Are Existing Out-Of-Distribution Techniques Suitable for Network Intrusion Detection?

Andrea Corsini
Department of Science and Methods for Engineering
University of Modena and Reggio Emilia, Modena, Italy
andrea.corsini@unimore.it

Shanchieh Jay Yang
Department of Computer Engineering
Rochester Institute of Technology, Rochester, USA
jay.yang@rit.edu

***Abstract*—Machine learning (ML) has become increasingly popular in network intrusion detection. However, ML-based solutions always respond regardless of whether the input data reflects known patterns, a common issue across safety-critical applications. While several proposals exist for detecting Out-Of-Distribution (OOD) in other fields, it remains unclear whether these approaches can effectively identify new forms of intrusions for network security. New attacks, not necessarily affecting overall distributions, are not guaranteed to be clearly OOD as instead, images depicting new classes are in computer vision. In this work, we investigate whether existing OOD detectors from other fields allow the identification of unknown malicious traffic. We also explore whether more discriminative and semantically richer embedding spaces within models, such as those created with contrastive learning and multi-class tasks, benefit detection. Our investigation covers a set of six OOD techniques that employ different detection strategies. These techniques are applied to models trained in various ways and subsequently exposed to unknown malicious traffic from the same and different datasets (network environments). Our findings suggest that existing detectors can identify a consistent portion of new malicious traffic, and that improved embedding spaces enhance detection. We also demonstrate that simple combinations of certain detectors can identify almost 100% of malicious traffic in our tested scenarios.**

I. Introduction

Network Intrusion Detection Systems (NIDS) monitor the network traffic for signs of potential threats with various techniques, including signature detection, anomaly detection, and behavioral analysis [1], [2]. Network traffic can be analyzed either at a Packet Capture or Network Flow [3] (NetFlow) level, though packet inspection has become less common due to encryption and the massive size of modern traffic. We focus on NetFlow inspection, where packets relating to a single communication [3] are analyzed by measuring aggregated features such as idle times and the amount of exchanged data.

Recently, Machine Learning has gained popularity in NIDS [4] as it enables automatic extraction of complex detection patterns, quick adaptation to changing environments [5], and easy personalization without expensive human expertise. However, ML-based solutions have limitations such as lacking interpretability and requiring well-crafted training data. Although a large body of research is addressing these issues [4], we focus on another drawback: *ML-based NIDSs always provide a response regardless of whether they recognize (are trained with) the input data pattern*.

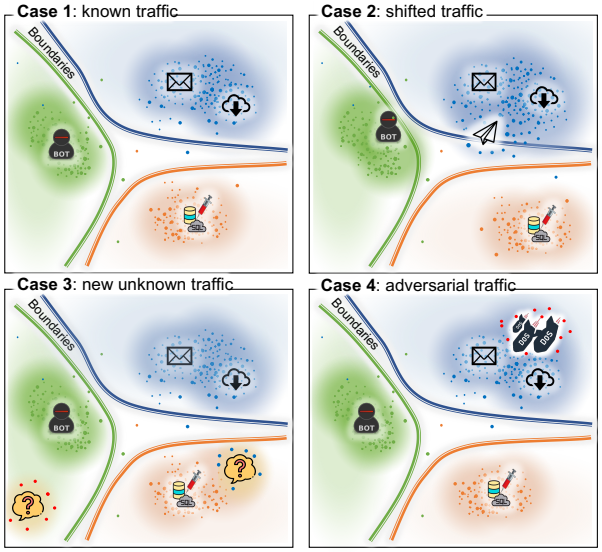

Fig. 1. Different situations in the decision space of a deployed ML-based NIDS. Case 1 is the expected situation where the new traffic respects the i.i.d. assumption. Case 2 depicts traffic that is (gradually) shifting due to changes in malicious and benign behaviors towards relatively known (usual) regions. Case 3 shows new unknown traffic (of potentially different classes) falling into unusual regions of the decision space. Case 4 describes a challenging situation where a new attack is crafted so that it is misclassified by the NIDS.

This issue is particularly relevant since network traffic tends to have *dynamic and non-stationary distributions*, either caused by normal behavioral shifts or adversaries, which can cause degradation in NIDS performance: a problem known as *concept drift* [6]. Another inherent problem of dynamic distributions is Out-Of-Distribution data, which is *unusual traffic markedly different from a reference distribution* not necessarily affecting the overall data distribution. In general, concept drift and OOD data are both caused by shifts in feature distributions, label distributions, or both [6], [7].

Based on our analysis, an ML-based NIDS may be affected in various ways after deployment. In Fig. 1, we present 4 exemplar cases in a NIDS trained to detect Botnet and SQL injections. Normally, the NIDS is expected to work as in Case 1, where new traffic is well represented by training data (i.i.d. assumption). However, it is normal for traffic to shift over time and this may affect NIDSs depending on the shift extent and

direction. For instance, the shift to SQL traffic in Case 2 does not compromise the NIDS, but the same is not true for benign traffic. Moreover, new traffic may also be adversarially crafted (Case 3) and be potentially mistaken as in Case 4.

We argue that shifts as those in Fig. 1 happen more in NetFlow features (*covariate shift* [7], [8]) rather than in labels (*actual or semantic shift* [6], [7]). As in Case 2, malicious traffic remains malicious if its features are adversarially crafted to evade detection, while shifts in normal user behaviors should not transform benign traffic into malicious. Even in cases of new attacks (Case 3), it might be possible to experience shifts in feature distributions [7]. Additionally, we argue that OOD techniques sensitive to small feature perturbations can also serve as drift detectors by monitoring the volume of alerts over time either with standard statistics or existing methods [6], [8], [9]. Therefore, in this work, we adapt and evaluate techniques to detect shifts primarily affecting features.

A perfect OOD detector should trigger an alert and ask the expert knowledge for further investigation in every situation but Case 1. However, Case 4 is extremely difficult to detect without any additional information besides the ML-based NIDS and training data. Whereas well-designed OOD detectors should identify cases like 2 and 3. We thus investigate whether traffic generated by new attacks, either similar to those in training or completely different, can be detected as OOD by existing techniques from other ML fields. Note that it is not guaranteed that effective techniques in other fields are suitable for network intrusion. As an example, well-working detectors on data like images with bounded and discrete domains might prove ineffective on NetFlows, where features are generally a mix of continuous unbounded and discrete.

Therefore, we select detection techniques of different natures from other ML fields and evaluate whether such techniques can identify NetFlows of unknown attacks. As baseline model, we consider a standard FeedForward Neural Network, and we also assess the effect of different training regimes on the quality of detection techniques. Specifically, we train models in *binary* (different attacks in the same class) and *multi-class* (each attack makes a class) settings, with and without the aid of a simple Contrastive Learning approach: *Center-Loss* [10]. We expose various combinations of models and detection techniques to malicious traffic generated from attack types not seen in training, where such traffic may come from the same dataset (same network environment) and a different dataset (different network environment). Finally, we evaluate two ensembles of OOD techniques to enhance detection and further explore the complementarity of these techniques, providing guidelines for practical applications. All our code and the numerical results are freely available at https://github.com/AndreaCorsini1/CyberOOD

The contributions of this paper include:

- We investigate the effectiveness of treating the identification of unknown intrusions as an OOD detection problem and explore the applicability of existing OOD techniques.
- We identify the most effective techniques for detecting new intrusions and explore their potential for combination to enhance detection. We also discuss limitations of some techniques, providing insights for further development.
- We emphasize the significance of improving the model embeddings to achieve better detection, highlighting that:
  - *Contrastive Learning*, specifically the use of *Center-Loss*, enables the creation of embeddings that improve OOD techniques and their ensemble.
  - *Multi-class* training allows making semantically richer embeddings, which offer advantages over binary ones for OOD techniques and their ensembles.

The remainder is organized as follows: Sec. II presents existing OOD literature; Sec. III describes key concepts for our work; Sec. IV outlines our methodology; Sec. V describes the experimental setup; Sec. VI presents results; and Sec. VII closes with limitations and potential future directions.

II. Related Works

In this section, we present various Out-Of-Distribution detection techniques and we review recent proposals to identify and react to shifts in the NIDS literature.

A. Out-Of-Distribution in Machine Learning

Machine learning models are trained under the closed-world assumption, where test data is drawn i.i.d. from the same distribution as the training data. However, this assumption is often violated and several ML fields try to address the issue of identifying unknown/anomalous/out-of-distribution data:

- *Anomaly detection* [11] aims to detect anomalous inputs that deviate from normality, whether in features or labels. Anomaly detection assumes there might be abnormal data in the training set [12] and treat data as a whole, thus it does not strictly require the correct classification of inputs.
- *Novelty detection* is similar to anomaly detection, but assumes the presence of only normal data in the training set and focuses on inputs affected by semantic shift [7], hence not falling into any of the training classes. In addition, novel inputs are not treated as erroneous and are typically prepared for retraining and future constructive procedures.
- *Open Set Recognition* [13] goes beyond novelty detection and also requires the correct classification of in-distribution (ID) data. The goal is to detect inputs belonging to new classes and correctly classify those from known classes. Open Set Recognition is usually focused on semantic shifts.
- *Outlier detection* identifies inputs in a dataset that markedly differ from others. Outlier detection is a pre-processing step and is not applied during inference or training.

As introduced in Sec. I, the NIDS setting requires the classification of known traffic and the detection of shifts caused either by modifications in known traffic or the appearance of unknown traffic. This setting resembles the Open Set Recognition one, but it additionally comprises shifts not implying the appearance of new classes. Therefore, we speak of Out-Of-Distribution detection in general terms.

Confidence-based detectors use estimates derived from a model to quantify the level of certainty or trust in its predictions as an indicator of ID-ness. In [14], the authors observed

that well-trained models assign lower confidence scores to OOD data. Subsequent studies [15]–[17] have proposed techniques to enhance confidence estimation, while others have introduced modifications to the model architecture and training objectives [18], [19]. Although confidence is not always a reliable OOD indicator [20], [21], due to their simplicity and clarity, confidence-based detectors are commonly used in practice and serve as a baseline for OOD detection.

Density-based detectors explicitly model the distribution of ID data, either raw or latent features, and flag samples falling into low-density regions as OOD. In multi-class tasks, class-conditional distribution estimators are often employed so that the OOD samples can be identified based on their likelihood [22], [23]. To model the class-conditional distribution of ID data, parametric and non-parametric models such as a simple Mixture of Gaussian, Kernel Density Estimation, and deep generative models [7] are frequently used. However, modeling the distribution of complex data and estimating the likelihood may be challenging [23], imply a-priori assumptions that need validation, and do not always scale well like in kernel estimators. Therefore, we prefer to avoid these detectors and leave their evaluation to future work.

Distance-based detectors are based on the idea that the OOD samples should be relatively far away from centroids or prototypes of ID classes. Once a prototype is extracted for each training class, a distance metric like Mahalanobis, Euclidean, or Cosine can be used to estimate the class similarity and flag samples that are not similar enough to any of the prototypes [7], [22]. Recently, even a class-conditioned K-Nearest Neighbor approach [24] has been adopted to detect OOD samples based on the distance from the k-nearest neighbor.

B. Out-Of-Distribution in Network Intrusion

A significant portion of the network intrusion literature on ML applications focuses on anomaly detection [25], [26] and concept drift [6], [27]–[29]. Anomaly techniques, such as autoencoders [30], have gained interest due to their ability to detect unknown attacks using only normal traffic and without requiring labels. However, these methods often suffer from a high number of false alarms as they flag any anomalous sample as an attack [4]. In contrast, concept drift and OOD detectors are generally more effective but typically require labeled data [6], [7]. Therefore, recent works proposed ML-based solutions that ease the need for labels without increasing false alarms by leveraging anomaly detection techniques. For instance, [31] proposed an efficient and online ensemble of autoencoders that utilizes an ad-hoc feature extraction module to differentiate normal and abnormal patterns in packets. Similarly, [31] introduced an adaptive ensemble system that incorporates a packet-based feature extraction method and a sub-classifier generation module to create ensemble models from drifted data chunks and ground truth labels. [32] modified the extreme gradient boosting algorithm to detect and adapt to drifts in the presence of a large number of features. [29] employed active learning, label estimation, and an explainable ML framework to respectively update the model, reduce labeling overhead, and interpret model reactions to shifts.

In a context akin to ours, [5] utilized a contrastive loss signal alongside a distance function capturing instance and class-level fidelity to recursively update the encoder network. Similarly, [27] employed Contrastive Learning to create a compressed representation of training data which is used to detect drifting samples with class centroids. Both these works use a contrastive signal that pulls embeddings of the same class together and pushes those of different classes apart, we instead rely on Center-Loss [10]. Moreover, these works use autoencoders while we adopt a FeedForward Network.

III. Gradient Detection & Contrastive Learning

This section presents key components of our study, specifically gradient-based detectors and Contrastive Learning. We represent an ML-based NIDS with a parameterized model f that maps NetFlows $x_i \in \mathbb{R}^d$ into a class $\overline{y} = \arg\max_{j \in C} z_j$, where C is the set of training classes and $z_j = f_j(x_i)$ is the logit (pre-softmax) score produced by f for class $j \in C$. Additionally, we suppose the model gives in output the embedded representation $e_i \in \mathbb{R}^w$ of x_i constructed after the last embedding layer, i.e., the one before the classifier layer. Refer to the left part of Fig. 2 for a graphical representation.

A. Gradient-based Detection: ODIN and Mahalanobis

Most OOD detectors rely on information extracted from models to derive OOD scores, disregarding information on the gradient. In [15], the authors observed that adding a fixed perturbation to samples in the direction of the gradient amplifies the gap between ID and OOD softmax scores. Thus, the idea behind Out-of-DIstribution detectioN (ODIN) [15] is to jointly apply *temperature scaling* [33] and a *controlled perturbation* to detect OOD data. ODIN consists of the following steps:

1) **Temperature Scaling**: divides the logits z_j by a temperature T that reduces the sharpness of the softmax distribution and makes the model less confident.
2) **Perturbation**: involves adding a perturbation ϵ to x_i in the direction given by the sign of the gradient:

$$\widehat{x_i} = x_i - \epsilon \text{sign}(\nabla p_j^*) \tag{1}$$

   where $p_j^* = \max_{j \in C} p_j$ is the maximum softmax score for x_i after temperature scaling. This perturbation pushes samples toward their nearest class.
3) **Detection**: computes an ID score by feeding $\widehat{x_i}$ into the model again; if this score is above a threshold, x_i is ID.

Another similar gradient-based method is the Mahalanobis Detector (MD) [12], where the Mahalanobis distance is used to measure how "typical" a point is with respect to a learned latent distribution. The Mahalanobis distance requires an estimate of the mean μ and covariance matrix Σ for each ID class, which are normally extracted from the training set. After having these parameters, MD applies a controlled perturbation as in ODIN, but without temperature scaling and where the

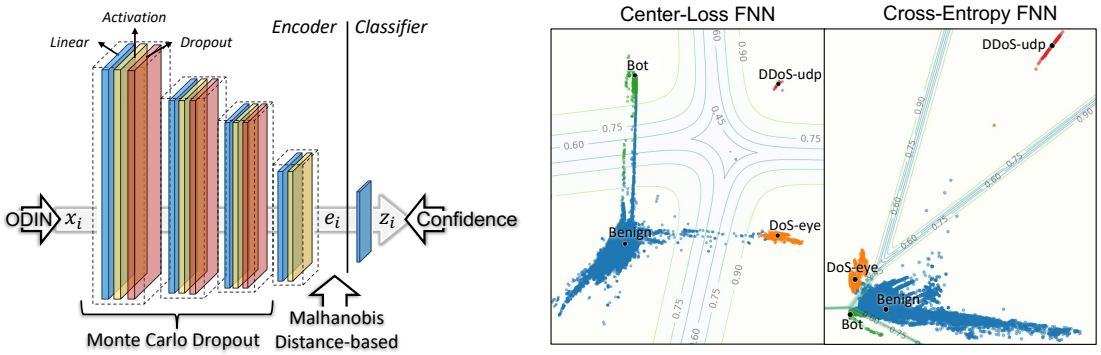

Fig. 2. On the left, the architecture of the considered FNN and the holistic view of where the different OOD detectors act. On the right, the 2D embeddings (e_i) created by the encoder inside the decision space of the FNN trained with and without Center-Loss on four traffic types. The lines highlight points of the decision space where the softmax scores produced by the classifier (i.e., the FNN confidence) change.

gradient is computed with respect to the distance between e_i and the nearest class distribution ($dist_{MD}(\cdot)$):

$$\widehat{x_i} = x_i - \epsilon \text{sign}(\nabla dist_{MD}(e_i)) \tag{2}$$

The final OOD detection is similar to ODIN: a threshold is first extracted from the validation, and every perturbed $\widehat{x_i}$ with a distance higher than this threshold is labeled as OOD.

B. Contrastive Learning and Center Loss

Contrastive Learning is a self-supervised technique designed to learn meaningful embedding representations. It achieves this by bringing similar input samples closer together in the learned embedding space while pushing dissimilar apart [34]. By doing so, Contrastive Learning encourages the model to capture discriminative features that can be useful for various downstream tasks. In a typical contrastive framework, each sample in a batch is augmented through ad-hoc transformations (such as random cropping and flipping for images) into new samples called the positives, while the original sample is referred to as the anchor. The objective is to maximize the similarity between the anchor and the positives while minimizing the similarity between the anchor and other batch samples.

One of the precursor techniques to Contrastive Learning is Center-Loss [10] (CL). CL encourages a model to learn discriminative embeddings e_i that cluster around their class centers. It accomplishes this by defining a center $c_j \in \mathbb{R}^w$ for each class $j \in C$ and introducing an additional term to the standard cross-entropy loss. This additional term minimizes the distance between the embeddings and their corresponding class centers, which are determined by the ground-truth labels. The class centers are learned alongside the model's parameters by minimizing the additional Center-Loss term:

$$L_{CL} = \frac{1}{2} \sum_{i \in B} ||e_i - c_j^*||^2 \tag{3}$$

where the sum is over the batch samples B and c_j^* is the ground-truth center of each input. The overall loss is thus a linear combination of Cross-Entropy (L_{CE}) and Center-Loss: $L = L_{CE} + \lambda L_{CL}$, where λ is a hyperparameter that controls the weight of L_{CL}.

IV. Methodology & Design Choices

Herein, we present our model architecture, the adapted Center-Loss for our settings, and the selected OOD detectors.

A. The Model Architecture

Although it might be possible to design ad-hoc architectures for OOD detection tasks [19], [35], we prefer to avoid them and make no particular assumption about the model. We only require for a NetFlow $x_i \in \mathbb{R}^d$ to have access to its pre-softmax score z_i and to an embedded representation $e_i \in \mathbb{R}^w$ produced within the model, like the one generated before the classification layer. Therefore, the architecture can comprise any layer like convolutional, linear, and recurrent ones [30].

We logically divide our model into two parts: (i) an encoder that transforms NetFlows x_i into embeddings e_i, and (ii) a classifier that uses e_i to produce a softmax score for each class. The proposed encoder is composed of four linear layers of decreasing size, each activated through a LeakyReLU non-linearity with a slope of 0.15. We also apply dropout after the first three layers. The classifier is a single linear layer that has as many neurons as the number of output classes. We will refer to such a model as Feedforward Neural Network (FNN) and provide in the left part of Fig. 2 a visual representation.

B. Improving the Model Embedding

Recently, Contrastive Learning has been widely adopted to improve the performance of ML in different tasks [34]. In our work, we propose to use Contrastive Learning to make embeddings learned by our FNN more discriminative, improving classification tasks [10], [34], [36] and potentially enhancing the effectiveness of OOD detectors. As an example, refer to the two plots on the right of Fig. 2, which represent the embedding spaces produced by our FNN encoder when trained with and without a contrastive learning signal, respectively. It is immediate to see that the projected NetFlows of individual attacks are less scattered and more separated in the Center-Loss plot. These discriminative embeddings may benefit detectors like distance-based ones that assume normality or well-representative prototypes to detect OOD.

Although many contrastive methods exist [10], [34], [36], most of them are primarily designed for other ML fields and rely on positive samples and ad-hoc augmentations [34], vague concepts in the NIDS literature. Therefore, we prefer to employ a simpler and more straightforward method: *Center-Loss* (described in Section III-B). The application of Center-Loss to our setting does not require any particular modifications; however, we need to account for the unique aspects of NIDSs such as heavily imbalanced training sets and noisy data.

To mitigate the effect of imbalanced sets, we adopt a combination of over- and under-sampling as further described in Sec. V-B. This is particularly important because CL works locally on batches, and with heavy unbalancing it is likely to have batches with only NetFlows of the majority benign class, thus focusing too much on improving their embeddings and its center. In addition, we propose to apply the CL term of Eq. 3 only on samples correctly classified by the model. This helps in mitigating the effect of noisy labels and data during training, which are common issues in NIDSs [37].

C. Adopted OOD Detectors

In this work, we consider OOD detectors of different natures that work beside classification models (pre-trained and not) and can be applied to any architecture. Our rationale for selecting detectors reviewed in Sec. II is to choose popular ones in related ML fields whose complexity (theoretical and implementation) is as low as possible. Wherever possible and not penalizing in terms of performance, we prefer to evaluate detectors as originally proposed.

Regarding confidence-based detectors, we adopt the baseline approach proposed in [38] (`CONF`). This straightforward solution involves applying a threshold to the softmax scores and labeling as OOD all the NetFlows with a score below this threshold. We also adopt Monte Carlo Dropout [17] (`MCD`) in a similar manner. Instead of relying on a single confidence estimate for a NetFlow x_i, we leverage MCD with a switch-off probability of 0.4 to obtain multiple softmax scores. Then, all those x_i for which the standard deviation of their softmax scores exceeds a predefined threshold are flagged as OOD. This allows a less biased estimate about x_i.

In addition, we adopt two cutting-edge gradient-based detectors in computer vision which are `ODIN` [15] and Malahnobis [12] (`MD`), introduced in Sec. III-A. Although there exist improvements over these proposals (see e.g. [19]), we prefer to keep them as originally proposed to avoid potential biases introduced by the assumptions of such improvements. As we demonstrate later, gradient-based detection seems less effective on NetFlows compared to images.

Lastly, we include two distance-based detectors. The first one (`SIM`) uses the simplified Silhouette [39] to measure the distance between test and training data. For each class $j \in C$, `SIM` first extracts a center by averaging the embeddings e_i of training data labeled with j. Then, it uses these centers to compute Silhouette values for testing NetFlows by flagging as OOD those having a maximum value below a threshold. Note that the simplified Silhouette is adopted here to reduce the computational complexity of the standard Silhouette [39]. The second detector is based on the K-Nearest Neighbor (`KNN`) proposal in [24], where a separate KNN model is fitted on the embeddings e_i of training classes and used to measure Euclidean distances at inference time. This detector works similarly to `SIM`, but it selects the KNN model to query for measuring the distance from the k^{th} nearest neighbor based on the class predicted by the FNN. If such distance is above a threshold, the NetFlow is OOD. After preliminary analysis, we set $k = 25$ and $\alpha = 100\%$ in all our experiments. We refer the reader to [24] for more detailed explanations.

All these detectors rely on thresholds extracted on ID NetFlows, except ODIN and MD, which also require OOD data. Details on threshold extraction are provided in Sec. V-B.

V. Experimental Setup

A. Datasets and Preprocessing

Datasets. In our experiments, we train models on benign traffic and specific attacks from one dataset. Then, we evaluate such models on remaining attacks from the same dataset as well as attacks from another one. Thus, we selected two similar labeled datasets: *IDS2017* [40] comprises synthetic traffic and common attacks like DoS (D) and DDoS (DD), while *IDS2018* [40] contains more attack variants and is created in a larger network. You can refer to Tab. IV for the list of their attacks. The traffic of these datasets is transformed into NetFlows with the CICFlowMeter [41], where each NetFlow is described by a set of more than 80 features. We purposely chose these datasets as they contain roughly the same attack families and their traffic comes from consecutive years, hence should not differ much. By training on some attacks and testing on all the others from both datasets, we can logically simulate all the cases described in Fig. 1. With a single dataset, it is hard to cover situations like those described in Case 2 of Fig 1, as inducing shifts in known training attacks requires artificial manual crafting of NetFlows. Contrary, with a dataset comprising the same attacks, we can try to simulate situations of Case 2 without explicit manual intervention. As an example, we are going to use the D-hulk traffic of IDS2018 for training and test detectors on the "shifted" D-hulk traffic of IDS2017. Lastly, note that solely including more diverse datasets does not help in better modeling Case 2.

Preprocessing. We have established with a simple feature selection procedure a common set of 20 features for both our datasets from the 80+ generated by the CICFlowMeter. Before applying our procedure, we log-scale continuous features, leave unaltered integer ones, and encode in one-hot port numbers by considering three intervals: well-known, registered, and ephemeral ports. Then, our feature selection procedure starts by considering each dataset per se and identifies the most important features with a Random Forest analysis [42]. On each dataset, we apply the following steps:

1) Remove IPs and quasi-constant (variance <0.05) features.
2) Keep an arbitrary feature between ones having a Pearson correlation coefficient higher than 0.8.

TABLE I
THE COMMON SET OF 20 FEATURES DESCRIBING A NETFLOW.

#	Name	Description
1	Dst wk	Whether destination port is well-known [0, 1023].
2	Dst reg	Whether destination port is registered [1024, 49151].
3	Num fwd pkts	Number of packets outgoing the network.
4	Num bwd pkts	Number of packets ingoing the network.
5	Max fwd pkt	Maximum size of outgoing packets in the NetFlow.
6	Max bwd pkt	Maximum size of ingoing packets in the NetFlow.
7	Ack cnt	Number of packets with ACK.
8	Syn cnt	Number of packets with SYN.
9	Rst cnt	Number of packets with RST.
10	Duration	NetFlow duration in seconds.
11	Pkts/s	Number of exchanged packets per second.
12	Fwd pkts/s	Number of packets outgoing the network per second.
13	Bwd pkts/s	Number of packets ingoing the network per second.
14	Avg IAT	Average Inter Arrival Time between packets.
15	Std IAT	Standard deviations of packet Inter Arrival Times.
18	Sflow fwd byts	Average number of outgoing packet bytes in sub-flows.[a]
19	Sflow bwd byts	Average number of ingoing packet bytes in sub-flows.[a]
16	Avg idle	Average idle time (between sub-flows) of the NetFlow.
17	Avg active	Average active time (length of sub-flow) of the NetFlow.
20	Fwd Seg min	Minimum segment size in outgoing packets.

[a] A sub-flow is a sequence of packets inside the NetFlow each received within a maximal inter-arrival time.

TABLE II
THE TRAINING SCENARIOS EXTRACTED FROM IDS2018 AND THEIR RATIONALE IN OUR EXPERIMENTS.

	Training Attacks	Rationale
Scenario 1	• FTP • D-hulk • DD-hoic	We select training attacks that generate a high-volume of normal (FTP) and obfuscating traffic. This setup tests detectors in identifying low-volume attacks and variations of training ones, e.g., SSH is a variant of FTP on a distinct protocol and D-hulk from IDS2017 may be a shifted version of 2018 one.
Scenario 2	• SSH • D-hulk • DD-http	We choose again training attacks of high-volume which may induce different classification patterns within the FNN compared to Scenario 1. Different patterns can potentially impact the capability of certain detectors to effectively identify unknown attacks.
Scenario 3	• D-eye • DD-udp • Bot	We create a diverse set of training attacks, spanning various malicious strategies, mostly relying on the HTTP protocol. This allows testing detectors in identifying attacks on different protocols and HTTP attacks with similar or distinct malicious strategies.

3) Fit a large Random Forest (200 trees with 20 as maximum depth) on the remaining features and evaluate Gini and Permutation importance [42].
4) Rank the features based on the normalized sum of Gini and Permutation importance.

After having the features ranked by their importance, our procedure automatically selects those that are among the 20 most important in both datasets. To ensure a satisfactory detection performance, we additionally ensure that the top-7 features on each dataset are selected. The final set of features is reported in Tab. I. Note that for open-source datasets not having all our features, Zeek with a customized script can be used to generate the required NetFlow features.

B. Model, Training, and Tuning details

Architecture. In our FNN, we use linear layers of decreasing size in the encoder, the first contains 128 neurons, the second 64, the third 32, and the fourth 2. All dropout layers switch off neurons with a probability of 0.3. Whereas the classifier contains as many neurons as classes in the training set. Note that we restrict the encoder to produce embeddings in a 2D space to easily plot them. We offline verified that this restriction does not limit the model classification performance, as theoretically stated in the universal approximation theorem [30].

Training. We train our FNN on scenarios extracted from IDS2018, the larger and more comprehensive dataset, where a scenario comprises all the benign traffic and three attacks (4 classes). Refer to Tab. II for the list of the scenarios and their attacks. Every training scenario is split 70/30 in a stratified manner. We use 70% of the traffic for training two separate models – one with Center-Loss and one with Cross-Entropy. The remaining 30% is used for validation purposes and detector tuning. All the models are trained for 25 epochs with the Adam optimizer [30], batch size of 512, and learning rate at 0.0005. The model producing the best F1-score on the validation traffic (always above 99% in all our scenarios) is saved for testing. Regarding Center-Loss, we use a separate Adam optimizer, a learning rate of 0.0001, and a weighting factor $\lambda = 1$ (see Sec. III-B). In addition, we use over- and under-sampling to make batches with roughly the same amount of NetFlows for each class. This is achieved by sampling with repetition a NetFlow with probability inversely proportional to the frequency of its class in the training set.

Metrics and Detector Tuning. Since our objective is to determine whether unknown malicious traffic can be identified as OOD, we evaluate detectors based on the True Positive Rate (TPR). In this context, a true positive refers to a NetFlow of an unknown attack labeled as OOD. We specifically avoid using the F1-score because our detectors are tuned to maintain a low False Positive Rate (FPR) of 5% on ID traffic. However, we do utilize the F1-score when assessing the performance of detector combinations, as the rejected ID traffic may exceed 5%. All the detectors selected in Sec. IV-C rely on pre-defined rejection thresholds. To set such thresholds, we followed a common practice in the literature, ensuring that 95% of the ID validation traffic (malicious included) is not rejected [12], [15], [19], [24]. The only exceptions are `ODIN` and `MD`, for which we also used OOD attacks to extract the threshold and select $\epsilon \in \{0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$ with $T = 20$. Specifically, we took advantage of attacks not used in our evaluation, like infiltration and attacks with a few NetFlows, and used them along with validation traffic to tune as in [15]. Note that we exclude infiltration as it is not well classified by the FNN nor well detected by OOD techniques with our features, i.e., an example of Case 4 in Fig. 1. We also see that using hard-to-discriminate attacks improves the detection capability of `ODIN` and `MD`. For other parameters such as mean and covariance matrix in `MD`, centers in `SIM`, and `KNN` models, we extracted them from the training set.

VI. Results & Analysis

In this section, we evaluate the chosen detectors in Sec. IV-C to identify previously unknown intrusions as OOD. Remember that these detectors work beside the ML-based NIDS, i.e., the FNN described in Sec. IV-A, that is trained to classify attacks of specific scenarios. These scenarios comprise only a few training attacks and are designed to test detectors under different circumstances, such as those presented in Fig. 1. The specific scenarios adopted are outlined in Tab. II. Although it is hard to precisely pinpoint which situation of Fig. 1 is in each scenario, we tried to design them to logically contain all. Every scenario comprises training attacks characterized by distinctive aspects. Within the pool of unknown testing attacks, encompassing all attacks not encountered during training, there are fairly similar and dissimilar ones. These testing attacks should end up in different regions of the FNN's decision space, effectively simulating the situations depicted in Fig. 1.

A. Detecting Unknown Attacks with OOD detectors

We begin by assessing detectors and their combination when applied to models trained in a *multi-class* setting both with and without Center-Loss (CL).

Single Detector Results. We first examine the performance of individual detectors. Tab. III presents in each horizontal section the TPR of detectors (columns) on a distinct scenario. Each cell contains the TPR of a detector on an unknown attack when applied to the FNN trained with and without Center Loss (TPR_{CL} / TPR_{CE}). The last row of a section (Total TPR) reports the global TPR, irrespective of the attack types.

Overall, we observe that all the unknown attacks are detected to some extent in their traffic. The best OOD detector appears to be KNN, followed by MCD and CONF, while other detectors exhibit lower average performance. Specifically, we see that ODIN and MD have generally lower TPRs than confidence-based detectors, contrary to what was discovered in computer vision [12], [15], [19]. This suggests that controlled perturbations rigidly derived from the gradient do not always benefit detection as expected. We suspect that NetFlow features, which do not have a bounded and discrete domain as pixels, may require more flexible per-feature perturbations that better conform to the domain of features. This might help in pushing ID NetFlows toward their class, better enlarging the gap between ID and OOD scores as in computer vision.

Furthermore, we find that applying CL to multi-class models does not always improve OOD detection. Although the embeddings produced with CL are in general more discriminative, this benefits detectors such as CONF, MD, and SIM, but not as much KNN. Our explanation is that a multi-class FNN has already semantically rich embeddings, reducing the effect of CL. In addition, training with CL may sometimes force the model to produce embeddings closer to known classes, which would not be without CL (refer to the right of Fig. 2 for a graphical comparison). This may benefit the assumptions of certain detectors like the representativeness of mean and covariance matrix in MD and the centers of SIM. However, tighter neighborhoods may negatively impact the performance of KNN in certain situations. Consequently, we conclude that CL is a simple method to enhance OOD detection in NIDSs [24], [27], although its effectiveness may vary on certain detectors.

TABLE III
The True Positive Rate Percentage of OOD detectors when applied to models trained with and without Center Loss.

	IDS2018 Attacks	CONF	MCD	ODIN	MD	KNN	SIM
		Scenario 1: FTP - D-hulk - DD-hoic					
unknown attacks	SSH	50.0 / 50.0	53.2 / 70.9	50.0 / 48.2	50.0 / 48.2	100 / 100	50.0 / 48.2
	D-eye	95.7 / 81.5	98.7 / 98.7	17.1 / 61.7	92.7 / 60.2	98.6 / 85.9	75.2 / 60.6
	D-http	9.4 / 9.4	8.1 / 7.3	0.0 / 100	100 / 100	1.0 / 1.0	2.0 / 2.0
	D-loris	8.3 / 0.4	52.9 / 26.6	8.3 / 0.0	53.4 / 0.4	79.1 / 78.3	1.3 / 0.6
	Web	21.2 / 33.4	35.4 / 32.5	3.8 / 0.0	49.6 / 2.5	56.9 / 46.8	47.3 / 2.3
	Botnet	0.0 / 0.0	0.8 / 0.1	49.9 / 0.0	0.4 / 0.1	50.0 / 97.0	0.0 / 0.0
	DD-http	89.4 / 48.7	95.6 / 71.3	90.6 / 0.0	56.0 / 0.1	93.9 / 92.3	50.5 / 0.0
	DD-udp	0.0 / 0.0	98.1 / 0.8	0.0 / 0.0	100 / 99.0	100 / 100	100 / 0.0
	Total TPR	**53.2** / 33.9	**57.3** / 48.1	**61.6** / 20.6	**48.5** / 20.7	74.2 / **83.9**	**33.8** / 9.5
	IDS2018 Attacks	CONF	MCD	ODIN	MD	KNN	SIM
		Scenario 2: SSH - D-hulk - DD-http					
unknown attacks	FTP	100 / 100	100 / 100	99.2 / 100	100 / 100	100 / 100	100 / 100
	D-eye	66.5 / 73.2	83.9 / 100	19.5 / 52.7	95.9 / 70.4	100 / 100	76.3 / 7.5
	D-http	100 / 100	100 / 100	98.1 / 100	100 / 100	100 / 100	100 / 100
	D-loris	30.5 / 0.4	72.0 / 11.5	6.8 / 0.4	77.5 / 0.4	60.8 / 80.9	54.7 / 0.3
	Web	9.4 / 10.1	57.4 / 29.5	2.8 / 0.9	53.7 / 3.1	40.5 / 55.3	19.2 / 0.0
	Botnet	0.0 / 0.0	0.5 / 0.0	0.0 / 0.0	0.3 / 0.1	48.1 / 99.4	0.0 / 0.0
	DD-udp	0.0 / 0.0	74.1 / 59.7	14.4 / 0.1	99.5 / 100	100 / 100	99.0 / 0.6
	DD-hoic	76.1 / 76.1	76.4 / 74.1	64.2 / 50.2	73.0 / 10.9	60.4 / 20.2	57.9 / 1.4
	Total TPR	**65.1** / **65.1**	**66.4** / 65.1	**57.2** / 51.4	**65.1** / 32.3	**68.8** / 59.5	**56.6** / 25.4
	IDS2018 Attacks	CONF	MCD	ODIN	MD	KNN	SIM
		Scenario 3: D-eye - Bot - DD-udp					
unknown attacks	FTP	0.0 / 0.0	7.2 / 2.8	0.0 / 0.0	90.0 / 89.1	94.7 / 100	78.4 / 0.0
	SSH	0.0 / 0.0	10.3 / 1.5	0.0 / 0.0	0.0 / 1.3	0.1 / 60.0	0.0 / 0.0
	D-hulk	100 / 97.5	100 / 99.6	98.4 / 97.3	98.4 / 96.1	99.0 / 100	91.0 / 72.6
	D-http	0.0 / 0.0	7.3 / 2.4	0.0 / 0.0	89.8 / 88.3	99.5 / 100	45.1 / 0.0
	D-loris	66.5 / 45.5	67.0 / 55.7	45.7 / 46.1	88.7 / 49.7	65.8 / 80.6	3.8 / 1.9
	Web	3.6 / 28.6	11.4 / 23.8	1.0 / 9.2	53.7 / 29.1	37.5 / 60.1	29.2 / 2.1
	DD-http	91.7 / 49.2	72.7 / 65.7	46.3 / 46.0	44.2 / 0.1	80.0 / 70.5	43.8 / 3.5
	DD-hoic	73.1 / 94.0	61.3 / 96.2	47.4 / 87.0	0.0 / 0.9	4.8 / 20.5	0.0 / 7.1
	Total TPR	**66.4** / 61.3	59.9 / **67.2**	46.6 / **58.3**	**45.2** / 33.4	56.7 / **64.8**	**39.4** / 17.9

Ensembles Results. We proceed by aggregating detectors into two ensembles to improve overall performance and evaluate their complementarity. For this analysis, we use the previously considered scenarios and assess the ensembles' performance on unknown attacks also from the IDS2017 dataset.

To measure the maximum amount of unknown traffic that can be rejected, we use a simple ensemble (ENS_1) that flags a NetFlow as OOD if at least one detector predicts it as such. This ensemble comprises all the detectors applied to the FNN trained with and without CL, resulting in a total of 12 combinations. The second ensemble (ENS_2) consists of three detectors and flags a NetFlow as OOD if at least one predicts OOD. We use the CONF detector applied to the CL-trained FNN, along with the KNN and ODIN detectors applied to the FNN trained with Cross-Entropy. The goal of ENS_2 is to prove that it contains complementary detectors. We remark that these ensembles have been specifically designed to increase the detection (TPR) of unknown attacks.

Tab. IV presents the TPR of the ensembles on attacks from the two datasets (horizontal sections). The last two rows report the total TPR and total F1-Score on all the attacks from

TABLE IV
THE TRUE POSITIVE RATE PERCENTAGE OF THE OOD ENSEMBLES. CELLS MARKED WITH * CONTAIN ATTACKS SEEN IN TRAINING.

	Attacks	Support	Scenario 1 ENS_1	Scenario 1 ENS_2	Scenario 2 ENS_1	Scenario 2 ENS_2	Scenario 3 ENS_1	Scenario 3 ENS_2
IDS2018	FTP	193.4k	*	*	100	100	100	100
	SSH	187.6k	100	100	*	*	62.0	59.1
	D-eye	41.5k	100	100	100	100	*	*
	D-hulk	461.9k	*	*	*	*	100	100
	D-http	139.9k	100	100	100	100	100	100
	D-loris	11.0k	100	79.0	100	100	100	80.7
	Web	833	98.1	65.5	82.8	60.0	96.4	66.0
	Botnet	286.2k	99.7	99.5	99.3	99.2	*	*
	DD-http	576.3k	97.5	94.1	*	*	99.8	99.5
	DD-udp	1728	100	100	100	100	*	*
	DD-hoic	686.0k	*	*	82.8	81.1	97.4	96.6
IDS2017	FTP	3967	100	100	100	100	100	100
	SSH	2976	100	100	100	100	100	100
	D-eye	7560	100	95.6	100	100	100	100
	D-hulk	158.3k	100	89.8	100	96.7	100	100
	D-http	1740	100	99.3	100	100	100	100
	D-loris	3999	100	99.8	100	100	100	63.1
	Botnet	736	100	99.9	100	100	92.2	100
	PScan	159.1k	100	99.7	100	100	99.8	86.9
	DD-loit	95.1k	100	99.9	100	100	100	100
	Total TPR		**99.1**	96.7	**93.3**	92.3	**96.9**	95.3
	Total F1		75.9	**86.0**	77.4	**82.5**	73.5	**83.5**

both datasets. Remember that true positives refer to unknown attacks labeled as OOD while false positives correspond to benign NetFlows mistakenly marked as OOD. We exclude training attacks as the FNN detects them correctly.

We first highlight that ENS_1 achieves almost perfect TPRs in both datasets, indicating there are complementary detectors in our set. However, this ensemble strategy also increases the false positives, as remarked by the consistent gap between total TPR and F1-Score in all three scenarios. In fact, the false positive rate on benign validation traffic goes from 5% of single detectors (as resulting from the tuning described in Sec. V-B) up to 36% with the ensemble.

On the other hand, ENS_2 demonstrates similar total TPRs compared to ENS_1 but consistently achieves better F1-Scores. This improvement is attributed to significantly reduced false positive rates, which are halved compared to those of ENS_1. We observed that CONF and KNN contribute the most to this ensemble, aligning with the findings in Tab. III, while ODIN gives a smaller nevertheless important contribution. Overall, ENS_2 proves to be a superior ensemble that incorporates complementary detectors. This highlights the relevance of combining detectors of different natures (e.g., confidence-, distance-, and gradient-based) applied to models trained with different strategies. By doing this it is possible to fortify defense against the situations described in Fig. 1.

Additionally, we remark that detecting attacks from other datasets appears to be a relatively easier task, indicating experimental bias [43]. Although we verified the similarity of individual feature distributions between datasets, patterns extracted from IDS2018 differ from those of IDS2017. This is evident from the almost perfect rejection of IDS2017 attacks, the rejection of attack types included in the training set from the 2018 data (such as FTP and D-hulk in Scenario 1), and the high rejection rates (above 70%) for IDS2017 benign traffic. Note that the benign traffic from IDS2017 is for a different network and is expected that detectors will reject benign traffic incurred on a different network. In general, there is a need for a methodology that enables better integration of traffic from different networks (datasets) for the purpose of OOD in NIDSs, a topic we will cover in the future.

Finally, we also conducted experiments by training on IDS2017 attacks and testing on IDS2018. In these regards, we only observed lower detection rates on certain unknown attacks of IDS2017 like Bot in both individual detectors and ensembles. However, the performance on attacks from IDS2018 (Bot included) was almost perfect. This discrepancy vouches once again for the necessity of a better integration methodology. Due to space limitations, we do not report such extensive results.

TABLE V
THE TRUE POSITIVE RATE PERCENTAGE OF DETECTORS WHEN APPLIED TO BINARY MODELS TRAINED WITH AND WITHOUT CENTER-LOSS AND THE DIFFERENCE WITH MULTI-CLASS TOTAL TPRS OF TAB. III.

IDS2018	CONF	MCD	ODIN	MD	KNN	SIM
Scenario 1	**54.1** / 8.9	**61.0** / 47.8	**57.0** / 17.0	**52.6** / 15.9	**85.0** / 73.3	**53.2** / 13.1
Scenario 2	**65.4** / 64.4	**66.3** / 65.0	**63.9** / 61.0	**55.3** / 30.8	**57.9** / 50.2	**47.4** / 1.5
Scenario 3	**51.3** / 36.1	**52.7** / 46.3	**51.3** / 35.6	**48.5** / 34.0	**53.1** / 52.6	**33.4** / 13.9
ΔScenario 1	↑0.9 / ↓25.0	↑3.7 / ↓0.3	↓4.6 / ↓3.6	↑4.1 / ↓4.8	↑10.8 / ↓10.6	↑19.4 / ↑3.6
ΔScenario 2	↑0.3 / ↓0.7	↓0.1 / ↓0.1	↑3.9 / ↑9.6	↓9.8 / ↓1.5	↓10.9 / ↓9.3	↓9.2 / ↓23.9
ΔScenario 3	↓15.1 / ↓25.2	↓7.2 / ↓20.9	↑4.7 / ↓22.7	↑3.3 / ↑0.6	↓3.6 / ↓12.2	↓6.0 / ↓4.0

B. Better Embedding, Better Detection

Many supervised datasets in network intrusion detection contain information about the specific type of attack each NetFlow belongs to. Typically, this information is ignored as the task is treated as a binary classification one. However, we do demonstrate herein that leveraging the richer semantics of multi-class models can improve OOD detection and that Contrastive Learning can serve a similar goal. To this end, we compare the overall *multi-class* results from the previous section with those of the same detectors applied to models trained in a *binary task*, which is obtained by grouping NetFlows of training attacks into a single malicious class. We retrain a binary FNN with and without CL for each scenario of Tab. II, and evaluate OOD detectors on unknown attacks.

Single Detector Comparison. We first compare the results of individual detectors on unknown attacks of IDS2018. In Tab. V, the top section displays the total TPR of detectors applied to binary models, with and without CL (TPR_{CL} / TPR_{CE}). Whereas the bottom section presents the reduction in the rejections computed by subtracting the multi-class total TPRs from binary ones, with and without CL.

The results in the top section highlight that CL consistently improves all the TPRs in the binary case, resulting more effective than in multi-class settings. Therefore, the importance of CL and Contrastive Learning is more pronounced for ML-based NIDSs trained on binary tasks, since multi-class trainings already make embeddings more discriminative.

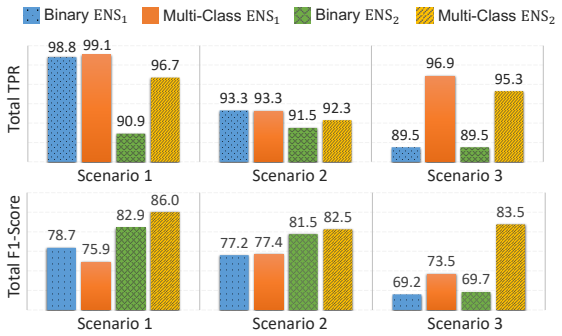

Fig. 3. The overall comparison of the four ensembles on unknown attacks from both datasets. Binary ENS_1 and ENS_2 are those created with detectors applied to the FNN trained in a binary task (benign vs. malicious), while Multi-Class ones are those created with the FNN trained in a multi-class setting (each training attack makes a distinct class). Better viewed in colors.

In the bottom section, we generally observe that detectors applied to the binary FNN without CL have significantly reduced ↓ TPRs compared to the multi-class case. This underlines that semantic information induced by multi-class training enables to make better embedding spaces for detecting unknown attacks. Whereas detectors applied to the binary FNN with CL have either similar detection rates or less pronounced reductions, suggesting that CL roughly gives the same enhancement despite the training regimes of the model.

Therefore, we conclude that better embeddings, such as those obtained from multi-class models or Contrastive Learning methods, enhance OOD detection.

Ensemble Comparison. Lastly, we present the overall results of the two ensembles described in Sec. VI-A in the binary case. Remember that ENS_1 comprises all the combinations of binary models and detectors, while ENS_2 combines `CONF` with the CL-trained FNN, as well as `KNN` and `ODIN` coupled with the FNN trained without CL. For this comparison, we consider the two ensembles made with the binary FNNs and also those created with the multi-class FNNs. Fig. 3 plots for each scenario of Tab. II the total TPR and F1-Score on unknown attacks from both IDS2018 and IDS2017.

Overall, we observe that ensembles of detectors applied to binary models still yield superior detection, but not as much as in the multi-class case. This suggests that combining OOD detectors is more effective when applied to models with semantically richer embeddings, such as those produced in multi-class settings. Furthermore, the better F1-scores of ENS_2 with respect to ENS_1 in both the binary and multi-class cases indicate that ENS_2 detectors positively complement each other. This demonstrates again the importance of leveraging OOD detectors of different natures, as they enable a broader coverage of unusual and potentially harmful regions of the model's decision space (see Fig. 1).

VII. Conclusion

In this work, we analyzed the ability of existing OOD techniques to detect traffic of unknown intrusions. We use a standard FeedForward Neural Network as ML-based NIDS and trained it on subsets of attacks in a binary and multi-class setting, by also applying a Contrastive Learning signal. Then, we use these models along with a set of six OOD techniques relying on different strategies to identify unknown attacks extracted from the same and a separate dataset (network).

Our findings reveal that existing OOD detectors constitute a valid means to identify portions of unknown attacks, although their effectiveness varies compared to other ML fields. Furthermore, we highlighted that employing training strategies such as multi-class supervision and Contrastive Learning improves the performance of most tested OOD detectors. Lastly, we demonstrated that combining detectors relying on different strategies leads to superior performance, especially when applied to differently trained models.

While our study has provided some insights into the potential of adopting OOD techniques for network intrusion detection, we acknowledge that there is still much to cover. Notably, one of the limitations of our work is the lack of a methodology that allows a more realistic integration of unknown attacks extracted from diverse datasets (networks). As many datasets offer only limited coverage of cyberattacks, this methodology is of utmost importance to comprehensively assess OOD techniques. Additionally, we recognize the prospective value of a visualization tool derived from our plotting strategy used for Fig. 2 to inspect models' decision spaces. Such a tool could prove beneficial for network inspection in practical use cases and aid the categorization of attacks in the context of Fig. 1.

Therefore, in future works, we will focus on these points and also explore the influence of different features on the efficacy of OOD detectors. Furthermore, we intend to improve less effective detectors, like ODIN and MD, and evaluate others.

References

[1] D. Chou and M. Jiang, "A survey on data-driven network intrusion detection," *Computing Survey*, vol. 54, no. 9, 2021.

[2] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Springer Cybersecurity*, vol. 2, no. 1, 2019.

[3] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, 2010.

[4] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *Journal of Network and Computer Applications*, vol. 169, 2020.

[5] A. Kuppa and N. Le-Khac, "Learn to adapt: Robust drift detection in security domain," *Computers and Electrical Engineering*, vol. 102, 2022.

[6] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, 2019.

[7] J. Yang, K. Zhou, Y. Li, and Z. Liu, "Generalized out-of-distribution detection: A survey," *ArXiv*, 2021.

[8] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. Lawrence, *Dataset Shift in Machine Learning*. The MIT Press, 2009.

[9] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," *International Conference on Data Mining*, 2007.

[10] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *Computer Vision–ECCV*. Springer International Publishing, 2016.

[11] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *Computing Survey*, vol. 54, 2021.

[12] M. Salehi, H. Mirzaei, D. Hendrycks, Y. Li, M. H. Rohban, and M. Sabokrou, "A unified survey on anomaly, novelty, open-set, and out of-distribution detection: Solutions and future challenges," *Transactions on Machine Learning Research*, 2022.

[13] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, "Toward open set recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, 2013.

[14] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," *Proceedings of International Conference on Learning Representations*, 2017.

[15] S. Liang, Y. Li, and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks," *ArXiv*, 2017.

[16] Y. Sun, C. Guo, and Y. Li, "React: Out-of-distribution detection with rectified activations," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[17] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on Machine Learning*, vol. 48, 2016.

[18] T. DeVries and G. W. Taylor, "Learning confidence for out-of-distribution detection in neural networks," *ArXiv*, 2018.

[19] Y.-C. Hsu, Y. Shen, H. Jin, and Z. Kira, "Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[20] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[21] A. Schwaiger, P. Sinhamahapatra, J. Gansloser, and K. Roscher, "Is uncertainty quantification in deep learning sufficient for out-of-distribution detection?" in *AI Safety (IJCAI)*, 2020.

[22] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[23] Z. Xiao, Q. Yan, and Y. Amit, "Likelihood regret: An out-of-distribution detection score for variational auto-encoder," in *International Conference on Neural Information Processing Systems*, 2020.

[24] Y. Sun, Y. Ming, X. Zhu, and Y. Li, "Out-of-distribution detection with deep nearest neighbors," in *Proceedings of the 39th International Conference on Machine Learning Research*, vol. 162, 2022.

[25] D. Kwon, H. Kim, J. Kim, S. Suh, I. Kim, and K. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, vol. 22, 2019.

[26] T. Su, H. Sun, J. Zhu, S. Wang, and Y. Li, "Bat: Deep learning methods on network intrusion detection using nsl-kdd dataset," *IEEE Access*, vol. 8, 2020.

[27] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and explaining concept drift samples for security applications," in *30th USENIX Security Symposium*, 2021.

[28] X. Wang, "Enidrift: A fast and adaptive ensemble system for network intrusion detection under real-world drift," in *Proceedings of the 38th Annual Computer Security Applications Conference*. ACM, 2022.

[29] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro, "Insomnia: Towards concept-drift robustness in network intrusion detection," in *Proceedings of the 14th Workshop on Artificial Intelligence and Security*. New York, USA: ACM, 2021.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT, 2016.

[31] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Network and Distributed System Security Symposium*, 2018.

[32] S. G. Totad, D. C. Mulimani, and P. R. Patil, "Concept drift adaptation in intrusion detection systems using ensemble learning," *International Journal of Natural Computing Research*, vol. 10, no. 4, 2021.

[33] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International Conference on Machine Learning*. PMLR, 2017.

[34] A. Jaiswal, A. Babu, M. Zadeh, D. Banerjee, and F. Makedon, "A survey on contrastive self-supervised learning," *Technologies*, vol. 9, 2021.

[35] C. Cortes, G. DeSalvo, and M. Mohri, "Learning with rejection," in *International Conference on Algorithmic Learning Theory*, 2016.

[36] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[37] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the cicids2017 case study," in *IEEE Security and Privacy Workshops*, 2021.

[38] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," *ArXiv*, 2016.

[39] F. Wang, H.-H. Franco-Penya, J. D. Kelleher, J. Pugh, and R. Ross, "An analysis of the application of simplified silhouette to the evaluation of k-means clustering validity," in *Machine Learning and Data Mining in Pattern Recognition*. Springer International Publishing, 2017.

[40] I. Sharafaldin, A. Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *Conference on Information Systems Security and Privacy*, vol. 1, 2018.

[41] A. Habibi Lashkari., G. Draper Gil., M. S. I. Mamun., and A. A. Ghorbani., "Characterization of tor traffic using time based features," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, 2017.

[42] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, "Variable selection using random forests," *Pattern Recognition Letters*, vol. 31, no. 14, 2010.

[43] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "Tesseract: Eliminating experimental bias in malware classification across space and time," in *Proceedings of the 28th USENIX Conference on Security Symposium*, 2019.