Successful Intrusion Detection with a Single Deep Autoencoder: Theory and Practice

Marta Catillo[1*†], Antonio Pecchia[1†] and Umberto Villano[1†]

[1]Dipartimento di Ingegneria, Università degli Studi del Sannio, Pal. Bosco Lucarelli C.so Garibaldi 107, Benevento, 82100 Italy.

*Corresponding author(s). E-mail(s): marta.catillo@unisannio.it;
Contributing authors: antonio.pecchia@unisannio.it;
villano@unisannio.it;
†These authors contributed equally to this work.

Abstract

Intrusion detection is a key topic in computer security. Due to the ever-increasing number of network attacks, several accurate anomaly-based techniques have been proposed for intrusion detection, wherein pattern recognition through machine learning techniques is typically used. Many proposals rely on the use of autoencoders, due to their capability to analyze complex, high-dimensional and large-scale data. They capitalize on composite architectures and accurate learning approaches, possibly in combination with sophisticated feature selection techniques. However, due to their high complexity and lack of transferability of the impressive intrusion detection results, they are hardly ever used in production environments. This paper is developed around the intuition that complexity is not necessarily justified because a single autoencoder is enough to obtain similar, if not better, intrusion detection results compared to related proposals. The wide study presented here addresses the effect of the seed, a deep investigation on the training loss and feature selection across the use of different hardware platforms. The best practices presented, regarding set-up and training, threshold setting and possible use of feature selection techniques for performance improvement, can be valuable for any future work on the use of autoencoders for successful intrusion detection purposes.

Keywords: Intrusion detection, Deep learning, Autoencoders, Denial of Service.

1 Introduction

The body of scientific literature on machine and deep learning applied to intrusion detection systems (IDS) is huge and ever-increasing [1]. IDSes aim to discover (and possibly block or divert) on-going attacks before any harm can be done. IDS research is pushed by several factors, which include, but are not limited to (i) availability of commercial and open source products to transform raw network packets into ready-to-use *flow records* suited for machine learning (e.g., Netflow, CICFlowMeter, Tranalyzer), (ii) increasing number of public, flow-based, intrusion detection datasets (e.g., UNSW-NB15, UGR16, CICIDS2017, USB-IDS-1) and (iii) specialized hardware and deep learning frameworks (e.g., Keras, TensorFlow and PyTorch). It is a fact that network traffic – transformed into fixed-length records of features – can be successfully leveraged to recognize potential attacks, which is the primary aim of an IDS. Currently, a wide community of academics and practitioners is conducting measurement studies at the intersection of deep learning and IDS. Due to the ever-growing occurrence and complexity of computer security incidents, IDS research is – and will steadily remain – a hot research topic. For a number of different reasons, present-day IDSes are only partially successful to avoid the occurrence of security incidents. These reasons include the high complexity and the huge bandwidth of currently used networks, the use of brand new or unknown exploits, the amplitude of the so-called attack surface. Moreover, often the security problem is blamelessly ignored, and suitable countermeasures are set up only when it is too late.

The "pattern" followed by many IDS papers is typically the same: (i) proposal of an algorithm or architecture based on neural networks (possibly deep ones and combined with feature selection), (ii) training-testing with one (more) public dataset(s), (iii) demonstration of impressive intrusion detection results (typically close to 100%). At the time being, intrusion detection may seem a perfectly solved problem with no room for further improvements. However, the frequent occurrences of security incidents reported by the press indicate the intrusion detection problem is still there, and that the experiments mentioned in the papers have only limited validity in real-world networks. In spite of the large availability of scientific proposals, there is a gap between "lab-made" machine learning and operations. For example, highly-complex deep networks proposed so far for intrusion detection, such as Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) and cascades/ensembles of AutoEncoders (AE), might find no or limited adoption in production environments. Not surprisingly, machine learning scholars tend to apply increasingly complex learning models and to "blindly" trust and use public intrusion detection datasets – the main source of benchmarking data for machine learning in IDS – but overlook the cybersecurity facets of traffic and attacks therein. For example, we found out that many Denial of Service (DoS) attacks from various public traffic captures are indeed ineffective under proper defense mechanisms and suitable configurations of the victim servers [2], which means that part of the community is spending a huge effort to develop learning approaches for

detecting harmless attacks. Another issue is the lack of transferability of the impressive results obtained on reference datasets (possibly outdated and not free from statistical biasing) to even slightly-different data collection settings, documented by previous work, such as [3–6].

Differently from existing proposals, this paper does not aim to address IDS through the application of increasingly complex deep learning models. Rather, we develop around the intuition that complexity is not necessarily justified because a single – and relatively simple – neural network, such as the autoencoder, is enough to obtain similar (if not better) intrusion detection results when compared to related IDS proposals. An autoencoder can be conveniently trained only by means of benign network traffic in order to learn a semi-supervised IDS model. It is unlikely that attacks are known beforehand; as such, semi-supervised approaches are more widely applicable in practice. It should be noted that among the wide corpus of related proposals, (multiple) autoencoders are often used in complex cascades/ensembles configurations, possibly complemented by sophisticated feature selection methods. Achieving a successful IDS through a single autoencoder underlies both *theory* and *practice*. While there are common beliefs that underlie, at least in theory, the design of AE-based IDSes (e.g., many autoencoders and more neurons imply better detection, the randomization of the training facets – initialization of the neurons and shuffling of the training data points – has a negligible effect on the learning process, the lower the training loss the better the IDS detection ability and that feature selection is beneficial to the results), the practice provides rather different outcomes. The gap between theory and practice, and the effectiveness of feature selection in the context of a single AE, are investigated through a large experimental campaign that aims to answer seven research questions (RQ) grouped into the following high-level groups:

- **TRAIN-RQs**: do the randomization of the initial weights and data shuffling (TRAIN-RQ1), the hardware platform (TRAIN-RQ2) and the final value of loss (TRAIN-RQ3) have an impact on the success of the autoencoder at detecting intrusions and how to achieve satisfactory detection performance (TRAIN-RQ4)?
- **FEATSEL-RQs**: do relevant features depend on the selection techniques and attacks (FEATSEL-RQ1) and does discarding less relevant features allow mitigating the effect of the initial randomization of the training process (FEATSEL-RQ2) and improving intrusion detection (FEATSEL-RQ3)?

Experiments are done with the benign traffic and DoS attacks from a recent update of the widely-used CICIDS2017 public intrusion detection dataset; the campaign is conducted with four hardware platforms and the ubiquitous Kera-Tensorflow framework to demonstrate the relevance of the problems addressed. The effectiveness of the AE-based IDS is assessed by computing the metrics of recall, precision, F1 score and false positive rate (FPR).

The results indicate that the randomization of the training facets, which we assess by varying the seed[1] of the pseudorandom number generator (PRNG), affects the detection performance. For example, different values of the seed make the F1 score to vary between 1.0 and 0.6 (if not less). This finding is confirmed – and even exacerbated – across the platforms assessed, as we note that different CPUs (or GPUs) lead to different exploitation of the threads used in training phase and, in turn, to different outcomes: in consequence, even when the seed is fixed, the results will be different from platform to platform. Another practical challenge raised is that obtaining low loss values at the end of the training process does not guarantee good IDS performance. As for the use of feature selection, we observe that it is possible to obtain marginal improvements of the detection metrics (e.g., F1 score from 0.982 to 0.987 by discarding 30 features of the original dataset) at a major risk of bad classification of the attacks that are not seen at the time features are selected. In a previous paper that appeared in the *security and privacy track* of an information and communication technologies (ICT) quality conference [7], we documented a preliminary experiment on the use of a single autoencoder for intrusion detection. Here we present a much wider study, which addresses the effect of the seed, a deeper investigation on the training loss and feature selection – as affected by the seed – and the use of four hardware platforms. Overall, they contribute to more comprehensive experiments and findings along different directions on the subject. Experiments indicate that, in spite of the good theory on the topic, successful IDSes can be achieved only through extensive practice.

The rest of this paper is organized as follows. Sect. 2 presents related work in the area. Sect. 3 deals with the basics of autoencoders, the proposed IDS approach with a single autoencoder and the reference dataset. Sect. 4 introduces the issues related to design and training of the autoencoder-based IDS. Sect. 5 and Sect. 6 discuss the experiments and results on the effect of the seed, the hardware platform and feature selection. Sect. 7 addresses the threats to validity of our study, while Sect. 8 closes with lessons, conclusions and directions of future research.

2 Related work

Intrusion detection systems (IDS) are part of the second defense line of a system. The first line of defense, according to [8], are the prevention techniques, such as authenticity, cryptography, secure routing and access control. If an intrusion breaks through the first line of defense, then the second line defense is the IDS, which prevents both inside and outside attacks harming the network resources directly or indirectly. Despite decades of research and development, existing intrusion detection systems still face challenges in improving the detection, reducing the false positives and – possibly – detecting unknown

[1] The seed is the *initial point* of the sequence of values generated by the pseudorandom number generator (PRNG)

attacks. In order to solve the above problems, researchers and practitioners have put forward many machine learning and deep learning approaches for intrusion detection [9]. These techniques can cope with large-scale and multi-dimensional data by automatically reducing the network traffic complexity and by finding the correlations among data without human intervention.

Unfortunately, due to obvious confidentiality and non-disclosure reasons, publicly-available data related to spontaneous security attacks and incidents are lacking. In consequence, research on network, system and application security is often carried out by experimentation on data collected through "lab-made" intrusions. In particular, research on intrusion detection systems customarily relies on public intrusion detection datasets [10], collected by network tools in a controlled testing environment trying to mimic realistic attack conditions. Traffic data might be available in a raw format, such as *pcap* packet data files, more "refined" formats, such as network flows organized in comma-separated values (*csv*) files – specially crafted to apply modern machine learning techniques – or both. In the latter case, each record pertains to a flow and the label states if it is an attack or not. An example of an intrusion detection dataset, flow-based and widely used in literature, is certainly CICIDS2017 [11]. Released by the Canadian Institute for Cybersecurity (CIC) in 2017, it simulates real-world network data and uses the tool CICFlowMeter – more on this later– to produce labeled flow records. Other known flow-based intrusion detection datasets are UNSW-NB15 [12], UGR'16 [13] and USB-IDS-1 [14].

Traditional network-centered security solutions have relied on predefined signatures or system models for known threats [15]. Over the last few years, there has been a growing interest in applying machine learning to network security for misuse detection as well as anomaly detection. In general, these approaches can be denoted as *supervised* or *unsupervised*, depending on whether there is the need to train the algorithm on labeled instances or not. Many machine learning solutions use supervised learning, where an algorithm is trained on labeled data points, thus determining a function to map points to classes. Many supervised approaches rely on a limited number of classifiers or a single classifier and achieve outstanding performance – detection rate close to 100% [16]. A large number of supervised methods also exploit artificial neural networks [17]. In the case of unsupervised techniques, instead, there is no need for labeled data points during the training phase. In this context the aim is to find the hidden structure of unlabeled data. Indeed, the vast majority of the unsupervised detection schemes proposed in the literature are based on clustering and outlier detection [18], [19].

Autoencoders are neural networks capable of learning features from unlabeled data by automatically uncovering the underlying structure of the data and by removing sources of variation in the input. Autoencoders were first developed as *nonlinear* extension of the standard linear principal component analysis (PCA) in order to make dimensionality reduction [20]. Kunang et al. in [21] use an autoencoder for feature extraction and the SVM algorithm for multi-class classification. Overall, the model achieves an accuracy of

86.96% and precision of 88.65%. In [22] the Authors propose a novel HCPDP (Heterogeneous Cross-Project Defect Prediction) method called Multi-Source Heterogeneous Cross-Project Defect Prediction (MHCPDP). MHCPDP uses an autoencoder to extract the intermediate features from the original datasets instead of simply removing redundant and unrelated features, and adopts a modified autoencoder algorithm to make instance selection for eliminating irrelevant instances from the source domain datasets.

Autoencoders can be used also for anomaly detection tasks. In this context they are mainly the components of more complex neural networks, specially crafted with the aim to design sophisticated detectors. In [23] the authors propose Kitsune, an unsupervised learning approach to detect attacks online. Kitsune's core algorithm is KitNet, which uses a collection of auto-encoder neural networks to distinguish between normative and abnormal traffic. The approach involves the integration of multiple autoencoders into a classifier. Experimental results show that Kitsune is effective with different attacks, and its performance is as outstanding as offline detectors. In [24] the authors show an effective deep learning method, namely autoencoder-IDS (AE-IDS) based on the random forest algorithm. In particular, they use random forest to select the actual features from the original dataset. The main innovation of the approach lies in the combination of 3-layer shallow autoencoders and traditional unsupervised machine learning clustering algorithm. The experimental results show that the proposed approach, evaluated by means of the CSE-CIC-IDS 2018 dataset, is superior to traditional machine learning methods in terms of easy training, strong adaptability and high detection accuracy. In [25], instead, an autoencoder-based anomaly detection method is proposed that extracts behavior snapshots of the network and uses deep autoencoders to detect anomalous network traffic produced from compromised IoT devices. Finally, Zhong et al. [26] propose an anomaly intrusion detection framework based on multiple deep learning methods. The authors use an autoencoder in order to identify anomaly scores of network traffic. The data with the anomaly score tag is then used to train a long short-term memory (LSTM) predictor. The measurement analysis shows that the approach gives better accuracy than other state-of-the-art algorithms. It is worth pointing out that all the aforementioned autoencoder systems adopt fairly sophisticated infrastructures. The objective of this paper is instead to discuss the adoption of a simpler solution based on the use of a single autoencoder.

As for the feature selection task, it can be performed by either *supervised* or *unsupervised* methods, depending on whether the label of the data points (if any) is used or not, respectively. The papers [27], [28] and [29] are general surveys on feature selection. Unsupervised feature selection methods are instead surveyed in [30]. A comprehensive study of feature selection aimed at intrusion detection systems is presented in [31] with experiments on the NSL-KDD and UNSW-NB15 datasets. In paper [32] a combination of filter-based feature reduction algorithms is applied to the CICIDS2017 DoS dataset, the same dataset used as a testbed in this paper. The proposed method results

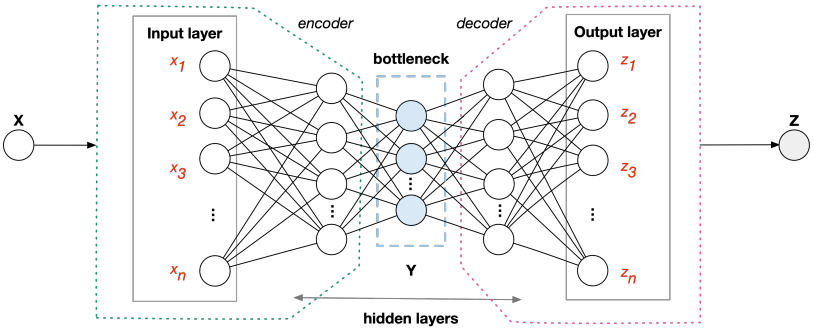


Fig. 1: Representation of an autoencoder.

in 24 reduced features and to an improvement of detection performance using a rule-based classifier. However, a reduction to only 24 features (out of 77) is excessive for an autoencoder, which by design is able to perform dimensionality reduction, as discussed in the following.

3 Proposed approach and dataset

3.1 Background on autoencoders

This study is based on the use of deep autoencoders, i.e., a specific type of feedforward neural network where the *input layer* has the same length as the *output layer*. The middle (hidden) layer of an autoencoder is also known as the *bottleneck layer* and its dimension is lower than the input/output layer. An autoencoder consists of two parts: encoder and decoder. Let $\mathbf{x}$ be a vector of n real numbers $[x_1,x_2,...,x_n]$, such as the flow records representing network traffic for the datasets used in the experiments. The encoder maps $\mathbf{x}$ to a code vector – or hidden representation – $\mathbf{y}$ at the bottleneck layer. On the other hand, the decoder transforms $\mathbf{y}$ into a vector of n, i.e., the same size of $\mathbf{x}$, real numbers $\mathbf{z} = [z_1, z_2, ..., z_n]$. Fig. 1 represents an autoencoder with three hidden layers.

Encoding-decoding formulas are given in Eq. 1 and Eq. 2. They represent the case of a "basic" autoencoder with only one hidden layer:

$$\mathbf{y} = \sigma(W\mathbf{x} + b) \tag{1}$$

$$\mathbf{z} = \sigma'(W'\mathbf{y} + b') \tag{2}$$

where W, W', b and b' are weight matrices and bias vectors, while σ and σ' are activation functions.

An autoencoder compresses the input into a lower-dimensional representation at the *bottleneck* layer and then it reconstructs the output from the representation. Deep learning can be applied to autoencoders. In particular,

multiple hidden layers can be used to provide depth: the resulting network is known as *deep* or *stacked autoencoder* [33].

In the autoencoder terminology, $\mathbf{z}$ is called the reconstruction of the input vector $\mathbf{x}$. In fact – once properly trained – the purpose of an AE is to "reconstruct" the input at the output layer. The "goodness" of the reconstruction is summarized by the reconstruction error (RE), which measures the difference between the output $\mathbf{z}$ and the originating input $\mathbf{x}$:

$$RE = \frac{1}{n}\sum_{i=1}^{n}(z_i - x_i)^2 \tag{3}$$

where z_i and x_i (with $1 \leq i \leq n$) denote the components of the output and input vector, and n is the dimensionality.

3.2 Single autoencoder-based IDS approach

Our IDS proposal, differently from most work currently available in the literature, relies on the use of a *single* autoencoder network for the detection of intrusions. In general, an autoencoder is trained by means of a given set of points, i.e., the training set of a typical machine learning experiment. Each point $\mathbf{x}$ of the training set is fed to the autoencoder, and weight matrices and bias vectors, i.e., W, W', b and b', are progressively adjusted in order to minimize the difference between $\mathbf{x}$ and its reconstruction $\mathbf{z}$. After training, the autoencoder will reconstruct accurately i.e., with low RE, future points "similar" to those used for training. Based on this principle, in order to pursue an IDS approach we train the autoencoder solely by means of normal (used as a synonym for benign through the rest of the paper) data points, i.e., flow records related to traffic collected under "normal" network operations (i.e., not subject to any attack); after training, the autoencoder – embedding a model of the "normal profile" – can identify any instance not conforming to the model as a potential intrusion.

The reconstruction error is a convenient indicator to detect intrusions. Since the autoencoder is trained using only normal data points, it will generate (i) *low* RE (good reconstructions) for future normal inputs, and (ii) *high* RE (bad reconstructions) for intrusions. In fact, when the autoencoder attempts to process a data point – a flow record in this study – that deviates from the norm, it will report an increment of the RE because it was not trained to reconstruct intrusions. The approach adopted in this paper does not need anomalies at training time and falls within the larger scope of semi-supervised anomaly detection [34]. As for any anomaly detection technique assigning a score (RE in this study) to the data points to be assessed, we rely on a cut-off detection threshold to discriminate normal points from intrusions. In particular, intrusion detection is based on the use of the threshold value: the data points producing RE values under the threshold are considered normal, and those with REs above the threshold are marked as intrusions. The value of the detection threshold is an outcome of the training phase: as such, it is inferred from normal data points as described in Sect. 3.3.

As previously said, the rationale underlying our approach is that any divergence with respect to the "benign behavior" – learned by the autoencoder – leads to a high reconstruction error, making it possible to recognize attack flow records. We will not discuss here if an AE could be some sort of "universal" detector. The issue is tough; however, at the state of the art there is evidence that, even if an AE can be fooled by a hand-crafted or adversarial learning-produced attack, it tends to behave better than supervised networks for unseen attack flows (see for example the experiments reported in [35] and [5]).

3.3 Threshold selection method

The idea of leveraging a single and relatively simple neural network and to train it with normal traffic (which makes it possible to detect intrusions because of their divergence from the "normal" behavior the AE was trained on) is indeed fascinating. In our previous proposals [35], the threshold was set in a *supervised* manner, reserving a small part of the dataset in hand to obtain an optimal balance between false positives (benign records with RE falling over the threshold) and false negatives (attack records with RE under the threshold). The results were encouraging (F1 score = 0.942), but lower than those obtained by supervised detection methods [36]; a further improvement exploiting three AEs led to better performance [37].

In this paper, thresholds are set in *semi-supervised manner* (without any cognition of the attack flows during training) by considering the outliers produced in the reconstruction of the benign flows used in the training step. It should be noted that there are several practical challenges that undermine the selection of a suitable threshold in a semi-supervised training setting. Assembling a "reliable" database of normal points for training purposes is a complex matter. For example, normal points might be fraught by uncommon behaviors and/or outliers being accidentally included within the normal points. The labeling might be imperfect, i.e., due to intrusions being occasionally labeled as benign behaviors. In consequence, the idea behind our method is to clear out as many "strange" – although normal – training points as possible *before* computing the threshold: though belonging to normal data, spurious "out-of-the-crowd" behaviors will be more similar to intrusions rather than to normal points. An incorrectly selected threshold (either too low or too high) might cause misclassifications.

The threshold is computed through a small (i.e., 10%) disjoint subset of the training set, which we call the **threshold set**. The method is represented in Fig. 2 and consists of the following steps:

1. AE training: the autoencoder undergoes the standard semi-supervised training procedure described above, which allows it to learn a profile of the benign flow records;
2. outlier detection: an *outlier detection* algorithm is applied to the threshold set in order to discriminate inliers from outliers;

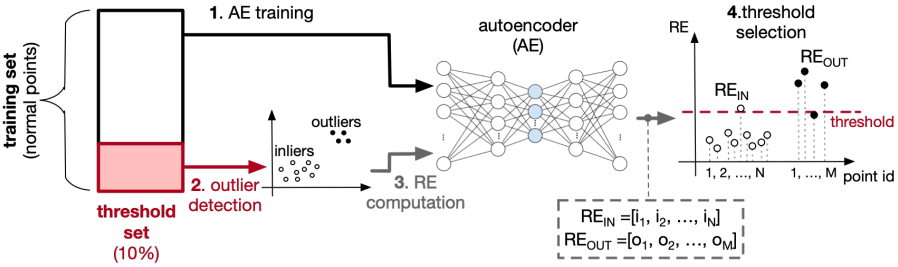


Fig. 2: Steps of the proposed threshold selection method.

3. RE computation: inliers and outliers are fed to the autoencoder to obtain the corresponding REs, which produces two separate vectors: RE_{IN} and RE_{OUT} of inliers and outliers, respectively;
4. threshold selection: the detection threshold is obtained through a sensitivity analysis performed with RE_{IN} and RE_{OUT}.

At first, the threshold is initialized with the maximum RE in RE_{OUT}; then, the threshold is progressively lowered until it is found an "equilibrium" between inliers and outliers, i.e., inliers whose RE is below the threshold against outliers characterized by a RE above the threshold. In this study we use the isolation forest [38], although the threshold selection method *does not* mandate a specific outlier detection algorithm.

3.4 IDS evaluation metrics

The overall performance of our autoencoder-based IDS is measured by analyzing the typical metrics of *recall* (R), *precision* (P), *false positive rate* (FPR), and *F1 score*. The metrics are computed with a labeled test set, i.e., a set of "held-out" benign and attack records, i.e., records not seen at all by the autoencoder during the training phase. These metrics are computed as follows:

$$R = \frac{TP}{TP + FN} \qquad P = \frac{TP}{TP + FP} \tag{4}$$

$$FPR = \frac{FP}{FP + TN} \qquad F1\ score = 2 \cdot \frac{P \cdot R}{P + R} \tag{5}$$

where True Positive (TP) and True Negative (TN) represent the records that are correctly classified, while False Positives (FP) and False Negatives (FN) indicate misclassifications. For example, TP is the set of attack points whose RE is higher than the detection threshold; similarly, TN is the set of normal points whose RE is lower than the threshold.

3.5 Dataset

The CICIDS2017 dataset consists of benign traffic synthesized by the abstract behavior of 25 users mixed with malicious traffic from many common attacks.

Table 1: The features of a network flow produced by CICFlowMeter.

Feature	short	Feature	short	Feature	short
Flow ID	f1	Fwd IAT Std	f29	ECE Flag Count	f57
Source IP	f2	Fwd IAT Max	f30	Down/Up Ratio	f58
Source Port	f3	Fwd IAT Min	f31	Average Packet Size	f59
Destination IP	f4	Bwd IAT Total	f32	Avg Fwd Segment Size	f60
Destination Port	f5	Bwd IAT Mean	f33	Avg Bwd Segment Size	f61
Protocol	f6	Bwd IAT Std	f34	Fwd Avg Bytes/Bulk	f62
Timestamp	f7	Bwd IAT Max	f35	Fwd Avg Packets/Bulk	f63
Flow Duration	f8	Bwd IAT Min	f36	Fwd Avg Bulk Rate	f64
Total Fwd Packets	f9	Fwd PSH Flags	f37	Bwd Avg Bytes/Bulk	f65
Total Backward Packets	f10	Bwd PSH Flags	f38	Bwd Avg Packets/Bulk	f66
Total Len. of Fwd Packets	f11	Fwd URG Flags	f39	Bwd Avg Bulk Rate	f67
Total Len. of Bwd Packets	f12	Bwd URG Flags	f40	Subflow Fwd Packets	f68
Fwd Packet Len. Max	f13	Fwd Header Len.	f41	Subflow Fwd Bytes	f69
Fwd Packet Len. Min	f14	Bwd Header Len.	f42	Subflow Bwd Packets	f70
Fwd Packet Len. Mean	f15	Fwd Packets/s	f43	Subflow Bwd Bytes	f71
Fwd Packet Len. Std	f16	Bwd Packets/s	f44	Init_Win_bytes_forward	f72
Bwd Packet Len. Max	f17	Min Packet Len.	f45	Init_Win_bytes_backward	f73
Bwd Packet Len. Min	f18	Max Packet Len.	f46	act_data_pkt_fwd	f74
Bwd Packet Len. Mean	f19	Packet Len. Mean	f47	min_seg_size_forward	f75
Bwd Packet Len. Std	f20	Packet Len. Std	f48	Active Mean	f76
Flow Bytes/s	f21	Packet Len. Variance	f49	Active Std	f77
Flow Packets/s	f22	FIN Flag Count	f50	Active Max	f78
Flow IAT Mean	f23	SYN Flag Count	f51	Active Min	f79
Flow IAT Std	f24	RST Flag Count	f52	Idle Mean	f80
Flow IAT Max	f25	PSH Flag Count	f53	Idle Std	f81
Flow IAT Min	f26	ACK Flag Count	f54	Idle Max	f82
Fwd IAT Total	f27	URG Flag Count	f55	Idle Min	f83
Fwd IAT Mean	f28	CWE Flag Count	f56		

In order to create the dataset, the proposing group used a laboratory environment with attacker and victim networks. The attacker was a Kali Linux node and the victim an Ubuntu 16.04 system running an Apache web server. The data capture period started at 9 a.m., Monday, July 3, 2017, and ended at 5 p.m., Friday, July 7, 2017, for a total of 5 days. *Monday* is the "normal day" and contains only benign traffic; in the morning and afternoon of Tuesday, Wednesday, Thursday, and Friday, in addition to normal traffic, attacks were performed. These attacks belong to the categories Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS. DoS attacks *hulk*, *slowloris*, *slowhttptest* and *goldeneye* were performed and captured on "Wednesday", i.e., the "DoS day". The dataset is a flow-based one; network flows are obtained by means of CICFlowMeter[2], which derives from a tool originally conceived to recognize the type of encrypted traffic and provides detailed information on the flow of packets occurring and their timing. In particular, network flow records consist of categorical and numeric features that provide context data and summary statistics computed from the packets exchanged between a source computer and a destination across a network. Each record of the dataset is a labeled flow, identified by 84 features (attack label included). The label states if the record belongs to normal traffic or attacks. Table 1 shows the 83 features associated with a flow by CICFlowMeter. It is a fact that this information can be successfully exploited to recognize malign flows, which is the primary aim of an IDS.

Extensive research on erroneously classified flows leads us to discover that often the attack flows contained in CICIDS2017 do not really harm a correctly-configured server [2] and that the flows produced by the original release of the CICFlowMeter tool – commonly used in the context of IDS research – contain

[2] https://github.com/ahlashkari/CICFlowMeter

inexplicable flows. These are actually fragments of an incorrectly truncated flow. A patch to the latter issue was recently provided in [39], along with a new version of both CICIDS2017 and CICFlowMeter: the experimentation presented in this paper is based on the fixed version of CICIDS2017[3].

We consider 490,968 flow records related to normal traffic and Denial of Service (DoS) attacks of the CICIDS2017 capture dating Wednesday, July 5, 2017. Overall, the records are split into three disjoint subsets used for *training*, *validation* and *test* by a stratified sampling strategy with no replacement. This means that the ratio between benign and attack classes of the total flow records is preserved in the splits. Flow records are divided as follows:

- CICIDS-TRAINING: *70%* of the total (i.e., 343,680) divided into 223,430 BENIGN and 120,250 ATTACK flow records. 10% of this set is the "threshold set", meant for the threshold selection process;
- CICIDS-VALIDATION: *15%* of the total (i.e., 73,644), divided into 47,877 BENIGN and 25,767 ATTACK flow records;
- CICIDS-TEST: *15%* of the total (i.e., 73,644), divided into 47,877 BENIGN and 25,767 ATTACK flow records.

The learning process leverages training-validation sets. It must be noted that attack records are removed from CICIDS-TRAINING and CICIDS-VALIDATION according to the adopted semi-supervised learning approach.

4 Implementation and training issues

The autoencoder has been implemented in Python with the ubiquitous Keras deep learning framework, which is backed by Tensorflow. As for any learning experiment, we start with an initial set of parameters (e.g., number of layers, neurons per layers, activation functions, optimizer and epochs) to form the autoencoder. The autoencoder is trained with CICIDS-TRAINING: the IDS model and threshold obtained after training are tested with CICIDS-VALIDATION in order to validate the choice of the parameters. The training-validation step is iterated until the optimal parameters are found. It is worth noting that CICIDS-TEST plays no role during training and validation, and it is used to compute the results presented in Sect. 5 and 6. The AE implementation, the dataset splits used in the experiments and a pre-trained model are publicly available on our website[4].

As for the features adopted at this stage of the experimentation, it is worth noting that out of the 83 features in Table 1, six of them (f1-f5, f7) must be discarded outright for detection purposes. For example, the Flow ID (f1) does not make any sense for detection; moreover, it is too easy to detect attack flows in a dataset by leveraging the IP of the attacker (f2) or the victim (f4). The initial set of experiments is thus conducted with the 77 features in Table 2, while the use of feature selection will be addressed in Sect. 6.

[3] https://downloads.distrinet-research.be/WTMC2021/tools_datasets.html
[4] http://idsdata.ding.unisannio.it/tools.html

Table 2: Features used in the initial experiments.

77-features set
f6, f8, f9, f10, f11, f12, f13, f14, f15, f16, f17, f18, f19, f20, f21, f22, f23, f24, f25, f26, f27, f28, f29, f30, f31, f32, f33, f34, f35, f36, f37, f38, f39, f40, f41, f42, f43, f44, f45, f46, f47, f48, f49, f50, f51, f52, f53, f54, f55, f56, f57, f58, f59, f60, f61, f62, f63, f64, f65, f66, f67, f68, f69, f70, f71, f72, f73, f74, f75, f76, f77, f78, f79, f80, f81, f82, f83

Table 3: Parameters of the autoencoder.

neurons per layer	*N*–48–24–8–24–48–*N*	validation_split	0.10
initializer	RandomNormal	shuffle	True
regularizer	default parameters	batch_size	1024
connection	fully connected	epochs	90
activation functions	*N/A*, relu, relu, relu, relu, relu, tanh		
optimizer	rmsprop (default parameters)		
loss function	mean squared error		

4.1 Parameter selection

In the following, we discuss our practices for setting up the autoencoder for successful intrusion detection based on the data in hand. Given the input and output layers (whose number of neurons is equal to the number of features considered) it is necessary to choose a suitable form factor, i.e., the number of hidden layers and neurons for each layer. Unfortunately, there are no rules to guide this choice, and so the only way is to proceed by iterating the training-validation step, as we did in this study.

Almost unexpectedly, we found out that the number of hidden layers is not a particularly critical parameter. It is possible to obtain low RE with 3 hidden layers or with 5 hidden layers as well. The main criticality is the number of neurons at the bottleneck, which is intended to hold the encoded flow state. In the case of network flows considered here the best results are obtained with a bottleneck layer made of 6 up to 8 neurons. In the rest of the paper, we will present results relative to a deep autoencoder with 5 hidden layers, each consisting of 48–24–8–24–48 neurons implementing the *rectified linear unit* (relu) activation function, as shown in Fig. 3.

The overall parameters of the autoencoder are reported in Table 3. The number of neurons of the input-output layer is denoted by N and depends on the number of features; the activation function of the output layer is the *hyperbolic tangent* (tanh). The loss function is the mean squared error between the output and input of the autoencoder: this matches the definition of the reconstruction error (RE) presented above.

4.2 Issues with the training process

The training procedure of the autoencoder is straightforward once the dataset has been suitably partitioned; more important, training is performed automatically by the Keras-Tensorflow deep learning software. The weights of the

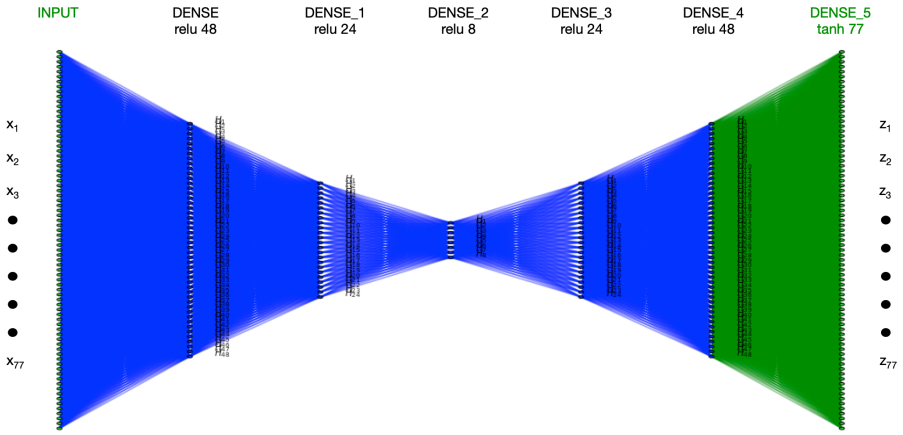


Fig. 3: Representation of the 77-input 48-24-8-24-48 autoencoder.

neurons are typically assigned random values by the pseudorandom number generator (PRNG) at the beginning of the training process. The seed is the initial point of the sequence of values generated by the PRNG. In Keras-Tensorflow the seed of the PRNG can be (i) either supplied by the user or (ii) just left unspecified. If *not* specified, the seed of the PRNG is handled transparently by the machine learning software at each program execution, which means that subsequent training runs – let all the remaining parameters and conditions fixed – will likely lead to different results.

Training consists in feeding CICIDS-TRAINING to the autoencoder in the form of fixed-size batches (whose size is regulated by the batch size parameter) for a given number of times (the number of epochs or simply epochs, for short, parameter). While doing so, the deep learning software adjusts the weights of the neurons by reducing progressively the loss function without user intervention, setting aside a small ratio of reserved data to validate the optimization actions performed – modifications of the weights in the network – so as to signal overfitting. However, things are not so simple, due to issues that could arise and that should be suitably considered in order to obtain good detection results from the autoencoder:

- the seed of the PRNG: while we would expect a "negligible" variability of the results from run to run due to not specifying the seed, this might not be the case for a semi-supervised autoencoder for intrusion detection;
- different CPUs (or GPUs) will lead to different exploitation of the threads used in the training phase and – at least in theory – might lead to different weights of the neurons in the network;
- there is no guarantee that obtaining low loss values as a result of the training phase leads to good intrusion detection performance.

These three issues – overlooked by the related literature – are separately dealt with in Sect. 5, so as to give practical advices for successful training of an autoencoder to be used for intrusion detection purposes.

5 Results on the training process

The phenomena giving rise to inconsistent intrusion detection results are described and proven experimentally across different seed values and computing platforms; moreover, we complement the experiments by a discussion on the validity of relying on "low" loss values as indicators of successful training. The experiments will answer these four research questions related to training issues (TRAIN-RQs):

- **TRAIN-RQ1**: has the seed of the PRNG used at training time a valuable effect on the autoencoder detection performance?
- **TRAIN-RQ2**: the weights of the autoencoder network produced are dependent upon the hardware platform used for the training procedure?
- **TRAIN-RQ3**: the final value of loss obtained at training time is directly correlated to the intrusion detection performance of the AE?
- **TRAIN-RQ4**: given a non-manageable effect of seed, hardware and loss value on the AE intrusion detection performance, which is the suggested learning procedure?

Both the experiments and RQs are supported by practical results obtained on typical computing hardware platforms:

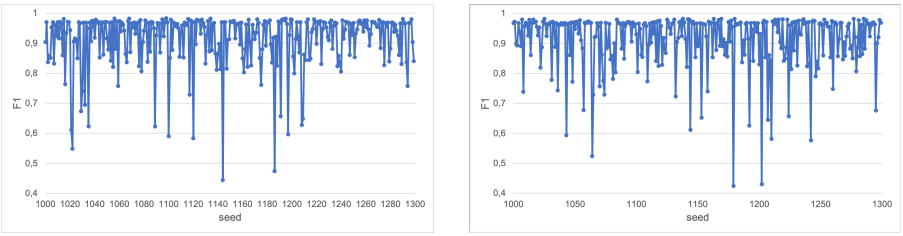
1. iMac model 14,1, 2,7 Ghz Intel Core i5 4-Core, 16GB RAM;
2. iMac model 19,1, 3 GHz Intel Core i5 6-Core, 16GB RAM;
3. workstation Lambda-VECTOR, 3.50 GHz AMD Ryzen Threadripper PRO 3975WX 32-Cores, GPU disabled;
4. workstation Lambda-VECTOR, 3.50 GHz AMD Ryzen Threadripper PRO 3975WX 32-Cores, NVIDIA RTX A5000 GPU.

In order to avoid any potential bias caused by the deep learning software, all the tests that follow have been performed with Python 3.8.2, Tensorflow 2.8.0, Keras 2.8.0, Scikit-Learn 1.0.2.

5.1 Effect of the seed of the PRNG

The effect of the seed used to start the random sequence on the weights of the neurons in the training phase is a relatively-well known phenomenon. If the seed is not specified when the training starts, the obtained weights are different across subsequent runs. One would accordingly aspect small variations in the obtained network behavior. Unfortunately, it is not so. The intrusion detection performance of the obtained networks will instead vary wildly with the seed.

Fig. 4 shows the effect of seed change in experiments performed on the platforms 1) and 3) above. Given a value of the seed, the autoencoder is (i) trained



(a) 2,7 GHz Intel Core i5 quad-core (b) 3,5 GHz AMD 3975WX 32-core

Fig. 4: F1 for different seed values.

with CICIDS-TRAINING, and (ii) tested with CICIDS-TEST in order to compute R, P, F1 and FPR. It should be noted that the remaining parameters – if not the seed – are fixed as per Table 3. We test the values of seed in 1000 to 1299 and the resulting detection performance with CICIDS-TEST (for sake of simplicity summarized here by the sole F1 index) is plotted in the diagrams. For each run, the specific value of the seed is set with the random.set_seed(s) function (with s denoting an integer) of TensorFlow. The results show a high variability of F1, that sometimes falls down from "satisfactory" values close to 1 down to 0.8 or even less. Furthermore, the plots obtained depend on the hardware platform (more on this later).

Answer to TRAIN-RQ1: The effect of the seed cannot be ignored. The detection capabilities of the AEs obtained with different values of seed present significant variations, sometimes falling under acceptable performance levels.

5.2 Effect of the hardware platform

As previously mentioned, the effect of the seed is intertwined with the variability of the detection performance due to different hardware platforms. In "no-GPU" systems, this is due to the use by the Tensorflow engine of a different number of threads depending on the number of available cores. As a result, the detection performance for the same seed value can be fairly different across different CPUs. A draconian solution is to force Tensorflow to exploit a single thread[5].

On systems provided with GPUs (which are typically exploited by default by Tensorflow) the problem is even tougher. In this case, not only the learning behavior varies with the hardware used, but it gives non-deterministic results on the same platform. Stated another way, each training on the same hardware produces different weights and, in the end, different detection performance. This phenomenon is a consequence of the non-additivity of floating point additions and of the use of asynchronous atomic sums in the Tensorflow run-time, and cannot be avoided without serious performance losses. A complete report on the status of Tensorflow non-determinism for NVIDIA GPUs can be found

[5]This can be obtained by calling tf.config.threading.set_inter_op_parallelism_threads(1) and tf.config.threading.set_intra_op_parallelism_threads(1), but this is detrimental for learning times.

Table 4: Classification performance for different architectures, seed=1134

Architecture	R	P	F1	FPR	training loss
2,7 GHz Intel Core i5 4-core	0.988	0.976	0.982	0.013	1.3240e-04
3 GHz Intel Core i5 6-core	0.802	0.977	0.881	0.0102	1.4006e-04
3,5 GHz AMD 3975WX 32-core	0.871	0.972	0.919	0.013	1.2080e-04
NVIDIA RTX A5000 (1)	0.785	0.983	0.873	0.007	1.2453e-04
NVIDIA RTX A5000 (2)	0.758	0.974	0.853	0.011	1.2768e-04
NVIDIA RTX A5000 (3)	0.991	0.968	0.979	0.018	1.2326e-04
NVIDIA RTX A5000 (4)	0.986	0.979	0.983	0.011	1.2087e-04
NVIDIA RTX A5000 (5)	0.498	0.960	0.656	0.011	1.1845e-04

on github[6]. The implication of this non-deterministic behavior is that a "successful" training (i.e., leading to good detection results) is not repeatable, and so the produced model has to be saved beforehand to be exploited in the future, if it gives good results at testing time.

Fig. 5 shows the plots of the reconstruction error for the records in CICIDS-TEST obtained with the same seed value – 1134, a "fairly good value" for platform 1) – of the four platforms at hand. The test has been executed multiple times on the GPU system to confirm the non-deterministic behavior. It is possible to see at a glance the extreme variability of the performance obtained, which ranges from very good (a, f, g) to decent (d, e) to absolutely unsatisfactory (h), which means the autoencoder generates a reconstruction error below the detection threshold for a significant number of attack records of CICIDS-TEST (thus classified as benign). For the sake of completeness, Table 4 shows the details of the performance results (recall, precision, F1 value, FPR) obtained in the eight experiments. The rows of Table 4 correspond orderly to the eight subfigures of Fig. 5, (a) to (h). The rightmost column of the table also shows the *final* values of the training loss obtained after the 90 epochs consistently used for training the eight networks. These values will be considered in the next subsection.

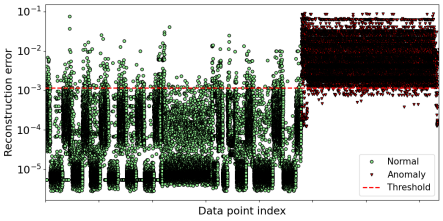
Answer to TRAIN-RQ2: The hardware platform has a non-negligible effect on the AE network obtained. Different CPUs lead to different trained networks. Moreover, on systems provided with GPU, the outcome of the training step is non-deterministic. This issue affects the final AE detection performance as well.

5.3 Considerations on the final loss value

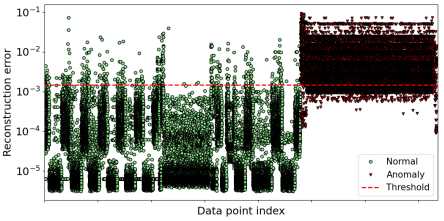
Training aims to reduce the loss value through a given number of optimization steps (again, epochs). Our experimentations, a small subset of which is reported here, clearly point out that all potential training attempts on the same input data lead to fairly similar loss values. Whatever the hardware used or the random seed, it works. The bad surprise is that at equal values of training losses do not reflect into equal abilities to detect attack flow records.

The analysis of the final loss values (i.e., the loss values at the completion of the training step) reported in the rightmost column of Table 4 shows

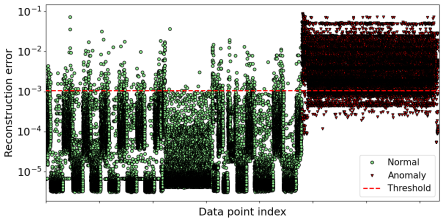
[6] https://github.com/NVIDIA/framework-determinism



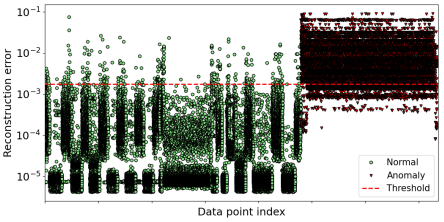
(a) 2,7 GHz Intel Core i5 4-core



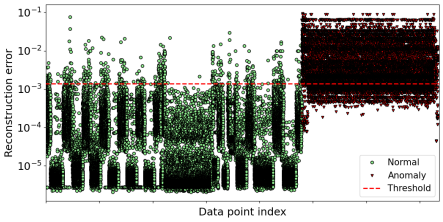
(b) 3 GHz Intel Core i5 6-core



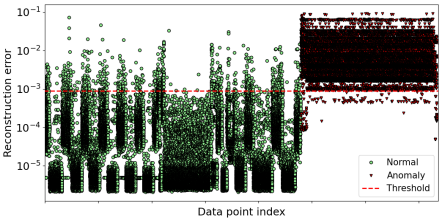
(c) 3,5 GHz AMD 3975WX 32-core



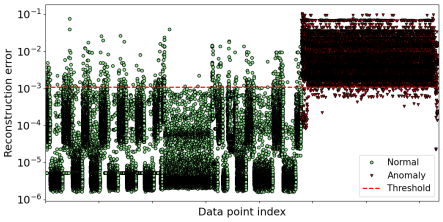
(d) NVIDIA RTX A5000 (1)



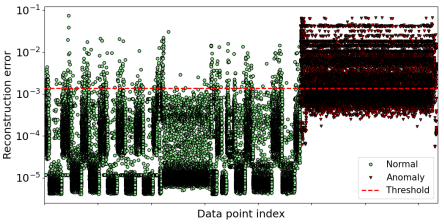
(e) NVIDIA RTX A5000 (2)



(f) NVIDIA RTX A5000 (3)



(g) NVIDIA RTX A5000 (4)



(h) NVIDIA RTX A5000 (5)

Fig. 5: Reconstruction error with the test set for different architectures, seed=1134.

that lower losses do not necessarily correspond to higher detection performance. For example, it can be noted that the best configuration presented –

NVIDIA RTX A5000 (4), the one corresponding to subfigure (g) in Fig. 5 – is characterized by a loss value (1.2087e-04) higher than the one of configuration NVIDIA RTX A5000 (5), subfigure (h), (1.1845e-04) offering very poor detection performance.

The reconstruction error (or the mean squared error measured by Keras, which is the same) is a mean of squares, extended to all the features fed to the autoencoder. The loss being equal, the contributions of the single features may be distributed differently. Possibly, one of the non-deterministic distribution of weights might lead to high error on the features that are most fit to recognize a given type of attack. There is no possible solution, as the attempt to provide an "universal" detector makes it impossible to assign higher weights to some of the features when computing the loss, simply because we do not know what could be the most relevant features for an unknown or new type of attack.

Answer to TRAIN-RQ3: The loss value obtained at the end of the training step is not directly linked to the quality of the AE obtained; lower losses do not necessarily correspond to higher detection performance.

5.4 Suggested learning procedure

The experiments presented in the preceding subsections have shown that many factors influence the detection performance of the AE in a non-negligible way. The non-determinism characterizing the learning procedure of neural networks does not necessarily allow to settle the weights in a satisfactory final state. Furthermore, the influencing factors are not under the control of the developer at learning time, as they appear to be linked to the development platform (Keras-Tensorflow) and not to the AE model itself. Further research is indeed necessary on this topic, which appears beyond the scope of this paper at the time. Deep autoencoders may be a great solution for intrusion detection, but they require a multiple-step learning procedure that could allow to discard "unfortunate", poorly-performing, AEs.

Our suggestion for obtaining satisfactory detection performance from the AE is to repeat the learning multiple times on the same hardware environment, checking the behavior of the autoencoders obtained from a not-too-narrow set of seeds using a (labeled) subset of the dataset. Once a sufficiently good seed for a given platform has been found, it has to be fixed for successive learnings or – better, in light of the GPU issues mentioned above – the model saved once and for all, and used as-it-is. However, it is clear that this process leads to a detector able to manage at best the attacks of the dataset in hand, but that *possibly* could be less successful for different types of attack.

Answer to TRAIN-RQ4: Given the uncertainty on the actual AE detection capabilities linked to non-deterministic training factors, it is convenient to reserve a validation subset of the dataset for choosing among the AEs obtained in correspondence of a sufficiently large set of seeds. If GPUs are used, every model obtained has to be saved, because the learning process performed is non-repeatable.

6 Results on the use of feature selection

Feature selection techniques are widely used in intrusion detection. They allow discarding the features that are either redundant or not significantly contributing to the classification. It should be noted that an autoencoder *per se* performs a dimensionality reduction as described in Sect. 3: in consequence – at least in theory – applying a preliminary feature selection "before" the autoencoder might seem redundant. However, given the problems linked to the mean of squares used to compute the loss discussed in Subsect. 5.3, a feature selection could possibly help to obtain (being equal to the final loss) an AE better tuned to the "significant" features and it is worth a try. Moreover, out of the features listed in Table 2, it is very unlikely that all of them are useful for intrusion detection; for example, a few ones are constant through all the dataset and so have no utility for classification. In any case it should be borne in mind that, without knowing the characteristics of the attacks, it is not possible to know which features can be useful for attack flow recognition or not.

In order to understand the role of feature selection in the context of an AE-based IDS, we assess three methods, i.e., ANOVA f-test, Mutual information statistic and Random forest. The experiments presented in the following aim to uncover the relevance of the features with respect to the DoS attacks listed in 3.5, the intertwinement between feature selection and the variability of the results caused by the seed, and the effect on IDS performance. Overall, they can be synthesized in the following research questions (FEATSEL-RQs):

- **FEATSEL-RQ1**: is the relevance of the features the same for every adopted technique and for any type of attack?
- **FEATSEL-RQ2**: discarding less relevant features makes the seed less influential on the intrusion detection performance of the AE?
- **FEATSEL-RQ3**: discarding less relevant features leads to better intrusion detection performance of the AE?

6.1 Analysis method and feature selection techniques

Feature selection can be performed by either *supervised* or *unsupervised* methods, depending on whether the label of the data points (if any) is used or not, respectively. Unsupervised feature selection is based on the examination of a set of flow records, without knowing if they are benign (normative) or malign (attacks). Most typically, clustering techniques are exploited. Due to the fairly high dimensionality of the flow records, we chose the more straightforward supervised techniques, which capitalize on the label (BENIGN or ATTACK). Moreover, consistently with the choice to avoid the use of attack data – even in this tuning phase, our AE never sees the attack flows of the reference dataset – we used the flow records taken from a different dataset, i.e., USB-IDS-1.

USB-IDS-1 is a novel synthetic intrusion dataset developed at the University of Sannio at Benevento (USB), Italy, and freely downloadable for research

use[7]. It contains normal *http* traffic, along with DoS attacks to an Apache web server carried out by means of publicly-available DoS scripts and command line utility programs widely used by the security community. In particular, the attacks available in the dataset are the well known *Hulk*, *Slowhttptest*, *Slowloris* and *TCPFlood*. This makes the USB-IDS-1 dataset *similar* – as far as the type of DoS attacks performed is concerned – to the CICIDS2017 Wednesday capture used in this paper. The details on the collection of USB-IDS-1 and the attacks performed can be found in [14].

As for the supervised feature selection techniques adopted, they are ANOVA f-test, Mutual information statistic and Random forest. The first two are *filter* methods, in that they use statistical techniques to evaluate the relationship between each feature and the target variable (the label of the flow record), and use the obtained scores to discard the features less useful to predict the target. ANOVA is based on the analysis of variance, whereas the Mutual information test is based on information gain. On the other hand, Random forest is an *embedded* method, in that feature selection is obtained as part of learning a data model. Describing the feature selection methods is out of the scope of this paper. The interested reader is referred to the survey papers mentioned in Sect. 2.

6.2 Relevant features by technique and attack

In the following, we leverage the feature selection techniques to find out the most relevant features among those listed in Table 2. The techniques have been implemented with Scikit-learn: they are used to analyze the following subsets of the USB-IDS-1 dataset: i) normal traffic plus *Hulk* attack data, ii) normal traffic plus *Slowhttptest* attack data, iii) normal traffic plus *Slowloris* attack data, iv) normal traffic plus *TCPFlood* attack data. Fig. 6 shows the obtained results, in the form of histograms reporting the score (y-axis) of the 77 features (x-axis). Higher score means higher contribution to the classification of the flow as an attack. As it can be seen at a glance comparing the histograms, the set of the most relevant features depends on the feature selection technique, and it is not uniform across all types of attack.

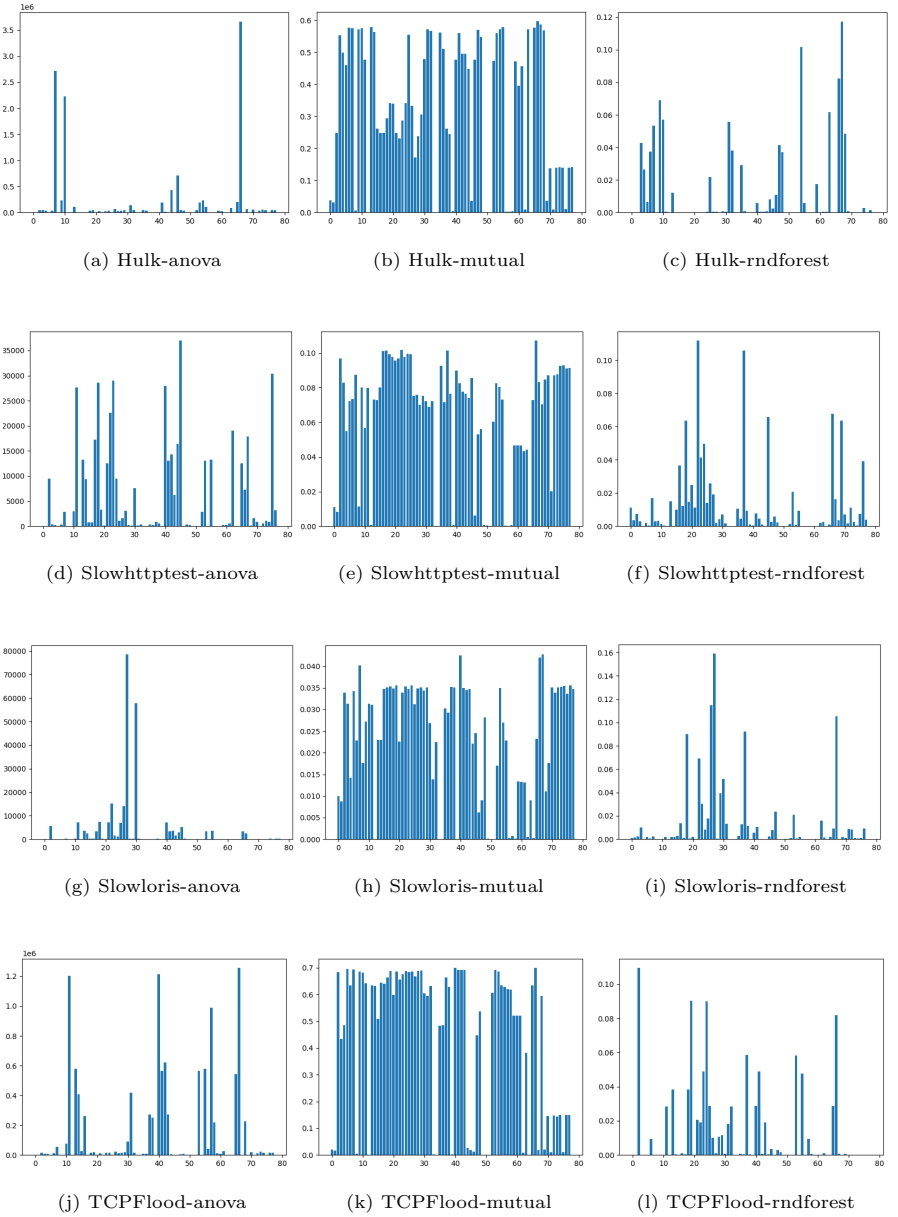
Answer to FEATSEL-RQ1: There no "uniform" view of the most relevant features across the three feature selection techniques, and the most relevant features to be used for detection vary with the type of attack.

Selecting only some of the 77 features makes the detector lose "universality", as (at least in principle) unknown attacks could be spotted by the discarded features. This possibly suggests that only the use of the full feature set can help to recognize never-seen-before attacks (0-day).

6.3 Feature selection and effect of the seed

In order to address FEATSEL-RQ2, we discard the lowest-ranked 10, 20 and 30 features, obtaining "reduced" sets of 67, 57 and 47 features, respectively,

[7] http://idsdata.ding.unisannio.it/



(a) Hulk-anova (b) Hulk-mutual (c) Hulk-rndforest

(d) Slowhttptest-anova (e) Slowhttptest-mutual (f) Slowhttptest-rndforest

(g) Slowloris-anova (h) Slowloris-mutual (i) Slowloris-rndforest

(j) TCPFlood-anova (k) TCPFlood-mutual (l) TCPFlood-rndforest

Fig. 6: Scores of the 77 features by technique: Hulk, Slowhttptest, Slowloris, TCPFlood attacks from the USB-IDS-1 dataset.

as shown in Table 5. It is worth noting that the first 10 features discarded are constant in the CICIDS2017 dataset, and so surely give no contribution

Table 5: Subsets of features used in experiments

67-features set
f6, f8, f9, f10, f11, f12, f13, f14, f15, f16, f17, f18, f19, f20, f21, f22, f23, f24, f25, f26, f27, f28, f29, f30, f31, f32, f33, f34, f35, f36, f37, f41, f42, f43, f44, f45, f46, f47, f48, f49, f50, f51, f52, f53, f54, f55, f57, f58, f59, f60, f61, f68, f69, f70, f71, f72, f73, f74, f75, f76, f77, f78, f79, f80, f81, f82, f83
57-features set
f6, f8, f9, f10, f11, f12, f13, f14, f15, f16, f17, f18, f19, f20, f21, f22, f23, f24, f25, f26, f27, f28, f29, f30, f31, f32, f33, f34, f35, f36, f37, f41, f42, f43, f44, f45, f46, f47, f48, f49, f50, f51, f52, f53, f54, f55, f57, f58, f59, f60, f61, f68, f70, f71, f72, f73, f75
47-features set
f6, f8, f9, f10, f11, f12, f13, f15, f16, f17, f19, f20, f21, f22, f23, f24, f25, f27, f28, f29, f30, f32, f33, f34, f35, f36, f37, f41, f42, f43, f44, f46, f47, f48, f49, f50, f51, f53, f54, f58, f59, f60, f61, f71, f72, f73, f75

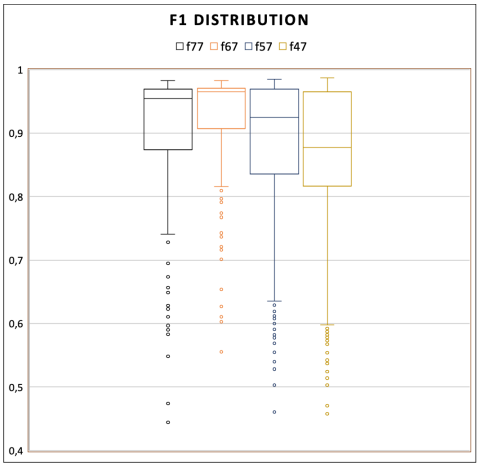


Fig. 7: Boxplot of F1 values for 77, 67, 57 and 47 feature sets.

to predict the label (BENIGN or ATTACK). Stated another way, they are non-informative variables that can potentially affect the effectiveness of the detector.

The experiment presented in 5.1 is repeated on platform 1) using the sets of features above. Given a set of features and the range of seeds 1000-1299, the autoencoder is (i) trained with CICIDS-TRAINING, and (ii) tested with CICIDS-TEST: the resulting F1 score on the test set recorded. It should be noted that – if not the seed – the parameters of the autoencoder are those in Table 3; N, i.e., the number of neurons of the input-output layers, is adjusted case-by-case and set to the number of features.

The boxplots in Fig. 7 show the F1 score obtained switching from the full feature set (F77) to the sets with 67, 57 and 47 (F67, F57, F47 in Fig. 7, respectively). Each boxplot summarizes 300 F1 scores. Interestingly, the lowest dispersion is obtained by the 67-feature set, where – trivially, with no specific

Table 6: Classification performance of the feature sets

	77-features set	67-features set	57-features set	47-features set
R	0.988	0.987	0.986	0.987
P	0.976	0.982	0.984	0.987
FPR	0.013	0.009	0.008	0.007
F1	0.982	0.984	0.985	0.987

selection technique – the constant features were discarded. The 57- and 47-features sets perform worse, with higher dispersion and lowest median of F1 scores. Hence the answer to FEATSEL-RQ2:

Answer to FEATSEL-RQ2: Applying feature selection "before" the autoencoder does not help to obtain a more stable behavior or to make the seed less influential on intrusion detection; the less the features, the higher the dispersion of the F1 score.

6.4 Feature selection and IDS performance

We now consider the IDS detection performance with feature selection, so as to answer RQ3. It turns out that, although feature selection does not mitigate the variability caused by the seed, it can help a bit to achieve good classification performance. Table 6 shows the P, R, FPR and F1 values obtained on the same platform (i.e., platform 1) of Sect. 5) with a different set of features; the best performance is obtained with the 47-features set, with seed = 1062. Most notably, the reduction of features makes it possible to halve the false positive rate.

In Figg. 8, 9, 10 and 11 we present the corresponding plots of RE over the test set and the confusion matrices, for each of the AE instances in Table 6. Even if the plots are very similar, on close examination subtle differences appear in the reconstruction error obtained for the different subsets of normative data. This led to slight performance improvement that characterizes the 47-features AE, signaled by the reduced false positive rate. Perhaps further tuning could help to obtain a slightly higher performance, but at these levels of precision and recall, it is likely to be simply a waste of time, since the misclassified flows are only 674 (i.e, 329 + 345 in Fig. 11) on a total of 73,644 test flows. Moreover, many of these misclassified flows are "tails" of long-duration flows (namely, those generated by slow DoS attacks) that have been truncated after the standard CICFlowMeter timeout of 120 s. This problem is well known and has been recently fixed in a further dataset derived from CICIDS2017 [40].

Answer to FEATSEL-RQ3 : Discarding less relevant features leads to marginal improvements of the intrusion detection performance of the AE; moreover, it is questionable if the marginal improvements obtained can be traded-off for a *potential* bad classification performance on "unseen" (e.g., 0-day) attacks.

The set of features obtained after selection is biased by the attacks available in the dataset adopted; unseen attacks (i.e., those not present in the dataset used to compute the feature rankings) may require a different set of features.

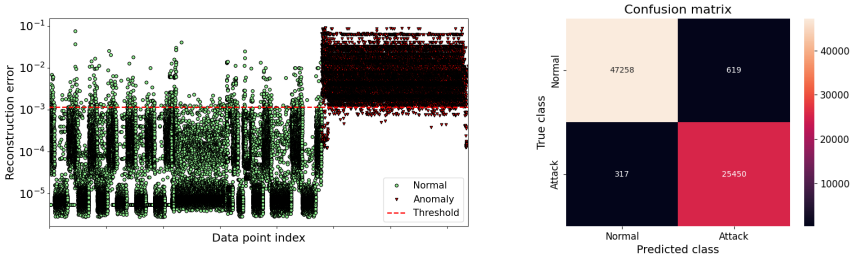


Fig. 8: Reconstruction error and confusion matrix of the 77-features AE.

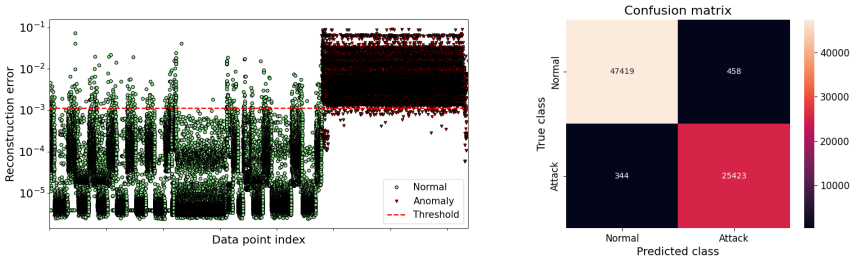


Fig. 9: Reconstruction error and confusion matrix of the 67-features AE.

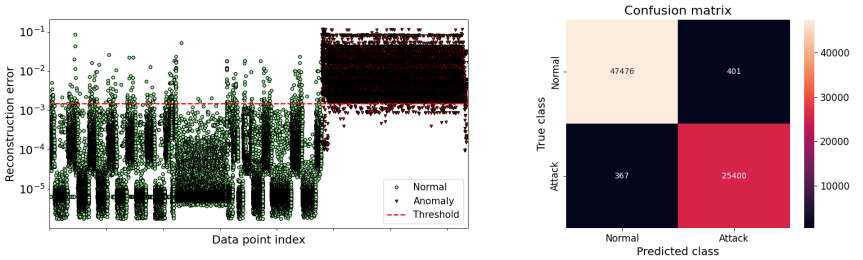


Fig. 10: Reconstruction error and confusion matrix of the 57-features AE.

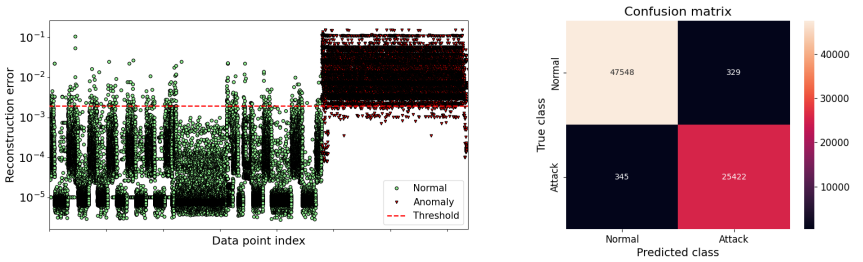


Fig. 11: Reconstruction error and confusion matrix of the 47-features AE.

In our opinion, the decision on whether to use the full feature set or just a subset depends on the frequency and the scope of 0-day attacks suffered.

7 Threats to validity

As for any data-driven study – especially if based on the use of machine and deep learning – there may be concerns regarding the validity and generalizability of the results. We discuss them based on four aspects of validity listed in [41].

Construct validity. The study builds around the intuition that a single autoencoder – if properly trained – can be successfully used for semi-supervised intrusion detection. The approach has the potential to inspire general IDS solutions that do not require attacks at training time. The construct is measured and investigated through a large experimental campaign based on the ubiquitous Keras-Tensorflow deep learning framework, different hardware platforms involving both CPUs and GPUs, and a widely-accepted benchmark by the related literature, i.e., the CICIDS2017 dataset (in particular, a recent fix of the dataset) supplemented by the additional dataset USB-IDS-1. The construct is assessed by considering both the training issues of a deep neural network and the use of feature selection. Overall, the study is supported by well-founded theory and practice, and standard evaluation metrics (recall, precision, false positive rate, F1 score).

Internal validity. Our study implements several countermeasures aiming to mitigate internal validity threats. For example, we made sure to test our autoencoder-based IDS by means of held-out data, i.e., not used at all for training and hyperperameters optimization. Experiments were replicated many times in order to account for the effect of the randomization of some training facets – initialization of neurons and training data shuffling – depending on the seed of the PNRG; moreover, experiments were replicated across different hardware platforms. As for feature selection, we based our considerations on three techniques, i.e., ANOVA f-test, Mutual information statistic and Random forest. The datasets adopted provide DoS activity generated through state-of-the-art attack tools, e.g., *hulk* and *slowhttptest*. The attacks are based on well-established tools and are typically used to validate IDS proposals; more importantly, the training of the autoencoder – semi-supervised – is not biased by specific attack types. The use of such a diverse mixture of controlled conditions and techniques aims to mitigate internal validity threats.

Conclusion validity. The conclusions of the study are consistent along the different dimensions of our experiments, which makes our finding perfectly reasonable and technically sound. For example, the variability of the IDS results caused by the seed of the PRNG is confirmed over different hardware platforms; moreover, the conclusion that the hardware platforms can affect the results – let the seed fixed – was inferred by running the same training-test experiment by using both standard computers and a full-fledged workstation featuring both CPUs and GPUs. We made sure to factor-out the non-determinism of the GPUs to confirm our conclusion was correct. Feature selection is based on three techniques: the conclusion that relevant features vary from attack to attack is consistent across all the techniques assessed. The interplay between the effect of the seed and the number of features is

also confirmed through experiments performed by lowering the number of features. Overall, the conclusions were inferred through a careful design of the experiments and by assessing the sensitivity of key results.

External validity. Our study can be replicated and applied to other types of neural networks, hardware platforms and attacks. Nowadays, there exist many deep learning frameworks and public repositories of neural networks and IDS datasets, which make our approach definitively feasible in practice. Our analysis approach does not interfere with system operations. There exist many products for capturing network packets and generating fixed-length records, which allow porting our method to other systems. In fact, in this paper we conducted our study with different hardware platforms by leveraging two independent datasets – providing data on several DoS attacks – collected within different network environments and published by different research teams. We are confident that the experimental details provided in the paper would support the replication of our study by future researchers and practitioners.

8 Lessons learned and conclusion

Semi-supervised learning approaches appear to be an interesting solution, since they alleviate the problem of the collection of training data. An autoencoder can be conveniently trained only by means of benign network traffic in order to learn a semi-supervised IDS model. The resulting autoencoder-based IDS is capable of detecting unknown types of attacks, simply because their patterns deviate from the learned normal patterns. Although various autoencoder models – often combined in complex architectures – have shown to be effective in detecting attacks, identifying the *optimal* model architecture to provide the best detection performance requires tremendous effort, and this hinders its practical application to IDS.

In this paper we have explored the use of a single autoencoder to classify network flows for intrusion detection purposes. We have presented the results of several years of research on this topic, the lessons learned and the open research issues. It is worth summarizing the main lessons learned throughout our research on AE for network flow classification:

- the seed used to start the random sequence generation matters, in that it leads to different trained models; therefore, the effect of the seed cannot be ignored. Successful intrusion detection requires checking the autoencoders obtained from a not-too-narrow set of seeds;
- if the AE is developed by Keras-Tensorflow, the trained models obtained with the same training data on different computing systems (alternative CPUs or GPUs) are likely to differ. The implication of this non-deterministic behavior is that a "successful" training is not repeatable, and so the produced model has to be saved beforehand to be exploited in the future, if it gives good results at testing time;
- trained models characterized by similar values of loss can be very different as far as their classification performance is concerned. Hence, multiple models

should be produced and suitably tested on a validation subset of the dataset so as to make it possible to choose the one with the best performance;
- a rigorous feature selection procedure requires information on the attacks to be detected. If this information is available, discarding scarcely significant features can improve classification accuracy. However, this is obtained at the expense of possible accuracy losses on zero-day attacks.
- applying feature selection "before" the autoencoder does not help to obtain a more steady behavior or to make the seed less influential on intrusion detection; the less the features, the higher the dispersion of the F1 score.

In our opinion, resorting to a single "basic" AE without any assistance from other neural networks or complex feature selection methods is a clear advantage in terms of simplicity of training and tuning, and use of processing power at recognition time. Another strong point of our solution is a good adaptability to unknown attacks, as only feature selection – which is just an option, not a strict requirement – requires a minimum notion of the attacks to be detected. A solution as ours requires no powerful or specialized processor, and is amenable to processing large quantities of data in real-time. This is why we claimed in a companion paper [7] that "simpler is better" and promote the use of a single autoencoder in future IDS designs. Our figures are comparable with the results obtained on the CICIDS2017 or similar datasets by supervised methods [16] or by AEs as components of more complex classification architectures. For example, the authors of HELAD system [26] – autoencoder combined with a long short-term memory – achieve an F1 score of 0.995. The performance is even worse – F1 score of 0.955 for DoS slowloris – for the approach using memory-augmented deep auto-encoder (MemAE) [42]. The already mentioned classifier assisted with aggressive feature reduction [32] performs better, but at the cost of large set-up times. Moreover, the ability of a rule-based classifier at detecting previously-unseen attacks is questionable.

The accuracy results obtained on the CICIDS2017 dataset leave little room for further improvements. Our future research will be oriented to the study of a training procedure and to the production of normal training data able to pave the way to the set up of an autoencoder able to recognize even unseen attack flows with reasonable performance figures. Our long-term objective is to give a set of guidelines and best practices able to leverage on a perfect balance between the *theory* and *practice* on autoencoders for intrusion detection. In the future, we will also extend our analysis to further datasets and attacks by also considering the rapidly-evolving IoT environments.

Author contribution

Marta Catillo, Antonio Pecchia and Umberto Villano contributed equally to this work.

Funding declaration

No funding to report.

Data availability

Publicly available datasets were analyzed in this study. The datasets can be found at the URLs mentioned in the paper.

Conflict of interest

The authors declare no competing interests.

References

[1] Dina, A.S., Manivannan, D.: Intrusion detection based on machine learning techniques in computer networks. Internet of Things **16**, 100462 (2021)

[2] Catillo, M., Pecchia, A., Rak, M., Villano, U.: Demystifying the role of public intrusion datasets: a replication study of DoS network traffic data. Computers & Security **108**, 102341 (2021)

[3] Verkerken, M., D'Hooge, L., Wauters, T., Volckaert, B., De Turck, F.: Towards model generalization for intrusion detection: Unsupervised machine learning techniques. Journal of Network and Systems Management **30**, 12 (2021)

[4] Apruzzese, G., Pajola, L., Conti, M.: The cross-evaluation of machine learning-based network intrusion detection systems. IEEE Transactions on Network and Service Management **19**(4), 5152–5169 (2022)

[5] Catillo, M., Del Vecchio, A., Pecchia, A., Villano, U.: Transferability of machine learning models learned from public intrusion detection datasets: the CICIDS2017 case study. Software Quality Journal **30**, 955–981 (2022)

[6] de Carvalho Bertoli, G., Alves Pereira Junior, L., Saotome, O., dos Santos, A.L.: Generalizing intrusion detection for heterogeneous networks: A stacked-unsupervised federated learning approach. Computers & Security **127**, 103106 (2023)

[7] Catillo, M., Pecchia, A., Villano, U.: Simpler is better: On the use of autoencoders for intrusion detection. In: Quality of Information and Communications Technology, pp. 223–238. Springer (2022)

[8] Zhang, Y., Lee, W., Huang, Y.: Intrusion detection techniques for mobile wireless networks. Wireless Networks **9**(5), 545–556 (2003)

[9] Kilincer, I.F., Ertam, F., Sengur, A.: Machine learning methods for cyber security intrusion detection: Datasets and comparative study. Computer Networks **188**, 107840 (2021)

[10] Ring, M., Wunderlich, S., Scheuring, D., Landes, D., Hotho, A.: A survey of network-based intrusion detection data sets. Computer & Security **86**, 147–167 (2019)

[11] Sharafaldin, I., Lashkari, A.H., Ghorbani., A.A.: Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In: Proc. International Conference on Information Systems Security and Privacy, pp. 108–116. SciTePress (2018)

[12] Moustafa, N., Slay, J.: UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: Proc. International Conference Military Communications and Information Systems Conference, pp. 1–6. IEEE (2015)

[13] Maciá-Fernández, G., Camacho, J., Magán-Carrión, R., García-Teodoro, P., Therón, R.: UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs. Computer & Security **73**, 411–424 (2017)

[14] Catillo, M., Del Vecchio, A., Ocone, L., Pecchia, A., Villano, U.: USB-IDS-1: A public multilayer dataset of labeled network flows for IDS evaluation. In: Proc. International Conference on Dependable Systems and Networks Workshops, pp. 1–6. IEEE (2021)

[15] Roesch, M.: Snort - lightweight intrusion detection for networks. In: Proc. International USENIX Conference on System Administration, pp. 229–238. USENIX Association (1999)

[16] Panigrahi, R., Borah, S., Bhoi, A.K., Ijaz, M.F., Pramanik, M., Jhaveri, R.H., Chowdhary, C.L.: Performance assessment of supervised classifiers for designing intrusion detection systems: A comprehensive review and recommendations for future research. Mathematics **9**(6), 690 (2021)

[17] Taher, K.A., Mohammed Yasin Jisan, B., Rahman, M.M.: Network intrusion detection using supervised machine learning technique with feature selection. In: Proc. International Conference on Robotics, Electrical and Signal Processing Techniques, pp. 643–646. IEEE (2019)

[18] Wei-Chao, L., Shih-Wen, K., Chih-Fong, T.: CANN: An intrusion detection system based on combining cluster centers and nearest neighbors. Knowledge-Based Systems **78**, 13–21 (2015)

[19] Jiang, J., Han, G., Liu, L., Shu, L., Guizani, M.: Outlier detection

approaches based on machine learning in the Internet-of-Things. IEEE Wireless Communications **27**(3), 53–59 (2020)

[20] Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE Journal **37**(2), 233–243 (1991)

[21] Kunang, Y.N., Nurmaini, S., Stiawan, D., Zarkasi, A., Firdaus, Jasmir: Automatic features extraction using autoencoder in intrusion detection system. In: Proc. International Conference on Electrical Engineering and Computer Science, pp. 219–224. IEEE (2018)

[22] Wu, J., Wu, Y., Niu, N., Zhou, M.: MHCPDP: multi-source heterogeneous cross-project defect prediction via multi-source transfer learning and autoencoder. IEEE Pervasive Computing **29**(2), 405–430 (2021)

[23] Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: An ensemble of autoencoders for online network intrusion detection. In: Proc. International Conference of Network and Distributed System Security Symposium (2018)

[24] XuKui, L., Wei, C., Qianru, Z., Lifa, W.: Building auto-encoder intrusion detection system based on random forest feature selection. Computers & Security **95**, 101851 (2020)

[25] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., Elovici, Y.: N-BaIoT-network-based detection of IoT botnet attacks using deep autoencoders. IEEE Pervasive Computing **17**(3), 12–22 (2018)

[26] Zhong, Y., Chen, W., Wang, Z., Chen, Y., Wang, K., Li, Y., Yin, X., Shi, X., Yang, J., Li, K.: HELAD: A novel network anomaly detection model based on heterogeneous ensemble learning. Computer Networks **169**, 107049 (2020)

[27] Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Computers & Electrical Engineering **40**(1), 16–28 (2014)

[28] Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R.P., Tang, J., Liu, H.: Feature Selection: A Data Perspective. ACM Comput. Surv. **50**, 1–45 (2018)

[29] Cai, J., Luo, J., Wang, S., Yang, S.: Feature selection in machine learning: A new perspective. Neurocomputing **300**, 70–79 (2018)

[30] Solorio-Fernández, S., Carrasco-Ochoa, J.A., Martínez-Trinidad, J.F.: A review of unsupervised feature selection methods. Artificial Intelligence Review **53**(2), 907–948 (2020)

[31] Binbusayyis, A., Vaiyapuri, T.: Comprehensive analysis and recommendation of feature evaluation measures for intrusion detection. Heliyon **6**(7), 04262 (2020)

[32] Kshirsagar, D., Kumar, S.: An efficient feature reduction method for the detection of DoS attack. ICT Express **7**(3), 371–375 (2021)

[33] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research **11**, 3371–3408 (2010)

[34] Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. **41**(3), 15 (2009)

[35] Catillo, M., Rak, M., Villano, U.: Discovery of DoS attacks by the ZED-IDS anomaly detector. Journal of High Speed Networks **25**(4), 349–365 (2019)

[36] Maseer, Z.K., Yusof, R., Bahaman, N., Mostafa, S.A., Foozy, C.F.M.: Benchmarking of machine learning for anomaly based intrusion detection systems in the CICIDS2017 dataset. IEEE Access **9**, 22351–22370 (2021)

[37] Catillo, M., Rak, M., Villano, U.: 2L-ZED-IDS: A two-level anomaly detector for multiple attack classes. In: Web, Artificial Intelligence and Network Applications, pp. 687–696. Springer (2020)

[38] Liu, F.T., Ting, K.M., Zhou, Z.: Isolation forest. In: Proc. International Conference on Data Mining, pp. 413–422. IEEE (2008)

[39] Engelen, G., Rimmer, V., Joosen, W.: Troubleshooting an intrusion detection dataset: the CICIDS2017 case study. In: Proc. Security and Privacy Workshops, pp. 7–12 (2021). IEEE

[40] Rosay, A., Carlier, F., Cheval, E., Leroux, P.: From CIC-IDS2017 to LYCOS-IDS2017: A corrected dataset for better performance. In: Proc. International Conference on Web Intelligence, pp. 570–575. ACM (2021)

[41] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: An Introduction. Kluwer Academic (2000)

[42] Min, B., Yoo, J., Kim, S., Shin, D., Shin, D.: Network anomaly detection using memory-augmented deep autoencoder. IEEE Access **9**, 104695–104706 (2021)