# Simpler is better: on the use of autoencoders for intrusion detection

Marta Catillo, Antonio Pecchia, and Umberto Villano

Dipartimento di Ingegneria
Università degli Studi del Sannio, Benevento, Italy
{marta.catillo,antonio.pecchia,villano}@unisannio.it

**Abstract.** The ever-growing occurrence of computer security incidents calls for advanced intrusion detection techniques. A wide body of literature dealing with Intrusion Detection Systems (IDSes) is based on machine learning; many proposals rely on the use of autoencoders (AEs), due to their capability to analyze complex, high-dimensional and large-scale data. Most of the times, AEs are used as building blocks of much more complex detection architectures, possibly in combination with sophisticated feature selection techniques. This paper summarizes several years of work in this field, suggesting that "simpler is better" and that a carefully tuned and trained AE can be used in isolation, obtaining recognition results comparable with those attained by more complex designs. The best practices presented here, regarding dataset production and sanitization, AE set-up and training, threshold setting, possible use of simple feature selection techniques for performance improvement can be valuable for any practitioner willing to use autoencoders for intrusion detection purposes.

**Keywords:** Intrusion detection · Autoencoders · Denial of Service.

## 1 Introduction

Due to the ever-growing occurrence and complexity of computer security incidents, intrusion detection is, and will steadily remain, a hot research topic. A wide body of literature aims at proposing effective solutions to the lack of security of the computer networks and devices our lives currently rely on, by presenting new proposals of Intrusion Detection Systems (IDSes) [12]. The aim of these systems is to discover (and possibly block or divert) on-going attacks before any harm can be done. 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.

In this situation, given the growing success of machine learning (ML) techniques and the availability of processors suitably designed for this domain, a

very large number of proposals in the cybersecurity field rely on ML [7]. The "pattern" followed by most papers on ML and IDSes is typically the same: an algorithm or an architecture based on neural networks (possibly deep ones) are proposed, then are tested on reference datasets, very high performance figures (often close to 100% intrusion detection capabilities) are proven by limited-width experiments. But the continuous flow of new proposals clearly indicates the intrusion detection problem is still there, and that the experiments mentioned in the papers have only limited validity in real-world networks.

Our previous work has tackled the problem of the lack of transferability of the impressive results obtained on reference datasets (possibly outdated and not free from statistical biasing) in even slightly-different data collection settings [4]. This paper will instead focus on the unnecessary complexity of many existing IDS proposals. Among the wide corpus of the existing proposals, multiple autoencoder (AEs) networks are often used in complex configurations, possibly complemented by sophisticated feature selection methods. We develop around the intuition that this complexity is not justified because a single autoencoder –suitably trained and correctly used– is enough to obtain similar (if not better) performance figures compared to existing proposals. This proposition is investigated in the context of the widely-used CICIDS2017 intrusion detection benchmarking dataset. The IDS solution proposed in this paper achieves 0.988 recall, 0.976 precision and 0.982 F1 with no feature selection and a single autoencoder; moreover, results indicate that the use of feature selection yields negligible improvements over the metrics at the cost of demanding tuning attempts. Based on the results, we discuss all our findings in several years of use of autoencoders for IDS, pointing out a number of best practices that can lead to successful performance results without unnecessary architectural complications.

The rest of this paper is organized as follows. Sect. 2 presents related work; Sect. 3 deals with the basics of autoencoders, its use for classification and the reference dataset. Sect. 4 discusses our proposal based on a single autoencoder for intrusion detection, the issues related to its design and training. Sect. 5 investigates the possibility to perform feature selection, and present the results obtained on the dataset. Sect. 6 closes with lessons, conclusions and directions of future research.

## 2 Related Work

Despite decades of research and development, existing intrusion detection systems still face challenges in improving the detection rate, reducing the false positives and –possibly– detecting unknown attacks. To solve the above problems, many researchers and practitioners have focused on developing intrusion detection systems that capitalize on **machine learning** and **deep learning** methods [13]. Moreover, in order to tune and test these techniques, many ready-to-use **public intrusion detection datasets** have been produced [23]. Most of these datasets are collected in synthetic environments under normative conditions and different intrusion scenarios. They emulate real network traffic and –at

least in theory– they do not contain any confidential data. Almost all datasets are released as labeled network flows, organized in comma-separated values files specially crafted to apply modern machine learning techniques. In particular, each record is 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 [24]. 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 datasets are USB-IDS-1 [1], UGR'16 [17] and UNSW-NB15 [21].

Over the last few years, a boundlessness of machine learning methods for misuse detection as well as anomaly detection have been proposed [16]. In general, these approaches can be delegated as *supervised* or *unsupervised* depending on whether there is a need to train the algorithm on labeled instances. In the case of supervised learning techniques, the algorithm is trained on labeled data points and it determines a function to map points to classes. Many supervised approaches rely on a limited number of classifiers or only one classifier by achieving outstanding performance –detection close to 100%– [22]. However, a large number of supervised methods also exploit artificial neural networks [25]. 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 outliers detection [28] [11].

**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. They are designed to map the input data points to an internal latent representation, which is then used to reconstruct the input. Autoencoders were first developed as *nonlinear* extension of the standard linear principal component analysis (PCA) in order to make **dimensionality reduction** [14]. For example, in [15] the authors use an autoencoder to perform automatic features extraction with the aim to reduce the dimensions of the data being processed. Thereafter, they classify the attacks by means of the support vector machine algorithm. Feng et al. [9] show a graph and autoencoder-based feature selection (GAFS) method, which projects the data to a lower-dimensional space using a single-layer autoencoder. The approach proved to be effective when compared with existing state-of-the-art methods. In [26] the authors propose a model which adopts two types of autoencoder. A generic autoencoder is used to capture the generic features which are common to all intrusions, while several ad-hoc autoencoders are trained with the aim to capture patterns that are specific only to particular groups of intrusions. From combining these two feature maps the authors propose a new feature map to classify the intrusions by means of the random forest classifier.

However, autoencoders are often used also in recent studies for **anomaly detection** purposes. In this context they are mainly components of a more complex network, specially crafted with the aim to design sophisticated detec-

tors. In [20] 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 [29], instead, the authors propose an effective deep learning method, namely autoencoder-IDS (AE-IDS) based on random forest. In particular, they use the random forest algorithm 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. A further heterogeneous ensemble method for intrusion detection is proposed by Zhong et al. [30]. In particular, the authors propose HELAD (Heterogeneous Ensemble Learning Anomaly Detection), an unsupervised approach where an autoencoder is combined with a long short-term memory (LSTM) predictor. The authors evaluate their approach by means of the MAWILab3 and CICIDS2017 datasets. The experimental results show that the HELAD algorithm has better adaptability and accuracy than other state-of-the-art algorithms. Min et al. [19] propose a network intrusion detection method using a memory-augmented deep autoencoder (MemAE), which can solve the over-generalization problem of autoencoders. MemAE solves this problem by bringing the reconstruction of the attack inputs closer to the normal sample through the memory module. Experiments are conducted on the NSL-KDD, UNSW-NB15, and CICIDS2017 datasets.

It is worth pointing out that all the aforementioned autoencoder systems adopt fairly sophisticated infrastructures. The detection of different classes of anomalies has been recently addressed by means of system log analysis and a deep autoencoder [2]: the proposed approach, called AutoLog, is based solely on a deep autoencoder network without any kind of artifice in the infrastructure.

## 3 Background and Datasets

### 3.1 Autoencoders (AE)

An autoncoder (AE) is a feedforward neural network where the *output layer* has the same dimension as the *input layer*. In fact, the purpose of an AE is to "reconstruct" the input at 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. Fig. 1 shows the representation of a basic autoencoder with three hidden layers.

It is possible to design different types of autoencoders [10]. In particular, deep learning can be applied to autoencoders: multiple hidden layers are used to

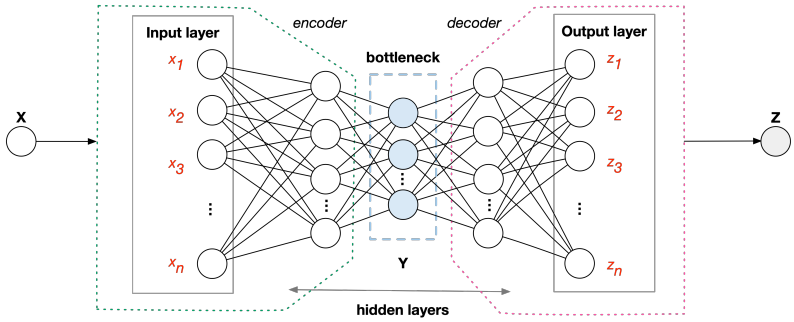



Fig. 1: Representation of an autoencoder.

provide depth. The resulting network is known as *deep* or *stacked autoencoder* [27].

An autoencoder consists of two parts: **encoder** and **decoder**. Let $\mathbf{x}$ be an input vector of $n$ real numbers $[x_1,x_2,...,x_n]$, 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 $n$ real numbers $\mathbf{z} = [z_1, z_2, ..., z_n]$. It tends to reconstruct the input vector $\mathbf{x}$ from $\mathbf{y}$. Encoding-decoding formulas are given in Eq. 1 and Eq. 2. They represent the case of a "basic" autoencoder with only one hidden layer:

$$y = \sigma(Wx + b) \tag{1}$$

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

where $W$, $W'$, $b$ and $b'$ are weight matrices and bias vectors, while $\sigma$ and $\sigma'$ are activation functions.

Regardless of the architecture, an autoencoder has one primary objective: reconstruct its input as accurately as possible. The goodness of the reconstruction is given by the **reconstruction error** (RE), which measures the difference between the reconstructed, i.e., $\mathbf{z}$, and the original version of the input, i.e., $\mathbf{x}$:

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

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 AE for classification and evaluation metrics

The rationale underlying the use of the AE for classification is that it can be trained to reconstruct a given set of inputs. After training, the autoencoder

will accurately reconstruct, i.e., obtain low RE, future points "similar" to those used for training, while it will badly reconstruct, i.e., high RE, future points "different" to those used for training.

**Threshold setting.** In order to discriminate *good* from *bad* reconstructions we rely on a cutoff threshold value. The **threshold** is the value of RE over which the flows are considered malign. In our first attempts [3], the threshold was set in a supervised manner, reserving a small part of the dataset in hand to obtain an optimal balance false positives (benign flows falling over the threshold) and false negatives (malign flows under the threshold). In our attempt to set up an autoencoder that never sees malign flows and it is potentially fit to detect any type of attack, we resorted successively to a threshold set at a given percentile of the RE [2]. This approach was not completely satisfactory, because it may produce some false positives. In this paper, thresholds are set *in unsupervised manner* (i.e., without any cognition of the attack flows) by considering the outliers produced in the reconstruction of the benign flows used in the training step.

**Evaluation metrics.** The overall performance of the classification is measured by analyzing the typical metrics of *recall* (R), *precision* (P), *false positive rate* (FPR), and *F1 score*. 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 points 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 that the threshold; similarly, TN is the set of normal points whose RE is lower that the threshold.

### 3.3 Reference Dataset: CICIDS2017

CICIDS2017 is a flow-based dataset based on `CICFlowMeter`[1]. The flows synthesize the characteristics of any interaction between two systems on the net, and can be generated from network captures by many existing tools. `CICFlowMeter` 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. 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 lead us to discover that often the attack flows contained in CICIDS2017 do not really harm a correctly-configured server [5] and that the flows produced by the original release of the

[1] `https://github.com/ahlashkari/CICFlowMeter`

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 Length of Fwd Packets | f11 | Fwd URG Flags | f39 | Bwd Avg Bulk Rate | f67 |
| Total Length of Bwd Packets | f12 | Bwd URG Flags | f40 | Subflow Fwd Packets | f68 |
| Fwd Packet Length Max | f13 | Fwd Header Length | f41 | Subflow Fwd Bytes | f69 |
| Fwd Packet Length Min | f14 | Bwd Header Length | f42 | Subflow Bwd Packets | f70 |
| Fwd Packet Length Mean | f15 | Fwd Packets/s | f43 | Subflow Bwd Bytes | f71 |
| Fwd Packet Length Std | f16 | Bwd Packets/s | f44 | Init_Win_bytes_forward | f72 |
| Bwd Packet Length Max | f17 | Min Packet Length | f45 | Init_Win_bytes_backward | f73 |
| Bwd Packet Length Min | f18 | Max Packet Length | f46 | act_data_pkt_fwd | f74 |
| Bwd Packet Length Mean | f19 | Packet Length Mean | f47 | min_seg_size_forward | f75 |
| Bwd Packet Length Std | f20 | Packet Length Std | f48 | Active Mean | f76 |
| Flow Bytes/s | f21 | Packet Length 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 | | |

`CICFlowMeter` tool –commonly used in the context of IDS research– contain inexplicable flows. These are actually fragments of an incorrectly truncated flow. A patch to the latter issue was recently provided in [8], along with a new version of both CICIDS2017 and `CICFlowMeter`: experimentation presented in this paper is based on the fixed version of CICIDS2017[2].

We consider 490,968 flows related to normal traffic and DoS attacks. Flows 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 flows is preserved in the splits. Flows are divided as follows:

- **CICIDS-TRAINING**: *70%* of the total (i.e., 343,680) divided into 223,430 `BENIGN` and 120,250 `ATTACK` flows;
- **CICIDS-VALIDATION**: *15%* of the total (i.e., 73,644), divided into 47,877 `BENIGN` and 25,767 `ATTACK` flows;
- **CICIDS-TEST**: *15%* of the total (i.e., 73,644), divided into 47,877 `BENIGN` and 25,767 `ATTACK` flows.

## 4 Proposed IDS approach with a single AE

The idea of leveraging a relatively simple neural network and to train it with normative traffic (thus using a *semi-supervised* approach), making it possible to

[2] `https://downloads.distrinet-research.be/WTMC2021/tools_datasets.html`

Table 2: Initial features used in 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 |

detect intrusions simply because of their divergence from the "normal" behavior the AE was trained on, is indeed fascinating. Our first results, obtained by setting the threshold with a supervised approach (balancing false positives and false negatives on a labeled portion of the dataset) and published in [3], were not so bad (F1 score = 0.942), but inferior to those obtained by supervised detection methods [18]. A successive attempt exploiting three AEs lead to better performance [6].

In the following, we discuss our best practices for setting up an AE for successful intrusion detection, using the above mentioned dataset as case study. It is worth noting that out of the 83 features in Table 1, six of them (f1-f5, f7) can be neglected outright for detection purposes –it is too easy to detect malign flows in a dataset by exploiting the IP of the attacker– as such, the initial experiment is conducted with the 77 features in Table 2. We will demonstrate that even a single-AE design can obtain remarkable results, avoiding unnecessary complications and undue overhead. Along with our indications for AE tuning and set-up, we will outline the research issues still open.

### 4.1 AE dimensions and depth

The first step to set-up an AE for intrusion detection is to choose a suitable form factor, i.e., the number of levels and neurons at each level. Given the number of input and output units, which are necessarily equal to the number of features considered, it is necessary to choose the number of hidden levels and the number of units at each level. Unfortunately, there are no rules to guide this choice, and so the only way is to proceed by trial and error.

Almost unexpectedly, we have found that the number of levels and units is not a particularly critical parameter. It is possible to obtain low RE with three hidden levels (encoding-bottleneck-decoding), or with five hidden levels as well. The only criticality is the number of units at the bottleneck, which have to hold the encoded flow state. In the case of network flow processing considered here we have found the best results by using a bottleneck made up of 6 up to 8 units. In the following, we will always present results relative to a 48-24-8-24-48 *relu* units *deep* AE.

It is interesting to note that there exist frameworks to automatize the search of a "good" network (e.g., *Keras optimizers*). For a simple network such as our AE, we think that the use of extensive optimization procedures is overkill.

### 4.2 Training and validation

Unlike the previous step, the modality of training is critical to obtain high detection performance. The AEs we have used for IDS are always trained in *semi-supervised* mode. By semi-supervised training we mean that a subset of the dataset in hand is reserved for training, from which only the normative traffic (*benign* label) is selected: during training, the AE sees no flows related to attacks. Benign flows are presented in input to the AE, whose weights are progressively tuned trying to obtain low RE, i.e., an output as close as possible to the flow presented in input, feature by feature.

The rationale is to instruct the AE to reproduce normative flows, hoping that any divergence with respect to a "benign behavior" could lead to a high reconstruction error, making it possible to recognize attack flows. 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 [3] and [4]).

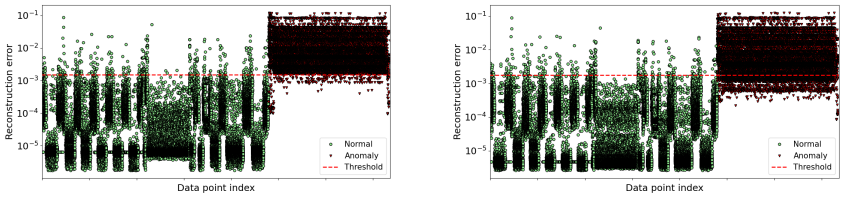
Our AEs are implemented by the ubiquitous deep learning framework *Keras*, which in its turn founds on *Tensorflow*. As for any learning framework, training is based on training-validation sets (i.e., CICIDS-TRAINING and CICIDS-VALIDATION after having filtered out the attack flows). When the training is started, the AE neurons are randomly initialized and input data are presented in batches through a given number of epochs. The systems tries to minimize the *loss*, 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. A solution is to compute the loss as the mean squared error at the output units; this matches the definition of reconstruction error (RE) above.

**Issue 1:** The training process is highly dependent on the hardware running Keras/Tensorflow.

Different CPUs (or GPUs) will lead to different schedules of the threads used for optimization in the training phase, and in the end to different weights in the network. The same is true for the *seed* of the pseudo-random generator used for units initialization. This is a physiologic characteristic of machine learning environment as Keras/Tensorflow: while we would expect only a slight variability of the results from run to run due to the seed, this is not the case for a semi-supervised autoencoder to be used for intrusion detection, as shown in the following subsection.

### 4.3 Results

It is a fact that all possible trainings on the same input data lead to fairly similar loss values. In other words, whatever the hardware used or the random seed, it works. The bad surprise is that at equal values training losses do not correspond equal abilities to detect malign flows. Fig. 2 shows two plots of the reconstruction error obtained on the CICIDS-TEST file (i.e., the split of benign




(a) seed = 1116, F1 = 0.9851 (b) seed = 4, F1 = 0.8904

Fig. 2: Reconstruction error for different seed values measured with the test set (CICIDS-TEST).

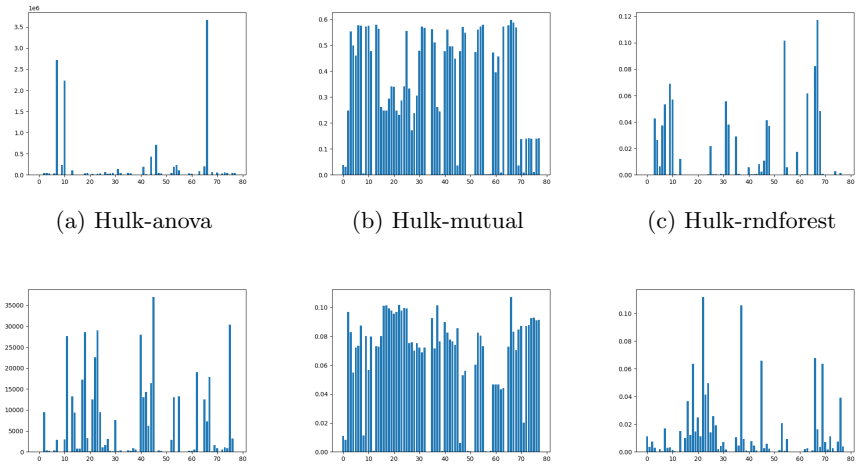
and malicious flows *held out* from training) by the same AE in perfectly equal conditions, let aside the random seed value. Both networks are perfectly able to recognize benign flows, which are mostly under the threshold. However, in the plot on the right the dots corresponding to malign flows are in a lower position, and so the detection performance is completely unsatisfactory. Simply by changing the seed, the F1 score falls from 0.9851 (a fairly good result) to 0.8904 (an unsatisfactory detection performance). Most notably, the two networks have similar final training loss values (1.2119e-04 and 1.3234e-04, respectively), but different detection performance.

**Issue 2:** A successful training leading to low loss values does not guarantee good classification performance.

The RE (or the loss measured by *Keras*, which is the same) is a mean of squares, extended to all features. 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 which could be the most relevant features for an unknown or new type of attack.

**Issue 3:** The seed used to start the random sequence generation counts, in that it leads to different trained models.

This is an open research issue. For the time being, the only viable solution is to validate the training performed using a (labeled) subset of the dataset (the *validation* file reserved for this purpose), and try to change the random seed until satisfactory results are obtained. It is clear that this process leads to a detector able to manage at best the attacks present in the dataset, but that *possibly* could be less successful for different types of attack.



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

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

Fig. 3: Scores of the 77 features, Hulk and Slowhttptest from USBIDS1 dataset

Table 3: Reduced sets 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 |

## 5 Feature selection

Feature selection techniques are widely used in the intrusion detection context to help obtain higher detection accuracy, neglecting the features which are redundant or statistically do not contribute significantly to the classification of flows. Out of the features listed in Table 1, it is very unlikely that all of them are useful for detection purposes. Sometimes a few ones are constant through all the dataset, and so have no utility for flow classification. Given the problems linked to the mean used to compute the loss discussed in Subsect. 4.3, any reduction of the number of features actually used can help to obtain, being equal the loss, an AE better tuned to the "significant" features. But, once again, without knowing the characteristic of attacks is not possible to know which features can be useful for malign flow recognition and which are useless.

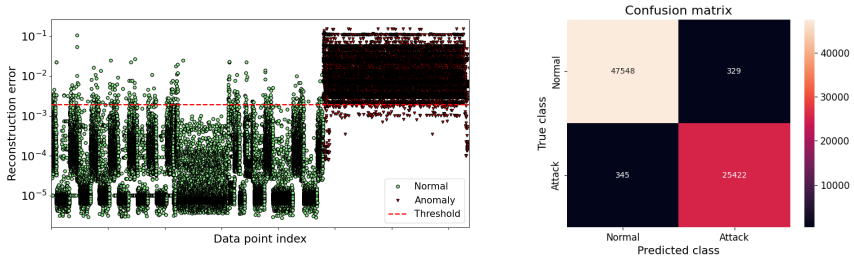



Fig. 4: Reconstruction error and confusion matrix of the 47-features AE

Table 4: 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 |
| **F1** | 0.982 | 0.984 | 0.985 | 0.987 |

Fig. 3 shows a sample of the results of widely used statistical tests (*ANOVA f-test*, *mutual information* statistic, *random forest*) performed on different type of attacks[3]. Each histogram reports the scores (y-axis) of the 77 features (x-axis). Higher score means higher contribution to the classification of the flow as a malign one. As can be seen at a glance comparing the histograms, the set of the most relevant features is not uniform across all types of attack, and also depend on the statistical test performed. In light of the above, selecting only some of the 77 features can help detection, but the detector loses "universality" as at least in principle unknown attacks could be spotted by the neglected features. However, a reasonable trade-off can be made by neglecting a small number of features which are ranked in the lowest positions according to the tests performed.

We have tried to discard the lowest-ranked 10, 20 and 30 features, obtaining "reduced" sets of 67, 57 and 47 features, respectively, as shown in Table 3. As expected, reducing the features helps a bit to obtain good classification performance. Table 4 shows the P, R, F1 values obtained; the best performance is obtained with the 47-features set. However, it should be noted that this might involve bad classification performance on unseen attacks (i.e., those not present in the dataset used to compute the feature rankings).

### 5.1 Results

As previously shown in Table 4, the best performance results have been obtained by the AE processing 47 features, with seed = 1062 and 90 training epochs. In Fig. 4 we present the graph of the RE over the testing set and corersponding the confusion matrix. Maybe further tuning could help to obtain a slightly higher performance, but at these levels of precision and recall, it is likely to would be

[3] Attacks are taken from the USB-IDS-1 dataset.

simply a waste of time, since the flows misclassified are only 674 (329 + 345) on a total of 73,644 test flows.

The figures obtained are comparable with the results obtained on the CICIDS2017 or similar datasets by supervised methods [22] or by AEs as components of more complex classification architectures. For example, the authors of HELAD system [30] –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) [19].

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 term of simplicity of training and tuning, use of processing power at recognition time. Another strong point of our solution is 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 quantity of data in real-time. This is why we claim that "simpler is better" and promote the use of a single autoencoder in future IDS designs.

## 6 Lessons learned and conclusion

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

- 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 seed used to start the random sequence generation counts, in that leads to different trained models;
- 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 unconsidered attacks.

The results obtained show that a single AE can obtain classifications accuracy comparable to the ones published in the research literature for supervised networks and for more complex designs built around one or several AEs. Our single autoencoder detection scheme is less probe to transferability problems than

supervised schemes and can be more easily tuned and managed than designs adopting AE as components.

The accuracy results obtained on the CICIDS2017 dataset leave little room for further improvements. Our future research will oriented to the study of a training procedure and to the production of normative training data able to pave the way to the set up of an autoencoder able to recognize even unseen attack flows with reasonable accuracy.

## References

1. 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. Int. Conf. on Dependable Systems and Networks Workshops (DSN-W). pp. 1–6. IEEE (2021)
2. Catillo, M., Pecchia, A., Villano, U.: Autolog: Anomaly detection by deep autoencoding of system logs. Expert Systems with Applications **191**, 116263 (2022)
3. 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)
4. 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 (2022)
5. 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)
6. 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, WAINA2020. pp. 687–696. Advances in Intelligent Systems and Computing, Springer International Publishing (2020)
7. Dina, A.S., Manivannan, D.: Intrusion detection based on machine learning techniques in computer networks. Internet of Things **16**, 100462 (2021)
8. Engelen, G., Rimmer, V., Joosen, W.: Troubleshooting an intrusion detection dataset: the cicids2017 case study. In: 2021 IEEE Security and Privacy Workshops (SPW). pp. 7–12. IEEE (2021)
9. Feng, S., Duarte, M.F.: Graph regularized autoencoder-based unsupervised feature selection. In: Proc. Int. Conf. on Signals, Systems, and Computers. pp. 55–59. IEEE (2018)
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
11. 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)
12. Khraisat, A., Gondal, I., Vamplew, P., Kamruzzaman, J.: Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity **2**(1), 20 (2019)
13. Kilincer, I., Ertam, F., Sengur, A.: Machine learning methods for cyber security intrusion detection: Datasets and comparative study. Computer Networks **188**, 107840 (2021)
14. Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE Journal **37**(2), 233–243 (1991)

15. Kunang, Y.N., Nurmaini, S., Stiawan, D., Zarkasi, A., Firdaus, Jasmir: Automatic features extraction using autoencoder in intrusion detection system. In: Proc. Int. Conf. on Electrical Engineering and Computer Science (ICECOS). pp. 219–224. IEEE (2018)
16. Kwon, D., Kim, H., Kim, J., Suh, S., Kim, I., Kim, K.J.: A survey of deep learning-based network anomaly detection. Cluster Computing **22**(1), 949–961 (2019)
17. 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)
18. 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** (2021)
19. Min, B., Yoo, J., Kim, S., Shin, D., Shin, D.: Network anomaly detection using memory-augmented deep autoencoder. IEEE Access **9** (2021)
20. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: An ensemble of autoencoders for online network intrusion detection. In: Proc. Int. Conf. of Network and Distributed System Security Symposium (NDSS) (2018)
21. Moustafa, N., Slay, J.: UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: Proc. Int. Conf. Military Communications and Information Systems Conference. pp. 1–6. IEEE (2015)
22. 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) (2021)
23. 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)
24. Sharafaldin, I., Lashkari, A.H., Ghorbani., A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: Proc. Int. Conf. on Information Systems Security and Privacy. pp. 108–116. SciTePress (2018)
25. Taher, K.A., Mohammed Yasin Jisan, B., Rahman, M.M.: Network intrusion detection using supervised machine learning technique with feature selection. In: Proc. Int. Conf. on Robotics, Electrical and Signal Processing Techniques (ICREST). IEEE (2019)
26. Thakur, S., Chakraborty, A., De, R., Kumar, N., Sarkar, R.: Intrusion detection in cyber-physical systems using a generic and domain specific deep autoencoder model. Computers & Electrical Engineering **91**, 107044 (2021)
27. 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)
28. 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)
29. 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)
30. 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)