Deep Anomaly Detection with Deviation Networks

Guansong Pang*
Australian Institute for Machine Learning
The University of Adelaide
Adelaide, Australia
guansong.pang@adelaide.edu.au

Chunhua Shen
Australian Institute for Machine Learning
The University of Adelaide
Adelaide, Australia
chunhua.shen@adelaide.edu.au

Anton van den Hengel
Australian Institute for Machine Learning
The University of Adelaide
Adelaide, Australia
anton.vandenhengel@adelaide.edu.au

ABSTRACT

Although deep learning has been applied to successfully address many data mining problems, relatively limited work has been done on deep learning for anomaly detection. Existing deep anomaly detection methods, which focus on learning new feature representations to enable downstream anomaly detection methods, perform indirect optimization of anomaly scores, leading to data-inefficient learning and suboptimal anomaly scoring. Also, they are typically designed as unsupervised learning due to the lack of large-scale labeled anomaly data. As a result, they are difficult to leverage prior knowledge (e.g., a few labeled anomalies) when such information is available as in many real-world anomaly detection applications.

This paper introduces a novel anomaly detection framework and its instantiation to address these problems. Instead of representation learning, our method fulfills an end-to-end learning of anomaly scores by a neural deviation learning, in which we leverage a few (e.g., multiple to dozens) labeled anomalies and a prior probability to enforce statistically significant deviations of the anomaly scores of anomalies from that of normal data objects in the upper tail. Extensive results show that our method can be trained substantially more data-efficiently and achieves significantly better anomaly scoring than state-of-the-art competing methods.

CCS CONCEPTS

• **Computing methodologies** → **Anomaly detection**; **Neural networks**; *Semi-supervised learning settings.*

KEYWORDS

Anomaly Detection, Deep Learning, Representation Learning, Neural Networks, Outlier Detection

ACM Reference Format:
Guansong Pang, Chunhua Shen, and Anton van den Hengel. 2019. Deep Anomaly Detection with Deviation Networks. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3292500.3330871

*Guansong Pang is the corresponding author.

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330871

1 INTRODUCTION

Anomalies are referred to as data objects that deviate significantly from the majority of data objects. Anomaly detection is the task of identifying these anomalies, which has important applications in broad domains, e.g., to detect network attacks in cybersecurity, to detect fraudulent transactions in finance, and to detect diseases in healthcare. Numerous anomaly detection methods have been introduced, but they often fail to detect anomalies in data with high dimensionality and/or intricate relations due to the curse of dimensionality and highly non-linear feature relations [19, 20].

In recent years, deep learning [11] has gained exceptional successes in discovering such intricate relations in high-dimensional data. However, deep learning for anomaly detection has been insufficiently explored due to the following two major challenges: (i) it is very difficult to obtain large-scale labeled data to train anomaly detectors due to the prohibitive cost of collecting such data in many anomaly detection application domains; and (ii) anomalies often demonstrate different anomalous behaviors, and as a result, they are dissimilar to each other, which poses significant challenges to widely-used optimization objectives that generally assume the data objects within each class are similar to each other.

Existing deep anomaly detection[1] methods [2, 7, 19, 20, 22, 29, 30] address these two challenges by using unsupervised deep learning to model the normal class in a two-step approach (i.e., the pipeline (a) in Figure 1): they first learn to represent data with new representations, e.g., intermediate representations in autoencoders [2, 7, 30], latent spaces in generative adversarial networks (GANs) [22, 29], or distance metric spaces in [19, 20]; and then they use the learned representations to define anomaly scores using reconstruction errors [2, 7, 22, 29, 30] or distance-based measures [19, 20]. However, in most of these methods [2, 7, 22, 29, 30], the representation learning is separate from anomaly detection methods, so it may yield representations that are suboptimal or even irrelevant w.r.t. specific anomaly detection methods. The very recent efforts [19, 20] address this problem by incorporating traditional anomaly scoring measures into the feature learning objective, which substantially improves the expressiveness of the feature representations. However, they still focus on optimizing the representations, which is an indirect optimization of anomaly scoring. This can lead to inefficient use of training data and low-quality anomaly scoring.

Also, they are mainly focused on unsupervised learning, which may lead to a common problem of unsupervised anomaly detection that many of the anomalies they identify are data noises or uninteresting data objects due to the lack of prior knowledge of the

[1]Deep anomaly detection refers to any methods that exploit deep learning techniques to learn feature representations or anomaly scores for anomaly detection.

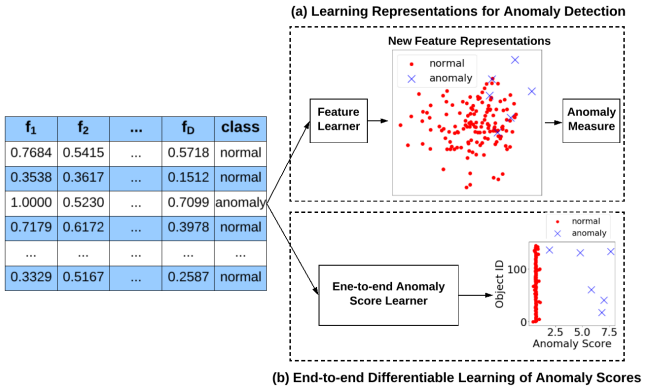

Figure 1: (a) Learning Features for Subsequent Anomaly Measures vs. (b) Direct Learning of Anomaly Scores

anomalies of interest [1, 19, 24]. A potential solution to this problem is to leverage a limited number of labeled anomalies as the prior knowledge to learn anomaly-informed models, since such prior knowledge is often available in many real-world anomaly detection applications. Those labeled anomalies may originally come from a deployed detection system, e.g., a few successfully detected network intrusion records, or they may be from users, such as a small number of fraudulent credit card transactions that are reported by clients and confirmed by the banks.

In this paper, we introduce a novel anomaly detection framework to fill these gaps by leveraging a few labeled anomalies to fulfill an end-to-end differentiable learning of anomaly scores. That is, as shown in the pipeline (b) in Figure 1, with the original data as inputs, we directly learn and output the anomaly scores rather than the feature representations. Specifically, as shown in Figure 2, given a training data object, the proposed framework first uses a neural anomaly score learner to assign it an anomaly score, and then defines the mean of the anomaly scores of some normal data objects based on a prior probability to serve as a reference score for guiding the subsequent anomaly score learning. Lastly, the framework defines a loss function, called deviation loss, to enforce statistically significant deviations of the anomaly scores of anomalies from that of normal data objects in the upper tail.

We further instantiate the framework into a method called deviation networks (DevNet). DevNet leverages multiple to dozens of labeled anomalies, accounting for only 0.005%-1% of all training data objects and 0.08%-6% of all anomalies per data set, and a Gaussian prior to perform a direct optimization of anomaly scores using a Z-Score-based deviation loss. By doing so, DevNet can not only achieve very data-efficient learning of the anomaly scores but also accommodate anomalies with different anomalous behaviors. Additionally, in contrast to most methods that produce hardly interpretable anomaly scores [10], the Z-Score-based deviation loss also allows DevNet to produce easily interpretable anomaly scores.

Accordingly, this paper makes the following major contributions.

- We introduce a novel framework to learn anomaly scores in an end-to-end fashion. In contrast to the current indirect optimization approach, our framework fulfills a direct optimization of anomaly scores. As far as we know, this is the first framework for leveraging limited labeled anomaly data to achieve end-to-end anomaly score learning.
- A novel anomaly detection method, namely deviation networks (DevNet[2]), is instantiated from the framework. DevNet synthesizes neural networks, Gaussian prior and Z-Score-based deviation loss to perform data-efficient and effective learning of the anomaly scores, resulting in well optimized and easily interpretable anomaly scores.

Extensive empirical results on nine large and/or high-dimensional real-world data sets show that (i) DevNet significantly outperforms four state-of-the-art competing methods in terms of both the Area Under Receiver Operating Characteristic Curve (AUC-ROC) and Precision-Recall curve (AUC-PR), with 3%-29% average AUC-ROC improvement and 21%-309% average AUC-PR improvement; and (ii) DevNet obtains a substantially better data efficiency than the competing methods, e.g., it can use 75%-88% less labeled anomalies to achieve the accuracy that is comparably good to, or substantially better than, the best contenders.

2 RELATED WORK

2.1 Traditional Anomaly Detection

Most traditional anomaly detection approaches, e.g., distance-based approach and density-based approach, are ineffective in handling irrelevant features or non-linear separable classes due to the curse of dimensionality and the deficiency in capturing the non-linear relations. Recently ensemble methods (e.g., iForest [14] and many others [9, 18]) showed some large improvement over these approaches by working on selected feature subspaces, but how to efficiently and effectively identify the relevant subspaces and model the intricate relations is still an open problem in anomaly detection.

2.2 Deep Anomaly Detection

Current popular deep anomaly detection methods are unsupervised approach, including autoencoder-based methods and GANs-based methods. Autoencoder-based methods [2, 7, 30] use a bottleneck network architecture to learn a low-dimensional representation space, and then use the learned representations to define reconstruction errors as anomaly scores. GANs-based methods [22, 29] also use the reconstruction error as anomaly score, but they leverage two competing networks, a generator and a discriminator, to adversarially learn a latent space of the training data and use this latent space to compute the reconstruction errors. These deep methods can capture more complex feature interactions than traditional shallow methods such as random projection [12], but they learn the representations separately from the subsequent anomaly detection, leading to suboptimal or unstable detection performance [19, 20]. To address this issue, very recent work [19, 20] focuses on unifying the representation learning and anomaly detection. The REPEN method [19] exploits triplet networks to integrate the representation learning with distance-based detectors, while deep Support Vector Data Description (SVDD) [20] aims to learn representations for the one-classifier, SVDD [27]. Both REPEN and deep SVDD achieve substantial improvement over the previous methods. However, their optimization objective still focuses on

[2]Our code is made available at https://sites.google.com/site/gspangsite/sourcecode.

feature representations, so they optimize the anomaly scoring in an indirect manner. DevNet is fundamentally different from these methods in that DevNet performs a direct differentiable learning of the anomaly scores in an end-to-end fashion.

2.3 Anomaly Detection with Limited Data

Only a few studies have been done on performing anomaly detection with a few labeled anomalies. In [16, 26], a small set of labeled anomalies is incorporated into a belief propagation process to achieve more reliable anomaly scoring, but they are only applicable to graph data. In [19], REPEN leverages a few labeled anomalies to learn more application-relevant feature representations, resulting in over 30% accuracy improvement compared to its fully unsupervised version.

This research line is relevant to few-shot classification [5, 25] and PU learning (i.e., learning from positive and unlabeled examples) [4, 13, 21]. Few-shot classification is relevant because it also aims to leverage a few labeled examples to identify incoming objects of the same class. However, they are very different because (i) in few-shot classification, we have a large number of labeled data of the seen classes during training, but we do not know any class information of the training data in anomaly detection; and (ii) few-shot classification implicitly assumes that the few labeled objects and incoming objects of each of the unseen classes share the same manifold, whereas the few labeled anomalies and the unseen anomalies may be from very different manifolds. The second difference is also the key difference between our task and PU learning, because PU learning also has the same assumption as few-shot classification since they are both focused on classification. Also, most PU learning techniques typically require a substantially large percentage of positive examples to work well, e.g., 45% in [13], 20%-50% in [4] and 20% in [21], which is often not practical or too costly to collect that much anomaly data in many anomaly detection applications. Therefore, both few-shot and PU learning techniques are significantly challenged by the studied problem.

3 END-TO-END ANOMALY SCORE LEARNING

3.1 Problem Statement

Instead of taking the current two-step approach that first learns new representations and then applies some anomaly measures to the new representations to compute anomaly scores, we aim to leverage a small number of labeled anomalies to directly learn the anomaly scores. Specifically, given a set of $N + K$ training data objects $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N, \mathbf{x}_{N+1}, \mathbf{x}_{N+2}, \cdots, \mathbf{x}_{N+K}\}$ with $\mathbf{x}_i \in \mathbb{R}^D$, in which $\mathcal{U} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ is unlabeled data and $\mathcal{K} = \{\mathbf{x}_{N+1}, \mathbf{x}_{N+2}, \cdots, \mathbf{x}_{N+K}\}$ with $K \ll N$ is a very small set of labeled anomalies that provide some prior knowledge of anomalies, our goal is to learn an anomaly scoring function $\phi : \mathcal{X} \mapsto \mathbb{R}$ that assigns anomaly scores to data objects in a way that we have $\phi(\mathbf{x}_i) > \phi(\mathbf{x}_j)$ if $\mathbf{x}_i$ is an anomaly and $\mathbf{x}_j$ is a normal data object.

3.2 The Proposed Framework

To solve this problem, we introduce a novel framework that synthesizes neural networks, a prior probability distribution of anomaly scores, and a new loss function to train a deep anomaly detector in an end-to-end fashion, with an objective to assign statistically significantly larger anomaly scores to anomalies than normal objects. The resulting model is expected to yield more optimized anomaly scores and be more data-efficient than the two-step approach.

3.2.1 The Procedure of the Framework. As shown in Figure 2, our framework consists of three major modules:

(1) We first use an *anomaly scoring network*, i.e., a function ϕ, to yield a scalar anomaly score for every given input $\mathbf{x}$.
(2) To guide the learning of anomaly scores, we then use a *reference score generator* to generate another scalar score termed as *reference score*, which is defined as the mean of the anomaly scores $\{r_1, r_2, \cdots, r_l\}$ for a set of l randomly selected normal objects, denoted as $\mu_{\mathcal{R}}$. The reference score $\mu_{\mathcal{R}}$ may be either learned from a model or determined by a prior probability F. The latter one is chosen so as to efficiently generate $\mu_{\mathcal{R}}$ and obtain interpretable anomaly scores.
(3) Lastly $\phi(\mathbf{x})$, $\mu_{\mathcal{R}}$ and its associated standard deviation $\sigma_{\mathcal{R}}$ are input to the *deviation loss* function L to guide the optimization, in which we aim to optimize the anomaly scores so that the scores of anomalies statistically significantly deviate from $\mu_{\mathcal{R}}$ in the upper tail while at the same time having the scores of normal objects as close as possible to $\mu_{\mathcal{R}}$.

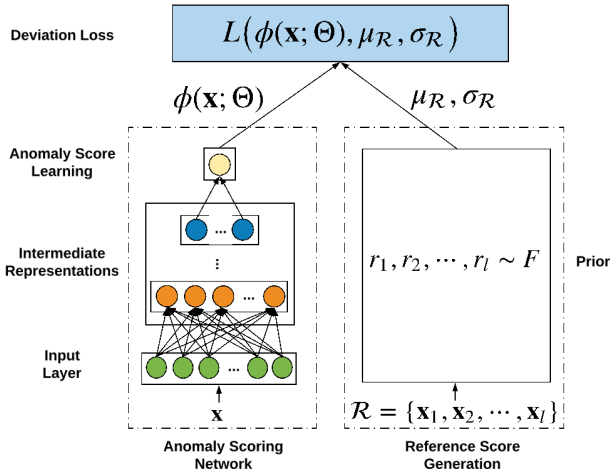

Figure 2: The Proposed Framework. $\phi(\mathbf{x};\Theta)$ is an anomaly score learner with the parameters Θ. $\mu_{\mathcal{R}}$ is the mean of the anomaly scores of some normal objects, which is determined by a prior F. $\sigma_{\mathcal{R}}$ is a standard deviation associated with $\mu_{\mathcal{R}}$. The loss $L(\phi(\mathbf{x};\Theta), \mu_{\mathcal{R}}, \sigma_{\mathcal{R}})$ is defined to guarantee that the anomaly scores of anomalies statistically significantly deviate from $\mu_{\mathcal{R}}$ in the upper tail while enforce normal objects have anomaly scores as close as possible to $\mu_{\mathcal{R}}$.

One problem here is how to effectively obtain a sufficient number of normal objects to train our model, since we only have a few labeled anomalies in $\mathcal{K}$ but do not know the class label of the objects in $\mathcal{U}$. We will discuss how to address this problem in Section 4.3.

3.2.2 How Does The Proposed Framework Address the Aforementioned Two Main Challenges of Deep Anomaly Detection? The deviation loss-based optimization in our framework forces the normal objects cluster around F in terms of their anomaly scores but

pushes anomalies statistically far away F, which well optimizes the anomaly scores and also empowers the intermediate representation learning to discriminate normal objects from the rare anomalies with different anomalous behaviors. In other words, our deep anomaly detector leverages a few labeled anomalies and the prior of anomaly scores to learn a high-level abstraction of normal behaviors, enabling it to assign a large anomaly score to an object as long as the object's behaviors significantly deviate from the learned abstraction of being normal. This offers an effective detection of dissimilar anomalies, e.g., anomalies due to different reasons or previously unknown anomalies; and in turn the optimization also requires substantially less labeled anomalies to train the detector.

4 DEVIATION NETWORKS

The proposed framework is instantiated into a method called Deviation Networks (DevNet), which defines a Gaussian prior and a Z-Score-based deviation loss to enable the direct optimization of anomaly scores with an end-to-end neural anomaly score learner.

4.1 End-to-end Anomaly Scoring Network

Let $\mathcal{Q} \in \mathbb{R}^M$ be an intermediate representation space, an anomaly scoring network $\phi(\cdot;\Theta) : \mathcal{X} \mapsto \mathbb{R}$ can be defined as a combination of a feature representation learner $\psi(\cdot;\Theta_t) : \mathcal{X} \mapsto \mathcal{Q}$ and an anomaly scoring function $\eta(\cdot;\Theta_s) : \mathcal{Q} \mapsto \mathbb{R}$, in which $\Theta = \{\Theta_t, \Theta_s\}$.

Specifically, $\psi(\cdot;\Theta_t)$ is a neural *feature learner* with $H \in \mathbb{N}$ hidden layers and their weight matrices $\Theta_t = \{\mathbf{W}^1, \mathbf{W}^2, \cdots, \mathbf{W}^H\}$, which can be represented as

$$\mathbf{q} = \psi(\mathbf{x};\Theta_t), \tag{1}$$

where $\mathbf{x} \in \mathcal{X}$ and $\mathbf{q} \in \mathcal{Q}$. Different hidden network structures can be used here based on the type of data inputs, such as multilayer perceptron networks for multidimensional data, convolutional networks for image data, or recurrent networks for sequence data.

$\eta(\cdot,\Theta_s) : \mathcal{Q} \mapsto \mathbb{R}$ is defined as an *anomaly score learner* which uses a single linear neural unit in the output layer to compute the anomaly scores based on the intermediate representations:

$$\eta(\mathbf{q};\Theta_s) = \sum_{i=1}^{M} w_i^o q_i + w_{M+1}^o, \tag{2}$$

where $\mathbf{q} \in \mathcal{Q}$ and $\Theta_s = \{\mathbf{w}^o\}$ (w_{M+1}^o is the bias term).

Thus, $\phi(\cdot;\Theta)$ can be formally represented as

$$\phi(\mathbf{x};\Theta) = \eta(\psi(\mathbf{x};\Theta_t);\Theta_s), \tag{3}$$

which directly maps data inputs to scalar anomaly scores and can be trained in an end-to-end fashion.

4.2 Gaussian Prior-based Reference Scores

Having obtained the anomaly scores using $\phi(\mathbf{x};\Theta)$, a *reference score* $\mu_{\mathcal{R}} \in \mathbb{R}$, which is defined as the mean of the anomaly scores of a set of some randomly selected normal objects $\mathcal{R}$, is fed into the network output to guide the optimization. There are two main ways to generate $\mu_{\mathcal{R}}$: data-driven and prior-driven approaches. Data-driven methods involve a model to learn $\mu_{\mathcal{R}}$ based on $\mathcal{X}$, while prior-driven methods generate $\mu_{\mathcal{R}}$ from a chosen prior probability F. The prior-based approach is chosen here because (i) the chosen prior allows us to achieve good interpretability of the predicted anomaly scores and (ii) it can generate $\mu_{\mathcal{R}}$ constantly, which is substantially more efficient than the data-driven approach.

The specification of the prior is the main challenge of the prior-based approach. Fortunately, extensive results in [10] show that Gaussian distribution fits the anomaly scores very well in a range of data sets. This may be due to that the most general distribution for fitting values derived from Gaussian or non-Gaussian variables is the Gaussian distribution according to the central limit theorem. Motivated by this, we define a Gaussian prior-based reference score:

$$r_1, r_2, \cdots, r_l \sim \mathcal{N}(\mu, \sigma^2), \tag{4}$$

$$\mu_{\mathcal{R}} = \frac{1}{l}\sum_{i=1}^{l} r_i, \tag{5}$$

where each r_i is drawn from $\mathcal{N}(\mu, \sigma^2)$ and represents an anomaly score of a random normal data object. We found empirically that DevNet was not sensitive to the choices of μ and σ as long as σ was not too large. We set $\mu = 0$ and $\sigma = 1$ in our experiments, which help DevNet to achieve stable detection performance on different data sets. DevNet is also not sensitive to l when l is sufficiently large due to the central limit theorem. $l = 5000$ is used here.

4.3 Z-Score-based Deviation Loss

A deviation loss is then defined to optimize the anomaly scoring network, with the deviation specified as a Z-Score

$$dev(\mathbf{x}) = \frac{\phi(\mathbf{x};\Theta) - \mu_{\mathcal{R}}}{\sigma_{\mathcal{R}}}, \tag{6}$$

where $\sigma_{\mathcal{R}}$ is the standard deviation of the prior-based anomaly score set, $\{r_1, r_2, \cdots, r_l\}$. The deviation can then be plugged into the contrastive loss [6] to specify our deviation loss as follows

$$L\big(\phi(\mathbf{x};\Theta), \mu_{\mathcal{R}}, \sigma_{\mathcal{R}}\big) = (1-y)|dev(\mathbf{x})| + y\max\big(0, a - dev(\mathbf{x})\big), \tag{7}$$

where $y = 1$ if $\mathbf{x}$ is an anomaly and $y = 0$ if $\mathbf{x}$ is a normal object, and a is equivalent to a Z-Score confidence interval parameter. This loss enables DevNet to push the anomaly scores of normal objects as close as possible to $\mu_{\mathcal{R}}$ while enforce a deviation of at least a between $\mu_{\mathcal{R}}$ and the anomaly scores of anomalies. Note that if $\mathbf{x}$ is an anomaly and it has a negative $dev(\mathbf{x})$, the loss is particularly large, which encourages large *positive deviations* for all anomalies. Therefore, the deviation loss is equivalent to enforcing a statistically significant deviation of the anomaly score of all anomalies from that of normal objects in the upper tail. We use $a = 5$ to achieve a very high significance level (i.e., 5.73303e-07) for all labeled anomalies.

Similar to the contrastive loss, the deviation loss is monotonically increasing in $|dev(\mathbf{x})|$ and is monotonically deceasing in $\max\big(0, a - dev(\mathbf{x})\big)$, so it is convex w.r.t. both cases. However, they are also very different, because the contrastive loss uses pairs of intra-class/inter-class data objects as training samples to learn a similarity metric, whereas our deviation loss is built upon the deviation function and dedicated to the direct learning of anomaly scores.

One problem for using Eqn. (7) is that we do not have the labeled normal objects. We address this problem by simply treating the unlabeled training data objects in $\mathcal{U}$ as normal objects. Our empirical results showed that DevNet and also its competing deep methods performed very well by using this simple strategy, even

when there was a large *anomaly contamination level* (i.e., the proportion of anomalies in the unlabeled training data set $\mathcal{U}$). This may be because anomalies are rare data objects and their impacts become very limited on the stochastic gradient descent-based optimization in these deep detectors. Therefore, this training strategy is used by DevNet and its competing deep methods throughout our experiments. This can be seen as training the model with noisy data sets. We will evaluate the impact of different noise levels on the detection performance in Sections 5.7.

4.4 The DevNet Algorithm

Algorithm 1 presents the procedure of DevNet. After a random weight initialization in Step 1, DevNet performs stochastic gradient descent-based optimization to learn the weights in Θ in Steps 2-10. Particularly, Step 4 first samples a mini-batch $\mathcal{B}$ of size b using stratified random sampling, followed by sampling the anomaly scores of l normal objects from the prior $\mathcal{N}(\mu, \sigma^2)$ in Step 5. After obtaining $\mu_{\mathcal{R}}$ and $\sigma_{\mathcal{R}}$ in Step 6, Step 7 performs the forward propagation of the anomaly scoring network and computes the loss. Step 8 then performs gradient descent steps w.r.t. the parameters in Θ. We finally obtain the optimized scoring network ϕ.

Algorithm 1 *Training DevNet*

Input: $\mathcal{X} \in \mathbb{R}^D$ - training data objects, i.e., $\mathcal{X} = \mathcal{U} \cup \mathcal{K}$ and $\emptyset = \mathcal{U} \cap \mathcal{K}$
Output: $\phi : \mathcal{X} \mapsto \mathbb{R}$ - an anomaly scoring network

1: Randomly initialize Θ
2: **for** $i = 1$ to *n_epochs* **do**
3: **for** $j = 1$ to *n_batches* **do**
4: $\mathcal{B} \leftarrow$ Randomly sample b data objects with a half of objects from $\mathcal{K}$ and another half from $\mathcal{U}$
5: Randomly sample l anomaly scores from $\mathcal{N}(\mu, \sigma^2)$
6: Compute $\mu_{\mathcal{R}}$ and $\sigma_{\mathcal{R}}$ of the l anomaly scores: $\{r_1, r_2, \cdots, r_l\}$
7: $loss \leftarrow \frac{1}{b} \sum_{\mathbf{x} \in \mathcal{B}} L\big(\phi(\mathbf{x}; \Theta), \mu_{\mathcal{R}}, \sigma_{\mathcal{R}}\big)$
8: Perform a gradient descent step w.r.t. the parameters in Θ
9: **end for**
10: **end for**
11: **return** ϕ

The core computation of training DevNet is the forward and backward propagation of the anomaly scoring network ϕ, so the time complexity of DevNet depends on the network architecture used. For example, for multilayer perceptron networks, both the forward and backward propagation have the same complexity of $O(Dh_1 + h_1h_2 + \cdots + h_H * 1)$, where h_i is the number of neural units in the i-th hidden layer, so DevNet has an overall time complexity of $O\big(n\_epochs * n\_batches * b * (Dh_1 + h_1h_2 + \cdots + h_H)\big)$ for its training and $O\big(I(Dh_1 + h_1h_2 + \cdots + h_H)\big)$ for its testing, where I is the data size of the test set.

4.5 Interpretability of Anomaly Scores

At the testing stage, like other anomaly detection methods, DevNet uses the optimized ϕ to produce an anomaly score for every test object and returns an anomaly ranking of the data objects based on the anomaly scores, in which the top-ranked objects are anomalies. However, the anomaly scores returned by most anomaly detectors are often not easily interpretable [10]. As a result, given a data object's anomaly score, it is not clear what is the probability of this object being an anomaly, and it is also difficult to determine a specific threshold to select the appropriate top-ranked objects. Therefore, if users need more than an anomaly ranking in practice, some types of separate anomaly score unification methods [10] are required for those methods to transform their scores into more interpretable ones. However, the anomaly scoring and the score unification are two independent modules in such cases, which may lead to untrustworthy explanation of the scores. By contrast, DevNet directly yields highly interpretable anomaly scores.

Proposition 4.1. *Let $\mathbf{x} \in \mathcal{X}$ and z_p be the quantile function of $\mathcal{N}(\mu, \sigma^2)$, then $\phi(\mathbf{x})$ lies outside the interval $\mu \pm z_p\sigma$ with a probability $2(1-p)$.*

This proposition of DevNet is due to the Gaussian prior and Z-Score-based deviation loss. The probability $2(1-p)$ offers a straightforward explanation to the anomalousness of any given score $\phi(\mathbf{x})$. Particularly, we have the probability $(1-p)$ when only focusing on the upper tail $\mu + z_p\sigma$, e.g., by applying $p = 0.95$, we have $z_{0.95} = 1.96$, which states that having anomaly scores over 1.96 (as $\mu = 0$ and $\sigma = 1$ are used in DevNet) indicates the object only has a probability of 0.05 generated from the same mechanism as the normal data objects. Users can also easily choose a threshold to determine anomalies with a desired confidence level, e.g., given the anomaly score distribution shown in Figure 1(b), it is easy to use $z_{0.95}$ to identify the anomalies with a 95% confidence level.

5 EXPERIMENTS

5.1 Data Sets

As shown in Table 1, nine publicly available real-world data sets are used, which are from diverse critical domains, e.g., intrusion detection, fraud detection, malicious URL detection, and disease detection. Five data sets contain real anomalies, i.e., exceptionally exciting projects in *donors*, fraudulent credit card transactions in *fraud*, backdoor network attacks in *backdoor*, malicious URLs in *URL*, and hypothyroid patients in *thyroid*. The other four data sets contain semantically real anomalies, i.e., they are rare or very different from the majority of data objects. The detailed information of accessing and preprocessing the data sets can be found in Appendix A.1.

5.2 Competing Methods

DevNet is compared with four methods, including REPEN [19], *adaptive* Deep SVDD (DSVDD) [20], prototypical networks (denoted as FSNet) [25], and iForest [14]. These four methods are chosen because they are the state-of-the-art in the relevant areas, i.e., REPEN in deep anomaly detection with limited labeled data, DSVDD in feature learning for anomaly detection, FSNet in few-shot classification, and iForest in unsupervised anomaly detection.

The original DSVDD is designed to minimize the distance between a fixed one-class center vector $\mathbf{c}$ and the training data in the projected space, in which the labeled anomalies cannot be used. To have a fair comparison to DevNet, we modified DSVDD to fully leverage the labeled anomalies based on [27], by adding an additional term into its objective function to guarantee a large margin between normal objects and anomalies in the new space while minimizing the $\mathbf{c}$-based hypersphere's volume. This adaption significantly enhances the original DSVDD.

Table 1: AUC-ROC and AUC-PR Performance (with ± standard deviation) of DevNet and Four Competing Methods. #obj. is the overall data size, D is the dimensionality size, f_1 and f_2 denote the percentage that the labeled anomalies respectively comprise in the training data and the total anomalies. D in *URL* and *news20*, i.e., '3M' and '1M', are short for 3,231,961 and 1,355,191, respectively. The best performance is boldfaced.

Data Characteristic					AUC-ROC Performance					AUC-PR Performance				
Data	#obj.	D	f_1	f_2	DevNet	REPEN	DSVDD	FSNet	iForest	DevNet	REPEN	DSVDD	FSNet	iForest
donors	619,326	10	0.01%	0.08%	**1.000**±0.000	0.975±0.005	0.995±0.005	0.997±0.002	0.874±0.015	**1.000**±0.000	0.508±0.048	0.846±0.114	0.994±0.002	0.221±0.025
census	299,285	500	0.01%	0.16%	0.828±0.008	0.794±0.005	**0.835**±0.014	0.732±0.020	0.624±0.020	**0.321**±0.004	0.164±0.003	0.291±0.008	0.193±0.019	0.076±0.004
fraud	284,807	29	0.01%	6.10%	**0.980**±0.001	0.972±0.003	0.977±0.001	0.734±0.046	0.953±0.002	**0.690**±0.002	0.674±0.004	0.688±0.004	0.043±0.021	0.254±0.043
celeba	202,599	39	0.02%	0.66%	**0.951**±0.001	0.894±0.005	0.944±0.003	0.808±0.027	0.698±0.020	**0.279**±0.009	0.161±0.006	0.261±0.008	0.085±0.012	0.065±0.006
backdoor	95,329	196	0.04%	1.29%	**0.969**±0.004	0.878±0.007	0.952±0.018	0.928±0.019	0.752±0.021	**0.883**±0.008	0.116±0.003	0.856±0.016	0.573±0.167	0.051±0.005
URL	89,063	3M	0.04%	1.69%	**0.977**±0.004	0.842±0.006	0.908±0.027	0.786±0.047	0.720±0.032	**0.681**±0.022	0.103±0.003	0.475±0.040	0.149±0.076	0.066±0.012
campaign	41,188	62	0.10%	0.65%	**0.807**±0.006	0.723±0.006	0.748±0.019	0.623±0.024	0.731±0.015	**0.381**±0.008	0.330±0.009	0.349±0.023	0.193±0.012	0.328±0.022
news20	10,523	1M	0.37%	5.70%	**0.950**±0.007	0.885±0.003	0.887±0.000	0.578±0.050	0.328±0.016	**0.653**±0.009	0.222±0.004	0.253±0.001	0.082±0.010	0.035±0.002
thyroid	7,200	21	0.55%	5.62%	**0.783**±0.003	0.580±0.016	0.749±0.011	0.564±0.017	0.688±0.020	**0.274**±0.011	0.093±0.005	0.241±0.009	0.116±0.014	0.166±0.017
				Average	**0.916**±0.004	0.838±0.006	0.888±0.011	0.750±0.028	0.708±0.018	**0.574**±0.008	0.263±0.010	0.473±0.025	0.270±0.037	0.140±0.015
				P-value	-	0.004	0.023	0.004	0.004	-	0.004	0.004	0.004	0.004

All methods are implemented in Python: DevNet, DSVDD and FSNet are implemented using Keras [3], REPEN is taken from its authors and is also built upon Keras, and iForest is taken from the scikit-learn package.

5.3 Parameter Settings

Since our experiments focus on unordered multidimensional data, multilayer perceptron (MLP) network architectures are used. Specifically, we tested two architectures for all neural methods. Motivated by the success of REPEN [19], our first network uses the same architecture as REPEN, i.e., one hidden layer with 20 neural units. The second architecture consists of three hidden layers to learn more intricate data interactions, with 1,000 units in the first hidden layer, followed by 250 and 20 units in the second and third hidden layers, respectively. The ReLu function $g(z) = max(0, z)$ is used because of its efficient computation and gradient propagation, and an ℓ_2-norm regularizer is applied to every hidden layer to avoid overfitting.

All DevNet, REPEN, DSVDD and FSNet were tested using these two architectures on all the data sets, and we found all of them performed best with the one hidden layer structure. This may be due to the limit of the available labeled data. Due to the page limitation, we report the results based on the architecture with one hidden layer by default. We show the results of DevNet using the three hidden layers in our ablation study in Section 5.8.

In training, DevNet, DSVDD and FSNet use the Root Mean Square propagation (RMSprop) optimizer [8] to perform gradient descents, and they are trained using 50 epochs, with 20 min-batches in each epoch. These settings enable the three deep detectors to achieve stable performance across the data sets. iForest is a non-neural ensemble method. It is used with the recommended settings, i.e., subsampling size set to 256 and ensemble size set to 100 [14]. iForest cannot work in data with millions of features, so we use the sparse random projection [12] to map *URL* and *news20* into a 1,000-dimensional space before applying iForest, which obtains better performance than other projection options.

5.4 Performance Evaluation Methods

We use two popular and complementary performance metrics, the Area Under Receiver Operating Characteristic Curve (AUC-ROC) and the Area Under Precision-Recall Curve (AUC-PR), to have a comprehensive evaluation of anomaly detectors. AUC-ROC summarizes the ROC curve of true positives against false positives, while AUC-PR is a summarization of the curve of precision against recall. Specifically, an AUC-ROC value of one indicates the best performance, while a value close to 0.5 indicates a random ranking of the objects. AUC-ROC is widely used due to its good interpretability.

However, AUC-PR is more suitable than AUC-ROC in many anomaly detection applications which require excellent performance on the positive class and do not care much of the performance on the negative class. This is because AUC-ROC is affected by the performance on both anomaly and normal classes and the performance on the normal class can bias AUC-ROC due to the class-imbalance nature of anomaly detection data. By contrast, AUC-PR evaluates how many positive predictions are correct (precision), and how many of the positive predictions that are truly positive compose the positive class (recall). We use a widely-used method, known as average precision in [23], to calculate AUC-PR. Large AUC-PR indicates better performance, but it is often very challenging to achieve large AUC-PR due to the skewed and heterogeneous distributions of anomalies.

The reported AUC-ROC, AUC-PR, and runtimes are averaged results over 10 independent runs. The paired *Wilcoxon* signed rank test [28] is used to examine the significance of the performance of DevNet against its competing methods. All runtimes are calculated at a node in a 2.4GHz Phoenix cluster with 64GB dedicated memory using 8 cores and 1 Tesla K80 GPU accelerator.

5.5 Effectiveness in Real-world Data Sets

5.5.1 Experiment Settings. This section examines the performance of DevNet on common real-life application scenarios where there are a large number of unlabeled data objects with a very small set of labeled anomalies. To replicate such scenarios, the anomalies and normal objects in each data set are first splitted into two subsets, with 80% data as training data and the other 20% data as test set. To have controlled experiments w.r.t. anomaly contamination, we then randomly add/remove the anomalies in each training data set such that the anomalies account for 2% of the training data, i.e., 2% anomaly contamination (other contamination levels are further examined in Section 5.7). The resulted data forms the unlabeled training data set $\mathcal{U}$. We further randomly sample 30 anomalies from the anomaly class as the prior knowledge of the anomalies of

interest, i.e., the labeled anomaly set $\mathcal{K}$, which accounts for only 0.005%-1% of all training data objects and 0.08%-6% of the anomaly class (see f_1 and f_2 in Table 1 for detail). Since only the class label of $\mathcal{K}$ is used during training, the task is equivalent to unsupervised anomaly detection with a few additional labeled anomalies available as prior knowledge. We will investigate the detection performance w.r.t. different amount of the prior knowledge in Section 5.6.

5.5.2 Findings - The direct optimization of anomaly scores enables DevNet to achieve significant improvement over other deep methods. The AUC-ROC and AUC-PR performance of DevNet and four competing methods are shown in Table 1. DevNet performs best on eight and nine data sets in the respective AUC-ROC and AUC-PR performance, and it performs comparably well to the best performer on *census* in AUC-ROC where it ranks in second. In terms of AUC-ROC, DevNet obtains substantially better average improvement than REPEN (9%), DSVDD (3%), FSNet (22%) and iForest (29%) and the improvement is statistically significant at the 95% or 99% confidence interval; in terms of AUC-PR, the improvement DevNet achieves is much more substantial than REPEN (118%), DSVDD (21%), FSNet (113%) and iForest (309%), which is all statistically significant at the 99% confidence interval. These results are due to the reason that DevNet efficiently leverages the limited available anomalies to well optimize the anomaly scores, resulting in high-quality anomaly rankings, i.e., substantially high precision and recall of detecting anomalies; while the competing methods have an indirect learning of anomaly scores, resulting in weak capability of discriminating some intricate anomalies from normal objects and thus high false positives and low recall rates.

5.6 Data Efficiency

5.6.1 Experiment Settings. This section examines the data efficiency of the deep methods, which is a critical factor as it is very difficult to obtain labeled anomalies in most anomaly detection applications. The number of available labeled anomalies varies from 5 to 120, with the anomaly contamination level fixed to be 2%. iForest is used as the baseline, which is an unsupervised method and thus insensitive to the amount of the labeled data. We aim to answer the following two key questions:

- How data-efficient are the DevNet and other deep methods?
- How much improvement can the deep methods gain from the labeled anomalies compared to the unsupervised iForest?

5.6.2 Findings - DevNet is the most data-efficient method; and the improvement due to the limited labeled anomalies is very substantial. Figure 3 shows the AUC-PR results w.r.t. different number of labeled anomalies available. Similar results can also be observed in AUC-ROC. The performance of these four deep methods generally increases with increasing number of labeled anomalies, since more labeled data generally helps train the model better. However, the AUC-PR of some competing deep detectors drops with more labeled data in some cases, e.g., FSNet in *census* and *backdoor*, REPEN in *celeba* and *news20*, DSVDD in *backdoor* and *thyroid*. This may be due to the scattered and dissimilar distributions of anomalies, because when the added labeled anomalies have very different anomalous behaviors and carry information conflicting to the other labeled anomalies for the optimization, they may then downgrade the detection performance. Compared to the counterparts, DevNet is more stable in such cases.

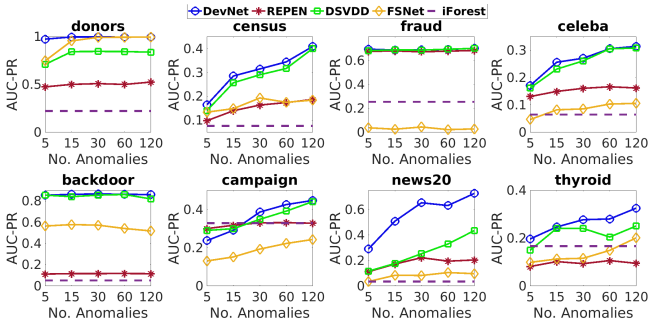

Figure 3: AUC-PR w.r.t. No. Labeled Anomalies. The results on *URL* are omitted due to prohibitively expensive computation.

DevNet is the most data-efficient method, which obtains the best average performance w.r.t. different number of labeled anomalies and achieves the fastest increase rate of AUC-PR against the number of labeled anomalies. Impressively, DevNet needs 75%-88% less labeled data to achieve comparably better performance to the best competing method in several cases, e.g., DevNet requires 83% less labeled data to achieve comparably good performance to the best contender FSNet on *donors*, and outperforms the best contender DSVDD on *news20* and *thyroid* using respective 88% and 75% less labeled data. The DevNet's superiority is due to its end-to-end differentiable learning of the anomaly scores, because it allows DevNet to directly optimize the anomaly scores with the limited labeled data, which can leverage the data much more efficiently than the counterpart two-step approach.

Compared to the unsupervised method iForest, even when only a very few labeled anomalies (e.g., 5 or 15) are used, the improvement of the prior knowledge-driven deep methods, especially DevNet and DSVDD, is very substantial on most data sets, such as *donors*, *census*, *fraud*, *celeba*, *backdoor*, *news20* and *thyroid*; for example, the average improvement of DevNet and DSVDD using 5 labels over iForest is more than 400%. In the case of *campaign* that may have very intricate distributions of anomalies, the deep methods need slightly more labeled data to achieve the similarly large improvement.

5.7 Robustness w.r.t. Anomaly Contamination

5.7.1 Experiment Settings. Recall that we use a simple training strategy to train DevNet and the other deep methods, i.e., all unlabeled training data objects in $\mathcal{U}$ are used as normal data objects and we sample negative data objects from this set of objects to comprise a half of data objects in each mini-batch (see Step 4 in Algorithm 1). This section investigates the robustness of DevNet w.r.t. different anomaly contamination levels in the unlabeled training data. We vary the contamination level from 0% up to 20%, with the number of available labeled anomalies fixed to be 30. We aim to examine the following two key questions:

- How robust are the deep anomaly detectors?
- Can the deep methods still substantially beat the unsupervised method iForest when the contamination level is high?

5.7.2 Findings - DevNet is consistently more robust than the other deep methods; and the substantially better improvement of DevNet over iForest persists even when a very large anomaly contamination is presented in the unlabeled training data. The AUC-PR results w.r.t. different anomaly contamination levels are presented in Figure 4. Similar results can also be observed in AUC-ROC. The performance of all deep anomaly detectors decreases with increasing contamination levels. This is because the probability of falsely sampling anomalies from the unlabeled data as normal objects gets larger in the mini-batch construction, which can mislead the stochastic gradient descent-based optimization and downgrade the detection accuracy. Nevertheless, it is clear that DevNet performs consistently better and achieves remarkably better average AUC-PR performance than REPEN (200%), DSVDD (28%) and FSNet (336%) over the different contamination levels. This demonstrates a strong capability of DevNet in tapping the limited prior knowledge to well optimize the anomaly scores in challenging noisy environments.

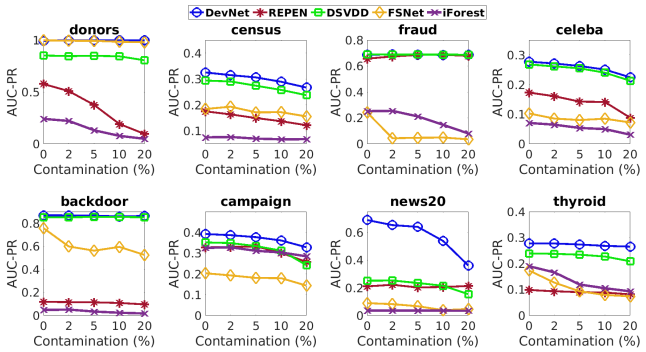

Figure 4: AUC-PR w.r.t. Different Contamination Rates. The results on *URL* are omitted due to prohibitively expensive computation.

Compared to iForest, the four deep methods obtain substantially better average AUC-PR improvement across the eight data sets, e.g., DevNet and DSVDD have respectively more than 800% and 600% average improvement. This is because although the large anomaly contamination in the unlabeled data presents many noises to the deep model training, the small set of labeled anomalies empowers the deep methods and help them to largely defy the noises. By contrast, the unsupervised method iForest does not have any prior knowledge of anomalies and thus returns many noisy or uninteresting objects as anomalies, leading to very large false positives; also, its performance still decreases with increasing anomaly contamination rate, because the unsupervised methods like iForest typically assume that anomalies are rare in the unlabeled data and thus they perform less effectively when the increasing anomaly contamination violates the assumption.

5.8 Ablation Study

5.8.1 Experiment Settings. We examine the importance of the key components of DevNet by comparing DevNet to its three variants. Recall that the default DevNet (denoted as Def) has one hidden layer with 20 ReLu units and a linear unit in the output layer.

- The first variant is DevNet-Rep, which removes the output layer of Def and uses our deviation loss to learn the representations only. In this case, the reference in the loss function is a 20-dimensional vector rather than a scalar.
- The second variant is DevNet-Linear, which removes the non-linear learning hidden layer of Def, making it equivalent to learning a direct linear mapping from the original data space to the anomaly score space.
- The third variant is DevNet-3HL, in which three hidden layers with respective 1000, 250 and 20 ReLu units are used.

5.8.2 Findings - The end-to-end learning of anomaly scores, deviation loss, and learning of non-linear features all have some major contributions to the superior performance of DevNet. Table 2 shows the performance of DevNet and its three variants. The end-to-end learning of anomaly scores enables Def to obtain more accurate and stable performance than Rep that focuses on feature learning. Def performs less effectively than Rep in *census*. This may be due to that some normal objects and anomalies in *census* are quite similar, which can mislead the score learning in Def more severely than the representation learning in Rep.

Table 2: AUC-ROC and AUC-PR Results of DevNet and Its Variants.

	AUC-ROC Performance				AUC-PR Performance			
Data	Def	Rep	Linear	3HL	Def	Rep	Linear	3HL
donors	**1.000**	0.999	0.978	**1.000**	**1.000**	0.976	0.827	**1.000**
census	0.828	**0.858**	0.832	0.686	0.321	**0.338**	0.297	0.241
fraud	**0.980**	0.975	0.937	0.926	0.690	0.684	0.659	**0.701**
celeba	**0.951**	0.949	0.949	0.877	0.279	**0.283**	0.281	0.239
backdoor	**0.969**	0.913	0.928	0.968	**0.883**	0.846	0.555	0.843
URL	**0.977**	0.954	0.872	0.941	0.681	**0.687**	0.347	0.595
campaign	**0.807**	0.759	0.757	0.679	**0.381**	0.371	0.357	0.259
news20	0.950	**0.953**	0.819	0.817	**0.653**	0.552	0.447	0.421
thyroid	0.783	0.729	0.717	**0.787**	0.274	0.216	0.205	**0.383**
Average	**0.916**	0.899	0.865	0.853	**0.574**	0.550	0.442	0.520
P-value	-	0.129	0.012	0.023	-	0.106	0.008	0.133

Note that Rep and DSVDD actually share a similar objective, but Rep uses the deviation loss while DSVDD uses the SVDD-based loss. Compared to DSVDD in Table 1, Rep performs slightly better in AUC-ROC (1% improvement) and substantially better in AUC-PR (16% improvement). This indicates that our deviation loss offers a much better capability in capturing different anomalous behaviors.

Compared to Linear, Def obtains significantly better average AUC-ROC (6%) and AUC-PR (30%) improvement, indicating a significant role of the intermediate non-linear feature learning before the learning of the anomaly scores. However, as illustrated by the substantial average improvement of Def over 3HL, deepening the hidden layers from one layer to three layers is not always beneficial, because we have only a few labeled anomalies, which are often not sufficient to well train a deeper model.

5.9 Scalability Test

5.9.1 Experiment Settings. We examine the scalability w.r.t. data size by generating four synthetic 1,000-dimensional data sets with varying data sizes. Similarly, the scaleup test w.r.t. dimension uses a fixed data size (i.e., 5,000) and varying dimensions. Each detector is trained and tested in a data set of the same size. The runtime below includes both training and testing execution time.

5.9.2 Findings - DevNet has a linear time complexity w.r.t. both data size and dimension. The scaleup test results are presented in Figure 5. These results show that the overall runtime of DevNet increases linearly w.r.t. both data size and dimension, which justifies the complexity analysis w.r.t. multilayer perceptron networks in Section 4.4. Particularly, although REPEN, FSNet and iForest also have linear time complexity, DevNet runs considerably faster than them by a factor of 10 to 20 on the large data sets. This is because the loss function in DevNet is very computationally efficient, whereas REPEN and FSNet involves extensive distance computation in both training and testing, and iForest needs much time on constructing isolation trees. On the high-dimensional data, DevNet runs comparably fast to REPEN and DSVDD but slightly slower than FSNet. This may be due to the fact that the computation in the bottom layers that project original very high-dimensional data into low-dimensional space dominates the overall runtime, as it is much more costly than the top layer that calculates the loss. As a result, FSNet, which uses a much smaller mini-batch size, requires less time to process each batch data and obtains a better computation efficiency than other methods like DevNet and DSVDD. iForest requires considerable time to perform random data space partition when the dimension is large, leading to the most costly method here.

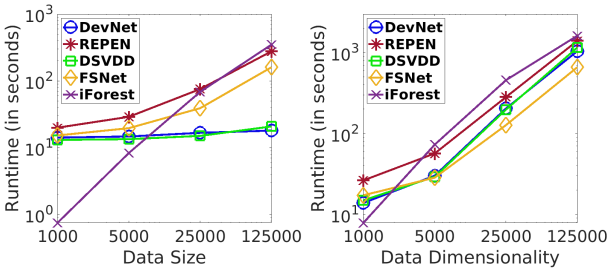

Figure 5: Scalability Test w.r.t. Data Size and Dimensionality.

6 CONCLUSIONS

This paper introduces a novel framework and its instantiation DevNet for leveraging a few labeled anomalies with a prior to fulfill an end-to-end differentiable learning of anomaly scores. By a direct optimization of anomaly scores, DevNet can be trained much more data-efficiently, and performs significantly better in terms of both AUC-ROC and AUC-PR, compared to the two-step deep anomaly detectors that focus on optimizing feature representations. We also find empirically that deep anomaly detectors can be well trained by randomly sampling negative examples from the anomaly-contaminated unlabeled data and positive examples from the small labeled anomaly set. Even when the anomaly contamination level is high, the deep detectors, especially DevNet, can still perform very well and achieve significant improvement over the state-of-the-art unsupervised anomaly detectors. This may provide a new perspective for optimizing anomaly detection methods.

We are testing DevNet on image and sequence data using convolutional/recurrent network architectures, and plan to extend DevNet by a hybrid of data-driven and prior-driven reference score generation approach for extremely challenging real-world applications where only one or two labeled anomalies are available.

ACKNOWLEDGEMENTS

This work is partially supported by the ARC Discovery Project DP180103023.

REFERENCES

[1] Charu C Aggarwal. 2017. Supervised outlier detection. In *Outlier Analysis*. Springer, 219–248.
[2] Jinghui Chen, Saket Sathe, Charu Aggarwal, and Deepak Turaga. 2017. Outlier detection with autoencoder ensembles. In *SDM*. SIAM, 90–98.
[3] François Chollet et al. 2015. Keras. https://keras.io.
[4] Charles Elkan and Keith Noto. 2008. Learning classifiers from only positive and unlabeled data. In *KDD*. ACM, 213–220.
[5] 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.
[6] R. Hadsell, S. Chopra, and Y. LeCun. 2006. Dimensionality Reduction by Learning an Invariant Mapping. In *CVPR*, Vol. 2. 1735–1742.
[7] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. 2002. Outlier detection using replicator neural networks. In *DaWaK*. Springer, 170–180.
[8] Geoffrey Hinton. 2012. Overview of mini-batch gradient descent. (2012). https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
[9] Fabian Keller, Emmanuel Muller, and Klemens Bohm. 2012. HiCS: High contrast subspaces for density-based outlier ranking. In *ICDE*. IEEE, 1037–1048.
[10] Hans-Peter Kriegel, Peer Kroger, Erich Schubert, and Arthur Zimek. 2011. Interpreting and unifying outlier scores. In *SDM*. SIAM, 13–24.
[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.
[12] Ping Li, Trevor J Hastie, and Kenneth W Church. 2006. Very sparse random projections. In *KDD*. ACM, 287–296.
[13] Xiaoli Li and Bing Liu. 2003. Learning to classify texts using positive and unlabeled data. In *IJCAI*, Vol. 3. 587–592.
[14] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data* 6, 1 (2012), 3.
[15] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Identifying suspicious URLs: An application of large-scale online learning. In *ICML*. ACM, 681–688.
[16] Mary McGlohon, Stephen Bay, Markus G Anderle, David M Steier, and Christos Faloutsos. 2009. SNARE: A link analytic system for graph labeling and risk detection. In *KDD*. ACM, 1265–1274.
[17] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Military Communications and Information Systems Conference, 2015*. 1–6.
[18] Guansong Pang, Longbing Cao, Ling Chen, Defu Lian, and Huan Liu. 2018. Sparse modeling-based sequential ensemble learning for effective outlier detection in high-dimensional numeric data. In *AAAI*. AAAI press, 3892–3899.
[19] Guansong Pang, Longbing Cao, Ling Chen, and Huan Liu. 2018. Learning Representations of Ultrahigh-dimensional Data for Random Distance-based Outlier Detection. In *KDD*. 2041–2050.
[20] Lukas Ruff, Nico Görnitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Robert Vandermeulen, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep one-class classification. In *ICML*. 4390–4399.
[21] Emanuele Sansone, Francesco GB De Natale, and Zhi-Hua Zhou. 2018. Efficient training for positive unlabeled learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018).
[22] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. 2017. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *IPMI*. Springer, Cham, 146–157.
[23] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
[24] Md Amran Siddiqui, Alan Fern, Thomas G. Dietterich, Ryan Wright, Alec Theriault, and David W. Archer. 2018. Feedback-Guided Anomaly Discovery via Online Optimization. In *KDD*. ACM, 2200–2209.
[25] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *NeurIPS*. 4077–4087.
[26] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. 2014. Guilt by association: Large scale malware detection by mining file-relation graphs. In *KDD*. 1524–1533.
[27] David MJ Tax and Robert PW Duin. 2004. Support vector data description. *Machine Learning* 54, 1 (2004), 45–66.
[28] RF Woolson. 2007. Wilcoxon signed-rank test. *Wiley Encyclopedia of Clinical Trials* (2007), 1–3.
[29] Houssam Zenati, Manon Romain, Chuan-Sheng Foo, Bruno Lecouat, and Vijay Chandrasekhar. 2018. Adversarially Learned Anomaly Detection. In *ICDM*. IEEE, 727–736.
[30] Chong Zhou and Randy C Paffenroth. 2017. Anomaly detection with robust deep autoencoders. In *KDD*. ACM, 665–674.

A SUPPLEMENTARY MATERIAL FOR REPRODUCIBILITY

A.1 Data Accessing and Preprocessing

The *donors* data is taken from KDD Cup 2014 for predicting excitement of projects proposed by K-12 school teachers, in which exceptionally exciting projects are used as anomalies (5.92% data). The *census* data is extracted from the US census bureau database, in which we aim to detect the rare high-income person (i.e., the person who earns over 50K dollars a year), which is about 6% of the data. The *fraud* data is for fraudulent credit card transaction detection, in which the fraudulent transactions are used as anomalies. The *celeba* data is a large-scale image data set which contains more than 200K celebrity images, each with 40 attribute annotations. We use the *bald* attribute as our detection target, in which the scarce bald celebrities, less than 3% celebrities, are treated as anomalies, and the other 39 attributes form the learning feature space. The *backdoor* data is a backdoor attack detection data set with the attacks as anomalies against the 'normal' class, which is extracted from the UNSW-NB 15 data set [17]. The *URL* data is for malicious URL detection, which consists of 120-day collection of malicious and benign URLs [15]. Following [19], the first-week subset of this collection is used and the malicious URLs are used as anomalies. The *campaign* data is a data set of direct bank marketing campaigns via phone calls, in which the rarely successful campaigning records, accounting for about 10% records, are used as anomalies. The *news20* data is a balanced text classification data set. Following the literature [9, 19], *news20* is converted to anomaly detection data with 5% anomalies by downsampling the small class. The *thyroid* data is a disease detection data set, in which the anomalies are the patients diagnosed with hypothyroid. All these data sets can be publicly accessed via the links provided in Table 3.

Table 3: Links for Accessing the Data Sets

Data	Link
donors	https://www.kaggle.com/c/kdd-cup-2014-predicting-excitement-at-donors-choose
census	https://archive.ics.uci.edu/ml/datasets/Census-Income+(KDD)
fraud	https://www.kaggle.com/mlg-ulb/creditcardfraud
celeba	http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
backdoor	https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/
URL	http://www.sysnet.ucsd.edu/projects/url/
campaign	https://archive.ics.uci.edu/ml/datasets/bank+marketing
news20	https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/
thyroid	http://archive.ics.uci.edu/ml/datasets/thyroid+disease

For these data sets, missing values are replaced with the mean value in the corresponding feature, and categorical features are encoded by one-hot encoding.

A.2 Algorithm Implementation

This section provides the detailed information of our implementation of algorithms. Relevant key information is also presented in Section 5.3.

A.2.1 Implementation of Competing Methods. We use the implementation of iForest available at the scikit-learn Python package. REPEN is directly taken from the authors. Its codes are publicly accessible at https://sites.google.com/site/gspangsite/sourcecode. We implement and further enhance DSVDD by adding an additional margin term into the one-class SVDD objective to enforce a margin between the center $\mathbf{c}$ and the labeled anomalies in the new representation space. Similar to DevNet, the contrastive loss [6] is used in DSVDD to fulfill this margin-based optimization. The anomaly score is defined as the distance to the one-class center $\mathbf{c}$, which is exactly the same as in its original paper. Due to the incorporating of the few labeled anomalies, the modified DSVDD substantially improves the original DSVDD by more than 30% detection accuracy. For FSNet, since we do not have the finer-grained class information in the training data, we cannot construct the training episodes in the same way as in [25]. Instead we randomly sample the same number of data objects from the unlabeled training data and from the limited labeled anomalies to form the desired episodes for training FSNet. The anomaly score is then calculated as a softmax over distances to the respective normal and anomaly prototypes.

A.2.2 Optimization Settings. In optimizing the deep anomaly detection methods, the default settings of the layers or optimizer in Keras are used, and they are as described in Section 5.3 otherwise. Particularly, for the hidden layer, we use the dense layer with an uniform Glorot weight initialization and an ℓ_2-norm weight decay regularizer (as recommended in Keras, the hyperparameter setting $\lambda = 0.01$ is used in the regularizer). No constraints are applied to the kernels or biases. The activation function is the default ReLu function. The Root Mean Square propagation (RMSprop) optimizer is used with the recommended settings in Keras, i.e., $lr = 0.001$, $\rho = 0.9$, $\epsilon = None$, and $decay = 0.0$. The mini-batch size is probed using a set of commonly used options, $\{8, 16, 32, 64, 128, 256, 512\}$. The best fits, 512 in DevNet and DSVDD, and 256 in FSNet, are used by default. Since REPEN was designed for a similar problem scenario as DevNet, it is used with the recommended optimization settings as in [19].

A.2.3 Packages Used in Our Implementation. The relevant packages and their versions used in our algorithm implementation are listed as follows:

- python==3.6.6
- keras==2.2.4
- keras-applications==1.0.6
- keras-preprocessing==1.0.5
- tensorflow-gpu==1.10.0
- scikit-learn==0.20.0
- numpy==1.14.5
- pandas==0.23.4
- scipy==1.1.0
- tensorboard==1.10.0