GOGGLES: Automatic Image Labeling with Affinity Coding

Nilaksh Das
nilakshdas@gatech.edu

Sanya Chaba
sanyachaba@gatech.edu

Renzhi Wu
renzhiwu@gatech.edu

Sakshi Gandhi
sakshi@gatech.edu

Duen Horng Chau
polo@gatech.edu

Xu Chu
xu.chu@cc.gatech.edu

Georgia Institute of Technology

ABSTRACT

Generating large labeled training data is becoming the biggest bottleneck in building and deploying supervised machine learning models. Recently, the data programming paradigm has been proposed to reduce the human cost in labeling training data. However, data programming relies on designing labeling functions which still requires significant domain expertise. Also, it is prohibitively difficult to write labeling functions for image datasets as it is hard to express domain knowledge using raw features for images (pixels).

We propose *affinity coding*, a new domain-agnostic paradigm for automated training data labeling. The core premise of affinity coding is that the affinity scores of instance pairs belonging to the same class on average should be higher than those of pairs belonging to different classes, according to some affinity functions. We build the GOGGLES system that implements affinity coding for labeling image datasets by designing a novel set of reusable affinity functions for images, and propose a novel hierarchical generative model for class inference using a small development set.

We compare GOGGLES with existing data programming systems on 5 image labeling tasks from diverse domains. GOGGLES achieves labeling accuracies ranging from a minimum of 71% to a maximum of 98% without requiring any extensive human annotation. In terms of end-to-end performance, GOGGLES outperforms the state-of-the-art data programming system Snuba by 21% and a state-of-the-art few-shot learning technique by 5%, and is only 7% away from the fully supervised upper bound.

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3380592

CCS CONCEPTS

• **Mathematics of computing → Probabilistic inference problems**; • **Computing methodologies → Computer vision representations**; *Cluster analysis*; Learning settings.

KEYWORDS

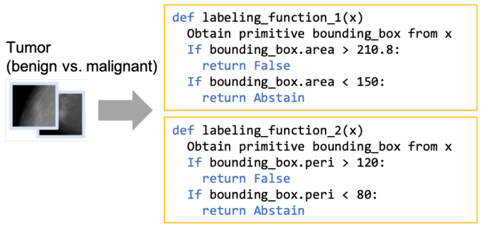
affinity coding, probabilistic labels, data programming, weak supervision, computer vision, image labeling

ACM Reference Format:
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 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3318464.3380592

1 INTRODUCTION

Machine learning (ML) is being increasingly used by organizations to gain insights from data and to solve a diverse set of important problems, such as fraud detection on structured tabular data, identifying product defects on images, and sentiment analysis on texts. A fundamental necessity for the success of ML algorithms is the existence of sufficient high-quality labeled training data. For example, the current ConvNet revolution would not be possible without big labeled datasets such as the 1M labeled images from ImageNet [23]. Modern deep learning methods often need tens of thousands to millions of training examples to reach peak predictive performance [28]. However, for many real-world applications, large hand-labeled training datasets either do not exist, or is extremely expensive to create as manually labeling data usually requires domain experts [6].

Existing Work. We are not the first to recognize the need for addressing the challenges arising from the lack of sufficient



```
def labeling_function_1(x)
  Obtain primitive bounding_box from x
  If bounding_box.area > 210.8:
    return False
  If bounding_box.area < 150:
    return Abstain
```

```
def labeling_function_2(x)
  Obtain primitive bounding_box from x
  If bounding_box.peri > 120:
    return False
  If bounding_box.peri < 80:
    return Abstain
```

Figure 1: Example LFs in data programming [29].

training data. The **ML community** has made significant progress in designing different model training paradigms to cope with limited labeled examples, such as semi-supervised learning techniques [38], transfer learning techniques [16] and few-shot learning techniques [3, 10, 32, 35]. In particular, the most related learning paradigm that shares a similar setup to us, few-shot learning techniques, usually require users to preselect a source dataset or pre-trained model that is in the same domain of the target classification task to achieve best performance. In contrast, our proposal can incorporate as many available sources of information as affinity functions.

Only recently, the *data programming* paradigm [20] and the Snorkel [19] and Snuba system [29] that implement the paradigm were proposed in the **data management community**. Data programming focuses on reducing the human effort in training data labeling, particularly in unstructured data classification tasks (images, text). Instead of asking humans to label each instance, data programming ingests domain knowledge in the form of labeling functions (LFs). Each LF takes an unlabeled instance as input and outputs a label with better-than-random accuracy (or abstain). Based on the agreements and disagreements of labels provided by a set of LFs, Snorkel/Snuba then infer the accuracy of different LFs as well as the final probabilistic label for every instance. The primary difference between Snorkel and Snuba is that while Snorkel requires human experts to write LFs, Snuba learns a set of LFs using a small set of labeled examples.

While data programming alleviates human efforts significantly, it still requires the construction of a new set of LFs for every new labeling task. In addition, we find that it is extremely challenging to design LFs for image labeling tasks primarily because raw pixels values are not informative enough for expressing LFs using either Snorkel or Snuba. After consulting with data programming authors, we confirmed that Snorkel/Snuba require images to have associated metadata, which are either text annotations (e.g., medical notes associated with X-Ray images) or primitives (e.g., bounding boxes for X-Ray images). These associated text annotations or primitives are usually difficult to come by in practice.

Example 1. *Figure 1 shows two example labeling functions for labeling an X-Ray image as either benign or malignant [29]. As we can see, these two functions rely on the bounding box primitive for each image and use the two properties (area and perimeter) of the primitive for labeling. We observe that these domain-specific primitives are difficult to obtain. Indeed, [29] states, in this particular example, radiologists have pre-extracted these bounding boxes for all images.*

Our Proposal. We propose *affinity coding*, a new domain-agnostic paradigm for automated training data labeling without requiring any domain specific functions. The core premise of the proposed affinity coding paradigm is that the *affinity scores of instance pairs belonging to the same class on average should be higher than those of instance pairs belonging to different classes, according to some affinity functions.* Note that this is quite a natural assumption — if two instances belong to the same class, then by definition, they should be similar to each other in some sense.

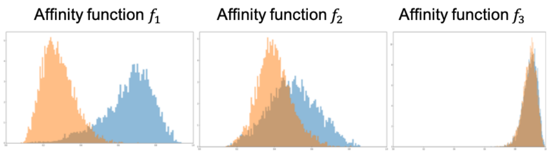


Figure 2: Affinity score distributions. Blue and yellow denote the affinity scores of instance pairs from the same class and different classes, respectively.

Example 2. *Figure 2 shows the affinity score distributions of a real dataset we use in our experiments (CUB) using three of the 50 affinity functions discussed in Section 3. In this particular case, affinity function f_1 is able to distinguish pairs in the same class from pairs in different classes very well; affinity function f_2 also has limited power in separating the two cases; and affinity function f_3 is not useful at all in separating the classes.*

We build the GOGGLES system that implements the affinity coding paradigm for labeling image datasets (Figure 3). First, GOGGLES includes a novel set of affinity functions that can capture various kinds of image affinities. Given a new unlabeled dataset and the set of affinity functions, we construct an affinity matrix. Second, using a very small set of labeled examples (development set), we can assign classes to unlabeled images based on the affinity score distributions we can learn from the affinity matrix. Compared with the state-of-the-art data programming systems, our affinity coding system GOGGLES has the following distinctive features.

- Data programming systems need some kinds of metadata (text annotations or domain-specific primitives) associated with each image to express LFs, while GOGGLES makes no such assumptions.
- Assuming the existence of metadata, data programming still requires a new set of LFs for every new dataset. In contrast, GOGGLES is a domain-agnostic system that leverages affinity functions, which are populated once and can be reused for any new dataset.

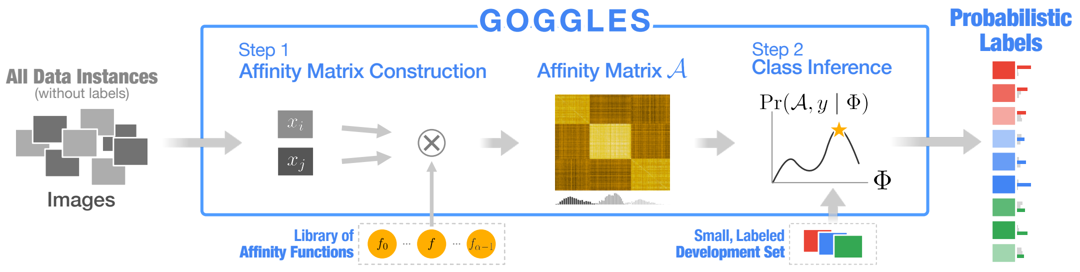

GOGGLES
All Data Instances
(without labels)
Images
Step 1
Affinity Matrix Construction
Affinity Matrix $\mathcal{A}$
Step 2
Class Inference
$\Pr(\mathcal{A}, y \mid \Phi)$
Φ
Library of Affinity Functions
Small, Labeled Development Set
Probabilistic Labels

Figure 3: Overview of the GOGGLES framework.

- Both Snorkel/Snuba and GOGGLES can be seen as systems that leverage many sources of weak supervision to infer labels. Intuitively, the more weak supervision sources a system has, the better labeling accuracy a system can potentially achieve. In data programming, the number of sources is the number of LFs. In contrast, affinity coding uses affinity scores between instance pairs under many affinity functions. Therefore, the number of sources GOGGLES has is essentially the number of instances multiplied by the number of affinity functions, a significantly bigger set of weak supervision sources.

Challenges. We address the following major challenges with GOGGLES:

- The success of affinity coding depends on a set of affinity functions that can capture similarities of images in the same class. However, without knowing which classes and labeling task we may have in the future, we do not even know what are the potential distinctive features for each class. Even if we have the knowledge of the particular distinctive features, they might be spatially located in different regions of images in the same class, which makes it more difficult to design domain-agnostic affinity functions.
- Given an affinity matrix constructed using the set of affinity functions, we need to design a robust class inference module that can infer class membership for all unlabeled instances. This is quite challenging for multiple reasons. First, some of the affinity functions are indicative for the current labeling, while many others are just noise, as shown in Example 2. Our class inference module needs to identify which affinity functions are useful given a labeling task. Second, the affinity matrix is high-dimensional with the number of dimension equals to the number of instances multiplied by the number of affinity functions. In this high-dimensional space, distance between any two rows in the affinity matrix becomes extremely small, and thus making it even more challenging to infer class assignments. Third, while we can infer from the affinity matrix which instances belong to the same class by essentially performing clustering, we still need to figure out which cluster corresponds to which class, relying only on a small development set.

Contributions. We make the following contributions:

- **The affinity coding paradigm**. We propose *affinity coding*, a new domain-agnostic paradigm for automatic generation of training data. Affinity coding paradigms consists of two main components: a set of affinity functions and a class inference algorithm. To the best of our knowledge, we are the first to propose a domain-agnostic approach for automated training data labeling.
- **Designing affinity functions.** GOGGLES features a novel approach that defines affinity functions based on a pre-trained VGG-16 model [26]. VGG-16 is a commonly used network for representation learning. Our intuition is that different layers of the VGG-16 network capture different high-level semantic concepts. Each layer may show different activation patterns depending on where a high-level concept is located in an image. We thus leverage all 5 max-pooling layers of the network, extracting 10 affinity functions per layer, for a total of 50 affinity functions.
- **Class inference using hierarchical generative model.** GOGGLES proposes a novel hierarchical model to identify instances of the same class by maximizing the data likelihood under the generative model. The hierarchical generative model consists of two layers: the base layer consists of multiple Gaussian Mixture Models (GMMs), each modeling an affinity function; and the ensemble layer takes the predictions from each GMM and uses another generative model based on multivariate Bernoulli distribution to obtain the final labels. We show that our hierarchical generative model addresses both the curse of dimensionality problem and the affinity function selection problem. We also give theoretical justifications on the size of development set needed to get correct cluster-to-class assignment.

GOGGLES achieves labeling accuracy ranging from a minimum of 71% to a maximum of 98%. In terms of end-to-end performance, GOGGLES outperforms the state-of-the-art data programming system Snuba by 21% and a state-of-the-art few-shot learning technique by 5%, and is only 7% away from the fully supervised upper bound. We also make our implementation of GOGGLES open-source on GitHub[1].

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

2 PRELIMINARY

We formally state the problem of automatic training data labeling in Section 2.1. We then introduce *affinity coding*, a new paradigm for addressing the problem in Section 2.2.

2.1 Problem Setup

In traditional supervised classification applications, the goal is to learn a classifier h_θ based on a labeled training set (x_i, y_i), where $x_i \in \mathcal{X}_{train}$ and $y_i \in \mathcal{Y}_{train}$. The classifier is then used to make predictions on a test set.

In our setting, we only have $\mathcal{X}_{train}$ and no $\mathcal{Y}_{train}$. Let n denote the total number of unlabeled data points, and let y_i^* denote the unknown true label for x_i. Our goal is to assign a *probabilistic label* $\tilde{y}_i^k$ for every $x_i \in \mathcal{X}_{train}$, where $\tilde{y}_i^k = \Pr([y_i^* = k]) \in [0, 1]$, where $k \in \{1, 2, \ldots, K\}$ with K being the number of classes in the labeling task, and $\sum_k \tilde{y}_i^k = 1$.

These probabilistic labels can then be used to train downstream ML models. For example, we can generate a discrete label according to the highest $\tilde{y}_i^q$ for every instance x_i. Another more principled approach is to use the probabilistic labels directly in the loss function $l(h_\theta(x_i), y)$, i.e., the expected loss with respect to $\tilde{y}$: $\hat{\theta} = \text{argmin}_\theta \sum_{i=1}^n E_{y \sim \tilde{y}_i}[l(h_\theta(x_i), y)]$. 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 [20].

2.2 The Affinity Coding Paradigm

We propose *affinity coding*, a domain-agnostic paradigm for automatic labeling of training data. Figure 3 depicts an overview of GOGGLES, an implementation of the paradigm.
Step 1: 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 i^{th} row of $\mathcal{A}$ corresponds to instance x_i and every j^{th} column of $\mathcal{A}$ corresponds to the affinity function $f_{j/(n+m)}$ and the instance $x_{j\%(n+m)}$, namely, $\mathcal{A}[i, j] = f_{j/(n+m)}(x_i, x_{j\%(n+m)})$.
Step 2: Class Inference. Given $\mathcal{A}$, we would like to infer the class membership for all unlabeled instances. For every unlabeled instance $x_i, i \in [0, n-1]$, we associate a hidden variable $\tilde{y}_i$ representing its unknown class. We aim to maximize the data likelihood $\Pr(\mathcal{A}, \tilde{y}|\Phi)$, where Φ denotes the parameters of the generative model used to generate $\mathcal{A}$.

Discussion. The affinity coding paradigm offers a domain-agnostic paradigm for training data labeling. 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 the same class. We verify that our assumption holds on all five datasets we tested. It is particularly worth noting that, out of the five 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.

3 DESIGNING AFFINITY FUNCTIONS

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 [26] 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 [37].

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 3.1, and the computation of affinity functions based on prototypes in Section 3.2.

Algorithm 1 Coding multiple affinity functions $f_0, f_1, \dots f_{\alpha-1}$ 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 $f_0, f_1, \dots f_{\alpha-1}$.
1: for all each max-pooling layer $L$ in VGG-16 do
2:     For image $x_j$, extract all of its prototypes $\boldsymbol{\rho}_j = \{v_j^{(1,1)}, v_j^{(1,2)}, \dots, v_j^{H\times W}\}$ 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 $\{v_j^1, v_j^2, \dots, v_j^Z\}$
4:     Similarly, for image $x_i$, extract all of its prototypes $\boldsymbol{\rho}_i = \{v_i^{(1,1)}, v_i^{(1,2)}, \dots, v_i^{H\times W}\}$
5:     for all $v_j^k \in \{v_j^1, v_j^2, \dots, v_j^Z\}$, 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
```

3.1 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:

$$\boldsymbol{\rho}_i = \{v_i^{(1,1)}, v_i^{(1,2)}, \dots, v_i^{(H,W)}\}$$

Example 3. *Figure 4 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.*

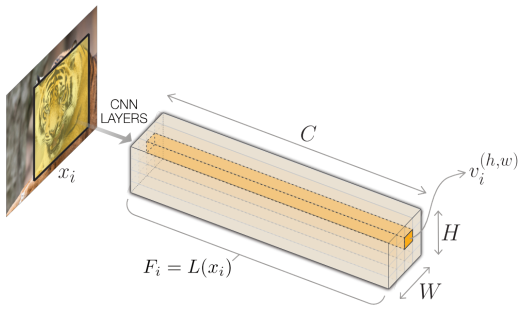


Figure 4: Extracting all prototypes in a layer of the VGG-16 network. Each prototype corresponds to a patch in the input image's raw pixel space.

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 $z \in \{1, \dots, Z\}$. Based on the top-Z channels, we can thus define the top-Z prototypes as follow:

$$v_i^z = v_i^{(h,w)}, \text{where } h, w = \operatorname*{argmax}_{h,w} F_i[c_z; h; w]. \quad (1)$$

The top-Z prototypes we extract for image x_i are:

$$\boldsymbol{\rho}_i = \{v_i^1, v_i^2, \dots, v_i^Z\}$$

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 4. *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:*

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

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.

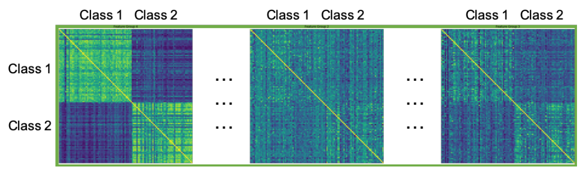


Figure 5: An affinity matrix visualized as a heatmap.

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* $v^2 = \{C_1[h_2, w_2], C_2[h_2, w_2], C_3[h_2, w_2]\} = \{0.5, 0.7, 0.9\}$.

3.2 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 z^{th} most informative prototype of x_j extracted from layer L, we define an affinity function as follows:

$$f_L^z(x_i, x_j) = \max_{h,w} sim(v_j^z, v_i^{(h,w)}) \tag{2}$$

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 $sim(\cdot)$ defined over two vectors a and b as follows:

$$sim(a, b) = \frac{a^T b}{\|a\|_2 \|b\|_2}. \tag{3}$$

Example 5. *Figure 5 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, which we also used in Example 2. 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.*

Discussion. We use all 5 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.

4 CLASS INFERENCE

In this section, we describe GOGGLES' class inference module: given the affinity matrix $\mathcal{A} \in \mathbb{R}^{N \times \alpha N}$ constructed on $N = n + m$ examples using s affinity functions, where n is the number of unlabeled examples and m is a very small development set (e.g., 10 labeled examples), we would like to assign a class label for every examples $x_i, \forall i \in [1, n]$. In other words, our goal is to predict $P(y_i = k|\mathbf{s}_i)$, where $\mathbf{s}_i$ denote the feature vector for x_i, namely, the i^{th} row in $\mathcal{A}$, and $y_i = k \in \{1, 2 \ldots, K\}$ is a hidden variable representing the class assignment of x_i.

Generative Modelling of the Inference Problem. Recall that our main assumption of affinity coding is that the affinity scores of instance pairs belonging to the same class should be different than affinity scores of instance pairs belonging to different classes. In other words, the feature vector $\mathbf{s}_i$ of one class should look different than that of another class. This suggests a *generative approach* to model how $\mathbf{s}_i$ is generated according to different classes. Generative models obtain $P(y_i = k|\mathbf{s}_i)$ by invoking the Bayes rules:

$$P(y_i = k|\mathbf{s}_i) = \frac{P(y_i = k)P(\mathbf{s}_i|y_i = k)}{P(\mathbf{s}_i)} = \frac{\pi_k \times P(\mathbf{s}_i|y_i = k)}{\sum_{k'=1}^{K} \pi_{k'} \times P(\mathbf{s}_i|y_i = k')} \tag{4}$$

where $\pi_k = P(y_i = k)$ is known as the *prior probability* with $\sum_{k'=1}^{K} \pi_{k'} = 1$, which is the probability that a randomly chosen instance is in class k, and $P(y_i = k|\mathbf{s}_i)$ is known as the *posterior probability*. To use Equation (4) for labeling, we need to learn π_k and $P(\mathbf{s}_i|y_i = k)$ for every class k. $P(\mathbf{s}_i|y_i = k)$ is commonly assumed to follow a known distribution family parameterized by θ_k, and is written as $P(\mathbf{s}_i|\theta_k)$. Therefore, the entire set of parameter we need to have to compute Equation (4) is $\Theta = \{\pi_1, \ldots, \pi_K, \theta_1, \ldots, \theta_K\}$. A common way to estimate Θ is by maximizing the log likelihood function:

$$L(\mathcal{A}, Y|\Theta) = \log \prod_{i=1}^{N} P(\mathbf{s}_i, y_i|\Theta) = \sum_{i=1}^{N} \log \left(P(y_i|\Theta)P(\mathbf{s}_i|\Theta, y_i)\right)$$
$$= \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{1}_{y_i=k} \log \left(\pi_k P(\mathbf{s}_i|\theta_k)\right) \tag{5}$$

where $\mathbb{1}_.$ is the identity function that evaluates to 1 if the condition is true and 0 otherwise. Therefore, the main questions we need to address include (i) what are the generative models to use, namely, the paramterized distributions $P(\mathbf{s}_i|\theta_k)$; and (ii) how do we maximize Equation (5).

Limitations of Existing Models. A commonly used distribution is multi-variate Gaussian distribution, where $\theta_k = \{\mu_k, \Sigma_k\}$, where μ_k is the mean vector and Σ_k is the covariance matrix, and $P(\mathbf{s}_i|\theta_k)$ is the Gaussian PDF:

$$P(\mathbf{s}_i|y_i = k) = P(\mathbf{s}_i|\theta_k) = \frac{\exp\{-\frac{1}{2}(\mathbf{s}_i - \mu_k)^T \Sigma_k^{-1}(\mathbf{s}_i - \mu_k)\}}{(2\pi)^{sn/2}\det(\Sigma_k)^{1/2}} \quad (6)$$

This yields the popular Gaussian Mixture Model (GMM), and there are known algorithm for maximizing the likelihood function under GMM. However, a naive invocation of GMM on our affinity matrix $\mathcal{A}$ is problematic:

- **High dimensionality.** The number of feature in the affinity matrix $\mathcal{A}$ is αN. In the naive GMM, the mean vectors and covariance matrices for all classes (components) have $K(\binom{\alpha N}{2} + \alpha N)$ number of parameters, which is much larger than the number of examples N. It is widely known that the eigenstructure in the estimated covariance matrix Σ_j will be significantly and systematically distorted when the number of features exceeds the number of examples [7, 30].
- **Affinity function selection.** Not all affinity functions are useful, as shown in Figure 5. If the number of noisy functions is small, GMM naturally handles 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 GMM or other generic clustering methods.

4.1 A Hierarchical Generative Model

A fundamental reason for the above two challenges when using GMM is that GMM needs to model correlations between all pairs of columns, which creates a huge number of parameters and makes it difficult for GMM to determine which affinity functions are more informative. In light of this observation, we propose a *hierarchical generative model* which consists of a set of *base models* and an *ensemble model*, as shown in Figure 6. Each base model captures the correlations of a subset of columns in $\mathcal{A}$ that originate from the same affinity function f, and we denote this "subset" matrix as $\mathcal{A}_f \in \mathbb{R}^{N \times N}$. The output of each base model is a *label prediction matrix* $LP_f \in \mathbb{R}^{N \times K}$, where the i^{th} row stores the probabilistic class assignments of x_i using affinity function f. All label prediction matrices are concatenated together to form the *concatenated label prediction matrix* $LP \in \mathbb{R}^{N \times \alpha K}$. The ensemble model takes LP and models the correlations of all affinity functions, and produces the final probabilistic labels for each unlabeled instance.

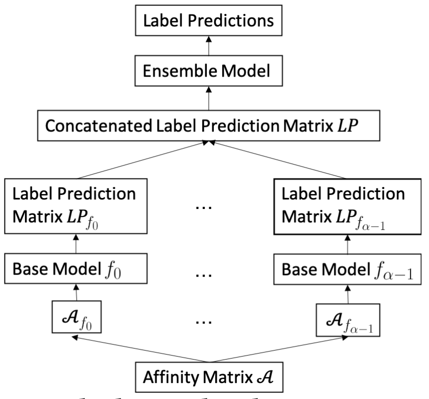

Label Predictions
Ensemble Model
Concatenated Label Prediction Matrix LP
Label Prediction Matrix LP_{f_0}
...
Label Prediction Matrix $LP_{f_{\alpha-1}}$
Base Model f_0
Base Model $f_{\alpha-1}$
$\mathcal{A}_{f_0}$
$\mathcal{A}_{f_{\alpha-1}}$
Affinity Matrix $\mathcal{A}$

Figure 6: The hierarchical generative model.

The Base Models. Given the part of the affinity matrix $\mathcal{A}_f \in \mathbb{R}^{N \times N}$ generated by a particular affinity function f, the base model aims to predict $P(y_i = k|\mathbf{s}_i^f)$, where $\mathbf{s}_i^f$ denotes the subset of the feature vector $\mathbf{s}_i$ corresponding to f.

We design a base generative model for computing $P(y_i = k|\mathbf{s}_i^f)$. As discussed before, a generative model requires specifying the class generative distributions $P(\mathbf{s}_i^f|\theta_k)$, parameterized by θ_k. We use the popular GMM for this purpose with an important modification. Instead of using the full covariance matrix Σ_k that models the correlations between all pairs of columns in $\mathcal{A}_f$, we use the diagonal covariance matrix, which reduces the number of parameters significantly from $\binom{N}{2}$ to N. Note that this simplification is only possible under the base generative model, as each column of $\mathcal{A}_f$ corresponds to an independent example.

The output of the base model for affinity function f is a label prediction matrix $LP_f \in \mathbb{R}^{N \times K}$, where $LP_f[i, k] = P(y_i = k|\mathbf{s}_i^f)$, namely, the probability that affinity function f believes example x_i is in class k.

The Ensemble Model. We concatenate α label prediction matrices $LP_{f_0}, \ldots, LP_{f_{\alpha-1}}$ from α base models to obtain the concatenated label prediction matrix $LP \in \mathbb{R}^{N \times \alpha K}$. Let $\mathbf{s}'_i$ denote the new feature vector of the i^{th} row in LP. The goal of the ensemble model is to predict $P(y_i = k|\mathbf{s}'_i)$.

We again design a generative model for performing the final prediction. As before, we need to decide on a class generative distribution $P(\mathbf{s}'_i|\theta'_k)$ parameterized by θ'_k. The Gaussian distribution used for the base models is not appropriate for the ensemble mode. This is because the values in the concatenated label prediction matrix LP are very close to either 0 or 1. Indeed, in an ideal scenario when all base models work perfectly, all values in LP will be 0 or 1 that correspond to the ground truth. This kind of discrete or close to discrete data is problematic for Gaussian distribution which is designed for continuous data. Fitting a Gaussian distribution on this kind of data typically incurs the singularity problem and provides poor predictions [2]. In light of this observation, we convert *LP* to a one-hot encoded matrix by converting the highest class prediction to 1 and the rest predictions to 0 for each instance and each affinity function, and we propose to use a categorical distribution to model *LP*.

After converting *LP* into a true discrete matrix, Multivariate Bernoulli distribution is a natural fit for modeling $P(\mathbf{s}'_i|\theta'_k)$, which is parameterized by $\theta'_k = \{b_{k,1}, \ldots, b_{k,\alpha K}\}$:

$$P(\mathbf{s}'_i|\theta'_k) = \prod_{l=1}^{\alpha K} b_{k,l}^{\mathbf{s}'_i[l]} (1 - b_{k,l})^{1-\mathbf{s}'_i[l]} \tag{7}$$

where $\mathbf{s}'_i[l]$ is the l^{th} dimension of the binary vector $\mathbf{s}'_i$, and we have a total of αK dimensions. The output of the ensemble model is the final label predictions $L \in \mathbb{R}^{N \times K}$, where $L[i,k] = P(y_i = k|\mathbf{s}'_i)$, namely, the probability that the ensemble model f believes example x_i is in class k.

Hierarchical Model Address the Two Challenges. First, the total number of parameters in the hierarchical model is $2\alpha KN + \alpha K$, much smaller than the number of parameters in the naive GMM $K(\binom{\alpha N}{2} + \alpha N)$, effectively addressing the high-dimensionality problem. Second, by consolidating the affinity scores produced by each affinity function to produce a binary *LP*, the ensemble model can only need to model the accuracy of the α affinity functions better instead of the original αN features, and thus can better distinguish the good affinity functions from the bad ones.

4.2 Parameter Learning

We need to learn the parameters of the base models and the ensemble model under their respective generative distributions. Expectation-maximization algorithm is the canonical algorithm for maximizing the log likelihood function in the presence of hidden variables [8]. We first show the EM algorithm for maximizing the general data log likelihood function in Equation (5), and then discuss how it needs to modified to learn the base models and the ensemble model.

EM for Maximizing Equation (5) Each iteration of the EM algorithm consists of two steps: an Expectation (E)-step and a Maximization (M)-step. Intuitively, the E-step determines what is the (soft) class assignment y_i for every instance x_i based on the parameter estimates from last iteration Θ^{t-1}. In other words, E-step computes the posterior probability. The M-step takes the new class assignments and re-estimates all parameters Θ^t by maximizing Equation (5). More precisely, the M-step maximizes the expected value of Equation (5), since the E-step produces soft assignments.

(1) **E Step.** Given the parameter estimates from the previous iteration Θ^{t-1}, compute the posterior probabilities:

$$\gamma_{i,k} = P(y_i = k|\mathbf{s}_i) = \frac{\pi_k \times P(\mathbf{s}_i|\theta_k)}{\sum_{k'=1}^{K} \pi_{k'} \times P(\mathbf{s}_i|\theta_{k'})} \tag{8}$$

(2) **M Step.** Given the new class assignments as defined by $\gamma_{i,k}$, re-estimate Θ_t by maximizing the following expected log likelihood function:

$$\mathbb{E}\{L(\mathcal{A}, Y|\Theta)\} = \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma_{i,k} \log \left(\pi_k p(\mathbf{s}_i|\theta_k)\right) \tag{9}$$

EM for Maximizing the Base Model. For each base model associated with affinity function f, $P(\mathbf{s}_i|\theta_k)$ in the EM algorithm is replaced with $P(\mathbf{s}^f_i|\theta^f_k)$, which is a multivariate Gaussian distribution as shown in Equation (6), but with a diagonal covariance matrix. The entire set of parameters is $\Theta = \{\pi^f_k, \mu^f_k, \Sigma^f_k\}$, where $k = 0, \ldots, K-1$, which update in each M-step as follows:

$$\begin{aligned} N_k &= \sum_{i=1}^{N} \gamma_{i,k}; \pi^f_k = N_k/N; \mu^f_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{i,k} \mathbf{s}^f_i \\ \Sigma^f_k &= \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{k,i} (\mathbf{s}^f_i - \mu^f_k)(\mathbf{s}^f_i - \mu^f_k)^T \end{aligned} \tag{10}$$

EM for Maximizing the Ensemble Model. For the ensemble model, $P(\mathbf{s}_i|\theta_k)$ in the EM algorithm is replaced with $P(\mathbf{s}'_i|\theta'_k)$, which is a multivariate Bernoulli distribution, as shown in Equation (7). The entire set of parameters is $\{\pi_k, b_{k,1}, \ldots, b_{k,\alpha K}\}$, where $k = 0, \ldots, K-1$, which we update in each M-step as follows:

$$\begin{aligned} N_k &= \sum_{i=1}^{N} \gamma_{i,k}; \pi_k = N_k/N \\ b_{k,l} &= \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{i,k} \mathbf{s}'_i[l], \text{ where } l \in \{1, 2, \ldots, \alpha K\} \end{aligned} \tag{11}$$

4.3 Exploiting Development Set

Consider a scenario without any labeled development set, in this case, the hierarchical model essentially clusters all unlabeled examples into K clusters without knowing which cluster corresponds to which class. Following the data programming system [29], we assume we have access to a small development set that is typically too small to train any machine learning models, but is powerful enough to determine the correct "cluster-to-class" assignment. Note that the theory developed here can also be used to provide theoretical guarantees on the mapping feasibility in the "cluster-then-label" category of semi-supervised learning approaches [17, 38].

Let $LS_{k'}$ denote the set of labeled examples for class k'. To make our analysis easier, we assume the size of $LS_{k'}$ is the same for all classes. Intuitively, we want to map cluster

k to class k' if most examples from $LS_{k'}$ are in cluster k. However, this simple cluster-to-class mapping strategy may create conflicting assignments, namely, the same cluster is mapping to multiple classes. We propose a more principled way to obtain the one-to-one mapping $g : k \mapsto k'$. We first define the "goodness" of the mapping L_g as:

$$L_g = \sum_{k=1}^{K} \sum_{l \in LS_{g(k)}} \gamma_{l,k} \tag{12}$$

To see why L_g can represent the "goodness" of a mapping. We represent development sets with a one-hot encoded ground truth matrix T where each element $t_{i,k'}$ is obtained by:

$$t_{i,k'} = \begin{cases} 1, & \text{if } i \in LS_{k'} \\ 0, & \text{otherwise} \end{cases} \quad i = 1, \ldots, N;\ k' = 1, \ldots, K \tag{13}$$

L_g is essentially the summation of the element-wise multiplication between the ground truth matrix T and label prediction matrix LP under a column mapping defined by g on the development set. Therefore, L_g is expected to be maximized when a mapping g makes the two matrices the most similar under cosine distance. Given L_g, the final mapping g is obtained by:

$$g = \arg\max_g \{L_g\}, \text{ and } g \text{ is a one-to-one mapping} \tag{14}$$

In other words, the final mapping is a one-to-one mappings that maximizes L_g. When $K = 2$, Equation (14) becomes

$$g(k) = \begin{cases} k, & \text{if } \sum_{l \in LS_1} \gamma_{l,1} \geq \sum_{l \in LS_0} \gamma_{l,1} \\ 1-k, & \text{otherwise} \end{cases} \tag{15}$$

Algorithm for Solving Equation (14). Instead of enumerating all possible mappings with a complexity of $O(K!)$ (which is actually feasible for a small K), we convert it to the *assignment problem* which can be solved in $O(K^3)$. Let $w_{k,k'}$ denote $w_{k,k'} = \sum l \in LS_{k'} \gamma_{l,k}$, then Equation (12) becomes:

$$L_g = \sum_{k=1}^{K} w_{k,g(k)} \tag{16}$$

Finding a g that maximizing Equation (16) is essentially the *Assignment problem*, and there are known algorithms [12] that solve it with a worst case time complexity of $O(K^3)$.

This "cluster-to-class" mapping is performed after obtaining base model predictions and the ensemble model predictions. After the mapping is obtained, we rearrange the the columns in the label prediction matrix LP_f produced by each base model, and the final label matrix L produced by the ensemble according to the mapping g, so that the true classes are aligned with the clusters.

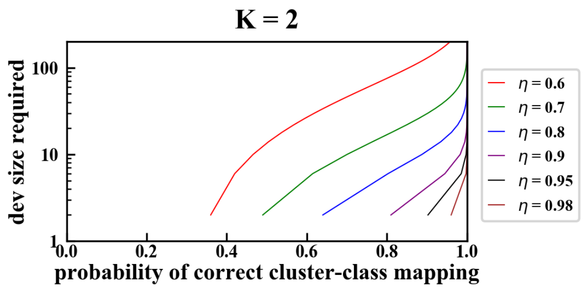


Figure 7: Size of the Development Set Needed.

4.4 The Size of Development Set Needed

In this section, we give an analysis about the size of the development set needed for GOGGLES to produce the correct "cluster-to-class" mapping, where the correct mapping is defined to be the mapping that achieves the highest labeling accuracy, which we denote as η. Intuitively, the higher η is, the less size we need. Consider an extreme scenario with $K = 2$ classes and our hierarchical generative model produces two clusters that perfectly separate the two classes. In this case, we only need one labeled example to determine which cluster corresponds to which class with 100% confidence. Figure 7 shows the size of development set required when $K = 2$ based on our theory to be discussed in the following, we can see when $\eta = 0.8$, only about 20 examples are required to produces the correct cluster-class mapping with a probability close to 1. However, as we will shown in the experiment section, the number of required development set size is actually much smaller in practice. This is because the theoretical lower-bound we will provide is a rather loose one, for ease of derivation.

A Theory on the Size of the Development Set. let us first assume the mapping of each class is independent, so the probability of a completely correct mapping is $P_{\text{ind}} = \prod_{k'=1}^{K} P_{k'}$, where $P_{k'}$ denote the probability that class k' is correctly mapped to its corresponding cluster.

To simplify derivation, we further assume "hard" assignment of classes labels: an example is only assigned to one cluster, in other words, γ only contains 0 and 1. This is a natural assumption because values in γ will be converted to be binary anyway when evaluating the accuracy of the algorithm. In the development set, we have a labeled set $LS_{k'}$ with a size of $d = m/K$ for every class k'. Let $d_{k',j},\ j = 1, \ldots K$, denote the number of examples in the development set $LS_{k'}$ that are in the jth cluster, so $\sum_{j=1}^{K} d_{k',j} = d$. Under the independence assumption, Equation (14) becomes

$$g^{-1}(k') = \arg\max_{1 \leq j \leq K} \sum_{i \in LS_{k'}} \gamma_{i,j} \tag{17}$$

where g^{-1} denote the inverse mapping of g, that is $k' \mapsto k$. Equation (17) means that each each class is mapped to the cluster in which its majority lies, so class k' is mapped to its correct cluster only when the majority of $LS_{k'}$ are in that cluster. Assume the kth cluster is the correct cluster for class k', so the probability of the k' class mapped correctly is:

$$\begin{aligned} P_{k'} > Pl_{k'} &= \sum_{d_{k',k}=0}^{d} P(d_{k',k} > \max_{1\le j\le K, j\ne k} d_{k',j}) \\ &= \sum_{d1,\dots,d_K} P(d_{k',1},\dots,d_{k',K}) \text{ s.t. } d_{k',k} > \max_{1\le j\le K, j\ne k} d_{k',j} \end{aligned} \tag{18}$$

The first > sign is because on the right side we don't consider the situations with ties in majority vote (the second > sign), where we break the ties randomly and a correct mapping is also possible. The P_{ind} is then lower bounded by:

$$P_{\text{ind}} > \prod_{k'=1}^{K} Pl_{k'} \tag{19}$$

Suppose the accuracy of our algorithm η is known, so the probability of an example being predicted to be its true label equals to η. An example in the development set $LS_{k'}$ is predicted to be its true label by the algorithm only when it is in the correct cluster, so the probability of it being in the correct cluster equals to η. In case of incorrect assignment, we assume the probability of assigning to every possible incorrect classes is equal, being $\rho = \frac{\eta}{K-1}$. $d_{k',1},\dots,d_{k',K}$ follow a multinomial distribution:

$$\begin{aligned} P(d_{k'1},\dots,d_{k',K}) &= \frac{d!}{\prod_{j=1}^{K} d_{k',j}!} \eta^{d_{k',1}} \rho^{d_{k',2}} \rho^{d_{k',3}} \dots \rho^{d_{k',K}} \\ &= \frac{d!}{\prod_{j=1}^{K} d_{k',j}!} \eta^{d_{k',k}} \rho^{d-d_{k',k}} \end{aligned} \tag{20}$$

The correct mapping under independent assumption requires the mapping of every class to be correct on their own. This is a rather strict assumption. Without assuming independence, Equation (14) is able to produce a completely correct mapping when some classes would otherwise fail to be mapped correctly on their own. In other words, the probability of a completely correct mapping is:

$$P_{\text{correct}} \ge P_{\text{ind}} \tag{21}$$

Combining Equation (21) and Equation (19), we get the following theorem.

Theorem 1. *The probability that Equation* (14) *gives the optimal mapping is lower bounded by $P_{correct} > \prod_{k'=1}^{K} Pl_{k'}$, where $Pl_{k'}$ is obtained by Equation* (18).

Therefore, the size of development set m^* that produces an optimal mapping with a probability of as least p is given by $m^* = Kd^*$, where d^* is the smallest value of d that makes $\prod_{k'=1}^{K} Pl_{k'} \ge p$.

The time complexity of solving the right hand side in Equation (18) by a brute-force iteration over all combinations of d_j is $O(d!)$, but it can be solved in $O(Kd^2)$ using a dynamic programming based approach.

For ease of of notation, we assume the 1st cluster is the correct cluster for class k'. Let $S(j, D_j)$ denote the following:

$$\begin{aligned} S(j, D_j) &= \sum_{d_{k',j},\dots,d_{k',K}} P(d_{k',1},\dots,d_{k',K}) \\ \text{s.t.} & \max_{j\le l\le K, l\ne k} d_{k',l} < d_{k',1} \text{ and } \sum_{l=j}^{K} d_{k',l} = D_j \end{aligned} \tag{22}$$

so $Pl_{k'} = S(1, d)$, and for each j:

$$S(j, D_j) = \sum_{d_{k',j}=0}^{d} S(j+1, D_j - d_j) \tag{23}$$

The time complexity of obtaining Pl_k by dynamic programming using Equation (23) is $O(Kd^2)$.

5 EXPERIMENTS

We conduct extensive experiments to evaluate the accuracy of labels generated by GOGGLES. Specifically, we focus on the following three dimensions:

- *Feasibility and Performance of GOGGLES (Section 5.2).* Is it possible to automatically label image datasets using a domain-agnostic approach? How does GOGGLES compare with existing data programming systems?
- *Ablation Study (Section 5.3).* How do the two primary innovations in GOGGLES (namely, affinity matrix construction and class inference) compare against other techniques?
- *Sensitivity Analysis (Section 5.4).* Is GOGGLES sensitive to the set of affinity functions? What is the size of the development set needed for GOGGLES to correctly determine the correct "cluster-to-class" mapping?

5.1 Setup

5.1.1 Datasets. We consider real-world image datasets with varying domains to evaluate the versatility and robustness of GOGGLES. Since our approach internally uses a pre-trained VGG-16 model for defining affinity functions, we select datasets which have minimal or no overlap with classes of images from the ImageNet dataset [23], on which the VGG-16 model was originally trained. Robust performance across these datasets show that GOGGLES is domain-agnostic with respect to the underlying pre-trained model. We perform our experiments on the following datasets, which are roughly ordered by domain overlap with ImageNet:

- **CUB**: The Caltech-UCSD Birds-200-2011 dataset [31] 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 etc. We use this metadata information for designing binary labeling functions which 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.

- **GTSRB**: The German Traffic Sign Recognition Benchmark dataset [27] contains 51,839 images for 43 classes of traffic signs. Again, for testing the performance of binary label generation, we sample 10 random class-pairs from the dataset and use them for all the experiments. Although this dataset contains images labeled by specific traffic signs, ImageNet contains a generic "street sign" class, and hence this dataset may also have some degree of domain overlap.
- **Surface**: The surface finish dataset [14] contains 1280 images of industrial metallic parts which 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 [11] 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 [13] 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. GOGGLES uses a small development set to determine the optimal class mapping for a given label assignment, the same assumption in Snuba [29]. By default, we use only 5 label annotations arbitrarily chosen from each class for this. Hence, for the task of generating binary labels, we use a development set having a size of 10 images for all the experiments. We report the performance of GOGGLES on the remaining images from each dataset.

5.1.2 Data Programming Systems. We compare GOGGLES with existing systems: Snorkel [19] and Snuba [29].

Snorkel is the first system that implements the data programming paradigm [20]. 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.) in order to design labeling functions. Hence, each attribute annotation in the union of the class-specific attributes acts as a labeling function which outputs a binary label corresponding to the class that the attribute belongs to. If an attribute belongs to both classes from the class-pair, 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 (c.f. Example 1), and the system automatically generates a set of labeling functions using a labeled small development set.

Since all 6 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 [9, 15]. For the VGG-16 model trained on ImageNet, this yields us feature vectors having 1000 dimensions for each image. To obtain densely rich primitives which 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 [34]. 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.

5.1.3 Few-shot Learning (FSL). Our affinity coding setup which uses 5 development set labels from each class is comparable to the 2-way 5-shot setup for few-shot learning from the computer vision domain. Hence, we compare GOGGLES's end-to-end performance with a recent FSL approach [3] that achieves state-of-the-art performance on domain adaptation.

We use the same development set used by GOGGLES as the few-shot labeled examples for training the FSL model.

The original FSL Baseline implementation uses a model trained on mini-ImageNet for domain adaptation to the CUB dataset, and achieves better performance than other state-of-the-art FSL methods. For a more comparable analysis, we use a VGG-16 model trained on ImageNet, which is the same pre-trained model GOGGLES uses for affinity coding. Note that our adaptation of the FSL Baseline method achieved a much better performance for domain adaptation on CUB than the original results reported in [3]. The FSL models as well as all end models are trained with the Adam optimizer with a learning rate of 10^{-3}, same as in [3].

5.1.4 Empirical upper-bound (supervised approach). We also would like to compare GOGGLES' performance with an empirical upper-bound, which is obtained via a typical supervised transfer learning approach for image classification. Specifically, 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.

5.1.5 Ablation Study: Other image representation techniques for computing affinity. GOGGLES computes affinity scores by extracting prototype representations from intermediate layers of a pre-trained model. We compare the efficacy of this representation technique with two other typical methods of image representation used in the computer vision domain. We compare the predictive capacity of each representation technique by constructing an affinity matrix from each candidate feature representation using pair-wise cosine similarity, and then using our class inference approach for labeling.

HOG. We compare with the histogram of oriented gradients HOG descriptor, which is a very popular feature representation technique used for recognition tasks in classical computer vision literature [33, 36]. The HOG descriptor [4] represents an image by counting the number of occurrences of gradient orientation in localized portions of the image.

Logits. We also compare with a modern deep learning-based approach, leveraged by recent works in computer vision [1, 25], that uses an intermediate output from a convolutional neural network as an image's feature representation. We use the logits layer from the trained VGG-16 model in our comparison, which is the output of the last fully connected layer, before it is fed to the softmax operation.

5.1.6 Ablation Study: Baseline methods for class inference. The class inference method in GOGGLES consists of a clustering step followed by class mapping. We compare our proposed hierarchical model for clustering with other baseline methods, including **K-means** clustering, Gaussian mixture modeling with expectation maximization (**GMM**) and spectral co-clustering (**Spectral**). Since these clustering methods are incognizant of the structural semantics of our affinity-based features which are derived from multiple affinity functions, we simply concatenate all affinity functions to create the feature set for each dataset, and then feed these features to the baseline methods. As we would like to see the absolute best performance of the baseline clustering approaches, we use the optimal "cluster-class" mapping for all baselines.

5.1.7 Evaluation Metrics. We use the train/test split as originally defined in each dataset. We report the *labeling accuracy* on the training set for comparing different data labeling systems, Snorkel, Snuba, and GOGGLES. We follow the same approach used in [19, 29] to train an end discriminative model by using the probabilistic labels generated from each data labeling system as training data and report the *end-to-end accuracy* as the end model's performance on the held-out test set. For labeling tasks, all experiments, including baselines, are conducted 10 times, and we report the average.

5.2 Feasibility and Performance

Table 1 shows the end-to-end system 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 71% to a maximum of 98%. GOGGLES shows an average of 21% 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 (c.f. Example 1). 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 (10x increase) for the PN-Xray dataset, the performance jumps from 55.50% to 67.84%. In comparison, GOGGLES still 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.

5.3 Ablation Study

We conduct a series of experiments to understand the goodness of different components in GOGGLES, including the proposed affinity functions and the proposed class inference method. Results are shown in Table 1.

Dataset	GOGGLES (our results)	Data Programming Snorkel	Data Programming Snuba	Representation HoG	Representation Logits	Class Inference Baselines K-Means	Class Inference Baselines GMM	Class Inference Baselines Spectral
CUB	97.83	89.17	58.83	62.93	96.35	98.67	97.62	72.08
GTSRB	70.51	-	62.74	75.48	64.77	70.74	69.64	62.40
Surface	89.18	-	57.86	85.82	54.08	69.08	69.14	60.82
TB-Xray	76.89	-	59.47	69.13	67.16	76.33	76.70	75.00
PN-Xray	74.39	-	55.50	53.11	71.18	50.66	68.66	75.90
Average	**81.76**	-	58.88	69.30	70.71	73.09	76.35	69.24

Table 1: Evaluation of GOGGLES labeling accuracy on training set. The '-' symbol represents cases where evaluation was not possible. GOGGLES shows on average an improvement of 23% over the state-of-the-art data programming system Snuba.

Goodness of Proposed Affinity Functions. We compare GOGGLES affinity functions with the two common methods of obtaining the distance between two images: HOG and Logits. We use the two baseline methods to generate affinity matrices and run GOGGLES' inference module on them. GOGGLES' affinity functions outperform the other two on almost all datasets. This is because GOGGLES's affinity functions covers features at different scales and locations.

Goodness of Proposed Class Inference. We compare GOGGLES' inference module with three representative clustering methods: K-means, GMM, and Spectral co-clustering. All methods use the GOGGLES affinity matrix as input data. Note that the three clustering methods are not able to map the clusters to the classes automatically. As we would like to see the absolute best performance of the baseline clustering approaches, we use the optimal "cluster-class" mapping for all baselines. GOGGLES's inference module has the best average performance. The primary reason for our improvement over generic clustering methods is that our generative model adapts to the design of our affinity matrix. Specifically, our generative model is better at (1) handling the high-dimensionality through using the hierarchical structure and reducing the parameters in the base model by using diagonal covariance matrices; and (2) selecting affinity functions through the ensemble model (c.f. Section 4.1).

In terms of running time, without parallelization, our generative model is α (the number of base models) slower than the GMM model (the best baseline method). However, in practice (and in our experiments), we can parallelize all of the base models using different slices of the affinity matrix.

5.4 Sensitivity Analysis

Varying Size of the Development Set. We vary the size of the development set from 0 to 40 to understand how it affects performance (Figure 8). As the development set size increases, the accuracy increases initially, but finally converges. This is expected as when the development set is small, the mapping obtained by Equation (14) has a low probability being the optimal as predicted in Figure 7. When the development set size is large enough, the mapping given by Equation (14) converges to the optimal mapping, so the accuracy converges. Another observation is that datasets with higher accuracy converge at a smaller development set size. For example, the CUB dataset has an accuracy of 97.63% and its accuracy converges at a development set size of 2, while the GTSRB dataset requires a development set size of 8 to converge as it achieves an lower accuracy of 70.75%. Finally, the empirical size of the development set required to converge is much smaller than the theory predicted in Figure 7. A development set with 5 examples per class enough for all datasets.

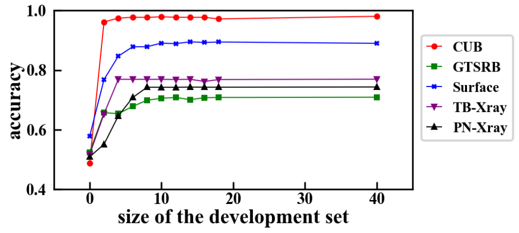


Figure 8: Accuracy w.r.t. development set size.

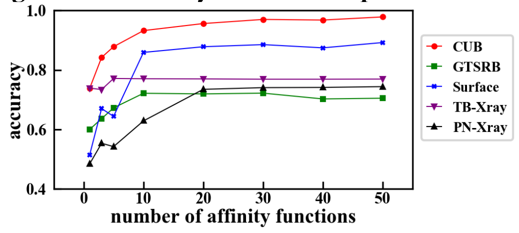


Figure 9: Accuracy w.r.t. varying # affinity functions.

Varying Number of Affinity Functions. We vary the number of affinity functions to study its affects on the results (Figure 9). 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.

5.5 End-to-End Performance Comparison.

We also use the probabilistic labels generated by Snorkel, Snuba and GOGGLES to train downstream discriminative models following the similar approach taken in [19, 29]. 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,

Dataset	FSL	Snorkel	Snuba	GOGGLES	Upper Bound
CUB	84.74	87.85	56.32	95.30	98.44
GTSRB	90.72	-	70.11	91.54	98.94
Surface	76.00	-	51.67	83.33	92.00
TB-Xray	66.42	-	62.71	70.90	82.09
PN-Xray	68.28	-	62.19	69.06	74.22
Average	77.23	-	60.60	**82.03**	89.14

Table 2: Comparison of end model accuracy on held-out test set. GOGGLES uses only 5 labeled instances per class but is only 7% away from the supervised upper bound (in gray) which uses the ground-truth labels of the training set.

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 2.

First, GOGGLES outperforms Snuba by an average of 21%, a similar number to the labeling performance improvement of 23% GOGGLES has over Snuba (c.f. Table 1), and the end model performance of Snuba is worse than FSL. This is because the labels generated by Snuba (59%) 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 (c.f. Section 5.1.3) by an average of 5%, 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.

6 RELATED WORK

ML Model Training with Insufficient Data. Semi supervised learning techniques [38] combine labeled examples and unlabeled examples for model training; and active learning techniques aim at involving human labelers in a judicious way to minimize labeling cost [24]. Though semi-supervised learning and active learning can reduce the number of labeled examples required to obtain a competent model, they still need many labeled examples to start. Transfer learning [16] and few-shot learning techniques [3, 10, 32, 35] often use models trained on source tasks with many labeled examples to help with training models on new target tasks with limited labeled examples. Not surprisingly, these techniques often require users to select a source dataset or pre-trained model that is in a similar domain as the target task to achieve the best performance. In contrast, our proposal can incorporate several sources of information as affinity functions.

Data Programming. Data programming [20], and Snuba [29] and Snorkel [19] 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). As we show in this paper, 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, affinity coding and GOGGLES offer a domain-agnostic and automated approach for image labeling.

Other Related Work in Database Community. Many problems in database community share similar challenges to our work. In particular,data fusion/truth discovery [18, 22], crowdsourcing [5], and data cleaning [21], in one form or another, all need to reconcile information from multiple sources to reach one answer. While the information sources are assumed as input in these problems, labeling training data faces the challenge of lacking enough information sources. In fact, one primary contribution of GOGGLES is the affinity coding paradigm, where each unlabeled instance becomes an information source.

7 CONCLUSION

We proposed affinity coding, a new paradigm that offers a domain-agnostic way of automated training data labeling. Affinity coding is based on the proposition that affinity scores of instance pairs belonging to the same class on average should be higher than those of instance pairs belonging to different classes, according to some affinity functions. We build the GOGGLES system that implements the affinity coding paradigm for labeling image datasets. GOGGLES includes a novel set of affinity functions defined using the VGG-16 network, and a hierarchical generative model for class inference. GOGGLES is able to label images with high accuracy without any domain-specific input from users, except a very small development set, which is economical to obtain.

8 ACKNOWLEDGEMENTS

We thank the SIGMOD'20 anonymous reviewers for their thoughtful and highly constructive feedback. This work was supported by NSF grants CNS-1704701 and IIS-1563816, and an Intel gift for ISTC-ARSA.

REFERENCES

[1] T Akilan, QM Jonathan Wu, Amin Safaei, and Wei Jiang. 2017. A late fusion approach for harnessing multi-CNN model high-level features. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 566–571.

[2] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.

[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. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

[4] Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection.

[5] Akash Das Sarma, Aditya Parameswaran, and Jennifer Widom. 2016. Towards globally optimal crowdsourcing quality management: The uniform worker setting. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 47–62.

[6] Allan Peter Davis, Thomas C Wiegers, Phoebe M Roberts, Benjamin L King, Jean M Lay, Kelley Lennon-Hopkins, Daniela Sciaky, Robin Johnson, Heather Keating, Nigel Greene, et al. 2013. A CTD–Pfizer collaboration: manual curation of 88 000 scientific articles text mined for drug–disease and drug–phenotype interactions. *Database* 2013 (2013).

[7] A. P. Dempster. 1972. Covariance Selection. *Biometrics* 28, 1 (Mar 1972), 157–175. https://doi.org/10.2307/2528966

[8] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1 (1977), 1–22.

[9] 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 *International conference on machine learning*. 647–655.

[10] Li Fei-Fei, Rob Fergus, and Pietro Perona. 2006. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence* 28, 4 (2006), 594–611.

[11] 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 transactions on medical imaging* 33, 2 (2013), 233–245.

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

[13] Daniel S Kermany, Michael Goldbaum, Wenjia Cai, Carolina CS 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.

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

[15] 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.

[16] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[17] Mohammad Peikari, Sherine Salama, Sharon Nofech-Mozes, and Anne L Martel. 2018. A cluster-then-label semi-supervised learning approach for pathology image classification. *Scientific reports* 8, 1 (2018), 7193.

[18] Ravali Pochampally, Anish Das Sarma, Xin Luna Dong, Alexandra Meliou, and Divesh Srivastava. 2014. Fusing data with correlations. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 433–444.

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

[20] 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.

[21] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1190–1201.

[22] Theodoros Rekatsinas, Manas Joglekar, Hector Garcia-Molina, Aditya Parameswaran, and Christopher Ré. 2017. Slimfast: Guaranteed results for data fusion and source reliability. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1399–1414.

[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. *International journal of computer vision* 115, 3 (2015), 211–252.

[24] Burr Settles. 2012. Active learning: Synthesis lectures on artificial intelligence and machine learning. *Long Island, NY: Morgan & Clay Pool* (2012).

[25] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. 2014. CNN features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 806–813.

[26] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[27] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* 0 (2012), –. https://doi.org/10.1016/j.neunet.2012.02.016

[28] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*. 843–852.

[29] Paroma Varma and Christopher Ré. 2018. Snuba: automating weak supervision to label training data. *Proceedings of the VLDB Endowment* 12, 3 (2018), 223–236.

[30] Santiago Velasco-Forero, Marcus Chen, Alvina Goh, and Sze Kim Pang. 2015. Comparative Analysis of Covariance Matrix Estimation for Anomaly Detection in Hyperspectral Images. *IEEE J. Sel. Top. Signal Process.* 9, 6 (Sep 2015), 1061–1073. https://doi.org/10.1109/JSTSP.2015.2442213

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

[32] Yaqing Wang and Quanming Yao. 2019. Few-shot Learning: A Survey. *CoRR* abs/1904.05046 (2019). arXiv:1904.05046 http://arxiv.org/abs/1904.05046

[33] Daniel Weinland, Mustafa Özuysal, and Pascal Fua. 2010. Making action recognition robust to occlusions and viewpoint changes. In *European Conference on Computer Vision*. Springer, 635–648.

[34] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.

[35] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, and Zeynep Akata. 2017. Zero-Shot Learning - A Comprehensive Evaluation of the Good,

the Bad and the Ugly. *CoRR* abs/1707.00600 (2017). arXiv:1707.00600 http://arxiv.org/abs/1707.00600

[36] Xiaodong Yang, Chenyang Zhang, and YingLi Tian. 2012. Recognizing actions using depth motion maps-based histograms of oriented gradients. In *Proceedings of the 20th ACM international conference on Multimedia*. ACM, 1057–1060.

[37] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.

[38] Xiaojin Jerry Zhu. 2005. *Semi-supervised learning literature survey*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences.