Out-of-Distribution Detection for Deep Neural Networks With Isolation Forest and Local Outlier Factor

SIYU LUAN1, ZONGHUA GU2, (Senior Member, IEEE), LEONID B. FREIDOVICH1,2, (Senior Member, IEEE), LILI JIANG3, AND QINGLING ZHAO4

1Department of Applied Physics and Electronics, Umeå University, 90187 Umeå, Sweden
2Department of Information Technologies and AI, Sirius University of Science and Technology, 354349 Sochi, Russia
3Department of Computing Science, Umeå University, 90187 Umeå, Sweden
4College of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing 210094, China

Corresponding author: Qingling Zhao (ada_zhao@njust.edu.cn)

This work was supported in part by the NSFC under Grant 61902185, and in part by Jiangsu Provincial NSF under Grant BK20190448.

ABSTRACT Deep Neural Networks (DNNs) are extensively deployed in today’s safety-critical autonomous systems thanks to their excellent performance. However, they are known to make mistakes unpredictably, e.g., a DNN may misclassify an object if it is used for perception, or issue unsafe control commands if it is used for planning and control. One common cause for such unpredictable mistakes is Out-of-Distribution (OOD) input samples, i.e., samples that fall outside of the distribution of the training dataset. We present a framework for OOD detection based on outlier detection in one or more hidden layers of a DNN with a runtime monitor based on either Isolation Forest (IF) or Local Outlier Factor (LOF). Performance evaluation indicates that LOF is a promising method in terms of both the Machine Learning metrics of precision, recall, F1 score and accuracy, as well as computational efficiency during testing.

INDEX TERMS Out-of-distribution, deep neural networks, runtime monitoring, outlier detection, isolation forest, local outlier factor.

I. INTRODUCTION

Machine Learning (ML), especially Deep Learning (DL) based on Deep Neural Networks (DNNs), has achieved tremendous success in many application domains. Thanks to their excellent performance, DNNs are deployed extensively in today’s safety-critical autonomous systems [1]. There are different approaches to incorporate DL components into the autonomous systems processing pipeline of perception, planning, and control. One approach in industry practice today is to use Convolutional Neural Networks (CNNs) for environment perception tasks, including object classification, object detection, object tracking, and semantic segmentation [2]. The other approach is to use Imitation Learning (IL) or Reinforcement Learning (RL) to train a DNN for End-to-End mapping from pixels to control commands, or half-way mapping from pixels to waypoints as input to subsequent planning and control stages [3].

While DNNs can achieve excellent performance in terms of the ML metrics (precision, recall, F1 score, and accuracy), they are not very interpretable or verifiable. (More interpretable and verifiable ML algorithms, e.g., decision trees, generally cannot achieve the same level of performance as DNNs.) A DNN is a complex, non-convex function that maps from its input (e.g., a set of image pixels) to output (e.g., classification or regression prediction). Its high complexity causes it to work as a black box with limited insight into its internal workings. DNNs are known to lack robustness, i.e., they are often brittle to input variations and may make mistakes in an unpredictable manner, e.g., a well-trained DNN with high accuracy may unpredictably misclassify an object if it is used for perception, or issue unsafe control commands if it is used for planning and control. Their complexity and opaqueness pose significant challenges to high levels of safety certification. Several high-profile fatal traffic accidents involving Autonomous Vehicles (AVs) have been caused by failures of DNNs used for environment perception, e.g., in the 2016 accident that
killed a Tesla driver, the Tesla vehicle’s brake was not applied since the DNN-based object recognition algorithm misclassified the side of a large white truck as the sky. Despite the widespread use of DNNs in autonomous systems, there is a lack of effective and practical Verification and Validation (V&V) techniques for them except extensive testing [4], [5].

One of the methods for achieving safety assurance of DNNs is Out-of-Distribution (OOD) detection for identifying test samples that do not belong to the training set’s data distribution, i.e., outliers/anomalies. The opposite of OOD is In-Distribution (ID), or inliers/normal data. OOD may be due to natural distribution shifts, e.g., if an AV is trained only in clear sunny weather, then adverse weather conditions of heavy fog, rain, or snow may be OOD for its perception system. Or it may be due to adversarial attacks, e.g., minor perturbations, such as stickers added to a Stop Sign to mimic vandalism, may trick the AV perception system into misclassifying it into a Speed Limit sign [6]. When faced with such OOD input samples, DNN classifiers often give incorrect yet over-confident predictions, with potentially serious safety consequences. Instead of blindly trusting the prediction results, the system should be equipped with an OOD detector, and “fail loudly” upon detecting OOD input, so that remedial actions may be taken subsequently to ensure safety, e.g., switching to a simple and fully-verified safety controller, or alerting the driver to take corrective actions.

In this paper, we present a framework for OOD detection for DNNs based on two outlier detection algorithms, Isolation Forest (IF) [7] and Local Outlier Factor (LOF) [8].

The rest of this paper is organized as follows: We discuss related work in Section II; present our approach using IF and LOF in Section III; discuss performance evaluation results in Section IV; give conclusions in Section V.

II. RELATED WORK

Traditional theorem proving and model-checking approaches for hardware/software verification, including theorem proving [9] and model checking [10] are generally not applicable to formal verification of DNNs due to mismatched modeling and property specification formalisms. Figure 1 shows an overview of V&V Techniques for DNNs. They can be broadly categorized into offline, design-time verification, including formal verification and testing; and online, runtime verification, including uncertainty estimation and OOD detection (our focus in this paper). Further details can be read in the recent survey paper [11] and in the report [12], which was published by the European Union Aviation Safety Agency (EASA) and the company Daedalean AG.

Testing for DNNs [5] can help find bugs but cannot guarantee system correctness. Formal verification techniques for DNNs [11], [13] can be broadly categorized into sound-and-complete, and sound-but-incomplete techniques. Sound-and-complete techniques solve the verification problem exactly by encoding it into a Satisfiability Modulo Theories (SMT) or a Mixed Integer Linear Programming (MILP) problem and invoking a solver. Such techniques generally have poor scalability to large DNNs that handle high-dimensional inputs such as images. Sound-but-incomplete techniques are typically based on Abstract Interpretation. They start from a bounded input range, compute a relaxed, conservative approximation of the reachable set layer-by-layer, e.g., encoded with bounding boxes or polyhedra. Such techniques have better scalability but may suffer from false positives due to over-approximation. While formal verification techniques for DNNs can provide strong guarantees, their industry acceptance has been hindered by the limitations of scalability or over-approximation.

Runtime verification [14] is an effective alternative approach to design-time verification for the safety assurance of complex/untrusted components. One runtime approach is uncertainty estimation, which aims to add uncertainty intervals or “error bars” to the prediction results to reflect the level of confidence. Well-known techniques, see e.g. [15] and [16], include Bayesian NN, which works by making multiple non-deterministic forward passes to sample from the distribution of DNN parameters (weights and biases); Monte-Carlo dropout, which works by adding dropout layers and making multiple non-deterministic forward passes to infer the output distribution; and Deep Ensembles, which works by aggregating results from multiple DNNs trained independently on the same training dataset. All of them rely on some form of model ensembles, i.e., multiple diverse models perform inference in parallel or sequentially on the same sample and compare their prediction results. The benefit of uncertainty estimation is that they provide quantitative uncertainty values. Their main drawback is high runtime overhead, e.g., an ensemble of $k$ DNNs cost $k$ times the computing power and memory size to make a prediction; Monte-Carlo dropout [17] typically requires a few hundred stochastic forward passes to get a reasonable estimate of model uncertainty.

An alternative runtime verification approach is OOD detection, highlighted in Figure 1. It solves a binary classification problem of ID vs. OOD, hence it has the potential to be more computationally efficient than uncertainty estimation methods that provide quantitative uncertainty values.

Certain OOD detection methods require OOD data samples for training the OOD detector. Hendrycks et al. proposed...
Outlier Exposer (OE) [18], which leverages a diverse set of OOD samples, and optimizes a loss function that includes an additional term that forces the DNN to output uniform distribution of Softmax scores for an OOD sample. Mohseni et al. proposed Self-Supervised Learning (SSL) [19], which adds additional nodes as reject functions in the last layer of the DNN, and uses a self-supervised approach to train reject functions with free unlabeled OOD samples and the classifier functions with a labeled ID training dataset.

Hendrycks and Gimpel [20] proposed a simple method of using the predicted Maximum Softmax Probability to separate OOD samples (with low MSP values) from ID samples (with high MSP values) based on a threshold, i.e., an input sample is determined to be OOD if its MSP does not exceed the given threshold. This method incurs no additional overhead beyond the forward inference time, but it has low accuracy since DNNs often give incorrect yet overconfident predictions, therefore it is typically used as the comparison baseline for other methods.

Some authors proposed monitoring neuron activations of one or more hidden layers of a DNN for outliers/anomalies as an indication of OOD input [21], [22]. Consider a CNN consisting of multiple layers that progressively extract more abstract and higher-level representations (features), which are used by downstream classification or regression tasks. The last few hidden layers contain the most high-level semantic features extracted by preceding convolutional layers. Since these features have much lower dimensions than the raw input sample in pixels, they are more amenable to outlier/anomaly detection algorithms. (The layer-wise feature extraction is broadly applicable to all types of feedforward DNNs, including CNNs, Multi-Layer Perceptrons (MLPs), and Transformers [23], hence this OOD detection approach is also applicable to all of them.) Typically, the last hidden layer is monitored, but additional layers may be monitored for improved performance at the cost of increased runtime overhead. If a given input is ID, then the activations in the monitored hidden layer(s) should be similar to other ID data seen during training; if the input is OOD, then the activations should form an outlier/anomaly. Cheng et al. [21] proposed to store all possible neuron activation patterns in the last hidden layer with an efficient data structure, Binary Decision Diagrams (BDD) [24]. Assuming the Rectified Linear Unit (ReLU) activation function, a Boolean variable is used to encode the output of each neuron being monitored and is assigned the value of 1 if the neuron is active (output from the ReLU is positive), or the value of 0 if the neuron is inactive (output from the ReLU is 0). During testing, check if a given sample’s activation pattern in the last hidden layer is similar to some stored patterns as measured by Hamming distance. The main drawback of this method is its high runtime overhead in terms of both CPU cycles and memory size, which can easily grow to tens of GB even for relatively small DNNs. Henzinger et al. [22] proposed Box abstraction-based OOD detection. They perform k-means clustering of activations in one or more hidden layers for each class during training, and construct Box abstractions for each combination of class and cluster to encode the lower and upper bounds of all the dimensions of the activation values. If an input sample’s activation in the monitored hidden layers falls outside of all the boxes of all the classes, then it is determined to be OOD. The main drawback of this method is its requirement for labeled ID samples.

In this paper, we propose to monitor one or more hidden layers of a DNN with two different outlier/anomaly detection methods: Isolation Forest (IF) [7] and Local Outlier Factor (LOF) [8]. Both are well-known techniques for anomaly detection [25], but they are typically applied to the input samples directly, while we apply them to neuron activations of one or more hidden layers of a DNN for OOD detection of input samples.

IF and LOF have the following advantages compared to related approaches: (1) They are unsupervised learning algorithms that require neither OOD data nor class labels for ID data during training, hence they can be applied to a pre-trained DNN without access to its training dataset. Even when the training dataset is available, the class labels may be partial or noisy, since labels may be costly to obtain for real applications [26], which motivates the need for semi-supervised learning and active learning [27]. In contrast, the Box method [22] requires the availability of class labels for ID samples to construct the box abstractions for each class; (2) They are non-parametric algorithms that are easy to train, and are tunable with flexible tradeoffs between OOD detection performance and runtime overhead, e.g., by selecting a different set of monitored layers in the DNN, or setting different hyperparameters, e.g., the number and maximum depth of trees in IF, the number of neighbors in LOF, etc. In contrast, The BDD-based method [21] does not have this flexibility, and has high runtime overhead for larger DNNs.

III. OUR APPROACH

A. OVERALL FRAMEWORK

A DNN is a function $F$ that maps from its input $x \in X$ to the corresponding output $y$, which may be a vector of probability values from a Softmax layer in case of classification, or a vector of continuous values in case of regression, as shown in (1)(2):

$$y = F (x; \theta) = f^{(n)} \left( f^{(n-1)} \left( \ldots \left( f^{(1)} (x) \right) \right) \right)$$

$$f^{(n)} (x_{n-1}) = a^{(n)} (w_n \cdot x_{n-1} + b_n)$$

where $f^{(n)}$ denotes the $n$-th layer of DNN $F$; $w_n, b_n$ denote weight matrix and bias vector of the $n$-th layer; $x_{n-1}$ denotes the vector of activation values as output from the $(n-1)$-th layer; $a^{(n)}$ denotes the nonlinear activation function at the $n$-th layer, e.g., ReLU. For simplicity, we adopt a uniform notation for both convolutional and fully-connected layers, since convolutional operations can also be expressed with matrix multiplication.

Figure 2 shows an overview of our proposed OOD detection framework.
Algorithm 1 shows the detailed workflow. Since the last hidden layer contains the most abstract high-level representation, it should always be monitored. In addition, we can often achieve better performance by monitoring multiple layers simultaneously. In this case, we train a separate OOD detector for each layer. For a given input sample \( x \), the activations \( f^\phi(x) \) at the \( \phi \)-th layer are passed to the OOD detector \( d^\phi \) for that layer. If any one of the monitored layers detects OOD, then the input sample \( x \) is determined to be OOD; if none of the monitored layers detects OOD, then \( x \) is determined to be ID. The order in which each layer is checked does not affect the algorithm’s correctness, but may affect the computational efficiency if the sample is OOD, since subsequent layers will be bypassed if any one layer detects OOD. Since the number of neurons in the hidden layers typically decreases going in the direction towards the output layer (left-to-right in Figure 2), we check the layers in the reverse order to maximize efficiency.

**Algorithm 1 OOD Detection With IF or LOF**

```
Input:
\( x \) : an input sample
\( F \) : DNN as defined in (1)(2)
\( \Phi \) : set of monitored layers in \( F \)
\( D \) : set of detectors, one for each layer \( \phi = \Phi \)

Output: ID or OOD

1: for all \( G <I> \) do
2: if \( \equiv \) OOD then
3: Return OOD
4: end if
5: end for
6: Return ID
```

### B. ISOLATION FOREST (IF)

IF [7] is a distance-based anomaly detection algorithm. IF isolates samples by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature to reach the next level of the decision tree. Repeat the process iteratively until the maximum height of a single tree is reached. The number of splits required to isolate a sample is equivalent to the path length from the root node to the terminating node. This path length, averaged over a forest of such random trees, is a measure of normality and our decision function. Outliers will be more easily isolated, hence have shorter paths in the decision tree.

Given a dataset of size \( n \), the anomaly score of a sample data point \( x_p \) is computed by (3) (from [7]):

\[
s(x_p) = 2 - \frac{E(h(x_p))}{c(n)}, \quad c(n) = 2H(n-1) - (2n-1)/n
\]

where \( h(x_p) \) is the path length of a single tree for the input \( x_p \); and \( E(h(x_p)) \) is the average of the path lengths of all trees; \( H(i) \) is the harmonic number estimated with \( H(i) = \ln(i) + 0.5772 \) (Euler’s constant). A higher anomaly score indicates a higher likelihood that the sample is an anomaly.

**Figure 3. A cluster \( C_1 \) and one outlier \( o_1 \), detected by IF.**

Figure 3 shows an example dataset with an outlier sample \( o_1 \). The number of random splits required to isolate \( o_1 \) is \( h(o_1) = 3 \) (assuming the left vertical split is selected before the other two splits), which should be much smaller than the number of splits required to isolate any sample within the cluster \( C_1 \). Hence the outlier \( o_1 \) is assigned a high anomaly score.

### C. LOCAL OUTLIER FACTOR (LOF)

LOF [8] is a density-based anomaly detection algorithm. It computes the local density of a given sample and compares it with the local density of its \( k \) nearest neighbors. The samples with much lower density than their neighbors are considered outliers. The number of neighbors considered (parameter \( k \), called \( n_{\text{neighbors}} \) in scikit-learn [28]) is an important hyperparameter. Here are the main steps of the LOF algorithm (based on the descriptions in [29]):

1) \( k \)-DISTANCE

For a given sample \( x_p \in X \), \( p = 1, 2, \ldots, N \), the \( k \)-distance \( d_k(x_p) \) of \( x_p \) is defined as: there are at least \( k \) points \( x_q \in X \setminus \{x_p\} \) that satisfy \( d(x_p, x_q) \leq d_k(x_p) \); and at most \( k-1 \) points of \( x_q \in X \setminus \{x_p\} \) that satisfy \( d(x_p, x_q) < d_k(x_p) \). (\( X \setminus \{x_p\} \) denotes the set of elements in set \( X \) excluding \( x_p \).) Such measure defines the way we locate neighbor points for a given sample \( x_p \).
2) k-DISTANCE NEIGHBORHOOD

Given \(d_k(x_p)\), the \(k\)-distance neighborhood \(N_k(x_p)\) of \(x_p\) contain samples that satisfy:

\[
N_k (x_p) = \{ x_q \in X \setminus \{x_p\} \mid d(x_p, x_q) \leq d_k(x_p) \} \tag{4}
\]

3) REACHABILITY DISTANCE

The reachability distance between any two points \(x_p\) and \(x_q\) is:

\[
rd_k(x_p, x_q) = \max \{d_k(x_q), d(x_p, x_q)\} \tag{5}
\]

4) LOCAL REACHABILITY DENSITY

The local reachability density is defined as the inverse of average reachability density based on the local neighborhood:

\[
lrd_k (x_p) = \frac{1}{|N_k (x_p)|} \sum_{x_q \in N_k (x_p)} rd_k (x_p, x_q)^{-1} \tag{6}
\]

5) LOCAL OUTLIER FACTOR

The LOF score (the anomaly score) of each sample, is defined as the ratio of the average local density of its \(k\)-nearest neighbors, and its own local density, computed by (7). An input \(x_p\) with a high LOF score has a lower density than its neighbors, hence is more likely to be an outlier.

\[
s(x_p) = LOF_k(x_p) = \frac{1}{|N_k (x_p)|} \sum_{x_q \in N_k (x_p)} lrd_k (x_q) \tag{7}
\]

\(LOF_k(x_p) < 1\) means \(x_p\) has higher density than its neighbors (likely to be an inlier); \(LOF_k(x_p) > 1\) means \(x_p\) has lower density than its neighbors (likely to be an outlier).

Figure 4 shows the same dataset as Figure 3. Assume parameter \(k=3\). Each circle centered at the sample \(x_p\) denotes the \(k\)-distance neighborhood of \(x_p\). We can see that the outlier sample \(o_1\) has a large \(k\)-distance, i.e., it is far away from its \(k=3\) neighbor points \(x_q\) (with purple color) within the large circle. To compute the LOF score of \(o_1\), \(LOF_k(o_1)\), we first compute the reachability distances from \(o_1\) to each of its 3 neighbor points \(rd_k(o_1, x_q)\) with (5) (which are all quite large), then compute its local reachability density \(lrd_k(o_1)\) with (6) (which is quite low). We then consider each of its 3 neighbor points \(x_q\) in turn, and compute its local reachability density \(lrd_k(x_q)\) in the same way (which are all quite high). Finally, we compute the LOF score \(LOF_k(o_1)\) with (7) (which is quite large), and determine \(o_1\) to be an outlier.

D. LOCAL VS. GLOBAL DETECTION

There are two categories of outlier detection algorithms: global vs. local detection [30]. Global outlier detection considers all samples, and the sample \(pt\) is considered an outlier if it is far away from all other samples. Local outlier detection covers a small subset of samples at a time. A local outlier is based on the probability of sample \(pt\) being an outlier as compared to its local neighborhood, determined with the \(k\)-Nearest Neighbors (kNN) algorithm in LOF. As an example, Figure 5 shows two clusters \(C_1\) and \(C_2\), where \(C_2\) has a higher density than \(C_1\); and three outliers, where \(o_1\) and \(o_2\) are global outliers, and \(o_3\) is a local outlier to \(C_2\). IF is sensitive to global outliers but weak in dealing with local outliers, whereas LOF can detect both global and local outliers well [31]. LOF takes into consideration both local and global properties of datasets and checks not just how isolated the sample is, but how isolated it is with respect to its surrounding neighborhood [28], hence it is able to detect the local outlier \(o_3\), despite the different densities of the two clusters \(C_1\) and \(C_2\). We believe this partially explains the better performance of LOF compared to IF in this paper.

IV. PERFORMANCE EVALUATION

A. DATASETS AND SETUP

We use the same dataset and experimental setup as the BDD-based method [21] and the Box-based method [22]. We use two well-known datasets, namely MNIST (handwritten digit recognition dataset with 10 classes) [32] and GTSRB (German Traffic Sign Recognition Benchmark dataset with 43 classes) [33]. We use the two DNN architectures shown in Table 1, for the two DNNs considered: NN1 for the MNIST dataset, and NN2 for the GTSRB dataset. The convolutional layers Conv(\cdot) have kernel size \(5 \times 5\) and unit stride, with the number of filters shown in parentheses; MaxPool denotes \(2 \times 2\) max-pooling layer; fc(\cdot) denotes Fully-Connected layer, with the number of neurons shown in parentheses; BN(\cdot) denotes Batch Normalization; ReLU denotes the ReLU activation function. We use the Tensorflow framework for training these DNNs, and the training setup and classification accuracy are the same as in [22] since we
build on top of their source code. All training and testing experiments are run on a workstation with an AMD Ryzen 2990WX CPU with 32 cores at frequency 3.0 GHz; 128 GB memory; and two GPUs (NVIDIA GeForce RTX 2080 Ti).

We emulate OOD samples with samples of one or more classes that are not included in the training dataset. Suppose the standard training dataset consists of labeled samples of \( n \) classes. The actual training dataset is the subset of the standard training dataset consisting of samples of \( k \) classes as the ID data. We exclude samples of the other \((n-k)\) classes from the training dataset, in order to use them as artificial OOD samples during testing. The test dataset is the same as the standard test dataset, and contains samples of all the \( n \) classes, both ID and OOD. For example, for the MNIST dataset: during training, we use samples (images) of 9 classes (digits 0 to 8) in the standard training dataset as the ID samples. During testing, the test dataset consists of 10,000 samples, including 8,991 ID samples (with ground-truth labels of digits 0 to 8) and 1,009 OOD samples (with ground-truth label of digit 9). For the GTSRB dataset: during training, we use samples of 20 classes (classes 0 to 19) in the standard training dataset as the ID samples. During testing, the test dataset consists of 12,630 samples, including 8,730 ID samples (with ground-truth labels of classes 0 to 19) and 3,900 OOD samples (with ground-truth labels of classes 20 to 42). With these settings, the two DNNs used in our experiments have different last layer sizes from the standard topologies in Table 1, i.e., NN\(_1\) has the last layer fc(9), and NN\(_2\) has the last layer fc(20). (This is an artifact of our experimental setup due to the need to create artificial OOD data with selected classes; the standard topologies in Table 1 should be adopted for actual training and deployment.)

Table 2 shows the definition of the confusion matrix. Note the difference from the definition of ID and OOD in [22], which determines a sample to be ID if the classifier predicts its correct class label, whereas we determine a sample to be ID or OOD based on its ground-truth class label. We have adopted our definition in the experiments, so our performance numbers are slightly different from theirs for the same methods. The numerical differences are small since the DNN classifiers for both datasets have high accuracy.

We use the standard classification performance metrics precision, recall, F1 score, and accuracy to evaluate the performance of OOD detectors. Different application domains may place more importance on different metrics, e.g., for medical imaging, high recall is preferred since false negatives may have severe consequences to patient health, and false positives can be resolved later by experienced professionals. But for real-time autonomous systems that must make online decisions, excessive false positives may be distracting and disruptive to system operation. As the harmonic mean of the precision and recall, the F1 score is typically more appropriate as the overall metric for evaluating classifiers, since it takes into account both factors.

\[
\text{Precision} = \frac{TP}{TP + FP} \quad (8)
\]
\[
\text{Recall} = \frac{TP}{TP + FN} \quad (9)
\]
\[
\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (10)
\]
\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (11)
\]

### B. EXPERIMENTAL RESULTS

We choose three comparison baselines: Softmax-based [20], BDD [21], and Box [22], which are the most relevant and similar works, since they are all based on the monitoring of hidden layers of a DNN.) For the Softmax-based method, the threshold value is set to be 0.99 or 0.9. For the BDD-based method, the Hamming distance threshold for judging pattern similarity is set to be 0. In terms of the monitored layers: For the BDD-based method, we always monitor the last hidden layer only, as specified in [21]. For the Box-based method, IF and LOF, we vary the number of monitored layers. (As a result, the performance metrics of Box, IF and LOF vary across different subfigures in Figures 6 and 7, whereas the metrics of Softmax and BDD stay the same.)

Important hyperparameters for IF include the following: 1) contamination: the proportion of outliers in the data set. It can be automatically determined as in the original paper on IF [7] or set to be in the range \([0, 0.5]\). The threshold on the anomaly score, called the offset in scikit-learn, is defined in such a way that we obtain the expected contamination in training; 2) max_samples: The number of samples to draw from the training dataset to train each base estimator. If max_samples exceeds the number of samples provided, all samples will be used for all trees (no sampling); 3) n_estimators: The number of base estimators (trees) in the ensemble (forest). Important hyperparameters for LOF include the following: 1) contamination: the proportion of outliers in the data set. 2) n_neighbors: the parameter \( k \) used to compute the average local density of a sample’s \( k \)-nearest neighbors.

| Table 1. Two DNN architectures used for performance evaluation. |
| --- |
| Architecture | Dataset |
| ReLU(Conv(40)), MaxPool, ReLU(Conv(20)), MaxPool, ReLU(fc(320)), ReLU(fc(160)), ReLU(fc(80)), ReLU(fc(40)), fc(10) | MNIST |
| ReLU(BN(Conv(40))), MaxPool, ReLU(BN(Conv(20))), MaxPool, ReLU(fc(240)), ReLU(fc(84)), fc(43) | GTSRB |

| Table 2. Confusion matrix for OOD detection. |
| --- |
| Predicted OOD | Predicted ID |
| Actual OOD | TP (True Positive) | FN (False Negative) |
| Actual ID | FP (False Positive) | TN (True Negative) |
Since the number of hyperparameters is relatively small, we use grid search to find the optimal hyperparameter configurations for each model and each set of monitored layers, instead of more sophisticated techniques such as Bayesian Optimization [34]. Some example hyperparameter settings are as follows: for NN1 for the MNIST dataset with 4 monitored layers: contamination for both IF and LOF is 0.01; max_samples for IF is $2^{14}=16,384$; n_estimators for IF is 500; n_neighbors for LOF is 100. For NN2 for the GTSRB dataset with 1 monitored layer: contamination for both IF and LOF is 0.03; max_samples for IF is $2^{15}=32,768$; n_estimators for IF is 300; n_neighbors for LOF is 20.

We make the following observations from Figures 6 and 7:

1. For the MNIST dataset, all methods achieve relatively high accuracy exceeding 90%. This is because accuracy is not a very discriminative performance metric for unbalanced datasets, where the vast majority of samples in the test dataset have the same label, e.g., only $\sim$10% of the test dataset are OOD samples. For example, a useless classifier that always predicts ID for all samples will get 90% accuracy (similar to IF in Figure 6(a)). This is not true for the GTSRB dataset, with $\sim$31% of OOD samples in the test dataset, where LOF achieves significantly higher accuracy than the other methods.

2. For both datasets, LOF consistently outperforms the other methods in terms of the F1 score, even though it does not always have the highest precision or highest recall. IF generally has worse performance than LOF, especially if only one hidden layer is monitored (Figures 6(a) and 7(a)). We attribute the better performance of LOF compared to IF to its better capability of detecting local outliers, which may be more prevalent than global outliers in the distributions of the activation values in the hidden layers.

3. Performance of OOD detection generally improves with increasing number of monitored layers, but the improvement is not significant for more than two layers, and the performance may even degrade with too many monitored layers. Since runtime overhead increases with the number of monitored layers, the designer should choose an appropriate number of monitored layers that achieves the best performance without too much runtime overhead, e.g., two layers seem to be the best choice for both NN1 and NN2.
We carried out an additional set of experiments of monitoring neurons in multiple layers as a single group with IF or LOF, and the results in Figure 8 indicate that this approach has much worse performance than our layer-wise monitoring approach in Algorithm 1. This justifies the layer-wise monitoring approach.

Table 3 shows the algorithm running times for training and testing of IF and LOF models with different numbers of monitored layers, and the times for building and testing the BDD that monitors the last hidden layer only (the BDD is built with a deterministic algorithm, not trained with a learning algorithm). The testing times are measured for the entire test dataset as one batch (10,000 samples for MNIST, and 12,630 for GTSRB). The BDD-based method has comparable testing time for NN$_1$ for MNIST, but a very long testing time for NN$_2$ for GTSRB, since the BDD data structure grows rapidly with the number of monitored neurons. This is not surprising, since the total number of possible activation patterns grows exponentially with the number of monitored neurons. Even though BDD is an efficient data structure, its size is highly dependent on the variable ordering when constructing the BDD [24], and in the worst-case, may also grow exponentially with the number of monitored neurons. It is obvious that the more layers are monitored, the longer both the training and testing times become, since a separate IF or LOF model is trained for each monitored layer. For the same number of monitored neurons, the running times for IF and LOF models with different numbers of monitored layers are shown in Table 3.

| # Monitored Layers | NN$_1$ for MNIST | NN$_2$ for GTSRB |
|--------------------|------------------|------------------|
| IF (training)      | 67               | 41               |
| IF (testing)       | 2484             | 1677             |
| LOF (training)     | 105              | 58               |
| LOF (testing)      | 31               | 63               |
| BDD (building)     | 257              | N/A              |
| BDD (testing)      | 40               | 742              |

Figure 7. OOD detection performance for the GTSRB dataset, with a varying number of monitored layers for Box, IF, and LOF.

Table 3. Running time for training and testing (time unit: second).
layers, the training time for LOF is slightly higher than IF; but the testing time for LOF is significantly lower than IF (1-2 orders of magnitude). Since testing time determines the runtime overhead, it is more important than training time for real-time autonomous systems, hence we conclude that LOF is more suitable than IF for deployment in such systems.

We measured the running times of the software implemented in Python on a powerful workstation. Furthermore, Python is a slow interpreted language, whereas actual deployment on the target embedded platform is likely to be in an efficient compiled language such as C. Hence the absolute magnitudes of the performance numbers in Table 3 are not very informative, and they are only intended for relative comparisons between the computational efficiencies of IF and LOF.

V. CONCLUSION

Since OOD samples may cause DNNs to make incorrect predictions, it is important to provide accurate and efficient algorithms for OOD detection for the safe deployment of DNNs in safety-critical systems. In this paper, we propose to perform OOD detection in a DNN by monitoring one or more of its hidden layers with two well-known outlier detection methods, Isolation Forest (IF) and Local Outlier Factor (LOF), and compare performance with closely-related works, including the Softmax-based method [20], the Box abstraction-based method [22], and the BDD-based method [21], for two well-known datasets, including MNIST and GTSRB. Performance evaluation demonstrates the effectiveness of LOF in terms of both the ML metrics of precision, recall, F1 score, and accuracy, as well as their computational efficiency during testing. Although IF is shown to be less effective than LOF for the case studies considered in this paper, the same conclusion may not generalize to other DNN models or other applications, hence we consider both to be promising techniques to be kept in the engineer’s toolbox. As part of future work, we plan to apply our OOD detection techniques to application case studies in safety-critical application domains ranging from autonomous driving [2] to medical imaging [35].

REFERENCES

[1] Y. Tang, C. Zhao, J. Wang, C. Zhang, Q. Sun, W. Zheng, W. Du, F. Qian, and J. Kurths, “An overview of perception and decision-making in autonomous systems in the era of learning,” 2020, arXiv:2001.02319. [Online]. Available: http://arxiv.org/abs/2001.02319
[2] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, “A survey of deep learning techniques for autonomous driving,” J. Field Robot., vol. 37, no. 3, pp. 362–386, Apr. 2020.
[3] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Perez, “Deep reinforcement learning for autonomous driving: A survey,” IEEE Trans. Intell. Transp. Syst., early access, Feb. 9, 2021, doi: 10.1109/TITS.2021.3054625.
[4] Q. Zhu, C. Huang, R. Jiao, S. Lan, H. Liang, X. Liu, Y. Wang, Z. Wang, and S. Xu, “Safety-assured design and adaptation of learning-enabled autonomous systems,” in Proc. 26th Asia South Pacific Design Autom. Conf., Jan. 2021, pp. 753–760.
[5] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, “Machine learning testing: Survey, landscapes and horizons,” IEEE Trans. Softw. Eng., early access, Feb. 17, 2020, doi: 10.1109/TSE.2019.2962027.
[6] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, “Robust physical-world attacks on deep learning visual classification,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 1625–1634.
[7] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in Proc. 8th IEEE Int. Conf. Data Mining, Dec. 2008, pp. 413–422.
[8] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “LOF: Identifying density-based local outliers,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2000, pp. 93–104.
[9] M. S. Nawaz, M. Malik, Y. Li, M. Sun, and M. I. U. Lali, “A survey on theorem provers in formal methods,” 2019, arXiv:1912.03028. [Online]. Available: http://arxiv.org/abs/1912.03028
[10] C. Baier and J.-P. Katoen, Principles of Model Checking, Cambridge, MA, USA: MIT Press, 2008.
[11] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, “Algorithms for verifying deep neural networks,” 2019, arXiv:1903.06758. [Online]. Available: http://arxiv.org/abs/1903.06758.
[12] J. M. Cluzeau, X. Henriquel, G. Rebender, G. Soudain, L. van Dijk, A. Gronskiy, D. Haber, C. Perret-Gentil, and R. Polak, “Concepts of design assurance for neural networks (CoDANN),” Eur. Union Aviation Saf. Agency, Daedalean AG, Zürich, Switzerland, Tech. Rep. v1, 2020.
[13] H.-D. Tran, W. Xiang, and T. T. Johnson, “Verification approaches for learning-enabled autonomous cyber-physical systems,” IEEE Des. Test. Comput., early access, Aug. 12, 2020, doi: 10.1109/MDAT.2020.3015712.
[14] Q. M. Rahman, P. Corke, and F. Dayoub, “Run-time monitoring of machine learning for robotic perception: A survey of emerging trends,” IEEE Access, vol. 9, pp. 20067–20075, 2021.
[15] D. Feng, A. Harakeh, S. Waslander, and K. Dietmayer, “A review and comparative study on probabilistic object detection in autonomous driving,” 2020, arXiv:2011.10671. [Online]. Available: http://arxiv.org/abs/2011.10671
[16] M. Henne, A. Schweiger, K. Roscher, and G. Weiss, “Benchmarking uncertainty estimation methods for deep learning with safety-related metrics,” in Proc. SafeAI AAAI, 2020, pp. 83–90.

[17] R. Michelmore, M. Kwiatkowska, and Y. Gal, “Evaluating uncertainty quantification in end-to-end autonomous driving control,” 2018, arXiv:1811.06817. [Online]. Available: http://arxiv.org/abs/1811.06817

[18] D. Hendrycks, M. Mazeika, and T. Dietterich, “Deep anomaly detection with outlier exposure,” 2018, arXiv:1812.04506. [Online]. Available: http://arxiv.org/abs/1812.04506

[19] S. Mohseni, M. Pitala, J. Yadawa, and Z. Wang, “Self-supervised learning for generalizable out-of-distribution detection,” in Proc. AAAI Conf. Artif. Intell., 2020, vol. 34, no. 4, pp. 5216–5223.

[20] D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” 2016, arXiv:1610.02136. [Online]. Available: http://arxiv.org/abs/1610.02136

[21] C.-H. Cheng, G. Nuhrenberg, and H. Yasuoka, “Runtime monitoring neuron activation patterns,” in Proc. Design, Autom. Test Eur. Conf. Exhib. DATE, Mar. 2019, pp. 300–303.

[22] T. A. Henzinger, A. Lukina, and C. Schilling, “Outside the box: Abstraction-based monitoring of neural networks,” 2019, arXiv:1911.09032. [Online]. Available: http://arxiv.org/abs/1911.09032

[23] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16 × 16 words: Transformers for image recognition at scale,” 2020, arXiv:2010.11929. [Online]. Available: http://arxiv.org/abs/2010.11929

[24] R. E. Bryant, “Symbolic Boolean manipulation with ordered binary-decision diagrams,” ACM Comput. Surv., vol. 24, no. 3, pp. 293–318, 1992.

[25] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” ACM Comput. Surv., vol. 41, no. 3, pp. 1–58, Jul. 2009.

[26] (Accessed: 2021). Data Labelling Pricing—Google AI Platform. [Online]. Available: https://cloud.google.com/ai-platform/data-labeling/pricing

[27] J. Zhu, Y. Wang, D. Zhou, and F. Gao, “Batch process modeling and decision recognition benchmark: A multi-class classification competition,” in Proc. SafeAI AAAI, 2018, pp. 1453–1460.

[28] J. E. van Engelen and H. H. Hoos, “A survey on semi-supervised learning,” ACM Comput. Surv., vol. 41, no. 3, pp. 1–58, Jul. 2009.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, “Scikit-learn: Machine learning in Python,” J. Mach. Learn. Res., vol. 12, pp. 2825–2830, Oct. 2011.

[30] T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, T. Nowotny, A. Kindermans, S. Gouws, and S. Gouws, “DeBERTa: Pre-training学术 models by fine-tuning,” 2020, arXiv:2010.11929. [Online]. Available: http://arxiv.org/abs/2010.11929

[31] L. Deng, “The MNIST database of handwritten digit images for machine learning research [best of the web],” IEEE Signal Process. Mag., vol. 29, no. 6, pp. 141–142, Nov. 2012.

[32] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “The German traffic sign recognition benchmark: A multi-class classification competition,” in Proc. Int. Joint Conf. Neural Netw., Jul. 2011, pp. 1453–1460.

[33] J. Snook, H. Larochelle, and R. P. Adams, “Practical Bayesian optimization of machine learning algorithms,” in Proc. Adv. Neural Inf. Process. Syst., vol. 25, 2012, pp. 1–9.

[34] X. Li, Y. Lu, C. Desrosiers, and X. Liu, “Out-of-distribution detection for skin lesion images with deep isolation forest,” in Proc. Int. Workshop Mach. Learn. Med. Imag. Cham, Switzerland: Springer, 2020, pp. 91–100.

SIVU LUAN is currently pursuing the Ph.D. degree with the Department of Applied Physics and Electronics, Umeå University. His research interests include cyber-physical systems and machine learning.

ZONGHUA GU (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, in 2004. He is currently a Professor with the Department of Applied Physics and Electronics, Umeå University, Sweden. His research interests include cyber-physical systems, real-time embedded systems, and machine learning.

LEONID B. FREIDOVICH (Senior Member, IEEE) received the M.Sc. degree in mechanics and engineering and the candidate degree in physical and mathematical sciences from Saint-Petersburg State Technical University, St. Petersburg, Russia, in 1996 and 1999, respectively, and the Ph.D. degree in mathematics from Michigan State University, East Lansing, in 2005. Since 2005, he has been with Umeå University, Umeå, Sweden, where he is currently an Associate Professor in control systems with the Department of Applied Physics and Electronics and has been appointed as a Docent in control systems, since 2010. He is a Co-Founder of RobotikUm AB. His research interests include nonlinear control systems and robotics. He has served as an Associate Editor for the Conference Editorial Board of the IEEE Control Systems Society, from 2015 to 2019. Since 2018, he has been an Associate Editor of IEEE TRANSACTIONS ON AUTOMATIC CONTROL.

LILI JIANG received the Ph.D. degree in computer science from Lanzhou University, China, in 2012. She is currently an Associate Professor with the Department of Computing Science, Umeå University, Sweden, and leading the Deep Data Mining Research Group. Before joining Umeå University, she was a Research Scientist at NEC Laboratories Europe, Germany, and a Postdoctoral Researcher with the Department of Databases and Information Systems, Max-Planck-Institut für Informatik, Saarbrücken, Germany. She has been dedicating to address academic challenges motivated by real applications by applying state-of-the-art data science techniques and exploring novel solutions. Her research interests include text mining, information retrieval, data fusion, natural language processing, machine learning, and privacy preservation.

QINGLING ZHAO received the Ph.D. degree from Zhejiang University, China, in 2015. She is currently an Associate Professor with Nanjing University of Science and Technology. Her current research interests include real-time systems and cyber-physical systems.

***