Prototype Propagation Networks (PPN) for Weakly-supervised Few-shot Learning on Category Graph

Lu Liu[1], **Tianyi Zhou**,[2] **Guodong Long**,[1] **Jing Jiang**,[1] **Lina Yao** [3] and **Chengqi Zhang**[1]
[1]Center for Artificial Intelligence, FEIT, University of Technology Sydney
[2]Paul G. Allen School of Computer Science & Engineering, University of Washington
[3]School of Computer Science and Engineering, University of New South Wales
lu.liu-10@student.uts.edu.au, tianyizh@uw.edu, guodong.long@uts.edu.au,
jing.jiang@uts.edu.au, lina.yao@unsw.edu.au, chengqi.zhang@uts.edu.au

Abstract

A variety of machine learning applications expect to achieve rapid learning from a limited number of labeled data. However, the success of most current models is the result of heavy training on big data. Meta-learning addresses this problem by extracting common knowledge across different tasks that can be quickly adapted to new tasks. However, they do not fully explore weakly-supervised information, which is usually free or cheap to collect. In this paper, we show that weakly-labeled data can significantly improve the performance of meta-learning on few-shot classification. We propose *prototype propagation network (PPN)* trained on few-shot tasks together with data annotated by coarse-label. Given a category graph of the targeted fine-classes and some weakly-labeled coarse-classes, PPN learns an attention mechanism which propagates the prototype of one class to another on the graph, so that the K-nearest neighbor (KNN) classifier defined on the propagated prototypes results in high accuracy across different few-shot tasks. The training tasks are generated by subgraph sampling, and the training objective is obtained by accumulating the level-wise classification loss on the subgraph. On two benchmarks, PPN significantly outperforms most recent few-shot learning methods in different settings, even when they are also allowed to train on weakly-labeled data.

1 Introduction

Machine learning (ML) has achieved breakthrough success in a great number of application fields during the past 10 years, due to more expressive model structures, the availability of massive training data, and fast upgrading of computational hardware/infrastructure. Nowadays, with the support of expensive hardware, we can train super-powerful deep neural networks containing thousands of layers on millions or even trillions of data within an acceptable time. However, as AI becomes democratized for personal or small business use, with concerns about data privacy, demand is rapidly growing for instant learning of highly customized models on edge/mobile devices with limited data. This brings new challenges since the big data and computational power that major ML techniques rely on are no longer available or affordable. In such cases, ML systems that can quickly adapt to new tasks and produce reliable models by only seeing few-shot training data are highly preferable.

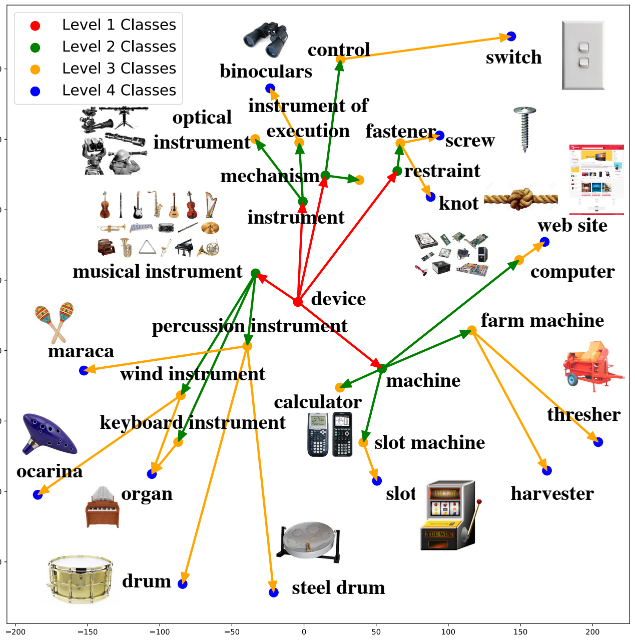

Figure 1: Prototypes learned by PPN and transformed to a 2D space by t-SNE [Maaten and Hinton, 2008]. Each edge connects a child class to a parent. The prototypes spread out as classes become finer, preserve the graph structure, and reflect the semantic similarity.

This few-shot learning problem can be addressed by a class of approaches called "meta-learning". Instead of independently learning each task from scratch, the goal of meta-learning is to learn the common knowledge shared across different tasks, or "learning to learn". The knowledge is at learning/algorithm-level and is task-independent, and thus can be applied to new unseen tasks. For example, it can be shared initialization weights [Finn *et al.*, 2017], an optimization algorithm [Ravi and Larochelle, 2017], a dis-

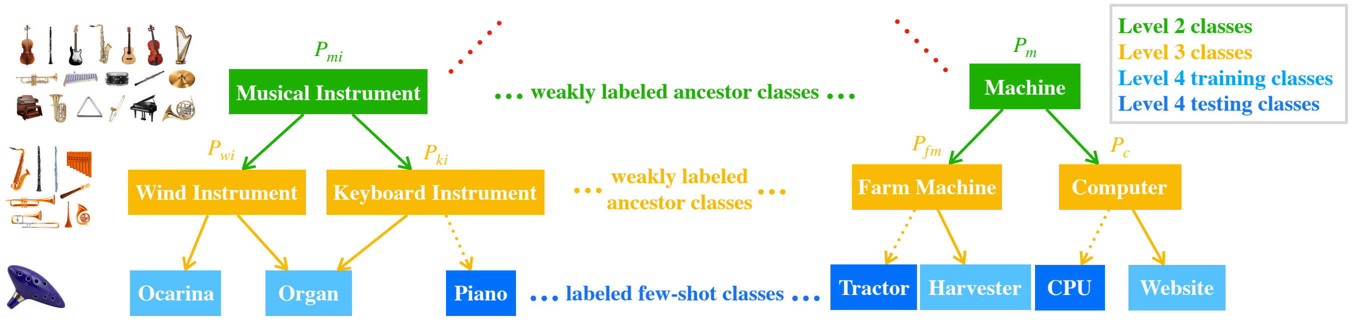

Figure 2: Weakly-supervised few-shot learning in PPN. The few-shot classes (leaf nodes) in training and test tasks are non-overlapping, but they are allowed to share some parent classes. Weakly-labeled data only has non-leaf class labels. As shown in the left side, finer class labels are more informative but more expensive to collect, so we assume that the number of weakly-labeled data exponentially reduces as their labels become finer. PPN is trained on classification tasks with both fine and coarse classes.

tance/similarity metric [Vinyals *et al.*, 2016], or a generator of prototypes [Snell *et al.*, 2017] that compose the support set of the K-nearest neighbor (KNN) predictor. Therefore, new tasks can benefit from the accumulated meta-knowledge extracted from previous tasks. In contrast to single-task learning, the "training data" in meta-learning are tasks, i.e., tasks from a certain distribution are sampled across all possible tasks. It then tries to maximize the validation accuracy of these sampled tasks. Meta-learning shares ideas with life-long/continual/progressive learning in that the meta-knowledge can be re-used and updated for future tasks. It generalizes multi-task learning [Caruana, 1997] since it can be applied to any new task drawn from the same distribution.

Although recent studies of meta-learning have shown its effectiveness on few-shot learning tasks, most of them do not leverage weakly-supervised information, which is usually free or cheap to collect, and has been proved to be helpful when training data is insufficient, e.g., in weakly-supervised [Zhou, 2017] and semi-supervised learning [Belkin *et al.*, 2006; Zhu and Ghahramani, 2002]. In this paper, we show that weakly-supervised information can significantly improve the performance of meta-learning on few-shot classification. In particular, we leverage weakly-labeled data that are annotated by coarse-class labels, e.g., an image of a Tractor with a coarse label of Machine. These data are usually cheap and easy to collect from web tags or human annotators. We additionally assume that a category graph describing the relationship between fine classes and coarse classes is available, where each node represents a class and each directed edge connects a fine-class to its parent coarse-class. An example of a category graph is given in Figure 2. It is not necessary for the graph to cover all the possible relationships and graph containing partial relationship is usually easy to obtain, e.g., the ImageNet category tree built based on the synsets from WordNet.

We propose a meta-learning model called prototype propagation network (PPN) to explore the above weakly-supervised information for few-shot classification tasks. PPN produces a prototype per class by propagating the prototype of each class to its child classes on the category graph, where an attention mechanism generates the edge weights used for propagation. The learning goal of PPN is to minimize the validation errors of a KNN classifier built on the propagated prototypes for few-shot classification tasks. The prototype propagation on the graph enables the classification error of data belonging to any class to be back-propagated to the prototype of another class, if there exists a path between the two classes. The classification error on weakly-labeled data can thus be back-propagated to improve the prototypes of other classes, which later contribute to the prototype of few-shot classes via prototype propagation. Therefore, the weakly-labeled data and coarse classes can directly contribute to the update of prototypes of few-shot fine classes, offsetting the lack of training data. The resulting graph of prototypes can be repeatedly used, updated and augmented on new tasks/classes as an episodic memory. Interestingly, this interplay between the prototype graph and the quick adaptation to new few-shot tasks (via KNN) is analogous to the complementary learning system that reconciles episodic memory with statistical learning inside the human hippocampus [Schapiro *et al.*, 2017], which is believed to be critical for rapid learning.

Similar to other meta-learning methods, PPN learns from the training processes of different few-shot tasks, each defined on a subset of classes sampled from the category graph. To fully explore the weakly-labeled data, we develop a level-wise method to train tasks, generated by subgraph sampling, for both coarse and fine classes on the graph. In addition, we introduce two testing/inference settings that are common in different practical application scenarios: one (PPN+) is allowed to use weakly-labeled data in testing tasks and is given the edges connecting test classes to the category graph, while the other (PPN) cannot access any extra information except for the few-shot training data of the new tasks. In experiments, we extracted two benchmark datasets from ILSVRC-12 (ImageNet) [Deng *et al.*, 2009], specifically for weakly-supervised few-shot learning. In different test settings, our method consistently and significantly outperforms the three most recently proposed few-shot learning models and their variants, which also trained on weakly-labeled data. The prototypes learned by PPN is visualized in Figure 1. The code is available at: https://github.com/liulu112601/PPN.

2 Related Works

A great number of meta-learning approaches have been proposed to address the few-shot learning problem. There are usually two main ideas behind these works: 1) learning a component of a KNN predictor applied to all the tasks, e.g., the support set [Snell *et al.*, 2017], the distance metric [Vinyals *et al.*, 2016], or both [Mishra *et al.*, 2018]; 2) learning a component of an optimization algorithm used to train different tasks, e.g., an initialization point [Finn *et al.*, 2017]. Another straightforward approach is to generate more training data for few-shot tasks by a data augmentation technique or generative model [Lake *et al.*, 2015; Wong and Yuille, 2015]. Our method follows the idea of learning a small support set (prototypes) for KNN, and differs in that we leverage the weakly-labeled data by relating prototypes of different classes.

Auxiliary information: unlabeled data [Ren *et al.*, 2018] and inter/intra-task relationship [Nichol and Schulman, 2018; Liu *et al.*, 2019; Ravi and Beatson, 2019] have recently been used to improve the performance of few-shot learning. Co-training on auxiliary tasks [Oreshkin *et al.*, 2018] has also been applied to improve the learning of the similarity metric. In contrast, to the best of our knowledge, we are the first to utilize the weakly-labeled data as the auxiliary information to bring significant improvement.

The prototype propagation in our method inherits ideas from random walk, message passing, belief propagation, and label propagation [Wu *et al.*, 2019; Zhou *et al.*, 2017; Dong and Yang, 2019]. A similar idea has also been used in more recent graph neural networks (GNN) such as graph attention networks (GAT) [Veličković *et al.*, 2018]. GNN are mainly designed for tasks on graph-structured data such as node classification [Hamilton *et al.*, 2017], graph embedding [Yu *et al.*, 2018], graph generation [Dai *et al.*, 2018] and graph clustering [Wang *et al.*, 2017]. Although our method uses an attention mechanism similar to GAT for propagation, we have a different training scheme (Algorithm 1) that only requires one-step propagation on a specific directed acyclic graph (DAG). GNN has been applied to meta-learning in [Garcia and Bruna, 2018], but the associated graph structure is defined on samples instead of classes/prototypes, and is handcrafted with fully connected edges.

3 Prototype Propagation Networks (PPN)

3.1 Weakly-Supervised Few-Shot Learning

In weakly-supervised few-shot learning, we learn from two types of data: the few-shot data $\mathcal{X}$ annotated by the target fine-class labels and the weakly-labeled data $\mathcal{X}^w$ annotated by coarse-class labels. Each $x \in \mathcal{X}$ is associated with a fine-class label $y \in \mathcal{Y}$, while each $x \in \mathcal{X}^w$ is associated with a coarse-class label $y^w \in \mathcal{Y}^w$. We assume that there exists a category graph describing the relationship between fine classes and coarse classes. This is a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{Y} \cup \mathcal{Y}^w, E)$, where each node $y \in \mathcal{Y} \cup \mathcal{Y}^w$ denotes a class, and each edge (or arc) $z \rightarrow y \in E$ connects a parent class z to one of its child classes y. An example of the category graph is given in Figure 2: the few-shot classes $\mathcal{Y}$ are the leaves of the graph, while the weakly-labeled classes $\mathcal{Y}^w$ are the parents [1] or ancestors [2] of the few-shot classes; for example, the parent class of "Tractor" is "Farm Machine", while its ancestors include "Farm Machine" and "Machine". A child class can belong to multiple parent classes, e.g., the class "Organ" has two parent classes, "Wind Instrument" and "Keyboard Instrument".

Notation	Definition
$\mathcal{X}$	Data for targeted fine-classes
$\mathcal{Y}$	Labels for targeted fine-classes
$\mathcal{X}^w$	Data for weakly-labeled classes
$\mathcal{Y}^w$	Labels for weakly labeled classes
$\mathcal{G}$	Category graph (a directed acyclic graph)
E	Edges of $\mathcal{G}$
$\mathcal{G}_i^j$	Level-j of $\mathcal{G}_i$ (a subgraph of $\mathcal{G}$)
$D^{\mathcal{G}_i^j}$	Distribution of data from classes on $\mathcal{G}_i^j$
$\mathcal{Y}^{\mathcal{G}_i}$	Set of classes on subgraph $\mathcal{G}_i$
$\mathcal{Y}^{\mathcal{G}_i^j}$	Set of classes from level-j of subgraph $\mathcal{G}_i$
T	a few-shot learning task
$\mathcal{T}$	Distribution that task T is drawn from
D^T	Distribution of data from classes in task T
w	a meta-learner producing learner models
Θ	Parameters of a meta learner
$\boldsymbol{P}_y$	Final prototype for class y
$\boldsymbol{P}_y^0$	Initialized Prototype for class y
$\boldsymbol{P}_y^+$	Propagated Prototype for class y

Table 1: Notations used in this paper.

We follow the setting of few-shot learning, which draws training tasks $T \sim \mathcal{T}$ from a task distribution $\mathcal{T}$ and assumes that the test tasks are also drawn from the same distribution. For few-shot classification, each task T is defined by a subset of classes, e.g., an N-way k-shot task refers to classification over N classes and each class only has k training samples. It is necessary to sample the training classes and test classes from two disjoint sets of classes to avoid overlapping. In our problem, as shown by Figure 2, the few-shot classes used for training and test (colored light blue and deep blue respectively) are also non-overlapping, but we allow them to share some parents on the graph. We also allow training tasks to cover any classes on the graph. Since finer-class labels can provide more information about the targeted few-shot classes but are more expensive to obtain, we assume that the amount of weakly-labeled data is reduced exponentially when the class becomes finer. The training aims to solve the following risk minimization (or likelihood maximization) of "learning to learn":

$$\min_{\Theta} \mathbb{E}_{T\sim\mathcal{T}} \left[\mathbb{E}_{(x,y)\sim\mathcal{D}^T} - \log \Pr(y|x; w(T;\Theta))\right], \qquad (1)$$

where each task T is defined by a subset of classes $\mathcal{Y}^T \subseteq \mathcal{Y} \cup \mathcal{Y}^w$, $\mathcal{D}^T$ is the distribution of data-label pair (x, y) with $y \in \mathcal{Y}^T$, $\Pr(y|x; w(T;\Theta))$ is the likelihood of (x, y) produced by model $w(T;\Theta)$ for task T, where Θ is the meta-learner parameter shared by all the tasks drawn from $\mathcal{T}$.

[1]**Parents**: directly connected coarse classes.

[2]**Ancestors**: coarse classes not connected but linked via paths.

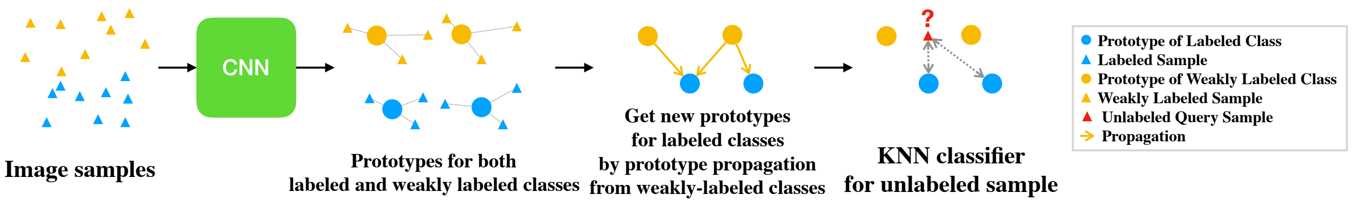

Figure 3: An overview for few-shot classifications using PPN. Prototypes are initialized by averaging sample representations (provided by a CNN) per class. They are then updated by weighted averaging of the prototypes of their parent classes (weakly-labeled classes). The above process generates a learner model (i.e., a KNN): the class whose prototype is the closest to a query sample is the output prediction.

In our method, $\Pr(y|x; w(T;\Theta))$ is computed by a soft-version of KNN, where $w(T;\Theta)$ is the set of nearest neighbor candidates (i.e., the support set of KNN) for task T, and Θ defines the mechanism generating the support set. Similar to prototypical networks [Snell *et al.*, 2017], the support set $w(T;\Theta)$ is composed of prototypes $\boldsymbol{P}_{\mathcal{Y}^T} \triangleq \{\boldsymbol{P}_y \in \mathbb{R}^d : y \in \mathcal{Y}^T\}$, each of which is associated with a class $y \in \mathcal{Y}^T$. Given a data point x, we first compute its representation $f(x) \in \mathbb{R}^d$, where $f(\cdot)$ is convolutional neural networks (CNN) with parameter Θ^{cnn}, then $\Pr(y|x; w(T;\Theta)) = \Pr(y|\boldsymbol{x}; \boldsymbol{P}_{\mathcal{Y}^T})$ is computed as:

$$\Pr(y|\boldsymbol{x}; \boldsymbol{P}_{\mathcal{Y}^T}) \triangleq \frac{\exp(-\|f(\boldsymbol{x}) - \boldsymbol{P}_y)\|^2)}{\sum_{z\in\mathcal{Y}^T}\exp(-\|f(\boldsymbol{x}) - \boldsymbol{P}_z)\|^2)}, \quad (2)$$

In the following, we will introduce prototype propagation which generates $\boldsymbol{P}_{\mathcal{Y}^T}$ for any task $T \sim \mathcal{T}$. An overall of all procedures in our model is shown in Figure 3.

3.2 Prototype Propagation

In PPN, each training task T is a level-wise classification on a sampled subgraph $\mathcal{G}_i \subseteq \mathcal{G}$, i.e., a classification task over $\mathcal{Y}^T = \mathcal{Y}^{\mathcal{G}_i^j}$, where $\mathcal{G}_i^j$ denotes the level-j of subgraph $\mathcal{G}_i$ and $\mathcal{Y}^{\mathcal{G}_i^j}$ is the set of all the classes on $\mathcal{G}_i^j$.

The prototype propagation is defined on each subgraph $\mathcal{G}_i$, which covers classes $\mathcal{Y}^{\mathcal{G}_i}$. Given the associated training data $\mathcal{X}^y$ for class $y \in \mathcal{Y}^{\mathcal{G}_i}$, the prototype of the class y is initialized as the average of the representations $f(\boldsymbol{x})$ of the samples $\boldsymbol{x} \in \mathcal{X}^y$, i.e.,

$$\boldsymbol{P}_y^0 \triangleq \frac{1}{|\mathcal{X}^y|}\sum_{\boldsymbol{x}\in\mathcal{X}^y} f(\boldsymbol{x}). \quad (3)$$

For each parent class z of class y on $\mathcal{G}_i$, we propagate $\boldsymbol{P}_z^0$ to class y with edge weight $a(\boldsymbol{P}_y^0, \boldsymbol{P}_z^0)$ measuring the similarity between class y and z, and aggregate the propagation (the messages) from all the parent classes by

$$\boldsymbol{P}_y^+ \triangleq \sum_{z\in\mathcal{P}_y^{\mathcal{G}_i}} a(\boldsymbol{P}_y^0, \boldsymbol{P}_z^0) \times \boldsymbol{P}_z^0, \quad (4)$$

where $\mathcal{P}_y^{\mathcal{G}_i}$ denotes the set of all parent classes of y on the subgraph $\mathcal{G}_i$, and the edge weight $a(\boldsymbol{P}_y^0, \boldsymbol{P}_z^0)$ is a learnable similarity metric defined by a dot-product attention [Vaswani *et al.*, 2017], i.e.,

$$a(\boldsymbol{p}, \boldsymbol{q}) \triangleq \frac{\langle g(\boldsymbol{p}), h(\boldsymbol{q})\rangle}{\|g(\boldsymbol{p})\| \times \|h(\boldsymbol{q})\|}, \quad (5)$$

where $g(\cdot)$ and $h(\cdot)$ are learnable transformations applied to prototypes with parameters Θ^{att}, e.g., linear transformations $g(\boldsymbol{p}) = \boldsymbol{W}_g\boldsymbol{p}$ and $h(\boldsymbol{q}) = \boldsymbol{W}_h\boldsymbol{q}$. The prototype after propagation is a weighted average of $\boldsymbol{P}_y^0$ and $\boldsymbol{P}_y^+$ with weight $\lambda \in [0,1]$, i.e.,

$$\boldsymbol{P}_y \triangleq \lambda \times \boldsymbol{P}_y^0 + (1-\lambda) \times \boldsymbol{P}_y^+. \quad (6)$$

For each classification task T on subgraph $\mathcal{G}_i$, $\boldsymbol{P}_y$ is used in Eq. (2) to produce the likelihood probability. The likelihood maximization in Eq. (1) aims to optimize the parameters Θ^{cnn} from $f(\cdot)$ and Θ^{att} from the attention mechanism across all the training tasks.

3.3 Level-wise Training of PPN on Subgraphs

The goal of meta-training is to learn a parameterized propagation mechanism defined by Eq. (3)-Eq. (6) on few-shot tasks. In each iteration, we randomly sample a subset of few-shot classes, which together with all their parent classes and edges on $\mathcal{G}$ form a subgraph $\mathcal{G}_i$. A training task T is drawn from each level $\mathcal{G}_i^j \sim \mathcal{G}_i$ as the classification over classes $\mathcal{Y}^T = \mathcal{Y}^{\mathcal{G}_i^j}$. The meta-learning in Eq. (1) is approximated by

$$\min_{\Theta^{cnn},\Theta^{att}} \sum_{\mathcal{G}_i\sim\mathcal{G}} \sum_{\mathcal{G}_i^j\sim\mathcal{G}_i} \sum_{(x,y)\sim\mathcal{D}^{\mathcal{G}_i^j}} -\log\Pr(y|x; \boldsymbol{P}_{\mathcal{Y}^{\mathcal{G}_i^j}}), \quad (7)$$

where $\mathcal{D}^{\mathcal{G}_i^j}$ is the data distribution of data-label pair (x,y) from classes $\mathcal{Y}^{\mathcal{G}_i^j}$. Since the prototype propagation is defined on the whole subgraph, it generates a computational graph relating each class to all of its parent classes. Hence, during training, the classification error on each class is back-propagated to the prototypes of its parent classes, which will be updated to improve the validation accuracy of finer classes and propagated to generate the prototypes of the few-shot classes later. Therefore, the weakly-labeled data of a coarse class will contribute to the few-shot learning tasks on the leaf-node classes.

The complete level-wise training procedure of PPN is given in Algorithm 1, each of whose iterations comprises two main stages: prototype propagation (lines 9-11) which builds a computational graph over prototypes of the classes on a sampled subgraph $\mathcal{G}_i$, and level-wise training (lines 12-16) which updates the parameters Θ^{cnn} and Θ^{att} on per-level classification task. In the prototype propagation stage, the prototype of each class is merged with the information from the prototypes of its parent classes, and the classification error

Algorithm 1 Level-wise training of PPN

```
Input: few-shot data $\mathcal{X}$ with labels from $\mathcal{Y}$,
       weakly-labeled data $\mathcal{X}^w$ with labels from $\mathcal{Y}^w$,
       category graph $\mathcal{G} = (\mathcal{Y} \cup \mathcal{Y}^w, E)$,
       learning rate scheduler for SGD, $\lambda$, $m$;
 1: Initialize: randomly initialize $\boldsymbol{P}$, $\Theta^{cnn}$, $\Theta^{att}$, $\tau \leftarrow 0$;
 2: while not converge do
 3:   if $\tau \bmod m = 0$ then
 4:     for class $y \in \mathcal{Y} \cup \mathcal{Y}^w$ do
 5:       Lazily update $\boldsymbol{P}_y^0$ by Eq. (3) and save it in buffer;
 6:     end for
 7:   end if
 8:   Sample a subgraph $\mathcal{G}_i \sim \mathcal{G}$;
 9:   for class $y \in \mathcal{Y}^{\mathcal{G}_i}$ do
10:     Get $\boldsymbol{P}_y^0$ from buffer, and propagation by Eq. (4)-(6);
11:   end for
12:   initialize loss $L \leftarrow 0$;
13:   for level-$j$ $\mathcal{G}_i^j \sim \mathcal{G}_i$ do
14:     $L \leftarrow L + \sum_{(x,y)\sim\mathcal{D}^{\mathcal{G}_i^j}} -\log \Pr(y|x; P_{\mathcal{Y}^{\mathcal{G}_i^j}})$ by Eq. (2);
15:   end for
16:   Mini-batch SGD to minimize $L$, update $\Theta^{cnn}$, $\Theta^{att}$;
17:   $\tau \leftarrow \tau + 1$;
18: end while
```

using the merged prototypes will be backpropagated to update the parent prototypes during the level-wise training stage. To improve the computational efficiency, we do not update $\boldsymbol{P}_y^0$ for every propagation. Instead, we lazily update $\boldsymbol{P}_y^0$ for all classes $y \in \mathcal{Y} \cup \mathcal{Y}^w$ every m epochs, as shown in lines 3-7.

3.4 Meta-Test: Apply PPN to New Tasks

We study two test settings for weakly-supervised few-shot learning, both of which will be used in the evaluation of PPN in our experiments. They differ in whether or not the weakly-supervised information, i.e., the weakly-labeled data and the connections of new classes to the category graph, is still accessible in the test tasks. The first setting (PPN+) is allowed to access this information within test tasks while the second setting (PPN) is unable to access the information. In other words, for PPN+, the edges (i.e., the yellow arrows in Figure 3) between each test class and any other classes are known during test phase, while these edges are unknown and needed to be predicted in the PPN setting. The second setting is more challenging but is preferred in a variety of applications, for example, where the test tasks happen on different devices, whereas the first setting is more appropriate for life-long/continual learning on a single machine.

In the second setting, we can still leverage the prototypes achieved in the training phase and use them for the propagation of prototypes for test classes. In particular, for an unseen test class y (and its associated samples $\mathcal{X}^y$), we find the K-nearest neighbors of $\boldsymbol{P}_y^0$ among all the prototypes achieved during training, and treat the training classes associated with the K-nearest neighbor prototypes as the parents of y on $\mathcal{G}$. These parent-class prototypes serve to provide weakly-supervised information to the test classes via propagation on the category graph.

In both settings, for each class y in a test task T, we apply the prototype propagation in Eq. (3)-(6) on a subgraph composed of y and its parents $\mathcal{P}_y^{\mathcal{G}}$. This produces the final prototype $\boldsymbol{P}_y$, which is one of the candidates for the nearest neighbors for $\boldsymbol{P}_{\mathcal{Y}^T}$ in KNN classification on task T. In the first setting (PPN+), the hierarchy covering both the training and test classes is known so the parent class of each test class y may come from either the training classes or the weakly-labeled test classes. When a parent class $y' \in \mathcal{P}_y^{\mathcal{G}}$ is among the training classes, we use the buffered prototype $\boldsymbol{P}_{y'}^0$ from training for propagation; otherwise, we use Eq. (3) to compute $\boldsymbol{P}_{y'}^0$ over weakly-labeled samples belonging to class y' for this task. In the second setting (PPN), since the edges connecting test classes are unknown and are predicted by KNN as introduced in the last paragraph, we assume that all the parents of y are from training classes. We directly use the parents' buffered prototypes from training for propagation.

4 Experiments

We compare PPN/PPN+ to three baseline methods, i.e., Prototypical Networks, GNN and Closer Look [Chen *et al.*, 2019], and their variants of using the same weakly-labeled data as PPN/PPN+. For their variants, we apply the same level-wise training on the same weakly-labeled data as in PPN/PPN+ to the original implementations, i.e., we replace the original training tasks with level-wise training tasks. We always tune the initial learning rate, the schedule of learning rate, and other hyperparameters of all the baselines and their variants on a validation set of tasks. The results are reported in Table 3 and Table 4, where the variants of baselines are marked by "*" following the baseline name.

In PPN/PPN+ and all the baseline methods (as well as their variants), we use the same backbone CNN (i.e., $f(\cdot;\theta^{cnn})$) that has been used in most previous few-shot learning works [Snell *et al.*, 2017; Finn *et al.*, 2017; Vinyals *et al.*, 2016]. It has 4 convolutional layers, each with 64 filters of 3×3 convolution, followed by batch normalization [Ioffe and Szegedy, 2015], ReLU nonlinearity, and 2×2 max-pooling. The transformation $g(\cdot)$ and $h(\cdot)$ in the attention module are fully connected linear layers.

In PPN/PPN+, the variance of $\boldsymbol{P}_y^0$ increases when the number of samples (i.e., the "shot") per class reduces. Hence, we set $\lambda = 0$ in Eq. (6) for N-way 1-shot classifications, and $\lambda = 0.5$ for N-way 5-shot classification. During training, we lazily update $\boldsymbol{P}_y^0$ for all the classes on the graph $\mathcal{G}$ every $m = 5$ epochs and choose the nearest $K = 3$ neighbours as parents among all prototypes gained after training for PPN. Adam [Kingma and Ba, 2015] is used to train the model for 150k iterations, with an initial learning rate of 10^{-3}, a weight decay of 10^{-4}, and a momentum of 0.9. We reduce the learning rate by a factor of $0.7\times$ every 15k iterations starting from the 10k-th iterations.

4.1 WS-ImageNet-Pure

WS-ImageNet-Pure is a subset of ILSVRC-12. On the ImageNet WordNet Hierarchy, we extract 80% classes from level-7 as leaf nodes of the category graph $\mathcal{G}$ and use them as

	WS-ImageNet-Pure					WS-ImageNet-Mix				
level	training		testing		#img	training		testing		#img
	#classes	#img	#classes	#img		#classes	#img	#classes	#img	
3	7	15140	1	3680	18820	7	92785	4	42882	135667
4	13	10093	6	2662	12755	13	76889	7	24592	101481
5	25	6904	9	1834	8738	23	50348	12	13462	63810
6	44	4350	18	1155	5505	42	22276	18	5849	28125
7	71	2538	18	646	3184	71	7546	18	1919	9465
all	160	39025	52	9977	49002	156	249844	59	88704	338548

Table 2: Statistics of WS-ImageNet-Pure and WS-ImageNet-Mix for weakly-supervised few-shot learning, where #classes and #img denote the number of classes and images respectively. Their classes are extracted from levels 3-7 of ImageNet WordNet hierarchy.

Model	**W-S**	**5way1shot**	**5way5shot**	**10way1shot**	**10way5shot**
Prototypical Net [Snell *et al.*, 2017]	N	33.17±1.65%	46.76±0.98%	20.48±0.99%	31.49±0.57%
GNN [Garcia and Bruna, 2018]	N	30.83±0.66%	41.33±0.62%	20.33±0.60%	22.50±0.62%
Closer Look [Chen *et al.*, 2019]	N	32.27±1.58%	46.02±0.74%	22.78±0.94%	28.04±0.36%
Prototypical Network*	Y	32.13±1.48%	44.41±0.93%	20.07±0.93%	30.87±0.56%
GNN*	Y	32.33±0.52%	45.67±0.87%	22.50±0.67%	32.67±0.37%
Closer Look*	Y	32.63±1.55%	43.76±0.93%	20.03±0.83%	30.67±0.36%
PPN (Ours)	Y	37.37±1.64%	50.31±1.00%	24.17±1.00%	36.21±0.57%
PPN+(Ours)	Y	**48.00±1.70**%	**52.36±1.02**%	**35.75±1.13**%	**38.18±0.63**%

Table 3: Validation accuracy (mean±CI%95) on 600 tasks of PPN and baselines on WS-ImageNet-Pure. W-S indicates weakly-supervised.

Model	**W-S**	**5way1shot**	**5way5shot**	**10way1shot**	**10way5shot**
Prototypical Net [Snell *et al.*, 2017]	N	31.93±1.62%	49.80±0.90%	21.02±0.97%	36.42±0.62%
GNN [Garcia and Bruna, 2018]	N	33.60±0.11%	45.87±0.12%	22.00±0.89%	34.33±0.75%
Closer Look [Chen *et al.*, 2019]	N	33.10±1.57%	40.67±0.73%	20.85±0.92%	35.19±0.43%
Protytypical Net*	Y	31.80±1.48%	49.03±0.93%	20.33±0.98%	34.79±0.58%
GNN*	Y	30.33±0.80%	47.33±0.28%	23.33±1.03%	31.33±0.80%
Closer Look*	Y	31.13±1.51%	44.90±0.78%	20.25±0.87%	34.01±0.40%
PPN (Ours)	Y	36.23±1.69%	52.38±0.92%	23.30±1.06%	38.20±0.55%
PPN+(Ours)	Y	**41.60±1.67**%	**53.95±0.96**%	**29.87±1.08**%	**39.17±0.58**%

Table 4: Validation accuracy (mean±CI%95) on 600 tasks of PPN and baselines on WS-ImageNet-Mix. W-S indicates weakly-supervised.

the targeted classes $\mathcal{Y}$ in few-shot tasks. The ancestor nodes of these classes on $\mathcal{G}$ are then sampled from level-6 to level-3, which compose weakly-labeled classes $\mathcal{Y}^w$. We sub-sample the data points for classes on $\mathcal{G}$ in a bottom-up manner: for any level-7 (bottom) level class y, we directly sample a subset from all the images belonging to y in ImageNet; for any class y on lower level-j with $j < 7$, we sample from all the images that belong to y and have not been sampled into its descendant classes. Hence, we know that any data point sampled into class y belongs to all the ancestor classes of y but we do not know its labels on any targeted class of y. In addition, we sample each candidate data point for any class on level j with probability 0.6^j. Hence, the number of data points associated with each class thus reduces exponentially when the level of the class increases. This is consistent with many practical scenarios, i.e., samples with finer-class labels can provide more information about targeted few-shot tasks, but they are much more expensive to obtain and usually insufficient.

	PPN	PPN+	Proto Net	GNN	Closer
Train	0.074	0.074	0.066	0.170	0.006
Test	0.025	0.029	0.023	0.011	0.115

Table 5: The average per-iteration time (in seconds) comparisons on PPN/PPN+ and other baselines on WS-ImageNet-Pure.

For training-test splitting of few-shot classes[3], we divide the classes from level-7 into two disjoint subsets with ratio 4:1 (4 for training and 1 for test). This ensures that any class in any test task has never been learned in training tasks. However, we allow training classes and test classes to share some parent classes. The detailed statistics of WS-ImageNet-Pure are given in Table 3.

The experimental results of PPN/PPN+ and all the base-

[3]Each training task is classification defined on a randomly sampled subset of training classes, while each test task is classification defined on a randomly sampled subset of test classes.

lines (and their weakly-supervised variants ending with "*") on WS-ImageNet-Pure are shown in Table 3. PPN/PPN+ outperform all other methods. The table shows that PPN/PPN+ are more advantageous in 1-shot tasks, and that PPN+ achieves $\sim 15\%$ improvement compared to other methods. This implies that the weakly-supervised information can be more helpful when supervised data is highly insufficient, and our method is able to significantly boost performance by exploring the weakly-labeled data. Although all the weakly-supervised variants of baselines are trained on the same data as PPN/PPN+, they do not achieve similar improvement because their model structures do not have such mechanisms as the prototype propagation in PPN which relates different classes and tasks. In addition, training on unrelated tasks can be distracting and even detrimental to performance. In contrast, PPN/PPN+ build a computational graph of prototypes associated with both coarse and fine classes, and the error on any class can be used to update the prototypes of other classes via backpropagation on the computational graph.

4.2 WS-ImageNet-Mix

To verify if PPN can still learn from weakly-labeled data that belong to other fine classes not involved in the few-shot tasks, we propose another subset of ILSVRC-12, WS-ImageNet-Mix, whose detailed statistics are given in Table 2. We extract WS-ImageNet-Mix by following the same procedure as extracting WS-ImageNet-Pure except that: 1) data points sampled for a coarse (non-leaf) class can belong to the remaining $\sim 20\%$ level-7 classes outside of the $\sim 80\%$ level-7 classes used for generating few-shot tasks; and 2) for any class on level-j, we sample each data point with probability 0.7^j instead of 0.6^j.

The experimental results are reported in Table 4, which shows that PPN/PPN+ still outperform all baselines, and PPN+ outperforms them by $\sim 10\%$ for 1-shot classification. This indicates that PPN/PPN+ is robust to (and might be able to leverage) weakly-labeled data unrelated to the final few-shot tasks.

4.3 Effects of Prototype Propagation

To study whether and how the propagation in Eq. (4) improves few-shot learning, we evaluate PPN using different λ in Eq. (6). Specifically, we try different weights $1-\lambda$ (x-axis in Figure 4) for $\boldsymbol{P}_y^+$ in Eq. (6) between $[0, 0.9]$, and report the validation accuracy (y-axis in Figure 4) on test tasks for Nway1shot tasks and the two datasets. In all scenarios, increasing the weight of $\boldsymbol{P}_y^+$ in the given range consistently improves the accuracy (although the accuracy might drop if the weight gets too close to 1 though), which demonstrates the effectiveness of prototype propagation.

4.4 Time Cost

The average per-iteration time (in seconds) on a single TITAN XP for PPN/PPN+ and the baselines are listed as Table 5. PPN has moderate time cost comparing to other baselines. Compared to prototypical network, our propagation procedure only requires $\sim$10% extra time cost but significantly improves the performance.

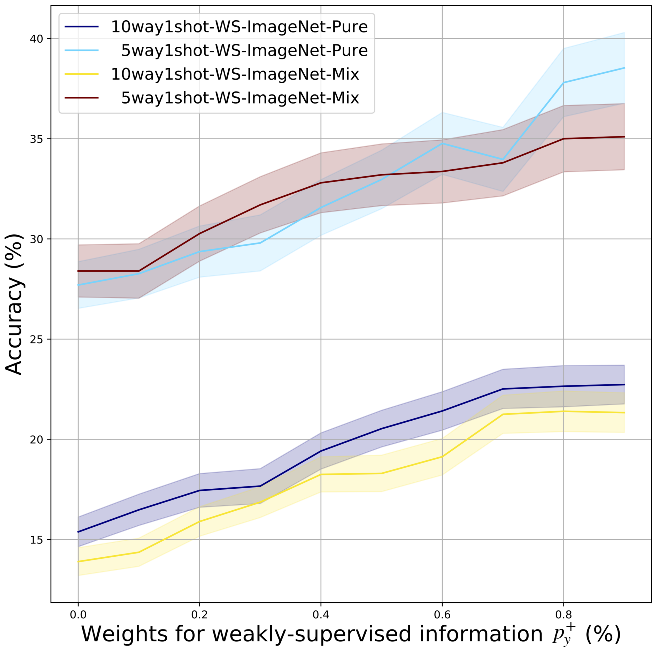

Figure 4: Validation accuracy averaged over 600 Nway1shot test tasks improves when the weight of $\boldsymbol{P}_y^+$ i.e., $1-\lambda$ in Eq. (6) increases, which implies the effectiveness of prototype propagation.

5 Future Studies

In this work, we propose to explore weakly-labeled data by developing a propagation mechanism generating class prototypes to improve the performance of few-shot learning. Empirical studies verifies the advantage of exploring the weakly-labeled data and the effectiveness of the propagation mechanism. Our model can be extended to multi-steps propagation and assimilate more weakly-labeled information. The graph can be more general than a class hierarchy, and the node on the graph is not limited to be a class: it can be extended to any output attribute.

Acknowledgements

This research was funded by the Australian Government through the Australian Research Council (ARC) under grants 1) LP160100630 partnership with Australia Government Department of Health and 2) LP150100671 partnership with Australia Research Alliance for Children and Youth (ARACY) and Global Business College Australia (GBCA). We also acknowledge the support of NVIDIA Corporation and MakeMagic Australia with the donation of GPUs.

References

[Belkin *et al.*, 2006] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR*, pages 2399–2434, 2006.

[Caruana, 1997] Rich Caruana. Multitask learning. *Machine Learning*, pages 41–75, 1997.

[Chen *et al.*, 2019] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Liu, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *ICLR*, 2019.

[Dai *et al.*, 2018] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. In *ICLR*, 2018.

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

[Dong and Yang, 2019] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *CVPR*, pages 1761–1770, 2019.

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

[Garcia and Bruna, 2018] Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. In *ICLR*, 2018.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1024–1034, 2017.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[Kingma and Ba, 2015] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Lake *et al.*, 2015] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, pages 1332–1338, 2015.

[Liu *et al.*, 2019] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, and Yi Yang. Transductive propagation network for few-shot learning. In *ICLR*, 2019.

[Maaten and Hinton, 2008] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, pages 2579–2605, 2008.

[Mishra *et al.*, 2018] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018.

[Nichol and Schulman, 2018] Alex Nichol and John Schulman. Reptile: A scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

[Oreshkin *et al.*, 2018] Boris N Oreshkin, Alexandre Lacoste, and Pau Rodriguez. Tadam: Task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, pages 721–731, 2018.

[Ravi and Beatson, 2019] Sachin Ravi and Alex Beatson. Amortized bayesian meta-learning. In *ICLR*, 2019.

[Ravi and Larochelle, 2017] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

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

[Schapiro *et al.*, 2017] Anna C Schapiro, Nicholas B Turk-Browne, Matthew M Botvinick, and Kenneth A Norman. Complementary learning systems within the hippocampus: A neural network modelling approach to reconciling episodic memory with statistical learning. *Philosophical Transactions of the Royal Society of London. Series B, Biological sciences*, page 20160049, 2017.

[Snell *et al.*, 2017] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, pages 4077–4087, 2017.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[Vinyals *et al.*, 2016] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, pages 3630–3638, 2016.

[Wang *et al.*, 2017] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. Mgae: Marginalized graph autoencoder for graph clustering. In *CIKM*, pages 889–898. ACM, 2017.

[Wong and Yuille, 2015] Alex Wong and Alan L Yuille. One shot learning via compositions of meaningful patches. In *ICCV*, pages 1197–1205, 2015.

[Wu *et al.*, 2019] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[Yu *et al.*, 2018] Wenchao Yu, Cheng Zheng, Wei Cheng, Charu C Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen, and Wei Wang. Learning deep network representations with adversarially regularized autoencoders. In *SIGKDD*, pages 2663–2671, 2018.

[Zhou *et al.*, 2017] Tianyi Zhou, Hua Ouyang, Yi Chang, Jeff Bilmes, and Carlos Guestrin. Scaling submodular maximization via pruned submodularity graphs. In *AISTATS*, pages 316–324, 2017.

[Zhou, 2017] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, pages 44–53, 2017.

[Zhu and Ghahramani, 2002] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Citeseer, 2002.