Comparison of Different Models in the Setting of Supervised Learning on *mcfs*ImageNet

Model	Hierarchy	#Params (MB)	Accuracy
Prototypical Net [8]	N	11	2.7%
ResNet18 [1]	N	11	48.6%
MahiNet w/o KNN	Y	11	49.1%
MahiNet (ours)	Y	12	**49.9%**

each class from tens of thousands of other classes (few-shot learning task only needs to distinguish it from 5-10 classes). Hence, dedicated modifications is necessary to make prototypical networks also work in the supervised learning setting. One motivation of this paper is to develop a model that can be trained and directly used in both settings.

The failure of Prototypical networks in supervised learning setting is not too surprising since for prototypical network the training and test task in the supervised setting is significantly different. We keep its training stage the same (i.e., target a task of 5 classes in every episode). In the test stage, we use the prototypes as the per-class means of training samples in each class to perform classification over all available classes on the test set data. Note the number of classes in the training tasks and the test task are significantly different, which introduces a large gap between training and test that leads to the failure. This can also be validated by observing the change on performance when we intentionally reduce the gap: the accuracy improves as the number of classes in each training task increases.

Another potential method to reduce the training-test gap is to increase the number of classes/ways per task to the same number of classes in supervised learning. However, prototypical net requires an prohibitive memory cost in such many class setting, i.e, it needs to keep 754(number of classes) $\times$ 2(1 for support set, 1 for query set)=1512 samples in memory for 1 shot learning setting and 754 $\times$ 6(5 for support set, 1 for query set) = 4536 samples for 5 shots setting. Unlike classical supervised learning models which only needs to keep the model parameters in memory, prototypical net needs to build prototypes on the fly from the few-shot samples in the current episode, so at least one sample should be available to build the prototypes and at least one sample for query.

To separately measure the contribution of the class hierarchy and the attention-based KNN classifier, we conduct an ablation study that removes the KNN classifier from MahiNet. The results show that MahiNet still outperforms ResNet18 even when only using the extra coarse-label information and an MLP classifier for fine classes during training. Using a KNN classifier further improves the performance since KNN is more robust in the few-shot fine-level classifications. In supervised learning scenario, memory update is applied in an efficient way. 1) For each epoch, the average clustering time is 30s and is only 7.6 percent of the total epoch time (393s). 2) Within an epoch, the memory update time (0.02s) is only 9 percent of the total iteration time (0.22s).

5.3 Meta-Learning Experiments

5.3.1 Setup

We use the same backbone CNN (i.e., ResNet18) and transformation functions (i.e., g, and h) for attention as in supervised learning. In each task, we sample the same number of classes for training and test, and follow the training procedure in [8]. For a fair comparison, all baselines and our model are trained on the same number of classes for all training tasks, since increasing the number of classes in training tasks improves the performance as indicated in [8]. We initialize the learning rate as 10^{-3} and halve it after every 10k iterations. Our model is trained by ADAM [64] using a mini-batch size of 128 (only in the pre-training stage), a weight decay of 0.0001, and a momentum of 0.9. We train the model for 25k iterations in total. Given the class hierarchy, the training objective sums up the cross entropy losses computed over the coarse class and over the fine classes, respectively. All hyper-parameters are tuned based on the validation set introduced in Table 2. Every baseline and our method use the same training set and test test, and the test set is inaccessible during training so any data leakage is prohibited. In every experimental setting, we make the comparison fair by applying the same backbone, same training and test data, the same task setting (we follow the most popular N way K shot problem setting) and the same training strategy (including how to sample tasks and the value of N and K for meta-learning, since higher N and K generally lead to better performance as indicated in [8]).

5.3.2 Experiments on mcfs*ImageNet*

Table 4 shows that MahiNet outperforms the supervised learning baseline (ResNet18) and the meta-learning baseline (Prototypical Net). For ResNet18, we use the fine-tuning introduced in [6], in which the trained network is first fine-tuned on the training set (i.e., the support set) provided in the new

TABLE 4
Test Accuracy (%) of Different Approaches in Meta-Learning Scenario on *mcfs*ImageNet

Model	Hierarchy	#params (MB)	5-10	20-5	20-10	50-5	50-10
ResNet18[1]	N	11	60.7	58.6	67.2	48.9	56.8
Prototypical Net [8]	N	11	78.48±0.66	67.78±0.37	70.11±0.38	57.74±0.24	62.12±0.24
Relation Net [12]	N	11	74.12±0.78	52.66±0.43	55.45±0.46	N/A	N/A
MAML [6]	N	11	61.67±0.01	47.24±0.01	48.10±0.00	11.43±0.00	11.88±0.00
Reptile [61]	N	11	36.21±0.01	29.13±0.01	15.23±0.01	18.03±0.00	9.16±0.00
MahiNet (Ours)	Y	12	**80.74**±0.66	**70.11**±0.41	**73.50**±0.36	**58.80**±0.24	**62.80**±0.24

Both the average accuracy over 600 test episodes and the corresponding 95 percent confidence intervals are reported. In the first row, "n-k" represents n-way (class) k-shot classification task. e.g., "20-10" refers to "20-way 10-shot". To make a fair comparison, all models use the same backbone: ResNet18. In 50-way experiments, Relation Net stops to improve after the first few iterations and fails to achieve comparable performance (more details in Section 5.4).

TABLE 5
Ablation Studies of MahiNet in the Meta-Learning Setting

Memory		Attention	Hierarchy	MLP-classifier	KNN-classifier	5-10	20-5	20-10	50-5	50-10
Mem-1	Mem-2									
✓				✓	✓	79.04±0.67	68.46±0.38	71.13±0.38	58.09±0.24	62.18±0.22
	✓			✓	✓	77.41±0.71	66.89±0.40	71.72±0.37	55.25±0.23	59.38±0.23
	✓	✓		✓	✓	76.85±0.67	66.43±0.41	70.01±0.38	55.13±0.23	59.22±0.23
✓	✓	✓		✓	✓	78.27±0.68	67.03±0.41	71.20±0.37	57.98±0.24	62.40±0.23
✓	✓	✓	✓		✓	79.61±0.67	69.49±0.40	72.52±0.37	58.48±0.24	62.62±0.24
✓	✓	✓	✓	✓		79.06±0.66	69.10±0.41	72.15±0.36	58.30±0.23	62.51±0.24
✓		✓	✓	✓	✓	80.64±0.64	68.99±0.40	72.78±0.37	58.56±0.25	62.70±0.24
✓	✓	✓	✓	✓	✓	**80.74**±0.66	**70.11**±0.41	**73.50**±0.36	**58.80**±0.24	**62.80**±0.24

Both the average accuracy over 600 test episodes and the corresponding 95 percent confidence intervals are reported. In the first row, "n-k" represents n-way (class) k-shot classification task. "Mem-1" stores the average feature of all training samples for each class in one task; "Mem-2" stores all features of the training samples in one task.

TABLE 6
Test Accuracy (%) of Basedline, Our Method and its Variants in the Meta-Learning Scenario on *mcfs*Omniglot

Model	Hierarchy	5-5	20-5	50-5
Reptile [61]	N	54.70±0.02	17.10±0.01	8.34±0.02
MAML [6]	N	74.60±0.01	21.61±0.00	9.27±0.20
Prototypical Net [8]	N	99.10±0.15	98.84±0.11	97.94±0.08
MahiNet (Mem-1)	N	99.17±0.16	98.82±0.11	97.96±0.09
MahiNet (Mem-3)	N	99.31±0.18	98.89±0.11	97.93±0.09
MahiNet (Mem-3)	Y	**99.40**±0.15	**99.00**±0.16	**98.10**±0.17

Both the average accuracy over 600 test episodes and the corresponding 95 percent confidence intervals are reported. In the first row, "n-k" represents n-way (class) k-shot classification task. e.g., "20-10" refers to "20-way 10-shot".

task in the test stage and then tested on the associated query set. To evaluate the contributions of each component in MahiNet, we show results of several variants in Table 5. "Attention" refers to the parametric functions for g and h, otherwise we use identity mapping. "Hierarchy" refers to the assist of class hierarchy. For a specific task, "Mem-1" stores the average embedding over all training samples for each class in one task; "Mem-2" stores all the embeddings of the training samples in one task; "Mem-3" is the union of "Mem-1" and "Mem-2". Table 5 implies: (1) Class hierarchy information can incur steady performance across all tasks; (2) Combining "Mem-1" and "Mem-2" outperforms using either of them independently; (3) Attention brings significant improvement to MCFS problem only when being trained with class hierarchy. Because the data is usually insufficient to train a reliable similarity metric to distinguish all fine classes, but distinguishing a few fine classes in each coarse class is much easier. The attention module is merely parameterized by the two linear layers g and h, which are usually not expressive enough to learn the complex metric without the side information of class hierarchy. With the class hierarchy as a prior knowledge, the attention module only needs to distinguish much fewer fine classes within each coarse class, and the learned attention can faithfully reflect the local similarity within each coarse class. We also show the number of parameters for every method in the table. Our method only has 9 percent more parameters compared to baselines.

5.3.3 *Experiments on* mcfs*Omniglot*

We conduct experiments on the secondary benchmark *mcfs*Omniglot. We use the same training setting as for *mcfs*ImageNet. Following [65], *mcfs*Omniglot is augmented with rotations by multiples of 90 degrees. In order to make an apple-to-apple comparison to other baselines, we follow the same backbone used in their experiments. Therefore, we use four consecutive convolutional layers, each followed by a batch normalization layer, as the backbone CNN and compare MahiNet with prototypical networks as in Table 6. We do ablation study on MahiNet with/without hierarchy and MahiNet with different kinds of memory. "Mem-3", i.e., the union of "Mem-1" and "Mem-2", outperforms "Mem-1", and "Attention" mechanism can improve the performance. Additionally, MahiNet outperforms other compared methods, which indicates the class hierarchy assists to make more accurate predictions. In summary, experiments on the small-scale and large-scale datasets show that class hierarchy brings a stable improvement.

5.4 Comparison to Variants of Relation Net

5.4.1 *Relation Network With Class Hierarchy*

In order to demonstrate that the class hierarchy information and the multi-output model not only improves MahiNet but also other existing models, we train relation network with class hierarchy in the similar manner as we train MahiNet. The results are shown in Table 7. It indicates that the class hierarchy also improves the accuracy of relation network by more than 1 percent, which verifies the advantage and generality of using class hierarchy in other models. Although the relation net with class hierarchy achieves a better performance, MahiNet still outperforms this improved variant due to its advantage in its model architecture, which

TABLE 7
The Improvement of Class Hierarchy on Relation Network on *mcfs*ImageNet

Model	Hierarchy	5 way 5 shot	5 way 10 shot
Relation Net [12]	N	63.02±0.87	74.12±0.78
Relation Net	Y	66.82±0.86	75.31±0.90
MahiNet (Mem-3)	Y	**74.98**±0.75	**80.74**±0.66

The average test accuracy over 600 test episodes and the corresponding 95 percent confidence intervals are reported.

are specially designed for utilizing the class hierarchy in the many-class few-shot scenario.

5.4.2 *Relation Network in Many-Class Setting*

When relation network is trained in many-class settings using the same training strategy from [12], we found that the network usually stops to improve and stays near a sub-optimal solution. Specifically, after the first few iterations, the training loss still stays at a high level and the training accuracy still stays low. They do not change too much even after hundreds of iterations, and this problem cannot be solved after trying different initial learning rates. The training loss quickly converges to around 0.02 and the training accuracy stays at $\sim 2\%$ no matter what learning rate is applied to the training procedure. Relation net on other low way settings have a competitive performance as shown in Table 7, which is also the settings reported in their papers. We use the same hyper-parameters as their paper for both few-class and many-class settings and the difference is only the parameter of the number of way. Even if relation network performs well in the few-class settings, its performance dramatically degrade when given the challenging high-way settings. One potential reason for this is that relation net uses MSE loss to regress the relation score to the ground truth: matched pairs have similarity 1 and mismatched pair have similarity 0. When the number of ways increases, the imbalance between the number of matched pairs and unmatched pairs grows and training is mainly to optimize the loss generated by the mismatched pairs instead of the matched ones, so that the objective focuses more on how to avoid mismatching instead of matching, i.e., predicting correctly, and makes the accuracy more difficult to get improvement. This suggests that relation networks may need more tricky way to train it successfully in a many-class setting compared to the few-class settings and traditional few-class few-shot learning approaches may not be directly applied in many-class few-shot problems. This is also one of the drawbacks for relation net and how to make the model more robust to hyper-parameter tuning is beyond the scope of our paper.

5.5 Analysis of Hierarchy

The main reasons of the improvement in our paper are: (1) our model can utilize prior knowledge from the class hierarchy; (2) we use an effective memory update scheme to iteratively and adaptively refine the support set. We show how much improvement the class hierarchy can bring to our model in Table 5 and to other baselines in Table 7. The upper half of Table 5 are variants of our model without using hierarchy while the lower half are variants with hierarchy. It shows that the variants with hierarchy consistently outperform the ones without using hierarchy. To see whether hierarchy can help to improve other baselines, in Table 7, we add hierarchy information to relation net (while keeping the other components the same) and it achieves 1-4 percent improvement on accuracy.

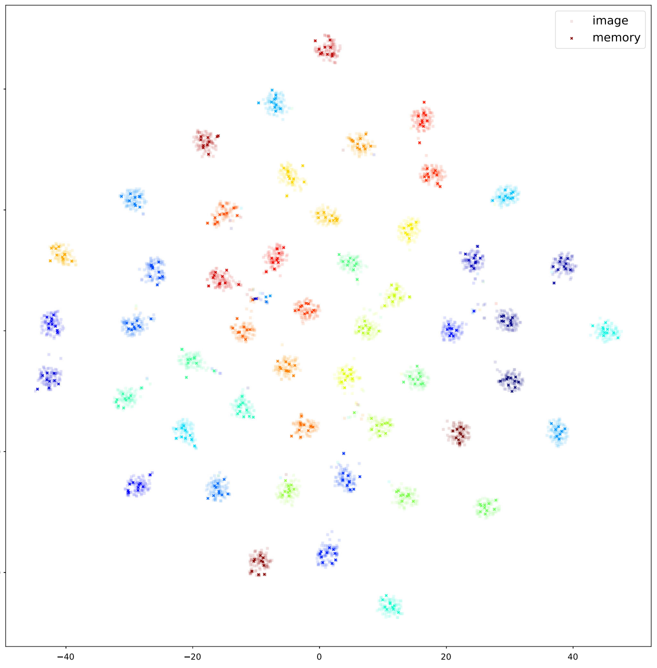


Fig. 3. The t-SNE visualization for memory maintained in the setting of supervised learning. We randomly sample 50 out of 754 fine classes shown as different colors. The sampled image feature of the training samples and the stored memory feature for one class share the same color while the image feature has higher transparency. For each class, the image features surround the memory feature and features from different classes are dispersed from each other for classification.

5.6 Analysis of Augmented Memory

5.6.1 *Visualization*

In order to show how representative and diverse the feature vectors selected into the memory slots are, we visualize them together with the unselected images' feature vectors using t-SNE in Fig. 3 [66]. In particular, we randomly sample 50 fine classes marked by different colors. Within every class, we show both the feature vectors selected into the memory and the feature vectors of the original images from the same class. The features in the memory (crosses with a lower transparency) can cover most areas where the image features (dots of a higher transparency) are located. It implies that the highly selected feature vectors in memory are diverse and sufficiently representative of the whole class.

5.6.2 *Memory Cost*

In the supervised learning experiments, the memory load required by MahiNet is only $754 \times 12/125,321 = 7.2\%$ (12 samples per class for all the 754 fine classes, while the training set includes 125,321 images in total) of the space needed to store the whole training set. We tried to increase the memory size to about 10 percent of the training set, but the resultant improvement on performance is negligible compared to the extra computation costs, meaning that increasing the memory costs is unnecessary for MahiNet. In contrast, for meta-learning, in each task, every class only has few-shot samples, so the memory required to store all the samples is very small. For example, in the 20-way 1-shot setting, we only need to store 20 feature vectors in the memory. Therefore, our memory costs in both supervised learning and meta-learning scenarios can be kept small and the proposed method is memory-efficient.

6 Conclusion

In this paper, we study a new problem "many-class few-shot" (MCFS), which is a more challenging problem commonly encountered in practice compared to the previously studied "many-class many-shot", "few-class few-shot" and "few-class many-shot" problems. We address the MCFS problem by training a multi-output model producing coarse-to-fine predictions, which leverages the class hierarchy information and explore the relationship between fine and coarse classes. We propose "Memory-Augmented Hierarchical-Classification Network (MahiNet)", which integrates both MLP and learnable KNN classifiers with attention modules to produce reliable prediction of both the coarse and fine classes. In addition, we propose two new benchmark datasets with a class hierarchy structure and multi-label annotations for the MCFS problem, and show that MahiNet outperforms existing methods in both supervised learning and meta-learning settings on the benchmark datasets without losing advantages in efficiency.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[2] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4700–4708.

[3] M. Andrychowicz *et al.*, "Learning to learn by gradient descent by gradient descent," in *Proc. Advances Neural Inf. Process. Syst.*, 2016, pp. 3981–3989.

[4] K. Li and J. Malik, "Learning to optimize," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–13. [Online]. Available: https://openreview.net/forum?id=ry4Vrt5gl

[5] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–11. [Online]. Available: https://openreview.net/forum?id=rJY0-Kcll

[6] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[7] O. Vinyals *et al.*, "Matching networks for one shot learning," in *Proc. Advances Neural Inf. Process. Syst.*, 2016, pp. 3630–3638.

[8] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proc. Advances Neural Inf. Process. Syst.*, 2017, pp. 4077–4087.

[9] M. Douze, A. Szlam, B. Hariharan, and H. Jégou, "Low-shot learning with large-scale diffusion," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 3349–3358.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[11] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum, "One shot learning of simple visual concepts," in *Proc. Annu. Meeting Cogn. Sci. Soc.*, 2011, pp. 2568–2573.

[12] F. S. Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1199–1208.

[13] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 594–611, Apr. 2006.

[14] A. Wong and A. L. Yuille, "One shot learning via compositions of meaningful patches," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1197–1205.

[15] B. M. Lake, R. R. Salakhutdinov, and J. Tenenbaum, "One-shot learning by inverting a compositional causal process," in *Proc. Advances Neural Inf. Process. Syst.*, 2013, pp. 2526–2534.

[16] Y. Wang, R. Girshick, M. Hebert, and B. Hariharan, "Low-shot learning from imaginary data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7278–7286.

[17] D. K. Naik and R. Mammone, "Meta-neural networks that learn by learning," in *Proc. Int. Joint Conf. Neural Netw.*, 1992, pp. 437–442.

[18] J. Schmidhuber, "Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook," Ph.D. dissertation, Dept. Institute Informatics, Tech. Univ. Munich, Munich, Germany, 1987.

[19] P. Zhao, Y. Zhang, M. Wu, S. C. Hoi, M. Tan, and J. Huang, "Adaptive cost-sensitive online classification," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 2, pp. 214–228, Feb. 2019.

[20] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[21] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Proc. Int. Conf. Mach. Learn. Workshop*, 2015, pp. 1–8. [Online]. Available: https://github.com/tensorfreitas/Siamese-Networks-for-One-Shot-Learning#references

[22] W. Yu, X. Lin, W. Zhang, J. Pei, and J. A. McCann, "SimRank*: Effective and scalable pairwise similarity search based on graph topology," *VLDB J.*, Jan. 2019. [Online]. Available: https://doi.org/10.1007/s00778-018-0536-3

[23] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–17. [Online]. Available: https://openreview.net/forum?id=B1DmUzWAW

[24] L. Metz, N. Maheswaranathan, B. Cheung, and J. Sohl-Dickstein, "Learning unsupervised learning rules," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–27. [Online]. Available: https://openreview.net/forum?id=HkNDsiC9KQ

[25] J. Gu, Y. Wang, Y. Chen, K. Cho, and V. O. Li, "Meta-learning for low-resource neural machine translation," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2018, pp. 3622–3631.

[26] E. Schwartz *et al.*, "RepMet: Representative-based metric learning for classification and one-shot object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5192–5201.

[27] K. R. Allen, E. Shelhamer, H. Shin, and J. B. Tenenbaum, "Infinite mixture prototypes for few-shot learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 232–241.

[28] L. Liu, T. Zhou, G. Long, J. Jiang, L. Yao, and C. Zhang, "Prototype propagation networks (PPN) for weakly-supervised few-shot learning on category graph," in *Proc. Int. Joint Conf. AI*, 2019, pp. 3015–3022. [Online]. Available: https://www.ijcai.org/Proceedings/2019/0418.pdf

[29] L. Liu, T. Zhou, G. Long, J. Jiang, and C. Zhang, "Learning to propagate for graph meta-learning," in *Proc. Advances Neural Inf. Process. Syst.*, 2019, pp. 1037–1048.

[30] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Int. J. Data Warehousing Mining*, vol. 3, no. 3, pp. 1–13, 2007.

[31] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2009, pp. 254–269.

[32] J. Read, L. Martino, and D. Luengo, "Efficient monte carlo methods for multi-dimensional learning with classifier chains," *Pattern Recognit.*, vol. 47, no. 3, pp. 1535–1546, 2014.

[33] J. Read, L. Martino, P. M. Olmos, and D. Luengo, "Scalable multi-output label prediction: From classifier chains to classifier trellises," *Pattern Recognit.*, vol. 48, no. 6, pp. 2096–2109, 2015.

[34] N. Spolaôr, E. A. Cherman, M. C. Monard, and H. D. Lee, "A comparison of multi-label feature selection methods using the problem transformation approach," *Electron. Notes Theor. Comput. Sci.*, vol. 292, pp. 135–151, 2013.

[35] W. Liu, D. Xu, I. Tsang, and W. Zhang, "Metric learning for multi-output tasks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 408–422, Feb. 2019.

[36] W. Liu and I. W. Tsang, "Making decision trees feasible in ultrahigh feature and label dimensions," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 2814–2849, 2017.

[37] J. Read, B. Pfahringer, and G. Holmes, "Multi-label classification using ensembles of pruned sets," in *Proc. IEEE Int. Conf. Data Mining*, 2008, pp. 995–1000.

[38] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in *Data Mining and Knowledge Discovery Handbook*. Berlin, Germany: Springer, 2009, pp. 667–685.

[39] S. Li, Y. Jiang, N. V. Chawla, and Z. Zhou, "Multi-label learning from crowds," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 7, pp. 1369–1382, Jul. 2019.

[40] D. Heider, R. Senge, W. Cheng, and E. Hüllermeier, "Multilabel classification for exploiting cross-resistance information in hiv-1 drug resistance prediction," *Bioinformatics*, vol. 29, no. 16, pp. 1946–1952, 2013.

[41] C. N. Silla and A. A. Freitas, "A survey of hierarchical classification across different application domains," *Data Mining Knowl. Discovery*, vol. 22, no. 1–2, pp. 31–72, 2011.

[42] A. D. Gordon, "A review of hierarchical classification," *J. Royal Statist. Soc.: Ser. General*, vol. 150, no. 2, pp. 119–137, 1987.

[43] J. Wehrmann, R. Cerri, and R. Barros, "Hierarchical multi-label classification networks," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5225–5234.

[44] S. Dumais and H. Chen, "Hierarchical classification of web content," in *Proc. Int. ACM SIGIR Conf. Res. Development Inf. Retrieval*, 2000, pp. 256–263.

[45] A. Sun and E.-P. Lim, "Hierarchical text classification and evaluation," in *Proc. IEEE Int. Conf. Data Mining*, 2001, pp. 521–528.

[46] H. G. Gauch Jr and R. H. Whittaker, "Hierarchical classification of community data," *J. Ecology*, vol. 69, pp. 537–557, 1981.

[47] Q. Zhang, C. Shi, Z. Niu, and L. Cao, "HCBC: A hierarchical case-based classifier integrated with conceptual clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 1, pp. 152–165, Jan. 2019.

[48] Y. Yang, Y. Duan, X. Wang, Z. Huang, N. Xie, and H. T. Shen, "Hierarchical multi-clue modelling for POI popularity prediction with heterogeneous tourist information," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 4, pp. 757–768, Apr. 2019.

[49] B. Lyu, L. Qin, X. Lin, L. Chang, and J. X. Yu, "Supergraph search in graph databases via hierarchical feature-tree," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 2, pp. 385–400, Feb. 2019.

[50] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, and Q. Zhu, "When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks," in *Proc. Int. Conf. Manage. Data*. New York, NY, USA: ACM, 2018, pp. 709–724. [Online]. Available: http://doi.acm.org/10.1145/3183713.3196913

[51] M. Ren *et al.*, "Meta-learning for semi-supervised few-shot classification," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–15. [Online]. Available: https://openreview.net/pdf?id=HJcSzz-CZ

[52] A. Li, T. Luo, Z. Lu, T. Xiang, and L. Wang, "Large-scale few-shot learning: Knowledge transfer with class hierarchy," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 7212–7220.

[53] H. Yao, Y. Wei, J. Huang, and Z. Li, "Hierarchically structured meta-learning," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 12189–12209.

[54] G. Valentini, "True path rule hierarchical ensembles for genome-wide gene function prediction," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 8, no. 3, pp. 832–847, May/Jun. 2011.

[55] N. Cesa-Bianchi, M. Re, and G. Valentini, "Synergy of multi-label hierarchical ensembles, data fusion, and cost-sensitive methods for gene functional inference," *Mach. Learn.*, vol. 88, no. 1–2, pp. 209–241, 2012.

[56] G. Yu, H. Zhu, C. Domeniconi, and J. Liu, "Predicting protein function via downward random walks on a gene ontology," *BMC Bioinf.*, vol. 16, no. 1, 2015, Art. no. 271.

[57] G. Yu, G. Fu, J. Wang, and Y. Zhao, "NewGOA: Predicting new go annotations of proteins by Bi-random walks on a hybrid graph," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 15, no. 4, pp. 1390–1402, Jul./Aug. 2018.

[58] B. Gholami and V. Pavlovic, "Probabilistic temporal subspace clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3066–3075.

[59] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Advances Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[60] A. A. Rusu *et al.*, "Meta-learning with latent embedding optimization," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–17. [Online]. Available: https://openreview.net/forum?id=BJgklhAcK7

[61] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," 2018, pp. 1–15, *arXiv: 1803.02999*.

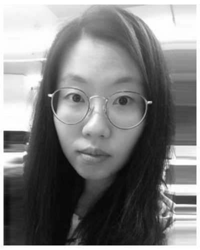
[62] Y. Wu and K. He, "Group normalization," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 3–19.

[63] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

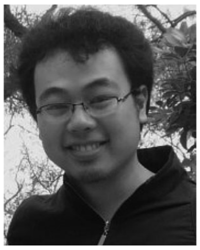
[64] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–15. [Online]. Available: https://openreview.net/forum?id=8gmWwjFyLj

[65] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1842–1850.

[66] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 2579–2605, 2008.



Lu Liu received the bachelor's degree from the South China University of Technology (SCUT), Guangzhou, China, in 2017. She is currently working toward the PhD degree at the Center for Artificial Intelligence, University of Technology Sydney (UTS), Australia. Her current research interests include deep learning, machine learning, few-shot learning, and meta-learning.



Tianyi Zhou is currently working toward the PhD degree with the Paul G. Allen School of Computer Science and Engineering, University of Washington, Seattle. His research covers several topics of machine learning, natural language processing, and statistics and data analysis. He has published more than 30 papers at top conferences and journals including NeurIPS, ICML, ICLR, AISTATS, ACM SIGKDD, IEEE ICDM, AAAI, IJCAI, IEEE ISIT, the *Machine Learning Journal* (Springer), *Data Mining and Knowledge Discovery* (Springer), *IEEE Transactions on Image Processing*, *IEEE Transactions on Neural Networks and Learning Systems*, etc. He is the recipient of the Best Student Paper Award at IEEE ICDM 2013.

Guodong Long received the PhD degree from the University of Technology Sydney (UTS), Australia, in 2014. He is a senior lecturer with the Centre for Artificial Intelligence (CAI), Faculty of Engineering and IT, UTS. His research focuses on data mining, machine learning, and natural language processing. He has more than 40 research papers published on top-tier journals and conferences, including the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Transactions on Cybernetics*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Neural Networks and Learning Systems*, NeurIPS, ICLR, AAAI, IJCAI, KDD, and ICDM.

Jing Jiang received the PhD degree from the University of Technology Sydney (UTS), Australia, in 2015. She is currently a senior lecturer with the Centre for Artificial Intelligence, Faculty of Engineering and IT, UTS. Her research interest include data mining and machine learning applications with the focuses on deep reinforcement learning, and sequential decision-making. She has more than 30 research papers published on top-tier journals and conferences including NeurIPS, AAAI, IJCAI, AAMAS, and ICDM.

Chengqi Zhang has been appointed as a distinguished professor with the University of Technology Sydney from 2017 to 2022, an executive director UTS Data Science from 2017 to 2021, an honorary professor with the University of Queensland from 2015 to 2017, an adjunct professor with the University of New South Wales from 2017 to 2020, and a research professor of Information Technology at UTS from December 2001. In addition, he has been selected as the Chairman of the Australian Computer Society National Committee for Artificial Intelligence since November 2005, and the Chairman of IEEE Computer Society Technical Committee of Intelligent Informatics (TCII) since June 2014.