A Cluster-then-label Approach for Few-shot Learning with Application to Automatic Image Data Labeling

RENZHI WU, NILAKSH DAS, SANYA CHABA, SAKSHI GANDHI, DUEN HORNG CHAU, and XU CHU, Georgia Institute of Technology, USA

Few-shot learning (FSL) aims at learning to generalize from only a small number of labeled examples for a given target task. Most current state-of-the-art FSL methods typically have two limitations. First, they usually require access to a source dataset (in a similar domain) with abundant labeled examples, which may not always be possible due to privacy concerns and copyright issues. Second, they typically do not offer any estimation of the generalization error on the target FSL task, because the handful of labeled examples must be used for training and cannot spare a validation subset. In this article, we propose a cluster-then-label approach to perform few-shot learning. Our approach does not require access to the labeled source dataset and provides an estimation of generalization error. We show empirically, on four benchmark datasets, that our approach provides competitive predictive performance to state-of-the-art FSL approaches and our generalization error estimation is accurate. Finally, we explore the application of our proposed method to automatic image data labeling. We compare our method with existing automatic data labeling systems. The end-to-end performance of our method outperforms the state-of-the-art automatic data labeling system Snuba by 26% and is only 7% away from the fully supervised upper bound.

CCS Concepts: • **Computing methodologies** → **Semi-supervised learning settings**; • **Information systems** → **Data mining;**

Additional Key Words and Phrases: Few-shot learning, cluster-then-label, data labeling

ACM Reference format:
Renzhi Wu, Nilaksh Das, Sanya Chaba, Sakshi Gandhi, Duen Horng Chau, and Xu Chu. 2022. A Cluster-then-label Approach for Few-shot Learning with Application to Automatic Image Data Labeling. *J. Data and Information Quality* 14, 3, Article 15 (May 2022), 23 pages.
https://doi.org/10.1145/3491232

1 INTRODUCTION

Deep learning methods have achieved breakthroughs on tasks with large quantity of labeled training data. However, in most cases, creating enough labeled data is either extremely expensive or even not possible, limiting the applicability of deep learning methods. In contrast, humans are able to learn to generalize a new class from only a few examples on a large variety of tasks. Therefore, learning to generalize from a few examples, formally known as **few-shot learning (FSL)**, is of great interest.

Authors' address: R. Wu, N. Das, S. Chaba, S. Gandhi, D. H. Chau, and X. Chu, Georgia Institute of Technology, Atlanta, GA; emails: renzhiwu@gatech.edu, nilakshdas@gatech.edu, sanyachaba@gatech.edu, sakshi@gatech.edu, polo@gatech.edu, xu.chu@cc.gatech.edu.

1936-1955/2022/05-ART15 $15.00
https://doi.org/10.1145/3491232

The current state-of-the-art few-shot learning methods include metric learning–based methods [27, 29], optimization-based methods [8, 17], hallucination-based methods [1, 31], model-based methods [24], and so on. Though taking different forms, most methods rely on some form of knowledge transfer: First, useful information from a source dataset or task is extracted; second, the extracted information allows the learning of a target model on the new dataset or task with only a few labeled examples. We observe two limitations of the current FSL methods.

One limitation of most of these current FSL methods is that they require access to the source dataset at the first step [14]. Accessing to the source dataset can cause privacy (e.g., raw pictures of people) and copyright (e.g., movie clips) issues. In addition, access may not even be possible when the source dataset is not publicly available. In these cases, a common alternative FSL solution is to fine-tune a pretrained source model, which may not work very well with very limited training examples and a large number of parameters to tune [8, 14].

Another limitation of the current few-shot learning methods is that they do not provide a natural way to estimate the generalization error on the target FSL task, though it is often necessary in practice for users, especially non-experts, to gauge how good the model is going to perform on unseen data. Since most of the FSL methods have the implicit assumption that the selected source domain matches that of the target task [3], one method is to estimate using the hold-out classes in the source dataset. This estimation is not accurate in realistic scenarios with domain shift between the source and target dataset and is not even possible without access to the source dataset. In traditional supervised learning settings, the generalization errors are typically estimated by using a hold-out validation set. In FSL setting, the few limited example may not even be sufficient for model training, let alone spare a validation subset for estimating the generalization error.

In this article, we propose a novel *cluster-then-label* approach to perform few-shot learning that (1) does not need access to the source datasets and (2) is able to provide generalization error estimation. Without access to the source datasets, we propose to incorporate additional knowledge into the target FSL task in the form of user-provided featurization functions, which take examples in the target task and give new feature representations. For example, the user can provide many featurization functions that use any layers of any pre-trained ML model, or the user can provide any domain-specific featurization functions.

While featurization functions provide a clean and flexible way to incorporate knowledge from other pre-trained models and any domain-specific knowledge the user might have, it is not clear how to leverage them for FSL. In particular, we face two challenges:

(1) Given a featurization function, how to use it to perform labeling with only a few labeled examples;
(2) How to identify which featurization function is best suitable for the target FSL task.

Our proposed "cluster-then-label" approach addresses the two challenges. Specifically, for a given featurization function, we first cluster all examples (both labeled and unlabeled) using the new feature representation. The assumption is that if the given featurization function is suitable for the target FSL task, then the clustering results would separate the classes well. We then use the few labeled examples to determine the cluster-to-class mapping, i.e., which class does each cluster represent. Since the few labeled examples are only used for deciding the mapping and are not used for learning class boundaries (like current FSL methods do), we can reuse the labeled examples for generalization error estimation. The most suitable featurization function is the one that provides the smallest estimated generalization error. Our key technical contributions consist of a maximum likelihood model for learning the most likely cluster-to-class mapping, as well as a principled way for estimating the generalization error of our results with confidence levels.

Since our proposed FSL method is able to incorporate various knowledge (e.g., pre-trained models and domain expertise) in the form of user-provided featurization functions to generate a large amount of labeled data, a natural application of our method is automatic data labeling. Therefore, we build an image data labeling system GOOGLES [5] based on our proposed FSL method with code open-sourced at github.[1]

This article is organized as follows: We formally define the problem and give an overview of our solution in Section 2, we describe the details of the proposed method in Section 3, we discuss the application of our method to automatic image data labeling in Section 4, we evaluated the proposed method experimentally in Section 5, we review related work in Section 6, and we conclude in Section 7.

This article is an extension of the conference paper of GOGGLES [5]. The proposed FSL method in Section 3 is a novel contribution though it originates from the class inference module in Reference [5] and most materials on the data labeling application in Section 4 are from Reference [5].

2 THE CLUSTER-THEN-LABEL APPROACH FOR FSL

2.1 Problem Setup

We consider a multi-class few-shot learning setting with K classes, and the kth class has m_k labeled examples, where m_k is too small to perform supervised learning. Therefore, we have a total of $m = \sum_{i=1}^{K} m_i$ labeled examples, which we call the *support set*. We also assume access to a large number n of unlabeled examples. Our goal is to assign a label $\hat{y_i}$ for every unlabeled example x_i, where $i \in [1, n]$, and provide an accuracy estimation of the assigned labels.

We also assume that the user (ML engineer) provides a set of featurization functions, where each featurization function $f_j : x \mapsto \mathbb{R}^{l_j}$ maps every example from the raw feature space to a new l_j dimensional feature space for performing clustering. For example, if our input examples are images, then one example f_j can take any image and pass it through a pre-trained VGG-16 model and use logits layers (the last fully connected layer) as a feature representation.

The user can also optionally provide a clustering algorithm, though by default we use a **Gaussian Mixture Model (GMM)** for clustering.

2.2 Solution Overview

Figure 1 illustrates the overall workflow of our *cluster-then-label* approach for performing FSL. Initially, the user is only required to provide one featurization function and a few labeled examples (the blue boxes in the figure). Our "cluster" step uses the provided featurization function to obtain features for performing clustering, and our "label" step uses the m labeled examples to determine a cluster-to-label assignment. Our approach is then able to guide the user in providing more labeled example or selecting a different featurization function until the user is satisfied with the estimated generalization error. Specifically, our approach works as follow:

Clustering Step: First, we apply the user provided featurization function to all examples, including both the m labeled examples and n unlabeled examples. We then apply a clustering algorithm (GMM by default) based on the new features to partition all examples into K non-overlapping clusters, i.e., each example is assigned to a unique cluster. Our assumption is that if the current featurization function is informative enough of the current classification task, then each cluster should correspond well enough to one of the K classes. This is in spirit similar to the Cluster Assumption used in some semi-supervised learning approaches [2, 22]. An illustration of the clustering result is shown in Figure 2, which shows that this clustering result separates the three classes fairly well.

[1] https://github.com/chu-data-lab/GOGGLES.

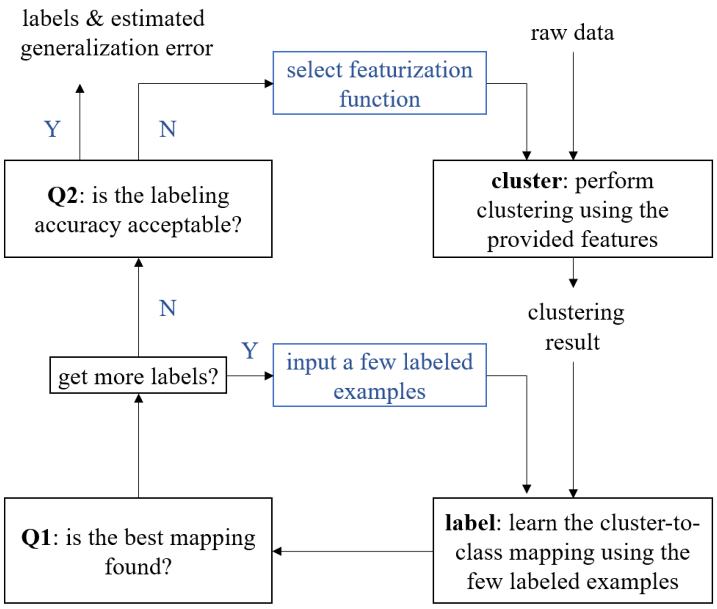
labels & estimated generalization error
Y
N
select featurization function
raw data
Q2: is the labeling accuracy acceptable?
cluster: perform clustering using the provided features
N
clustering result
Y
get more labels?
input a few labeled examples
Q1: is the best mapping found?
label: learn the cluster-to-class mapping using the few labeled examples

Fig. 1. A cluster-then-label approach for FSL. Blue denotes user-provided inputs.

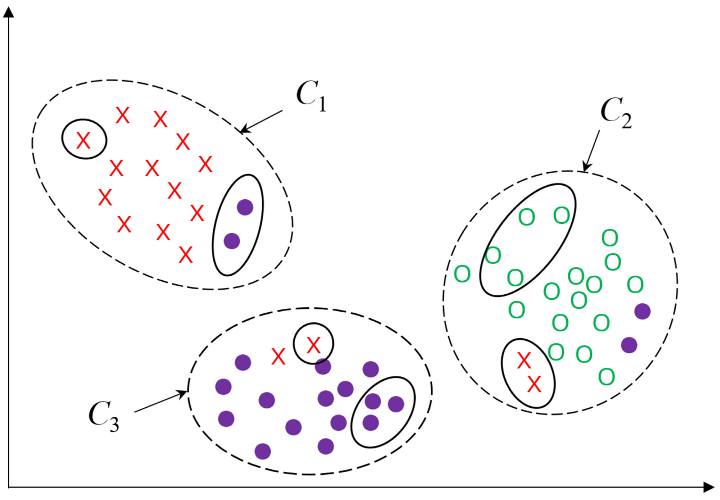

Fig. 2. Illustration of clustering result and support set. Dashed line circles are three clusters C_1, C_2, and C_3. The red cross, green empty dot, and purple solid dot denote the ground-truth class labels for three classes. The instances circled out by black solid line are in the support set.

Labeling Step: Next, based on a few labeled examples (i.e., the support set, e.g., the instances circled out by solid line in Figure 2), each cluster is mapped to a class label, and all examples in a particular cluster are labeled with the class associated with that cluster. We design a probabilistic model to learn such a mapping. There are two potential scenarios where the labels produced by our cluster-then-label approach may not be accurate:

(1) the current clustering results indeed separate all classes well; however, the learned cluster-to-class mapping is incorrect;
(2) the current clustering results simply cannot separate all classes well. In this case, no cluster-to-class mapping is ever going to produce accurate labeling decisions.

Q1: Is the Best Mapping Found? To identify the first potential scenario that leads to inaccurate labeling decisions, we provide a quantitative estimate. Specifically, we answer the following question: *What is the probability that the best mapping is found? how many more labeled examples per class is needed so that the probability of the found mapping being the best mapping can be increased to a certain threshold?* The user is then able to examine the answer to this question, and decides whether the current probability is acceptable or is willing to provide more labeled examples to obtain a mapping with a higher confidence of being the best mapping given the current clustering results.

Q2: Is the Labeling Accuracy Acceptable? When the user decides to not get more labels, we provide a quantitative estimate of the generalization error based on the current clustering results and the learned cluster-to-class mapping. Specifically, we answer the following questions: *What is the probability that the generalization error of the found mapping is below a certain threshold?* If the user is satisfied with the given answer, then we output the final labels; otherwise, the user is advised to select a different featurization function, and the entire cluster-then-label approach is started over.

Example 1. We provide an example to demonstrate the workflow in Figure 1. For example, a user wants to perform FSL for classification on the Surface dataset where each image has a label of "good" (smooth) or "bad" (rough) metallic surface (see details in Section 5.1.2). As a initialization step, the user has to select a featurization function (see the upper blue box in Figure 1) and has to provide a initial set of labeled examples (see the lower blue box in Figure 1). We assume the user has initially selected VGG as the featurization function and provided one labeled examples per class. In addition, the user uses GMM as the clustering method.

Following the workflow in Figure 1, after the *cluster* and the *label* step, the system provides a probabilistic answer to Q1. Specifically, the system will state that the probability that the best mapping is learned is 0.4, and four additional examples per class are required to obtain a mapping that has a probability of 0.8 of being the best mapping. Seeing the system's answer to Q1, the user thinks it is acceptable and provides four more labeled examples for each class. After additional labeled examples, the system updates the answer to Q1 stating that the best mapping is learned with 0.85 probability. Now, the user feels it is good enough and decides not to provide more labels.

Next, the system proceeds to Q2. Given the current clustering results and the current mapping, the system estimates the probability that the classification accuracy being greater than 0.7 is 0.9. The user thinks it is good enough and the system output labels for all unlabeled instances using the class label associated with each cluster.

We will show the details of our probabilistic model for learning the cluster-to-class mapping in Section 3.1. Based on the mapping model, we show how we address Q1 in Section 3.2 and how to address Q2 in Section 3.3.

3 SOLUTION DETAILS

3.1 Learning the Cluster-to-class Mapping

Given a clustering result, we need to determine the class label each cluster should be mapped to. We use the support set to map each cluster to a class. For the labeled examples in the support set, the information we observed includes their class labels and the cluster that they belong. Intuitively, we want to map a cluster to a class if most labeled examples in this cluster are from this class. However, this simple cluster-to-class mapping strategy may create conflicting assignments, namely, multiple clusters might be mapped to the same class. For example, cluster C_1 and cluster C_3 in Figure 2 will both be mapped to the purple class using this simple strategy. We propose to model the likelihood of the observed information to infer the one-to-one mapping in a principled way.

We first represent the mapping and the observed information in a compact way. Let C_i denote the ith cluster and T_i denote the ith class ($i = 1, \ldots, K$). A one-to-one mapping g maps each class to a unique cluster, i.e., $C_{g(i)}$ is the corresponding cluster of class T_i. Let $\mathbf{D}$ be a $K \times K$ matrix where each element $d_{i,j}$ denote the number of labeled instances from class T_i that fall into cluster C_j in the support set. Given a mapping g, we can rearrange the columns in $\mathbf{D}$ to obtain a new matrix $\mathbf{D}'$ so that in $\mathbf{D}'$ the ith row represents the class that is mapped to the cluster that the ith column represents. The rearrangement make the matrix easier to interpret, as each diagonal element $d'_{i,i}$ in $\mathbf{D}'$ denotes the number of labeled instances from class T_i that fall in cluster $C_{g(i)}$. The rearrangement can also be express in a matrix form by the following:

$$\mathbf{D}' = \mathbf{D}\mathbf{G}, \tag{1}$$

where $\mathbf{G}$ is a column-switching transformation matrix that encodes the mapping g. Each element $g_{i,j}$ in $\mathbf{G}$ is

$$g_{i,j} = \begin{cases} 1, & \text{if } g(i) = j \\ 0, & \text{otherwise} \end{cases}. \tag{2}$$

Maximum Likelihood Formulation. Matrix $\mathbf{D}'$ essentially encodes the observed information in the support set. We now model the likelihood that $\mathbf{D}'$ is generated. There are two factors affecting the likelihood of an observed $\mathbf{D}'$: (1) How good is the given one-to-one mapping $\mathbf{G}$? and (2) How well does the current clustering result seperate the classes under a mapping $\mathbf{G}$, i.e., the unknown classification accuracy α?

Intuitively, when classification accuracy is higher, the numbers in each row of $\mathbf{D}'$ are more "concentrated" to a particular cell. For example, in the extreme case when classification accuracy equals one, each row only has one non-zero number. Let α denote the unknown classification accuracy of the algorithm, which is the probability of each instance being assigned to its true label. An instance from class T_i is correctly labeled when and only when it falls into cluster $C_{g(i)}$, so the probability of each instance falling into the cluster $C_{g(i)}$ is α. We further assume the probability of an instance falling into any other clusters is equal, so it is $\beta = \frac{(1-\alpha)}{K-1}$. Under this setting, each row in $\mathbf{D}'$ follows a multinomial distribution:

$$P(d'_{i,1}, \ldots, d'_{i,K}) = \frac{m_i!}{d'_{i,1}! \ldots d'_{i,K}!} \alpha^{d'_{i,i}} \prod_{1 \le j \le K, j \ne i} \beta^{d'_{i,j}}. \tag{3}$$

We assume the label assignment for instances from different classes are independent, then the data likelihood of **D'** is

$$L(\mathbf{G}, \alpha) = P(\mathbf{D'}|\mathbf{G}, \alpha) = \prod_{i=1}^{K} P(d'_{i,1}, \ldots, d'_{i,K}). \tag{4}$$

We have modeled the likelihood of **D'** with **G** and α being the model parameters in Equation (4). The maximum likelihood estimation of **G** and α is obtained by

$$\mathbf{G}^*, \alpha^* = \arg\max_{\mathbf{G}, \alpha} L(\mathbf{G}, \alpha). \tag{5}$$

Substitute Equation (3) into Equation (4),

$$L(\mathbf{G}, \alpha) = \alpha^{\mathrm{tr}(\mathbf{D'})} \beta^{m-\mathrm{tr}(\mathbf{D'})} \times \prod_{i=1}^{K} \frac{m_i!}{d'_{i,1}! \dots d'_{i,K}!}. \tag{6}$$

The sequence product term $\prod_{i=1}^{K} \frac{m_i!}{d'_{i,1}! \dots d'_{i,K}!}$ is constant with respect to $\mathbf{G}$ and α. In addition, we assume the accuracy is better than random guess, that is $\alpha > 1/K$, so we have $\alpha > (1-\alpha)/(K-1) = \beta$. In this case, $L(\mathbf{G}, \alpha)$ is a monotonically increasing function with respect to $\mathrm{tr}(\mathbf{D'})$. By maximizing $L(\mathbf{G}, \alpha)$, we obtain $\mathbf{G}^*$ and α^* as

$$\begin{aligned} \mathbf{G}^* &= \arg\max_{\mathbf{G}} \mathrm{tr}(\mathbf{D'}) = \arg\max_{\mathbf{G}} \mathrm{tr}(\mathbf{DG}) \\ \alpha^* &= \frac{\mathrm{tr}(\mathbf{DG}^*)}{m}, \end{aligned} \tag{7}$$

where tr() denotes the trace of a matrix, i.e., the summation of the elements on the main diagonal of a matrix. Substitute the form of $\mathbf{G}$ from Equation (2) into Equation (7):

$$\mathbf{G}^* = \arg\max_{\mathbf{G}} \sum_{i,j} d_{i,j} g_{i,j} = \arg\max_{\mathbf{G}} \sum_{i=1}^{K} d_{i,g(i)}. \tag{8}$$

This is essentially the *assignment problem*, and there are known algorithms [11] that solve it with a worst case time complexity of $O(K^3)$.

As an example, when $K = 2$, the solution of g to Equation (8) is

$$g(i) = \begin{cases} i, \text{ if } d_{1,1} + d_{2,2} \geq d_{1,2} + d_{2,1} \\ 1 - i, \text{ otherwise} \end{cases} \qquad i = 1, 2. \tag{9}$$

Accordingly, $\mathbf{G}^*$ is

$$\mathbf{G}^* = \begin{cases} I, \text{ if } d_{1,1} + d_{2,2} \geq d_{1,2} + d_{2,1} \\ 1 - I, \text{ otherwise} \end{cases}, \tag{10}$$

where I is the identity matrix.

Modeling the Generalization Error. Given the clustering result and the learned mapping G^*, we are now ready to estimate the generalization error α of our results. While Equation (7) provides a point maximum likelihood estimation of α^* for α, it does not provide a confidence level for the estimation. We provide an estimation for α along with a confidence level by leveraging its distribution. Specifically, since we use $\mathbf{G}^*$ to assign labels for unlabeled data, the posterior distribution of α can be obtained by the Bayes rule:

$$\begin{aligned} P(\alpha|\mathbf{D}', \mathbf{G}^*) &= \frac{P(\alpha)P(\mathbf{D}'|\mathbf{G}^*, \alpha)}{P(\mathbf{D}'|\mathbf{G}^*)} \\ &= \frac{P(\alpha)P(\mathbf{D}'|\mathbf{G}^*, \alpha)}{\int_0^1 P(\alpha)P(\mathbf{D}'|\mathbf{G}^*, \alpha)d\alpha}. \end{aligned} \tag{11}$$

We use a non-informative uniform prior, that is $P(\alpha) = \mathrm{unif}(0, 1)$. The likelihood $P(\mathbf{D}'|\mathbf{G}^*, \alpha)$ can be obtained by substituting Equation (8) into Equation (4).

3.2 Q1: Is the Best Mapping Found?

In this section, we estimate the probability that the best mapping is found (which gives the best classification accuracy) and the number of additional labeled examples needed to find the best mapping (thus, achieve the best accuracy), so that users can make decision on whether to obtain more labeled examples or not.

Different cluster-to-class mapping $\mathbf{G}^*$ leads to different prediction accuracy on the unlabeled data. Given a support set, we would like to provide a lower-bound probability P_l that the estimated mapping $\mathbf{G}^*$ acquired by Equation (7) is the best. For that, we first give a formal definition of the best mapping. We define the mapping given by Equation (7) when the support set size is infinite is the best, because it is expected to give the highest classification accuracy. Let $\mathbf{G}^{**}$ denote the the optimal mapping, and it can be concretely defined as

$$\mathbf{G}^{**} = \lim_{\forall_i, m_i \to +\infty} \mathbf{G}^*. \tag{12}$$

In other words, the optimal mapping can be defined as the mapping found when the development set size is infinitely large.

The best mapping is found when $\mathbf{G}^* = \mathbf{G}^{**}$. We would like to find a P_l such that:

$$P(\mathbf{G}^* = \mathbf{G}^{**}) > P_l. \tag{13}$$

Since we can always reorder the clusters to make $\mathbf{G}^{**}$ an identity matrix, without loss of generality, we assume $\mathbf{G}^{**}$ to be the identity matrix I to simplify notation. A specific case that makes the $\mathbf{G}^*$ obtained by Equation (7) satisfy $\mathbf{G}^* = \mathbf{G}^{**} = I$ is that

$$\forall i, d_{i,i} > \max_{1 \le j \le K, j \ne i} d_{i,j}. \tag{14}$$

The probability of $\mathbf{G}^* = \mathbf{G}^{**}$ is greater than the probability of a specific case that makes $\mathbf{G}^* = \mathbf{G}^{**}$ hold, as there can be other cases that make it hold. Therefore

$$\begin{aligned} P(\mathbf{G}^* = \mathbf{G}^{**}) &> \prod_{i=1}^{K} P\left(d_{i,i} > \max_{1 \le j \le K, j \ne i} d_{i,j}\right) \\ &= \prod_{i=1}^{K} \left(\sum_{d_{i,1},\ldots,d_{i,K}} P\left(d_{i,1}, \ldots, d_{i,K}\right) \right. \\ &\qquad \left. \text{s.t. } d_{i,i} > \max_{1 \le j \le K, j \ne i} d_{i,j} \right) \\ &= P_l. \end{aligned} \tag{15}$$

As in Equation (3), $\forall i$ $P(d_{i,1}, \ldots, d_{i,K})$ follows a multinomial distribution, so the right-hand side of Equation (15) can be obtained. Thus, given a support set size, and the true accuracy α, we derived a lower-bound probability P_l that the best mapping is found in Equation (15). As an example, when $P_l = 0.8$, it is guaranteed that the probability of the estimated mapping $\mathbf{G}^*$ being the best mapping is greater than 0.8.

Figure 3 illustrates the lower-bound probability P_l for $K = 2$ under different classification accuracy α. In the figure, we assumed each class has equal number of instances in the support set.

However, computing the lower bound needs the true accuracy α (α is required in Equation (3)), which is unknown. We use the estimated distribution of α in Equation (11) to obtain the expected lower-bound probability $\mathbb{E}(P_l)$ as follows:

$$\mathbb{E}(P_l) = \int_0^1 P(\alpha | \mathbf{D}', \mathbf{G}^*) P_l \, d\alpha. \tag{16}$$

Number of more examples to find the best mapping. When the estimated lower-bound probability P_l is not satisfactory, e.g., $P_l = 0.6$, the user would like to know how many additional examples are needed in the support set to obtain a better guarantee, e.g., $P_l = 0.8$.

Let p_0 denote the threshold (e.g., 0.8) that the user would like P_l to surpass. Let m_i' denote the smallest number of instances in the support set for the ith class, $1 \le i \le K$, that makes $P_l \ge p_0$.

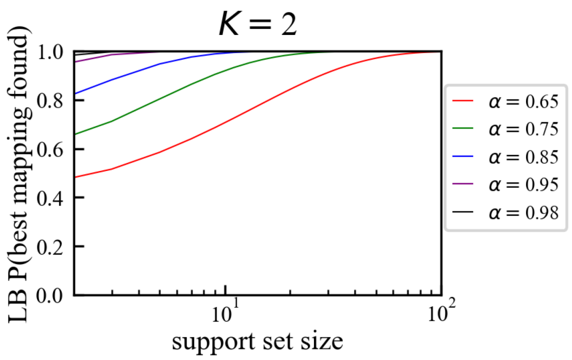

Fig. 3. Lower-bound probability of finding the best mapping when $K = 2$.

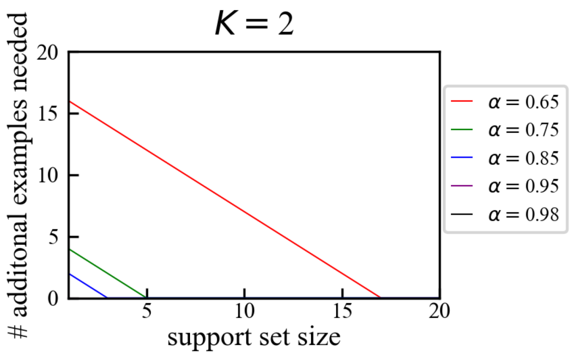

Fig. 4. Number of additional labeled examples required to ensure the probability of finding the best mapping is greater than 0.8.

$\{m'_1, m'_2, \ldots, m'_K\}$ can be obtained by

$$m'_1, \ldots, m'_K = \arg \min_{m_1, \ldots, m_K} \sum_{i=1}^{k} m_i, \text{ s.t. } P_l \geq p_0. \tag{17}$$

Equation (17) tries to minimize the support set size while ensuring $P_l \geq p_0$. The number of additional labeled examples required in the ith class is

$$\Delta m_i = m'_i - m_i, 1 \leq i \leq K. \tag{18}$$

Note that P_l is a lower-bound probability, so the additional size derived in Equation (18) is an upper-bound size, in other words, the actual size needed is often smaller. We also note we have no constraint on the required additional examples and they can simply be a random sample of the data points in the ith class.

Figure 4 illustrates the number of additional labeled examples needed when $p_0 = 0.8$ for different accuracy α and support set size m.

Again, computing Δm_i requires the accuracy α that is unknown. We use the estimated distribution of α to obtain the expected required size for each class $\mathbb{E}(\Delta m_i)$ as follows:

$$\mathbb{E}(\Delta m_i) = \int_0^1 P(\alpha | \mathbf{D}', \mathbf{G}^*) \Delta m_i d\alpha. \tag{19}$$

3.3 Q2: Is the Labeling Accuracy Acceptable?

The probability that the generalization error of the found mapping is below a certain threshold ϵ equals to the probability that the generalization accuracy is above the threshold $1 - \epsilon$. Given the estimated distribution of generalization accuracy α in Equation (11), the probability of the generalization accuracy being above the threshold $1 - \epsilon$ is

$$P(\alpha > 1 - \epsilon | \mathbf{D}',\ \mathbf{G}^*) = \int_{1-\epsilon}^{1} P(\alpha | \mathbf{D}', \mathbf{G}^*) d\alpha. \tag{20}$$

The user sets a threshold ϵ and looks at the probability provided by Equation (20) to decide whether he/she is satisfied. For example, when $\epsilon = 0.3$ and $P(\alpha > 1 - \epsilon | \mathbf{D}',\ \mathbf{G}^*) = 0.8$, i.e., the probability that the generalization error is below 0.3 is 0.8, the user may be satisfied with the results.

When the user provided multiple featurization functions, we select the best one automatically by choose the one that maximizes Equation (20).

4 APPLICATION TO IMAGE DATA LABELING

The application of our FSL method to data labeling is straightforward. After user provides some featurization functions, our method is able to to generate probabilistic labels for unlabeled data. These labels can then be used to train downstream ML models. It has been shown that as the amount of unlabeled data increases, the generalization error of the model trained with probabilistic labels will decrease at the same asymptotic rate as supervised models do with additional labeled data [21].

The quality of featurization functions decides the accuracy of labeling. Users can manually write featurization functions that encode their domain knowledge or directly use any pre-trained models. In this work, we only consider labeling image data. For image datasets, the pre-trained models are especially powerful. Typically, a direct invocation of pre-trained image model and using the output from the last logits layer will give fair good results. To better make use of the pre-trained models, we propose an Affinity Coding Paradigm that unitizes the knowledge encoded in a pre-trained model in a more comprehensive way. The Affinity Coding Paradigm also allows users to write similarity functions as featurization functions. This is especially useful, because often it is more straightforward to write distance/similarity functions to describe whether two instances belong to a same class. We build the system GOOGLES [5] that implements the Affinity Coding Paradigm for featurization (and our FSL method for label generation).

4.1 The Affinity Coding Paradigm

Affinity Matrix Construction. An affinity function takes two instances and output a real value representing their similarity. Given a library of α affinity functions $\mathcal{F} = \{f_0, f_1, \ldots, f_{\alpha-1}\}$, a set of n unlabeled instances $\{x_0, \ldots, x_{n-1}\}$, and a small m labeled examples $\{(x_n, y_n), \ldots, (x_{n+m-1}, y_{n+m-1})\}$ as the development set, we construct an affinity matrix $\mathcal{A} \in \mathbb{R}^{(n+m) \times \alpha(n+m)}$ that encodes all affinity scores between all pairs of instances under all affinity functions. Specifically, the ith row of $\mathcal{A}$ corresponds to instance x_i and every jth column of $\mathcal{A}$ corresponds to the affinity function $f_{j/(n+m)}$ and the instance $x_{j\%(n+m)}$, namely, $\mathcal{A}[i,j] = f_{j/(n+m)}(x_i, x_{j\%(n+m)})$.

Once we have $\mathcal{A}$, we use it as features to infer the class membership for all unlabeled instances using our FSL method to generate probabilistic labels.

Discussion. The affinity coding paradigm offers a domain-agnostic paradigm for creating good featurization functions. Our assumption is that, for a new dataset, there exists one or multiple affinity functions in our library $\mathcal{F}$ that can capture some kinds of similarities between instances in

ALGORITHM 1: Coding multiple affinity functions $f_0, f_1, \ldots f_{\alpha-1}$ based on the pre-trained VGG model

```
Input: Two unlabeled images $x_i$ and $x_j$
Output: Affinity scores between $x_i$ and $x_j$ under $f_0, f_1, \ldots f_{\alpha-1}$.
1: for all each max-pooling layer $L$ in VGG-16 do
2:     For image $x_j$, extract all of its prototypes $\boldsymbol{\rho}_j = \{v_j^{(1,1)}, v_j^{(1,2)}, \ldots, v_j^{H \times W}\}$ by passing it through
       the pre-trained VGG until layer $L$ to obtain a filter map of size $C \times H \times W$, where $C$, $H$ and
       $W$ are the number of channels, height and width of the filter map, respectively, and each
       prototype is vector of length $C$.
3:     Selecting $Z$ most activated prototypes of $x_j$, denoted as $\{v_j^1, v_j^2, \ldots, v_j^Z\}$
4:     Similarly, for image $x_i$, extract all of its prototypes $\boldsymbol{\rho}_i = \{v_i^{(1,1)}, v_i^{(1,2)}, \ldots, v_i^{H \times W}\}$
5:     for all $v_j^k \in \{v_j^1, v_j^2, \ldots, v_j^Z\}$, where $z \in [1, Z]$ do
6:         $f_L^z(x_i, x_j) \leftarrow \max_{h,w} sim(v_j^Z, v_i^{(h,w)})$
7:     end for
8: end for
```

the same class. We verify that our assumption holds on all four datasets we tested. It is particularly worth noting that, out of the four datasets, three of them are in completely different domains than the ImageNet dataset the VGG-16 model is trained on. This suggests that our current $\mathcal{F}$ is quite comprehensive. We acknowledge that there certainly exists potential new labeling tasks that our current set of affinity functions $\mathcal{F}$ would fail.

Our affinity coding paradigm is based on the proposition that examples belonging to the same class should have certain similarities. For image datasets, this proposition translates to *images from the same class would share certain visually discriminative high-level semantic features.* However, it is nontrivial to design affinity functions that capture these high-level semantic features due to two challenges: (1) without knowing which classes and labeling task we may have in the future, we do not even know what those features are, and (2) even assuming we know the particular features that are useful for a given class, they might be spatially located in different regions of images in the same class.

To address these challenges, GOGGLES leverages pre-trained convolutional neural networks (VGG-16 network [25] in our current implementation) to transplant the data representation from the raw pixel space to semantic space. It has been shown that intermediate layers of a trained neural network are able to encode different levels of semantic features, such as edges and corners in initial layers; and textures, objects and complex patterns in final layers [34].

Algorithm 1 gives the overall procedure of leveraging the VGG-16 network for coding multiple affinity functions. Specifically, to address the issue of not knowing which high-level features might be needed in the future, we use different layers of the VGG-16 network to capture different high-level features that might be useful for different future labeling tasks (Line 1). We call each such high-level feature a *prototype* (Line 2). As not all prototypes are actually informative features, we keep the top-Z most "activated" prototypes, which we treat as informative high-level semantic features (Line 3). For every one of the informative prototype v_j^k extracted from an image x_j, we need to design an affinity function that checks whether another image x_i has a similar prototype (Line 5). Since these prototypes might be located in different regions, our affinity function is defined to be the maximum similarity between all prototypes of x_i and v_j^k (Line 6).

We discuss prototype extraction and selection in Section 4.2, and the computation of affinity functions based on prototypes in Section 4.3.

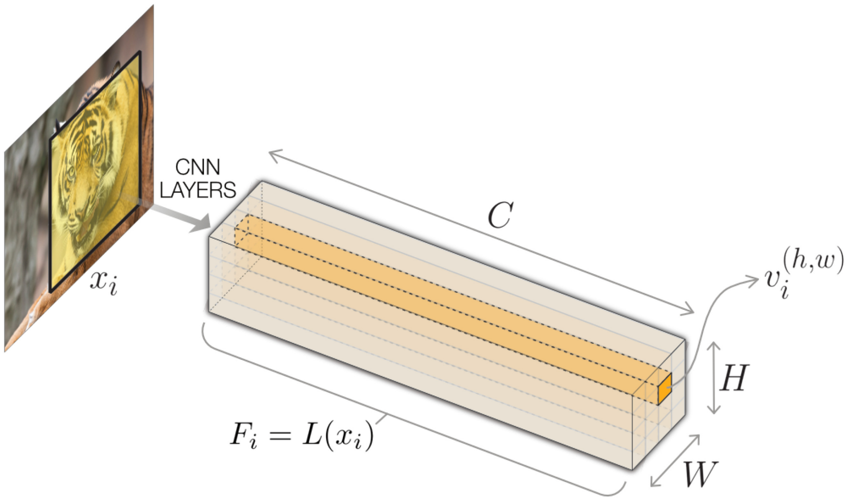

Fig. 5. Extracting all prototypes in a layer of the VGG-16 network. Each prototype corresponds to a patch in the input image's raw pixel space.

4.2 Extracting Prototypes

In this subsection, we discuss (1) how to extract all prototypes from a given image x_i using a particular layer L of the VGG-16 network and (2) how to select top-Z most informative prototypes amongst all the extracted ones.

Extracting all prototypes. To begin, we pass an image x_i through a series of layers until reaching a max-pooling layer L of a CNN to obtain the $F_i = L(x_i)$, known as a *filter map*. We choose max-pooling layers as they condense the previous convolutional operations to provide compact feature representations. The filter map F_i has dimensions $C \times H \times W$, where C, H, and W are the number of channels, height, and width of the filter map, respectively. Let us also denote indexes over the height and width dimensions of F_i with h and w, respectively. Each vector $v_i^{(h,w)} \in \mathbb{R}^C$ (spanning the channel axis) in the filter map F_i can be backtracked to a rectangular patch in the input image x_i, formally known as the *receptive field* of $v_i^{(h,w)}$. The location of the corresponding patch of a vector $v_i^{(h,w)}$ can be determined via gradient computation. Since any change in this patch will induce a change in the vector $v_i^{(h,w)}$, we say that $v_i^{(h,w)}$ encodes the semantic concept present in the patch. Formally, all prototypes we extract for x_i are as follows:

$$\boldsymbol{\rho}_i = \left\{ v_i^{(1,1)}, v_i^{(1,2)}, \ldots, v_i^{(H,W)} \right\}.$$

Example 2. Figure 5 shows the representation of an image patch in semantic space using a tiger image. An image x_i is passed through VGG-16 until a max-pooling layer to obtain the filter map F_i that has dimensions $C \times H \times W$. In this particular example, the yellow rectangular patch highlighted in the image is the receptive field of the orange prototype $v_i^{(h,w)}$, which as we can see, captures the "tiger's head" concept.

Selecting top-Z informative prototypes. In an image x_i, obviously not every patch and the corresponding prototype $v_i^{(h,w)}$ is a good signal. In fact, many patches in an image correspond to background noise that are uninformative for determining its class. Therefore, we need a way to intelligently select the top-Z most informative semantic prototypes from all the $H \times W$ possible ones.

In this regard, we first select top-Z channels that have the highest magnitudes of activation. Note that each channel is a matrix $\mathbb{R}^{H \times W}$, and the activation of a channel is defined to be the maximum value of its matrix (typically known as the 2D Global Max Pooling operation in CNNs). We denote the indexes of these top-Z channels as c_z, where $z \in \{1, \ldots, Z\}$. Based on the top-Z channels, we can thus define the top-Z prototypes as follows:

$$v_i^z = v_i^{(h,w)}, \text{where } h, w = \underset{h,w}{\operatorname{argmax}} F_i[c_z; h; w]. \tag{21}$$

The top-Z prototypes we extract for image x_i are

$$\boldsymbol{\rho}_i = \left\{ v_i^1, v_i^2, \ldots, v_i^Z \right\}.$$

The pair (h, w) may not be unique across the channels, yielding the same concept prototypes. Hence, we drop the duplicate $v_i^{(h,w)}$'s and only keep the unique prototypes.

Example 3. We illustrate our approach for selecting top-Z prototypes by an example. Suppose we would like to select top-2 prototypes in a layer that produces the following filter map of dimension $C \times H \times W = 3 \times 2 \times 2$. The three channels are as follows:

$$C_1 = \begin{bmatrix} 1 & 0.5 \\ 0.3 & 0.6 \end{bmatrix} C_2 = \begin{bmatrix} 0.1 & 0.7 \\ 0.4 & 0.3 \end{bmatrix} C_3 = \begin{bmatrix} 0.2 & 0.9 \\ 0.5 & 0.1 \end{bmatrix}.$$

First, we sort the three channels by the maximum activation in descent order i.e., the maximum element in the matrix: C_1, C_3, C_2. Then, we select the first $Z = 2$ channels: C_1, C_3. Next, for each of the selected channels we identify the index of its maximum element on the H and W axis: $(h_1, w_1) = (0, 0), (h_2, w_2) = (0, 1)$. Finally, we obtain the $Z = 2$ prototypes by stacking the values over all channels that share the same H and W axis index identified in the last step:
$v^1 = \{C_1[h_1, w_1],\ C_2[h_1, w_1],\ C_3[h_1, w_1]\} = \{1, 0.1, 0.2\}$, and $v^2 = \{C_1[h_2, w_2],\ C_2[h_2, w_2],$ $C_3[h_2, w_2]\} = \{0.5, 0.7, 0.9\}$.

4.3 Computing Affinity

Having extracted prototypes for each image, we are ready to define affinity functions and compute affinity scores for a pair of images (x_i, x_j). Affinity functions are supposed to capture various types of similarity between a pair of images. Intuitively, two images are similar if they share some high-level semantic concepts that are captured by our extracted prototypes. Based on this observation, we define multiple affinity functions, each corresponding to a particular type of semantic concept (prototype). Therefore, the number of affinity functions we can define is equal to the number of max-pooling layers (5) of the VGG-16 network multiplied by the number of top-Z prototypes extracted per layer.

Let us consider a particular prototype v_j^z, that is, the zth most informative prototype of x_j extracted from layer L, we define an affinity function as follows:

$$f_L^z(x_i, x_j) = \max_{h,w} sim\left(v_j^z, v_i^{(h,w)}\right). \tag{22}$$

As we can see, we calculate the similarity between a prototype v_j^z of x_j and the vector $v_i^{(h,w)} \forall (h, w) \in \{(1, 1), \ldots, (H, W)\}$ contained in $F_i = f(x_i)$ using a similarity function $sim(\cdot)$, and pick the highest value as the affinity score. In other words, our approach tries to find the "most similar patch" in each image x_i with respect to a given patch corresponding to one of the top-Z prototypes of image x_j. We use the cosine similarity metric as the similarity function $sim(\cdot)$

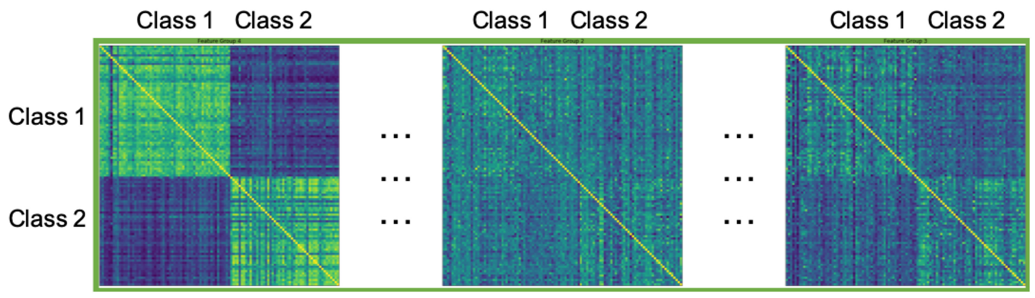

Fig. 6. An affinity matrix visualized as a heatmap.

defined over two vectors a and b as follows:

$$sim(a, b) = \frac{a^T b}{\|a\|_2 \|b\|_2}. \tag{23}$$

Example 4. Figure 6 shows an example affinity matrix $\mathcal{A}$ for the CUB dataset we use in the experiments. It only shows three of the 50 affinity functions. The rows and columns are sorted by class only for visual intuition. As we can see, some affinity functions are more informative than others in this labeling task.

We use all five max-pooling layers from the VGG-16. For each max-pooling layer, we use the top-10 prototypes, which we empirically find to be sufficient. Note that while we choose VGG-16 to define affinity functions in the current GOGGLES implementation, GOGGLES can be easily extended to use any other representation learning techniques.

In summary, our approach automatically identifies semantically meaningful prototypes from the dataset, and leverages these prototypes for defining affinity functions to produce an affinity matrix.

Noisy featurization/affinity functions. A good set of featurization functions is the foundation of good labeling accuracy of our system. It is natural that some featurization functions (or equivalently the affinity functions constructed upon them) can be noisy as it is possible that not all featurization/affinity functions are relevant to the task at hand. If the number of noisy functions is small, then some clustering methods like GMM naturally handle feature selection as the components will not be well separated by noisy functions and will be well separated by "good" functions. However, under such high dimensionality, there could exist too many noisy features that could form false correlations among them and eventually undermine the accuracy of the clustering methods and affect the learned cluster-to-class mapping and eventually decreases the accuracy of the generated labels. The current set of chosen featurization/affinity functions are empirically shown to be quite robust for various image labeling tasks as shown in the experiments.

Discussion: Extension to different modalities. While we currently indeed only focus on image data labeling, our approach can in principle be applied to other data modalities. Our current design of affinity functions is based on VGG-16 that is specific to image data. Extending to a different modality requires a different design of the affinity functions. For example for text data, one may design the affinity functions based on pre-trained language models (e.g., BERT [6]) or based on classic string similarity functions (e.g., Edit distance).

5 EXPERIMENTS

We conduct extensive experiments to evaluate the proposed method. Specifically, we focus on the following three questions:

(1) How good is the classification accuracy of the proposed method compared with the existing FSL methods?
(2) How accurate is the estimated generalization error compared with other ways of estimation?
(3) How good is the end-to-end performance of our proposed data labeling method GOGGLES compared with other data labeling systems?

The experiment section is organized as follows: In Section 5.1, we discuss the overall setup (e.g., datasets used) for the experiments; In Section 5.2, we evaluate the accuracy of FSL methods; In Section 5.3, we evaluate the accuracy of generalization error estimation. In Section 5.4, we evaluate the performance of GOGGLES compared with state-of-the-art data labeling systems.

5.1 Setup

We only consider image datasets in this work. All our experiments were performed on a machine with a 2.20-GHz Intel Xeon Gold 5120 CPU, $8 \times$ GeForce RTX 2080 GPU, and with 96 GB 2666 MHz RAM. The reported results are the averaged result of 10 runs.

5.1.1 Featurization Functions. We use one general featurization function to show our approach would work in a general task. Furthermore, to demonstrate our approach is able to incorporate user-provided information, we additionally select two domain-specific featurization functions to simulate the input of an user with domain knowledge.

For the general purpose featurization function, we use a pretrained VGG-16 network [26] that was trained on ImageNet [23]. For the evaluation of FSL in Sections 5.2 and 5.3, we use the last logits layer from the trained VGG-16 model, which is the output of the last fully connected layer, before it is fed to the softmax operation. This is for a pair comparison, since most FSL methods only use the output of last layer for prediction. For the evaluation of data labeling, we additionally use the featurization method (affinity coding) proposed in Section 4.

We use the **color histogram (CH)** and **histogram of oriented gradients (HOG)** as exemplar user-provided featurization functions that encode domain-specific knowledge. Color histogram represents the frequency of various color values in the image, so it is useful when the color is different in different classes. The HOG descriptor [4] represents an image by counting the number of occurrences of gradient orientation in localized portions of the image. Therefore, HOG is powerful only when the gradient orientation is significantly different in different classes.

5.1.2 Datasets. We are primarily concerned with realistic cross-domain FSL scenarios. Since our featurization function VGG-16 was trained on ImageNet, we select datasets that have little or no overlap with classes of images from the ImageNet dataset. We perform experiments on the following datasets, which are roughly ordered by domain overlap with ImageNet:

- **CUB**: The Caltech-UCSD Birds-200-2011 dataset [30] comprises of 11,788 images of 200 bird species. The dataset also provides 312 binary image-level attribute annotations that help explain the visual characteristics of the bird in the image, e.g., white head, grey wing, and so on. We use this metadata information for designing binary labeling functions that are used by a data programming system. To evaluate the task of generating binary labels, we randomly sample 10 class-pairs from the 200 classes in the dataset and report the average performance across these 10 pairs for each experiment. These sampled class-pairs are not present in the ImageNet dataset. However, since ImageNet and CUB contain common images of other bird species, this dataset may have a higher degree of domain overlap with the images that VGG-16 was trained on.
- **Surface**: The surface finish dataset [16] contains 1280 images of industrial metallic parts that are classified as having "good" (smooth) or "bad" (rough) metallic surface finish. This is

a more challenging dataset, since the metallic components look very similar to the untrained eye, and has minimal degree of domain overlap with ImageNet.

- **TB-Xray**: The Shenzhen Hospital X-ray set [10] has 662 images belonging to 2 classes, normal lung X-ray and abnormal X-ray showing various manifestations of tuberculosis. These images are of the medical imaging domain and have absolutely no domain overlap with ImageNet.
- **PN-Xray**: The pneumonia chest X-ray dataset [12] consists of 5,856 chest X-ray images classified by trained radiologists as being normal or showing different types of pneumonia. These images are also of the medical imaging domain and have no domain overlap with ImageNet.

Development Set. As in typical FSL work, we assume a small development set with ground-truth labels is available. This is also the same assumption in the data labeling system Snuba [28]. By default, we use only five label annotations arbitrarily chosen from each class for this. Hence, for the task with binary labels, we use a development set having a size of 10 images for all the experiments. We report the performance on the remaining images from each dataset.

5.2 Evaluating Classification Accuracy

Since we assume no access to any source dataset, the only eligible few-shot learning method is fine-tuning on pretrained networks. Though fine-tuning is a relatively simple practice, a recent survey reveals that it actually has comparable performance to other state-of-the-art few-shot learning algorithms in a realistic cross-domain evaluation setting [3]. Apart from fine-tuning, we also would like to compare with an empirical upper bound that shows the best possible results. The evaluated methods are as follows:

Fine-tuning: We implement fine-tuning referring to a recent work [3] that achieves state-of-the-art performance in realistic cross-domain scenarios. Specifically, we fix the all other layers and use the support set to fine-tune the final fully connected classifier in the VGG-16. We also modify the last fully connected layer of the architecture to our corresponding number of classes. We use a Adam optimizer with a learning rate of 10^{-3} as in Reference [3].

Empirical upper bound: The bound is obtained via a typical supervised transfer learning approach. Specifically, we split the dataset to training, validation and testing set by 60%, 20% and 20%. We freeze the convolutional layers of the VGG-16 model and only update the weights of the fully connected layers in the VGG-16 architecture while training. We also modify the last fully connected "logits" layer of the architecture to our corresponding number of classes.

Our "cluster-then-label" method: Our approach is designed to interact with users to decide the size of support set and featurization function to use. For a fair comparison with the other baseline methods, we fix the size of support set, i.e., skip Q1 in Figure 1. As for featurization functions, we select the one that gives the best generalization accuracy. Specifically, we select the one with higher probability that the generalization accuracy is above 0.5 by Equation (20).

Table 1 shows the classification accuracy for all methods. We can make the following observations:

- The proposed method obtained better accuracy than fine-tuning in all datasets. On average, the proposed method shows an improvement of 9%, which is significant considering the upper bound is only 17% better than fine-tuning.
- The proposed method is able to select the best featurization function automatically on all datasets relying on the proposed generalization accuracy estimation.

Table 1. Classification Accuracy for All Methods

Dataset	Our "cluster-then-label" method				Fine-tuning	Upper bound
	Auto selected	CH	HOG	VGG		
CUB	**0.964**	0.545	0.629	0.964	0.847	0.984
Surface	**0.858**	0.495	0.858	0.541	0.760	0.920
TB-Xray	**0.691**	0.621	0.691	0.672	0.664	0.821
PN-Xray	**0.712**	0.705	0.531	0.712	0.683	0.742
Average	**0.806**	0.592	0.678	0.722	0.739	0.867

Gray denote empirical upper bound of a supervised method.

- In three of four datasets, the proposed method with VGG as featurization function works better than fine-tuning VGG. This is primarily due to that the proposed method also incorporates the information in the unlabeled examples by clustering.
- HOG is much better than the general purpose VGG as well as fine-tuning on Surface. This is because the "good" class and "bad" class images in surface are different in their texture, which is captured by gradient orientation in HOG. This demonstrates that our system is able to incorporate user-provided knowledge.

5.3 Evaluating Generalization Error Estimation

We also compare the generalization error estimation of the proposed method and the fine-tuning method using several settings:

- Fine-tuning with generalization error estimation using the support set. We use the support set to fine-tuning VGG-16 and obtained the classification error of the fine-tuned model on the support set as the generalization error estimation.
- Fine-tuning with generalization error estimation using train-valid split. We split the support set to training set and validation set by 50-50, resulting in five labeled examples in the training set and five labeled examples in the validation set. We fine-tuning VGG-16 on the training set and obtain the classification error on the validation set as the generalization error estimation. The train-valid split for fine-tuning results in an increase the true generalization error. We also obtain the error increase using a hold-out test set.
- Our "cluster-then-label" method with generalization error estimation directly using the support set. We use the support set to learn the cluster-to-class mapping and assign labels to the support set. We use the classification error on the support set as a point estimation of generalization error.
- Our "cluster-then-label" method with generalization error estimation by Equation (11). Our proposed method provides a distribution estimation in stead of a point estimation. For a fair comparison to other methods, we obtain an expected value of the distribution in Equation (11) as a point estimation to compare with others.

Note that the true generalization error for all evaluated methods can be different. We use a hold out test test to calculate the true generalization error ϵ^* for all methods. To compare the accuracy of all the estimates, we obtain the absolute difference $\Delta\epsilon$ between the estimated error $\hat{\epsilon}$ and the true error ϵ^*, i.e., $\Delta\epsilon = |\hat{\epsilon} - \epsilon^*|$.

Table 2 shows the generalization error estimation results for all methods. We can make the following observations:

- The proposed method gives better generalization error estimation than fine-tuning on all datasets. On average, the proposed method is 34% better than fine-tuning.

Table 2. Generalization Error Difference (|Estimated Generalization Error−True Generalization Error on Hold-Out Set|)

Dataset	Our "cluster-then-label" method		Fine-tuning	
	distribution	support set eval	train-valid-set split	support set eval
CUB	0.064	**0.043**	0.154 (0.137)	0.153
Surface	**0.058**	0.075	0.086 (0.051)	0.240
TB-Xray	**0.141**	0.185	0.284 (0.163)	0.336
PN-Xray	**0.123**	0.163	0.185 (0.175)	0.337
Average	**0.096**	0.116	0.176 (0.132)	0.267

Gray numbers in the "train-valid-set split" column denote the test error increase when spliting the support set to training and validation set for generalization error estimation. Bold numbers show the best results for each dataset

- For fine-tuning, spliting the dataset into training set and validation set gives better generalization error estimation than naively using the support set. However, this causes the true generalization error increase significantly due to fewer data used for training. The true generalization error increased by 50% on average.
- For the proposed method, directly using the support set already gives decent estimation, because the support set is only used to determine the mapping. The derived distribution in Equation (11) is able to give better generalization error estimation on average. This is because figuring out the mapping is also fitting, so naively using the support set on average underestimates the generalization error and overestimates the generalization accuracy. By introducing a uniform prior in Equation (11), we are able to calibrate the overestimated generalization accuracy and give a better estimate on average.

5.4 Evaluating Data Labeling Performance

We compare our automatic data labeling system GOGGLES with existing systems: Snorkel [20] and Snuba [28].

Snorkel is the first system that implements the data programming paradigm [21]. Snorkel requires humans to design several *labeling functions* that output a noisy label (or abstain) for each instance in the dataset. Snorkel then models the high-level interdependencies between the possibly conflicting labeling functions to produce probabilistic labels, which are then used to train an end model. For image datasets, these labeling functions typically work on metadata or extraneous annotations rather than image-based features, since it is very hard to hand design functions based on these features.

Since CUB is the only dataset having such metadata available, we report the mean performance of Snorkel on the 10 class-pairs sampled from the dataset by using the attribute annotations as labeling functions. More specifically, we combine CUB's image-level attribute annotations (which describe visual characteristics present in an image, such as white head, grey wing, etc.) with the class-level attribute information provided (e.g., class A has white head, class B has grey wing, etc.) to design labeling functions. Hence, each attribute annotation in the union of the class-specific attributes acts as a labeling function that outputs a binary label corresponding to the class that the attribute belongs to. If an attribute belongs to both classes from the class-pair, then the labeling function abstains. We used the open-source implementation provided by the authors with our labeling functions for generating the probabilistic labels for the CUB dataset.

Snuba extends Snorkel by further reducing human efforts in writing labeling functions. However, Snuba requires users to provide per-instance primitives for a dataset, and the system automatically generates a set of labeling functions using a labeled small development set.

Table 3. Evaluation of GOGGLES Labeling Accuracy on Training Set

Dataset	GOGGLES		Data Programming	
	with affinity	without affinity	Snorkel	Snuba
CUB	0.978	0.964	0.892	0.588
Surface	0.892	0.858	—	0.579
TB-Xray	0.769	0.691	—	0.595
PN-Xray	0.744	0.712	—	0.555
Average	**0.846**	0.806	—	0.579

"With affinity" denotes using the affinity functions proposed in Section 4 and "without affinity" means only using the last logits layer from the pre-trained model. The "—" symbol represents cases where evaluation was not possible. GOGGLES shows on average an improvement of 0.26 over the state-of-the-art data programming system Snuba.

Since all datasets do not come with user-provided primitives, to ensure a fair comparison with Snuba, we consulted with Snuba's authors multiple times. They suggested that we use a rich feature representation extracted from images as their primitives, which would allow Snuba to learn labeling functions. As such, we use the *logits layer* of the pre-trained VGG-16 model for this purpose, as it has been well documented in the domain of computer vision that such feature representations encode meaningful higher order semantics for images [7, 18]. For the VGG-16 model trained on ImageNet, this yields us feature vectors having 1000 dimensions for each image. To obtain densely rich primitives that are more tractable for Snuba, we project the logits output onto a feature space of the top-10 principal components of the entire data determined using principal component analysis [32]. We use these projected features having 10 dimensions as primitives for Snuba. Empirical testing revealed that providing more components does not change the results significantly. We also use the same development set size for Snuba and GOGGLES. We used the open-source implementation provided by the authors for learning labeling functions with automatically extracted primitives and for generating the final probabilistic labels.

5.4.1 Running Time and Human Effort. Our system includes four steps as shown in Figure 1: cluster, label, answering Q1, and answering Q2. The label step can be done in a few minutes and generating estimates for Q1 and Q2 can be done within tens of seconds. The running time is dominated by the cluster step that is dependent on the chosen clustering method. In our experiment, the running time of cluster with GMM ranges from 5 minutes to 10 minutes on the four tested datasets.

The human effort is the cost to provide the labeled development set that contains five labeled examples per class. For example on the Surface dataset that has two classes, the human effort is the cost to label 10 examples, which takes about 1 minute.

5.4.2 Labeling Accuracy. Table 3 shows the labeling accuracy for GOGGLES, Snorkel, Snuba, and a supervised approach that serves as an upper-bound reference for comparison. (1) GOGGLES achieves labeling accuracies ranging from a minimum of 0.74 to a maximum of 0.98. GOGGLES shows an average of 0.26 improvement over the state-of-the-art data programming system Snuba. (2) To ensure a fair comparison, we consulted with authors of Snuba and took their suggested approach of automatically extracting the required primitives. As we can see, the performances of Snuba on all datasets are just slightly better than random guessing. This is primarily because Snuba is really designed to operate on human annotated primitives. Furthermore, Snuba's performance degrades if the size of the development set is not sufficiently high. Our experiments showed that, indeed, if we increase the development set size for Snuba from 10 to 100 (10× increase) for the PN-Xray dataset, then the performance jumps from 0.55 to 0.67. In comparison, GOGGLES still

Table 4. Comparison of End Model Accuracy on Held-out Test Set

Dataset	Fine-tuning	Snorkel	Snuba	GOGGLES	Upper bound
CUB	0.847	0.879	0.563	0.953	0.984
Surface	0.760	—	0.517	0.833	0.920
TB-Xray	0.664	—	0.627	0.709	0.821
PN-Xray	0.683	—	0.622	0.691	0.742
Average	0.738	—	0.582	**0.797**	0.867

GOGGLES uses only five labeled instances per class but is only 7% away from the supervised upper bound (in gray) that uses the ground-truth labels of the training set.

performs better with a development set size of only 10 images. (3) We can only use Snorkel on CUB, as CUB is the only dataset that comes with annotations that we can leverage as labeling functions. These labeling functions may be considered perfect in terms of coverage and accuracy, since they are essentially human annotations. GOGGLES uses minimal human supervision and still outperforms Snorkel on CUB.

5.4.3 *End-to-end Performance.* We also use the probabilistic labels generated by Snorkel, Snuba and GOGGLES to train downstream discriminative models following the similar approach taken in References [20, 28]. Specifically, we use the VGG-16 as the downstream ML model architecture, and tune the weights of the last fully connected layers using cross-entropy loss. For training the FSL model, we use the same development set used by Snuba and GOGGLES for labeling. For training the upper-bound model, we use the entire training set labels. The performance of each approach on hold-out test sets is reported in Table 4.

First, GOGGLES outperforms Snuba by an average of 0.21, a similar number to the labeling performance improvement of 0.26 GOGGLES has over Snuba (cf. Table 3), and the end model performance of Snuba is worse than FSL. This is because the labels generated by Snuba are only slightly better than random guessing, and having many extremely noisy labels can be more harmful than having fewer labels in training an end model. Second, GOGGLES outperforms the fine-tuned state-of-the-art FSL method by an average of 6%, which is significant considering GOGGLES is only 7% away from the upper bound. Third, not surprisingly, the more accurate the generated labels are, the more performance gain GOGGLES has over FSL (e.g., the improvements are more significant on CUB and Surface, which have higher labeling accuracies compared with other datasets).

This experiment demonstrates the advantage GOGGLES has over FSL and data programming systems—GOGGLES has the exact same inputs compared with FSL (both only have access to the pre-trained VGG-16 and the development set), and does not require dataset-specific labeling functions needed by data programming systems.

5.4.4 *Varying Number of Affinity Functions.* To study the impact of the quality of the given set of affinity functions on the labeling accuracy, we simulate quality change by varying the number of affinity functions used. When less affinity functions are given, less usefully information is provided, in other words, the quality of the set of given labeling functions is low. The results are shown in Figure 7. Accuracy increases as the number of affinity functions increases for all datasets. This is understandable as more affinity functions brings more information that the inference module can exploit.

6 RELATED WORK

Our work relates to few-shot learning, semi-supervised learning, and data programming.

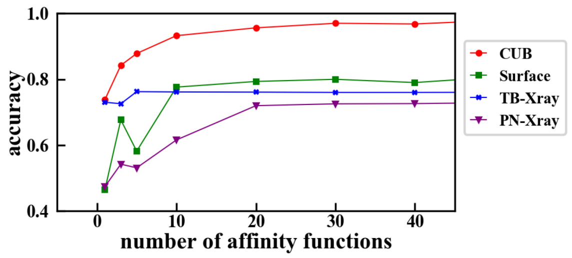

Fig. 7. Accuracy w.r.t. varying number of affinity functions.

There are many lines of methods on few-shot learning, include metric learning–based methods [15, 27, 29, 33], optimization-based methods [8, 17], hallucination-based methods [1, 31], model-based methods [24]. Most of them follow the meta-learning paradigm that extracts transferable knowledge from source datasets or tasks at meta-training phase to improve the generalization on new tasks at meta-testing phase. Two major differences of our approach are that we rely on the proposed generalization error estimation to select the best transferable knowledge (pre-trained models or domain-specific featurization functions) for new tasks and we don't require access to source datasets.

Our method also exploit the structure of the unlabeled data as in semi-supervised learning. The cluster-then-label procedure was originally proposed in semi-supervised learning [35] and it has been applied to varies fields [9, 13, 19]. Different from existing cluster-then-label semi-supervised methods, our approach utilizes transferable knowledge from source datasets. Most importantly, our approach provides an estimation of generalization error that no previous cluster-then-label semi-supervised learning approaches do.

Data programming [21] and Snuba [28] and Snorkel [20] systems that implement the paradigm were recently proposed in the data management community. Data programming focuses on reducing the human effort in training data labeling and is the most relevant work to ours. Data programming ingests domain knowledge in the form of labeling functions. Each labeling function takes an unlabeled instance as input and outputs a label with better-than-random accuracy (or abstain). However, using data programming for image labeling tasks is particularly challenging, as it requires images to have associated metadata (e.g., text annotations or primitives), and a different set of labeling functions is required for every new dataset. In contrast, our proposed method (affinity function for featurization and cluster-then-label for class inference) offers a domain-agnostic and automated approach for image labeling.

Since this work is an extension of the conference paper [5], we highlight the difference to the conference version as follows:

(1) In contrast to the heuristic objective in the conference paper (Equation (12) in Reference [5]), a more principled MLE formulation of the cluster-then-label approach is proposed.
(2) The proposed method in Section 3.1 is able to provide an estimate of generalization error, which has not been done before by any existing FSL methods as far as we know.
(3) The ability to estimate generalization error enables the possibility of providing feedback to users to involve humans in the loop, a core feature of this journal version. Specifically, the system estimates Q1 and Q2 to provide guidance for the user to choose the featurization function and to provide more labels.

7 CONCLUSION

We introduced a cluster-then-label approach for few-shot learning. Our approach has two major benefits: it does not require access to the source dataset and it provides generalization error estimation. In a realistic cross-domain setting on four datasets, we demonstrated experimentally that the proposed approach achieves better predictive performance than and provides more accurate generalization error estimation than existing methods.

We also explored the application of our proposed few-shot learning method to image data labeling. In addition, we propose the affinity coding paradigm for better featurization for image data labeling. We build the GOGGLES system that implements our few-shot learning method and affinity coding paradigm for labeling image datasets. GOGGLES is able to label images with high accuracy, requiring only a very small development set, which is economical to obtain.

REFERENCES

[1] Antreas Antoniou, Amos Storkey, and Harrison Edwards. 2017. Data augmentation generative adversarial networks. arXiv:1711.04340. Retrieved from https://arxiv.org/abs/1711.04340.

[2] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. 2010. *Semi-Supervised Learning* (1st ed.). The MIT Press.

[3] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. 2019. A closer look at few-shot classification. arXiv:1904.04232. Retrieved from https://arxiv.org/abs/1904.04232.

[4] Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. 886–893.

[5] Nilaksh Das, Sanya Chaba, Renzhi Wu, Sakshi Gandhi, Duen Horng Chau, and Xu Chu. 2020. GOGGLES: Automatic image labeling with affinity coding. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'20)*. Association for Computing Machinery, New York, NY, 1717–1732. https://doi.org/10.1145/3318464.3380592

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

[7] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the International Conference on Machine Learning*. 647–655.

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 1126–1135.

[9] Andrew Goldberg, Xiaojin Zhu, Aarti Singh, Zhiting Xu, and Robert Nowak. 2009. Multi-manifold semi-supervised learning. In *Artificial Intelligence and Statistics*. 169–176.

[10] Stefan Jaeger, Alexandros Karargyris, Sema Candemir, Les Folio, Jenifer Siegelman, Fiona Callaghan, Zhiyun Xue, Kannappan Palaniappan, Rahul K. Singh, Sameer Antani, et al. 2013. Automatic tuberculosis screening using chest radiographs. *IEEE Trans. Med. Imag.* 33, 2 (2013), 233–245.

[11] Roy Jonker and Anton Volgenant. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38, 4 (1987), 325–340.

[12] Daniel S. Kermany, Michael Goldbaum, Wenjia Cai, Carolina C. S. Valentim, Huiying Liang, Sally L. Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, et al. 2018. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell* 172, 5 (2018), 1122–1131.

[13] Santosh Kumar, Xiaoying Gao, and Ian Welch. 2017. Cluster-than-label: Semi-supervised approach for domain adaptation. In *Proceedings of the IEEE 31st International Conference on Advanced Information Networking and Applications (AINA'17)*. IEEE, 704–711.

[14] Joshua Lee, Prasanna Sattigeri, and Gregory Wornell. 2019. Learning new tricks from old dogs: Multi-source transfer learning from pre-trained networks. In *Advances in Neural Information Processing Systems*. 4372–4382.

[15] Huaiyu Li, Weiming Dong, Xing Mei, Chongyang Ma, Feiyue Huang, and Bao-Gang Hu. 2019. Lgm-net: Learning to generate matching networks for few-shot learning. arXiv:1905.06331. Retrieved from https://arxiv.org/abs/1905.06331.

[16] Wafa Louhichi. 2019. *Automated Surface Finish Inspection Using Convolutional Neural Networks*. Ph.D. Dissertation. Georgia Institute of Technology.

[17] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms. arXiv:1803.02999. Retrieved from https://arxiv.org/abs/1803.02999.

[18] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1717–1724.

[19] Mohammad Peikari, Sherine Salama, Sharon Nofech-Mozes, and Anne L. Martel. 2018. A cluster-then-label semi-supervised learning approach for pathology image classification. *Sci. Rep.* 8, 1 (2018), 1–13.

[20] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proc. VLDB Endow.* 11, 3 (2017), 269–282.

[21] Alexander J. Ratner, Christopher M. De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. In *Advances in Neural Information Processing Systems*. 3567–3575.

[22] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B. Tenenbaum, Hugo Larochelle, and Richard S. Zemel. 2018. Meta-learning for semi-supervised few-shot classification. arXiv:1803.00676. Retrieved from https://arxiv.org/abs/1803.00676.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.* 115, 3 (2015), 211–252.

[24] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *Proceedings of the International Conference on Machine Learning*. 1842–1850.

[25] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. Retrieved from https://arxiv.org/abs/1409.1556.

[26] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*.

[27] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*. 4077–4087.

[28] Paroma Varma and Christopher Ré. 2018. Snuba: Automating weak supervision to label training data. *Proc. VLDB Endow.* 12, 3 (2018), 223–236.

[29] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*. 3630–3638.

[30] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. 2011. The caltech-ucsd birds-200-2011 dataset.

[31] Yu-Xiong Wang, Ross Girshick, Martial Hebert, and Bharath Hariharan. 2018. Low-shot learning from imaginary data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7278–7286.

[32] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometr. Intell. Lab. Syst.* 2, 1–3 (1987), 37–52.

[33] Sung Whan Yoon, Jun Seo, and Jaekyun Moon. 2019. Tapnet: Neural network augmented with task-adaptive projection for few-shot learning. arXiv:1905.06549. Retrieved from https://arxiv.org/abs/1905.06549.

[34] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*. Springer, 818–833.

[35] Xiaojin Zhu and Andrew B. Goldberg. 2009. Introduction to semi-supervised learning. *Synth. Lect. Artif. Intell. Mach. Learn.* 3, 1 (2009), 1–130.

Received March 2021; revised August 2021; accepted October 2021