



## Article

# Benchmarking Deep Learning Methods for Behaviour-Based Network Intrusion Detection

Mário Antunes <sup>1,2,\*</sup> , Luís Oliveira <sup>3,\*</sup> , Afonso Seguro <sup>4</sup>, João Veríssimo <sup>4</sup>, Ruben Salgado <sup>4</sup> and Tiago Murteira <sup>4</sup><sup>1</sup> Computer Science and Communication Research Centre (CIIC), School of Technology and Management, Polytechnic of Leiria, 2411-901 Leiria, Portugal<sup>2</sup> Institute for Systems and Computer Engineering, Technology and Science (INESC TEC, CRACS), 4200-465 Porto, Portugal<sup>3</sup> Smart Cities Research Center (Ci2), Polytechnic Institute of Tomar, 2300-313 Tomar, Portugal<sup>4</sup> Polytechnic Institute of Tomar, 2300-313 Tomar, Portugal; aluno21086@ipt.pt (A.S.); aluno21072@ipt.pt (J.V.); aluno21374@ipt.pt (R.S.); aluno21087@ipt.pt (T.M.)

\* Correspondence: mario.antunes@ipleiria.pt (M.A.); loliveira@ipt.pt (L.O.)

**Abstract:** Network security encloses a wide set of technologies dealing with intrusions detection. Despite the massive adoption of signature-based network intrusion detection systems (IDSs), they fail in detecting zero-day attacks and previously unseen vulnerabilities exploits. Behaviour-based network IDSs have been seen as a way to overcome signature-based IDS flaws, namely through the implementation of machine-learning-based methods, to tolerate new forms of normal network behaviour, and to identify yet unknown malicious activities. A wide set of machine learning methods has been applied to implement behaviour-based IDSs with promising results on detecting new forms of intrusions and attacks. Innovative machine learning techniques have emerged, namely deep-learning-based techniques, to process unstructured data, speed up the classification process, and improve the overall performance obtained by behaviour-based network intrusion detection systems. The use of realistic datasets of normal and malicious networking activities is crucial to benchmark machine learning models, as they should represent real-world networking scenarios and be based on realistic computers network activity. This paper aims to evaluate CSE-CIC-IDS2018 dataset and benchmark a set of deep-learning-based methods, namely convolutional neural networks (CNN) and long short-term memory (LSTM). Autoencoder and principal component analysis (PCA) methods were also applied to evaluate features reduction in the original dataset and its implications in the overall detection performance. The results revealed the appropriateness of using the CSE-CIC-IDS2018 dataset to benchmark supervised deep learning models. It was also possible to evaluate the robustness of using CNN and LSTM methods to detect unseen normal activity and variations of previously trained attacks. The results reveal that feature reduction methods decreased the processing time without loss of accuracy in the overall detection performance.

**Keywords:** network security; intrusion detection systems; convolutional neural networks; long short-term memory; deep learning; CSE-CIC-IDS2018 dataset



**Citation:** Antunes, M.; Oliveira, L.; Seguro, A.; Veríssimo, J.; Salgado, R.; Murteira, T. Benchmarking Deep Learning Methods for Behaviour-Based Network Intrusion Detection. *Informatics* **2022**, *9*, 29. <https://doi.org/10.3390/informatics9010029>

Academic Editor: Antony Bryant

Received: 6 February 2022

Accepted: 17 March 2022

Published: 20 March 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Computer network security entails a broad set of technologies, applications, and protocols to protect a given organization's assets and user operations. For example, at the application layer, a set of secure protocols (e.g., HTTPS or SMTPS) can be used to implement confidential channels between client and server and hide the exchanged messages. The configuration of virtual private networks (VPN) and firewalls are also two examples of technologies to implement confidentiality and protection in client/server communication.

Regarding intrusion detection systems (IDSs), their primary function is to monitor the network traffic and distinguish between regular and malicious activity. The sense of normal network activity is defined by recurrent patterns, easily identified by common network

flows features, such as the number of packets exchanged and error rate. On the other hand, network intrusions usually explore software vulnerabilities and correspond to a drift of context, observed by abnormal fluctuations of patterns or features values, when compared with the normal network activity patterns.

The signature-based IDSs are largely used in real-world networks. They take advantage of a database of known attacks and vulnerability exploits signatures, but fail to detect malicious unseen activity that never appeared. In this type of IDS, an alert is triggered when a match is found between the pattern analysed and an entry in the signatures database. Behaviour-based IDSs are designed to detect zero-day attacks and exploits and take advantage of machine learning (ML) algorithms characteristics to detect unknown and previously unseen patterns. Although they may theoretically detect unknown attacks, their applicability in real-world networks is low, mainly due to the complexity of the development and the high number of false positives they usually produce.

The benchmark and assessment of ML methods are pivotal to leveraging behaviour-based IDS adoption and implementation. Besides implementing ML models in real-world scenarios, the assessment and tuning of this type of IDSs take advantage of realistic datasets that represent normal and abnormal network activities. Several incursions have been made in recent decades to develop realistic datasets [1–3]. However, the narrow scope, the lack of heterogeneity, the reduced available formats, and the level of criticism for some of them [4] have allowed researchers to adopt and evaluate other promising and recent ones.

Despite being recent, CSE-CIC-IDS2018 dataset has been adopted to test and benchmark several approaches to develop behaviour-based IDS. It is very well organized and is publicly available at [5]. The scientific community has widely used the dataset to benchmark IDS, as it includes a wide range of attacks, executed with different tools, organized in a timeline, and mixes normal and anomalous network packet flows. In addition, the traffic was dynamically generated to simulate a corporate network. Besides the features collected from the network flows, the dataset also includes the original PCAP files with all the packets collected, which increases the flexibility to apply different preprocessing and processing methods.

The CSE-CIC datasets have been evaluated by several authors, namely by benchmarking distinct methods. In [6], the authors benchmark the CSE-CIC-IDS2017 dataset against several ML and deep learning models. Ferreira et al. [7] evaluate CSE-CIC-IDS2017 dataset with two bio-inspired ML methods, namely CLONALG artificial immune system, learning vector quantization (LVQ), and back-propagation multi-layer perceptron (MLP). The detailed survey in [8] analysed several intrusion detection methods evaluated with the CSE-CIC-IDS2018 dataset. In [9], Ferrag et al. present a survey and a comparative study of deep learning approaches for behaviour-based intrusion detection. The authors describe 35 well-known datasets and provide their classification into 7 categories. In [10], the authors compare a comprehensive set of deep learning frameworks in detecting network intrusions and also in classifying common network attack types through the CSE-CIC-IDS2018 dataset.

Among the multiplicity of existing ML methods, this paper evaluates a subset of deep-learning-based algorithms widely implemented by ML common tools, namely convolutional neural networks (CNN) and long short-term memory (LSTM) ML methods. Additionally, autoencoder and principal component analysis (PCA) methods were used to reduce the features involved in the dataset processing. The former reduces the number of perceptrons by compressing all the information, where the input and output layers are the same but in the hidden layers in the neural network. The latter correlates with the variables creating components that enable the clustering of several characteristics shared by themselves. The CSE-CIC-IDS2018 dataset was used to assess the deep learning methods, mainly due to the diversity of network traffic it encloses and by being massively adopted to benchmark behaviour-based IDS.

CNN and LSTM methods have already been successfully applied to evaluate other public IDS datasets, besides CSE-CIC-IDS2018. Some recent works described below reinforce the appropriateness of using CNN and LSTM methods to detect and classify intrusion

detection in a wide range of public datasets, such as CSE-CIC-IDS2018, which was used in our experiments.

In [11], the authors evaluate these methods evaluated on NSL-KDD and ISCX datasets. In [12] the authors evaluate bi-directional LSTM deep learning method with KDDCUP-99 and UNSW-NB15 datasets, which result in an average of 99.5% accuracy for softmax and ReLu. NSL-KDD dataset has also been evaluated by several authors [12,13]. The former applies an RNN method with feature reduction, for combining a correlation and information gain, while the latter train an IDS model based on CNN and benchmark the performance of the model with traditional machine learning methods, such as random forest (RF) and support vector machine (SVM), and LSTM.

The significant contributions of this paper are the following: (i) the evaluation of the CSE-CIC-IDS2018 dataset processing with CNN and LSTM; (ii) the evaluation of the dataset with two features reduction methods, namely autoencoder and PCA; and (iii) a set of publicly available Python scripts to assess the dataset and replicate the experiments described in this paper.

The outcomes of the paper, namely the overall architecture and the processing pipeline, are the foundation to evaluate this learning strategy with other public datasets. The aim is to benchmark CNN and LSTM methods and to evaluate the influence of features reduction in the overall performance.

The rest of the paper is organized as follows: Section 2 describes the background behind the IDS, deep learning methods, and features reduction techniques. Next, the overall architecture, the tools and the tests setup is described in Section 3. The results analysis is detailed in Section 4 and, finally, the conclusions and future work suggestions are delineated in Section 5.

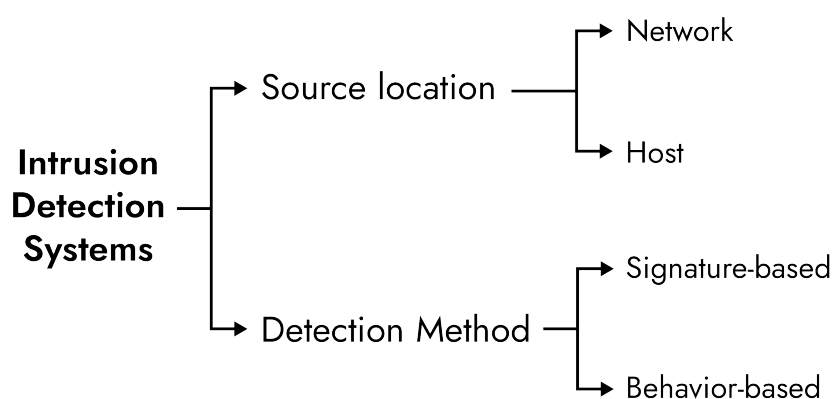
## 2. Fundamentals

This section details the most relevant topics related to this paper's research, namely the fundamentals behind the intrusion detection systems (IDS) and the deep learning methods used to evaluate the dataset.

### 2.1. Behaviour-Based IDS

IDSs aims to process logs generated by the active network equipment, such as computers or routers, to detect deviations to the normal network behaviour. Generally, this anomalous activity is named network intrusions, and, initially, the intrusion detection software was designed to process and analyse the logs generated by different systems [14]. The same motivation has been applied to network traffic processing, being IDSs the most popular application to collect, analyse, and classify network traffic content.

An IDS taxonomy was proposed by several authors [14,15], to group these systems according to relevant characteristics. Figure 1 depicts the overall classification of IDSs according to two of those characteristics, namely audit source location and detection techniques used.



**Figure 1.** Intrusion detection systems taxonomy.

The audit source location distinguishes the IDSs based on the source of the information they analyse. Host- and network-based IDSs are among the most relevant sources from where the IDS can obtain information. A host IDS collects continuously data from an active equipment (e.g., PC or router) to identify deviations to normal behaviour, namely by processing and analysing log files of the services running or by reading activity data, such as CPU or memory activity.

Regarding the detection methods that can be used, behaviour-based and signature-based IDSs are the most commonly used techniques. Behaviour-based IDS are also named as detection by behaviour or anomaly detection IDSs, while signature-based IDSs are also known as knowledge-based or misuse detection IDSs.

Signature-based IDSs are based on the description of known attacks by the means of a signature or a pattern, generally termed a “rule”. It examines the network and compares the collected data (e.g., network traffic) with a rules database (knowledge base) looking for a match with a sample of an already known misuse. An alert is generated if a match is found, being otherwise considered not intrusive, all the events that do not match any signature. The most used approaches for misuse detection are expert system, and pattern matching analysis. The same methodology can be identified in anti-malware functioning. Signature-based IDSs are highly efficient in detecting known attacks, that is, those for which a signature is available. However, these IDSs fail to detect those attacks that correspond to zero-day exploits to which signatures do not yet exist. A well-known and widely used signature-based IDS is snort [16].

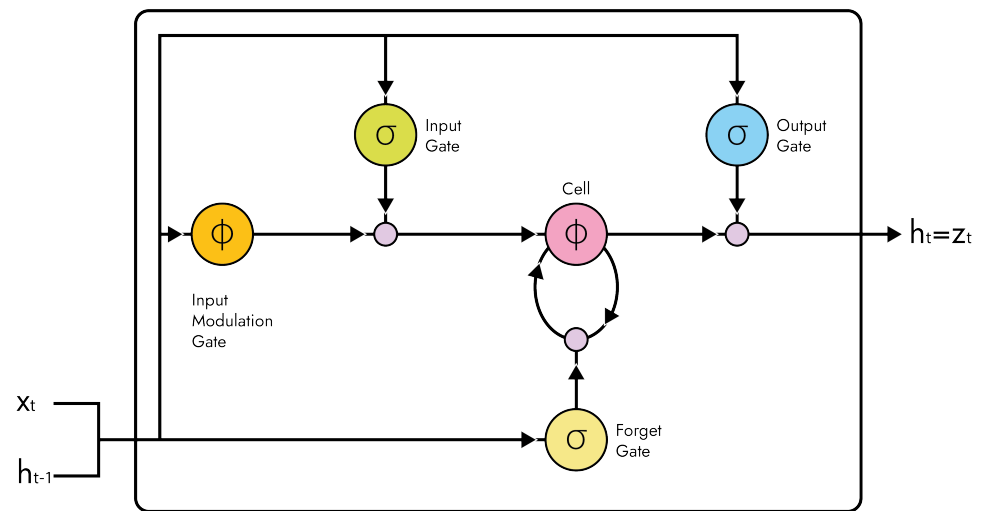
Behaviour-based IDSs presuppose that all intrusive activities are necessarily anomalous. These systems start by building a model that represents a trained normality profile for network activity. They then proceed by looking for anomalous activities, which by definition are those that do not match the previously established profile. Behaviour-based IDSs have two major advantages over those based on signatures. On the one hand, they have the ability to detect unknown anomalies and “zero-day” attacks. On the other hand, the normality profile can be trained for each contextual environment, which potentially increases the difficulty for an attacker to explore the vulnerabilities that a network may have. A detailed description of the most relevant profiling techniques, as well as a list of some behaviour-based IDSs recent developments, can be found in [17,18].

However, while enabling the reaction to unknown attacks, this type of IDS generates a high rate of false positives, which implies a continuous and efficient tuning of the training dataset and adjustment of reaction levels [7].

## 2.2. Long Short-Term Memory

LSTM is an artificial recurrent neural network (RNN) used in deep learning. LSTM aims to overcome the short-term memory issue observed in RNN by predicting entire sequences of organized data while maintaining a certain level of memory regarding past operations [19]. Unlike normal feed-forward neural networks, an LSTM network allows previous outputs to be used as inputs of next layers while having hidden states at arbitrary intervals. This way, we have the possibility of using input of any length; therefore, it is pretty suitable for classifying, processing, and predicting time series, as the traffic collected in a computers network, with time intervals of unknown length [20].

An LSTM architecture is a chain-based structure, as depicted in Figure 2. Information is computed in “cells”, while memory management is carried out by “gates”, generally classified into the following types: forget gate, input gate, and output gate. These gates contain a sigmoid activation function to compress the data received by cells and learn which data is essential to keep or forget. Finally, the predictions are made by passing relevant information to the chain of sequences in the neural network. These networks are widely used and very useful today, being at the forefront of language modelling, such as in translation or text generation, in short, any activity related to reading and writing, because of their ability to recognize patterns throughout time.



**Figure 2.** Long short-term memory architecture.

### 2.3. Convolutional Neural Network

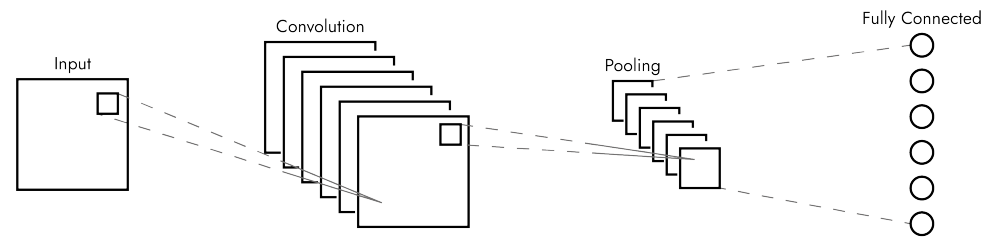
A CNN is a neural network that is suitable to automatically detect and group unique characteristics of the neural network. It is a deep learning algorithm that uses a feed-forward network, which means that it takes an input and assigns the importance (weights and biases that can be learned) to its features [21].

Each neuron of this artificial neural network has a non-linear activation function that produces the final output (Figure 3). Multiple connected neurons in the neighbourhood produce a kernel or weight matrix, and multiple kernels will produce a so-called “convolution layer” (CONV), each one producing an output. The CONV layer carries out convolution operations as it scans an input. It does so by taking advantage of its various hyper-parameters (a parameter that is set before the learning process begins, which is tunable and can affect how well a model trains), such as the filter size (denotes the dimensions of a filter which is applied in the process) and the stride (designates the number of pixels by which the window moves after each operation). The resulting output is referred as feature map or activation map. After that, the processing flows through the pooling layer (POOL), which is a down-sampling operation that takes the average and maximum values of a given region doing some spatial invariance. After gathering these values, the network reaches the fully connected layer (FC) where each input is connected to all neurons.

A popular input to the CNN is an image vector with a set of enclosed features, which, after going through all of these layers, end up in a probabilistic approximation to recognize the desired image. A significant benefit of this algorithm is that the preprocessing required on a regular CNN is much smaller than the other classification algorithms. The use of CNN in IDSs is usually due to the accuracy needed in a specific classification related with patterns. Attack types that happen with a specific pattern are easier to identify using CNN [22].

The use of CNN and LSTM networks to classify data streams in this dataset has been carried out a few times [23–25]. From several articles, the results were always identical, usually above 90% in accuracy, precision, and/or recall. This always depends on the architecture of the network used, but it is easy to conclude that both are equally good for stream classification, even without the application of any method for feature reduction.





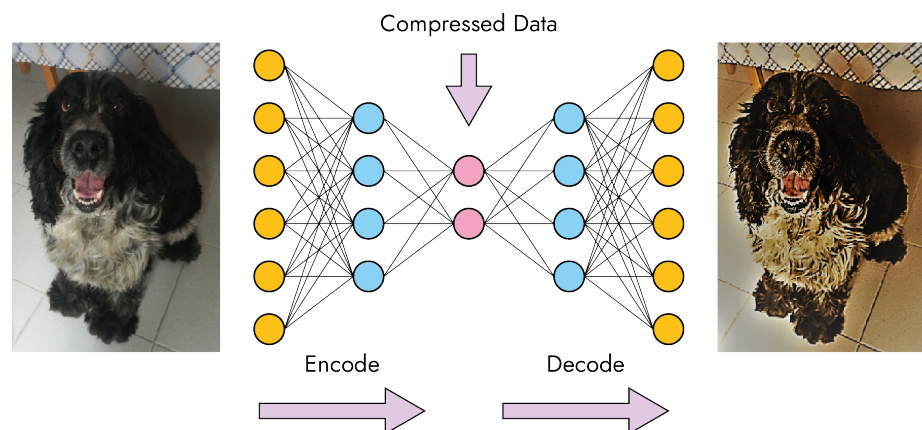
**Figure 3.** Convolutional neural network architecture.

#### 2.4. Features Reduction Techniques

Autoencoder and PCA are two widely used techniques to reduce the features set of a dataset to a stage that does not compromise the overall classification performance.

Autoencoder is an unsupervised learning technique that aims to reduce the number of inputs imposed on a neural network without “damaging” its performance. Simply put, the autoencoder tries to copy its input to output using compression and reconstruction techniques. Generally, autoencoder has two major components: an encoder and a decoder. The encoder is the part of the network that compresses the input and transforms it into its encoded representation. The decoder is the component that aims to reconstruct the given input through the previously mentioned codified representation [26,27].

An autoencoder is used for unsupervised training methods, knowing that it has the ability to compress the information to the minimum number of nodes in the inner layers, this allows, with a proven encoder, the ability to store the same information in a reduced number of features. Since it is built from a neural networks, its basis are from multi-layer perceptrons (MLP) or CNN, among others. The methodology applied in this paper is based on MLP, which is made up of nodes that result from the sum of all the weights plus the bias wrapped in an activation function. This technique has been used for image compression or noise reduction, as depicted in Figure 4. It is able to extract given characteristics from a dataset, as an unsupervised training method, or even to generate new images from noise, as in videos, predicting the next frame by taking the previous frame as input.



**Figure 4.** Autoencoder architecture.

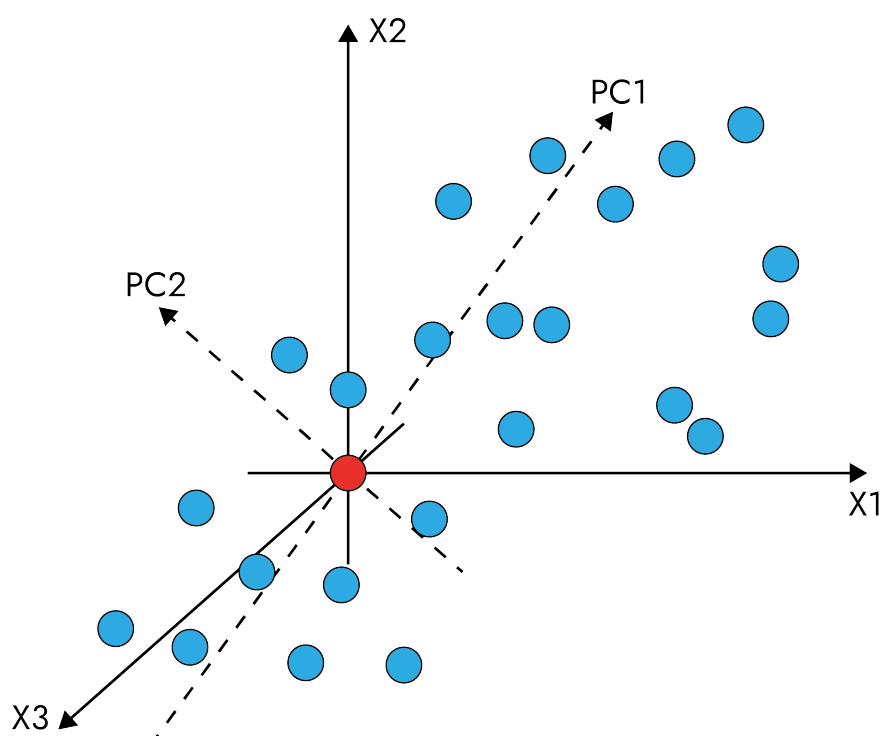
PCA is a mathematical method that converts a set of possibly correlated observable variables into values of linearly uncorrelated variables called principal components. It is important to note that the number of principal components is always less (or equal) to the number of original variables, hence why this method was chosen to test it as a possible input reducer in the network.

PCA functioning is based on decomposing values into correlative synonyms and is sensitive to the original variables’ relative scale or inputs. The axes of the PCA are categorized in order of importance, which means that in clusters, or groups of values that

are separated, their difference is more critical the further apart they are on the x-axis or the horizontal axis [28,29].

Diving to how it works practically, considering a matrix with  $N$  rows, labelled as “observations” and  $K$  columns, labelled as “variables”, PCA constructs a variable space with as many dimensions as there are variables, being that each one represents one coordinate axis (Figure 5). Next, mean centring is the following step, and it involves the subtraction of the variable averages from the data. The vector of averages formed is interpretable as a point in space which is situated in the middle of the point swarm. The subtraction of the averages corresponds to a re-positioning of the coordinate system.

In the “post mean centring” scenario, the data set’s first summary index, or the first principal component, will be computed. The first principal component is the line that best accounts for the shape of the point swarm, as it represents the maximum variance direction on the data. Each observation may be projected onto this line in order to obtain a coordinate value along the “Principal Component line”. This value is known as a score.



**Figure 5.** Principal component analysis architecture.

Usually, the computation of only one principal component is insufficient to complete the required process. Thus, a second principal component is calculated. It is oriented such that it reflects the second largest source of variation in the data while being orthogonal (contrary to being parallel) to the first principal component.

Lastly, when two principal components have been obtained, they together define a plane in the  $K$ -dimensional variable space. When projecting all of the data onto the previous window and plotting the results, it is possible to visualize the structure of the researched data set.

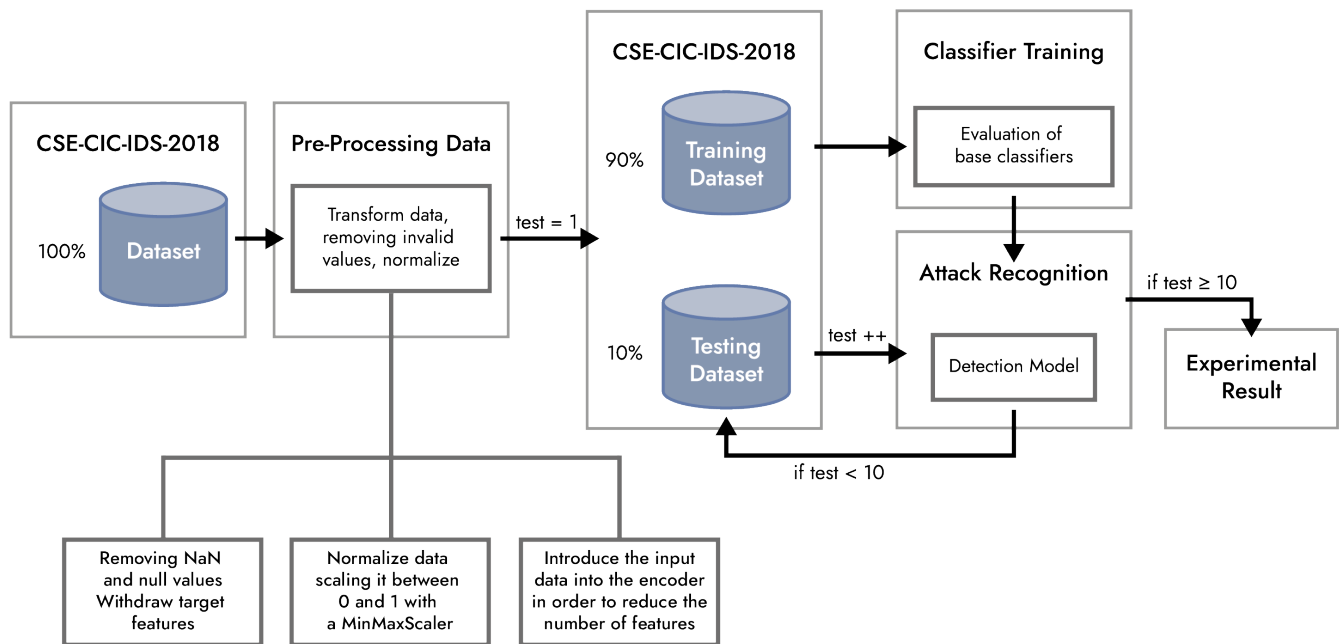
### 3. Overall Architecture

The overall architecture is depicted and explained in this section. A description of the CSE-CIC-IDS2018 dataset and the tools used to process CNN and LSTM methods are explained.

Figure 6 depicts the overall architecture for preprocessing and processing the dataset. The initial dataset can be divided and processed by each day or used as a whole. It is

initially preprocessed by the Orange data mining tool in order to remove inconsistencies (Infinity, NaN, and null values). The result is a clean CSV file loaded to Pandas framework, where the headers are removed and passed to a NumPy array.

The tests were made with the initial features set and by resulting set after autoencoder and PCA methods have been applied. The number of features of each network flow was reduced from 78 to 18.



**Figure 6.** Overall processing architecture.

The initial dataset is then split into training and testing sets. In the test setup described in Figure 6 which represents the validation process of the dataset, the training set is 90% length and the remaining examples correspond to 10% for testing. The training phase builds the separated models for LSTM and CNN deep learning methods. The testing phase process each model with the testing set to recognize the enclosed network attacks. The dataset is cross-validated ten times and the global results are presented. The results identify the metrics obtained by each run and the global result regarding accuracy, precision, recall, and F1 score.

The processing of CNN was made through Keras and the K-fold cross-validation use Python scripts developed under Scikit-learn library.

### 3.1. Dataset

CSE-CIC-IDS2018 is a dataset created by the Canadian Institute for Cybersecurity (CIC) and the Communications Security Establishment (CSE) and has been used to train predictive models to detect intrusions based on network traffic anomalies [5,30]. The dataset contains a different type of attack each day and is divided into two parts: (i) 1 TB of PCAP files with all the packets captured in the test network; (ii) an already processed CSV file, ready to be processed by machine learning algorithms.

Table 1 details the nine days of activity used to benchmark the dataset with the CNN and LSTM methods. It is possible to observe a more significant number of flows related to DoS and DDoS attacks, respectively, on 16 and 21 February. The brute force attack observed on 14 February is the third attack in the number of malicious flows. By their nature, these three attack types are expected to have a larger number of packets and flows compared with other malicious activities.



**Table 1.** Detailed of the attacks used for benchmark.

Date	Attack Type	Malicious Flows	Total Number of Flows
14 February 2018	BruteForce	380,943	1,044,751
15 February 2018	DoS	52,498	1,040,548
16 February 2018	DoS	601,802	1,048,574
21 February 2018	DDoS	687,746	1,048,575
22 February 2018	Web App Attack	362	1,042,965
23 February 2018	Web App Attack	566	1,042,867
28 February 2018	Infiltration	68,236	606,902
1 March 2018	Infiltration	92,403	328,181
2 March 2018	Botnet	286,191	1,044,525

The dataset has six different types of attacks, namely BruteForce, Botnet, denial of service (DoS), distributed DoS (DDoS), web attacks, and infiltration attacks, distributed by specific dates. Table 2 describes the different tools and frameworks used in the execution of each attack. For example, DoS attacks were executed with the following tools: GoldenEye, Slowloris, SlowHTTPTest, and Hulk [31]. Regarding BruteForce, two penetration testing tools of Patator BruteForce suite, available on Kali Linux were used, namely FTP-Patator and SSH-Patator [32]. Damn Vulnerable Web Application (DVWA) was used to generate malicious traffic [33] produced by various web attacks. Infiltration attacks were inflicted by nmap and portscan tools. Both scanners employ passive attacks and collect precious information to initiate a remote access for infiltration purposes [34]. IoT Botnet ARES communications were used to produce Botnet-related traffic [35]. DDoS traffic was generated through the Low Orbit Ion Cannon (LOIC) network stress testing and denial of service attack application [36].

**Table 2.** Tools used in each attack.

Attack Type	Tools Used
BruteForce	FTP-Patator and SSH-Patator
DoS	Hulk, GoldenEye, Slowloris, Slowhttpstest
Web Applications Attack	Damn Vulnerable Web Application (DVWA)
Infiltration	nmap and port scan
Botnet	Ares-botnet
DDoS	Low Orbit Ion Cannon (LOIC)

The tools used to generate the attacks have produced distinct network traffic patterns for the same vulnerability exploits. For example, the various tools used to generate DoS attacks, namely Hulk, GoldenEye, Slowloris, and Slowhttpstest, have the same goal and aim to attain similar results; however, the procedures employs and the network traffic generated by each one is necessarily different. This fact challenges the detector, as for the same attack distinct patterns should be trained to consequently have a better detection, regardless the tool used to generate the attack.

### 3.2. ML Tools

ML algorithms are widely implemented in various tools, as well as applications to support data preprocessing tasks.

Orange data mining [37] was used to remove incomplete and missing data in the dataset. It is an open-source data processing tool equipped with an appealing GUI and a set of machine learning algorithms. However, it is essential to mention that this tool does not have non-blocking interfaces, so handling large amounts of data is challenging.

Keras [38] is an open-source high-level Python API used for a deep neural network. It can run on top of TensorFlow [39] (our case), Microsoft Cognitive Toolkit, R, Theano, or PlaidML. It is a user-friendly and extensible tool, which helps to adjust the model if needed to try out different scenarios. It is also used alongside TensorFlow to implement into the pipeline any functionality that may not be present within Keras framework. TensorFlow [39], an open-source machine learning platform developed by Google, was used to train and test the chosen models.

Scikit-learn [40] is a machine learning library that encompasses several machine learning algorithms, namely for regression and classification as well as data processing functions. This tool was used to prepare training and testing sets, and to calculate ROC curves and confusion matrix. Matplotlib [41] is a library for Python that was used for data visualization and graphical plotting.

### 3.3. Evaluation Metrics

The following metrics were used to evaluate the results depicted in Section 4:

- False positive rate (FPR) refers to the ratio of wrongly predicted attacks to the total observations. This number should be as low as possible.
- Accuracy (A) is the ratio of correctly predicted observation to the total observations.
- Precision (P) means the ratio of correctly predicted positive observations to the total predicted positive observations. Higher precision is related to the low false positive rate—the higher the better.
- Recall (R) should be also the higher the better and indicates the ratio of correctly predicted positive observations to all observations.
- F1 is an harmonic and weighted average of Precision and Recall.

## 4. Results

This section presents the results obtained with the benchmark of CNN and LSTM. Their models were initially evaluated with an 80–20 rule (80% of the dataset for training and the remaining 20% for testing) and then validated by applying a 10-fold cross-validation.

### 4.1. Evaluation of the Models with 80–20 Rule

This subsection describes the results obtained with an 80–20 rule applied to the dataset, in which 80% was used for training and the remaining was used for testing. The input sequence size for the LSTM nets for the next iteration was set to 12 and the following parameters were also chosen for both models: activation, loss functions, and optimizers.

The activation functions used in the LSTM layers were “arctang”, “sigmoid”, and “ReLU”. For CNN layers, namely in the last layer of each model, the “softmax” function was used to make the model probabilistic. Since these models are probabilistic and devoted to classification purposes, the loss function chosen was “Sparse Categorical Crossentropy”. The chosen optimizer was “Adam” in both models, an already developed evolution of the “gradient descent” base algorithm. This optimizer was used in both models with a learning rate of 0.00025.

The results summarized in Table 3 express a high level of precision and accuracy attained with the experiments. As brute force attacks present a sparsity of malicious traffic when added to the total number of attacks in the dataset do not represent a large enough percentage to be noticeable in the end, it is hard to know if there was any improvement with the merging of the data. However, another conclusion that can be drawn is the significant improvement in recognizing infiltration attacks. This is due to the improvement in recognition of benign traffic. With the more significant amount and diversity of data in a single model, the recognition of attacks is substantially better than in individual models.

**Table 3.** Results obtained with LSTM and CNN models with a split of 80% training and 20% testing.

Models	Precision	Recall	Accuracy	FPR	F1
CNN	0.9967	0.9982	0.9993	0.0005	0.9974
LSTM	0.9985	0.9970	0.9994	0.0002	0.9977

#### 4.2. Evaluation of the Models with 10-Fold Cross-Validation

Cross-validation is a technique designed to evaluate predictive models, by partitioning the original dataset into a training set to train the model, and a test set to evaluate it. The models generated through LSTM and CNN were tested with a 10-fold cross-validation, in which in each one of the 10 interactions, 90% of the dataset is used for training, while the remaining 10% was used for testing.

Table 4 summarizes the results obtained with both models but applies a stratified 10-fold cross-validation method, where the dataset is randomly divided into 10 different parts. The training set corresponds to 90% of the dataset, while the remaining 10% is used to test and evaluate each one of the ten iterations.

**Table 4.** Results obtained with LSTM and CNN models by applying 10-fold cross-validation.

Models	Precision	Recall	Accuracy	FPR	AUC	F1
CNN	0.9929	0.9981	0.9987	0.0011	0.9984	0.9955
LSTM	0.9942	0.9932	0.9983	0.0009	0.9961	0.9937

The results attained still reveal a high level of precision and recall on detecting the attacks present in the dataset. The area under the curve (AUC) measures the classifier's performance to distinguish between two classes, in this case, normal traffic and network attack. AUC can be seen as a summary of the ROC curve, and the higher the AUC, the better the model's performance to distinguish between normal traffic and intrusions. The values attained for the AUC measure are above 99%, which reveals the high performance of both methods.

#### 4.3. Feature Reduction Methods Evaluation

A feature reduction attempt was made to reduce the number of features being processed and, at the same time, to decrease the training processing time of each model. The reduction in the parameters set was applied to each flow, and autoencoder and PCA techniques were evaluated.

To evaluate the gain in processing time for CNN model, we picked up the dataset of 2 March 2018. Table 5 summarizes the processing time and the accuracy with the initial set of 78 features, where it is possible to identify the processing time, accuracy, and loss for the first five epochs.

**Table 5.** Training dataset of 2 March 2018 with the initial features set.

Epoch	Time (s)	Accuracy	Loss
1	167	0.981	0.047
2	166	0.999	0.004
3	179	0.999	0.003
4	249	0.999	0.002
5	252	0.999	0.002

For this particular training/testing, an 80–20 distribution was used (80% of the data for training and 20% for testing). When applying the PCA and autoencoder reduction methods

to the same dataset (2 March 2018), the processing time reduces, and the performance remains intact compared with the initial features set results. Table 6 summarizes the results obtained with autoencoder, while Table 7 summarizes the results obtained with PCA. In both cases, the initial set of 78 features was reduced to 18 and only the results of the first 5 epochs are detailed.

**Table 6.** Training dataset of 2 March 2018 with the feature set reduction through autoencoder.

Epoch	Time (s)	Accuracy	Loss
1	45	0.933	0.157
2	44	0.998	0.010
3	44	0.998	0.008
4	44	0.999	0.006
5	44	0.999	0.005

**Table 7.** Training dataset of 2 March 2018 with the feature set reduction through PCA.

Epoch	Time (s)	Accuracy	Loss
1	44	0.945	0.129
2	43	0.998	0.010
3	43	0.999	0.006
4	42	0.999	0.005
5	40	0.999	0.005

Analysing both approaches, it is possible to conclude that, despite there being fewer data in training with autoencoder and PCA methods, they both attain almost the same level of accuracy as with the initial features set (Table 5). The processing time has greatly reduced in both models, as the size of the networks has also reduced. A reduction in convergence time is observed of more than five times the initial time.

#### 4.4. Intrusion Detection Evaluation

Intrusion detection is a compound technique with a deep learning method (CNN or LSTM) and a features reduction approach (autoencoder or PCA). Table 8 summarizes the results obtained with CNN model employing PCA to reduce the features set.

**Table 8.** Results obtained with a training CNN and PCA.

Day/Attack	F1	Precision	Recall	FPR	TNR	Accuracy
2 March 2018/Botnet	0.9992	0.9993	0.9993	0.0002	0.9997	0.9996
21 February 2018/DDoS	1.0000	1.0000	1.0000	0.0000	1.0000	1.0000
15 February 2018/DoS	0.9972	0.9951	0.9994	0.0002	0.9997	0.9997
16 February 2018/DoS	0.7293	0.5739	1.0000	1.0000	0.0000	0.5740
1 March 2018/Infiltration	0.3463	0.7896	0.2218	0.0232	0.9768	0.7642
28 February 2018/Infiltration	NaN	NaN	0.0000	0.0000	1.0000	0.8876
14 February 2018/SshFtpBruteForce	1.0000	1.0000	1.0000	0.0000	1.0000	1.0000
22 February 2018/WebXssBruteForce	0.6909	1.0000	0.5278	0.0000	1.0000	0.9998
23 February 2018/WebXssSQLBruteForce	0.4865	1.0000	0.3214	0.0000	1.0000	0.9996

Table 9 summarizes the results obtained with CNN model employing autoencoder method to reduce the features set.

**Table 9.** Results obtained with a training CNN and autoencoder.

Day/Attack	F1	Precision	Recall	FPR	TNR	Accuracy
2 March 2018/Botnet	0.9990	0.9984	0.9996	0.0005	0.9994	0.9995
21 February 2018/DDoS	1.0000	1.0000	1.0000	0.0000	1.0000	1.0000
15 February 2018/DoS	0.9960	0.9928	0.9992	0.0003	0.9996	0.9996
16 February 2018/DoS	0.7293	0.5739	1.0000	1.0000	0.0000	0.5739
1 March 2018/Infiltration	0.3421	0.7595	0.2208	0.0274	0.9726	0.7609
28 February 2018/Infiltration	NaN	NaN	0.0000	0.0000	1.0000	0.8876
14 February 2018/SshFtpBruteForce	0.9998	0.9995	1.0000	0.0002	0.9997	0.9998
22 February 2018/WebXssBruteForce	0.5600	1.0000	0.3889	0.0000	1.0000	0.9998
23 February 2018/WebXssSQLBruteForce	0.2769	1.0000	0.1607	0.0000	1.0000	0.9995

The same datasets were evaluated with the LSTM, PCA, and autoencoder methods. Table 10 summarizes the results obtained with LSTM model employing PCA method to reduce the features set.

**Table 10.** Results obtained with a training LSTM and PCA.

Day/Attack	F1	Precision	Recall	FPR	TNR	Accuracy
2 March 2018/Botnet	0.9994	0.9992	0.9997	0.0003	0.9997	0.9997
21 February 2018/Ddos	0.9987	1.0000	0.9975	0.0001	0.9999	0.9983
15 February 2018/Dos	0.9958	0.9945	0.9971	0.0003	0.9997	0.9996
16 February 2018/Dos	1.0000	1.0000	1.0000	0.0000	1.0000	1.0000
1 March 2018/Infiltration	0.4852	0.7757	0.3530	0.0400	0.9600	0.7891
28 February 2018/Infiltration	NaN	NaN	0.0000	0.0000	1.0000	0.8876
14 February 2018/SshFtpBruteForce	0.9998	0.9997	0.9999	0.0002	0.9998	0.9999
22 February 2018/WebXssBruteForce	0.8174	0.7224	0.9412	0.4444	0.0001	1.0000
23 February 2018/WebXssSQLBruteForce	0.4658	1.0000	0.3036	0.0000	1.0000	0.9996

Table 11 summarizes the results obtained with LSTM model employing autoencoder method to reduce the features set.

Table 12 summarizes the average F1 of the respective models with both feature reduction methods, for each trained attack.

By analysing the results obtained with the experiments, it is possible to elaborate the following main conclusions:

- The difference of the results obtained between PCA and autoencoder is not significantly large to be able to say that the data reduction method affects the performance of the deep learning models;
- The difference in the results obtained with the different attacks is minimal. The results obtained with LSTM on 16 February 2018 were a bit lower, but also with good performance;
- The results with brute force attacks reveal inconsistency, despite the good performance on day 14 February 2018;
- Infiltration attacks were the least successful, as none of the models were able to classify the traffic correctly.

**Table 11.** Results obtained with a training LSTM and autoencoder.

Day/Attack	F1	Precision	Recall	FPR	TNR	Accuracy
2 March 2018/Botnet	0.9745	0.9788	0.9702	0.0079	0.9921	0.9861
21 February 2018/DDoS	0.9990	1.0000	0.9979	0.0000	1.0000	0.9986
15 February 2018/DoS	0.9917	0.9894	0.9941	0.0006	0.9994	0.9992
16 February 2018/DoS	1.0000	1.0000	1.0000	0.0000	1.0000	1.0000
1 March 2018/Infiltration	0.4407	0.7878	0.3059	0.0323	0.9677	0.7814
28 February 2018/Infiltration	NaN	NaN	0.0000	0.0000	1.0000	0.8876
14 February 2018/SshFtpBruteForce	0.9998	0.9996	0.9999	0.0000	0.9998	0.9998
22 February 2018/WebXssBruteForce	0.4444	1.0000	0.2857	0.0000	1.0000	0.9998
23 February 2018/WebXssSQLBruteForce	0.3824	1.0000	0.2364	0.0000	1.0000	0.9996

**Table 12.** Averages of the F1 measure obtained for each pair of deep learning and feature reduction methods, and for each attack type.

Attack	CNN-PCA	CNN-Autoencoder	LSTM-PCA	LSTM-Autoencoder
Botnet	0.9992	0.9990	0.9994	0.9745
DDoS	1.0000	1.0000	0.9987	0.9990
DoS	0.8632	0.8626	0.9979	0.9958
Infiltration	-	-	-	-
BruteForce	0.7258	0.6122	0.7610	0.6089

The results obtained with the experiment denoted that both models generally achieved excellent detection performance for attacks that produce more network traffic flows, such as Botnet or DoS attacks. Infiltration attacks, however, fell short of expectations. Despite the good result on the 14 February 2018, there are not enough data to gather any well-supported conclusion for brute force attacks.

#### 4.5. Related Work Comparison

The CSE-CIC-IDS2018 dataset has been widely used by several authors to benchmark network intrusion detection techniques, mainly those based on machine learning and deep learning methods. Table 13 details the comparison of our best results with those obtained by other authors, for the CSE-CIC-IDS2018 dataset.

**Table 13.** Comparison with other works available in the literature.

Models	Accuracy	Precision	Recall	AUC
CNN—our results	0.9987	0.9929	0.9981	0.9984
LSTM—our results	0.9983	0.9942	0.9932	0.9961
CNN [23]	0.9999	0.8175	0.8225	n/a
LSTM [25]	0.9620	0.9600	0.9600	n/a
XGBoost [42]	0.9600	0.9900	0.7900	n/a
Deep autoencoder [43]	0.9920	0.9500	0.9890	n/a
Random Forest, Decision Tree [44]	0.9999	1.000	0.999	n/a
Logistic Regression, Decision Tree, and Gradient Boosting [45]	0.9880	0.9880	0.9710	0.9410



The results we have obtained with CNN and LSTM methods are competitive with those presented from other authors. The accuracy is above 0.96 in all the models and our results are also very competitive regarding the precision and recall, when comparing with the models that present the best results. Some of the models being compared do not use cross validation and, in some cases, the results are global and do not give information about which attacks are being detected. Although some additional experiments have to be made to compare our results with the methods described in the literature, the overall results obtained show the appropriateness of applying CNN and LSTM methods to detect intrusions in the dataset. The use of these methods together with PCA and autoencoder, as depicted in Table 4, while lowering slightly the performance obtained still keep the results benchmarked with other related work results.

## 5. Conclusions

This paper presented a comparative study about using deep learning methods to detect network intrusions. The dataset CSE-CIC-IDS2018 was used to benchmark CNN and LSTM deep learning models, employing feature reduction through autoencoder and PCA methods.

Globally, the results revealed a good detection performance with both CNN and LSTM. It is also possible to conclude that feature reduction methods did not impact the overall detection performance while reducing the processing time. Autoencoder only compresses the data, so the processing is not slower than PCA. Regarding the fact that PCA may lose an amount of information to the point that it is not suitable for detecting certain types of attacks, the results show a slight decrease in “hits” with PCA, but not too significant. Both feature reduction methods converged to almost the same results, using both CNN and LSTM.

Some datasets are too unbalanced, having too much more benign traffic compared with the traffic related to intrusions. Some examples refer to the XSS, brute force, and SQL Injection attacks. Expected models faced constraints with detecting attacks that have not been trained yet. The detection of variations of attacks, such as DoS and DDoS, is also challenging, frequently generating false positives.

Regarding future work, the following topics are being under development. Firstly, we are extending the learning architecture to evaluate the influence of an ensemble-based model, which aggregates the classification of distinct classifiers. We aim to evaluate the improvement made by an ensemble-based model and how it may benefit the overall classification results, mainly those misclassified examples by CNN and LSTM classifiers separately.

Secondly, we aim to evaluate the learning architecture proposed in this paper with other public IDS datasets, besides CSE-CIC-IDS2018.

Finally, as the dataset has both network flows and the corresponding packets, we intend to apply the same learning strategy and compare the results obtained with those of network flows.

**Author Contributions:** Conceptualization, M.A. and L.O.; data curation, A.S., J.V., R.S. and T.M.; formal analysis, M.A. and L.O.; funding acquisition, L.O.; investigation, M.A., L.O., A.S., J.V., R.S. and T.M.; methodology, M.A. and L.O.; software, A.S., J.V., R.S. and T.M.; supervision, M.A. and L.O.; validation, M.A. and L.O.; visualization, A.S., J.V., R.S. and T.M.; writing—original draft preparation, M.A., L.O., A.S., J.V., R.S. and T.M.; writing—review and editing, M.A. and L.O. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work has been funded by national funds through FCT—Fundação para a Ciência e Tecnologia, I.P. under the Project UIDB/05567/2020.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data presented in this study are available on request, from the corresponding authors.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

A	accuracy
AUC	area under the curve
CNN	convolutional neural network
CIC	Canadian Institute for Cybersecurity
CSE	Communications Security Establishment
DDoS	distributed denial of service
DoS	denial of service
DVWA	Damn Vulnerable Web Application
FPR	false positive rate
FP	false positive
LSTM	long short-term memory
LOIC	low orbit ion cannon
LVQ	learning vector quantization
MLP	multi-layer perceptron
NN	neural network
TNR	true negative rate
P	precision
PCA	principal component analysis
R	recall
RAM	random access memory
RF	random forest
ROC	receiver operating characteristic
SVM	support vector machine
TB	terabyte

## References

1. Al Tobi, A.M.; Duncan, I. KDD 1999 generation faults: A review and analysis. *J. Cyber Secur. Technol.* **2018**, *2*, 164–200. [CrossRef]
2. Tavallae, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A detailed analysis of the KDD CUP 99 data set. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, ON, Canada, 8–10 July 2009; pp. 1–6.
3. Massicotte, F.; Gagnon, F.; Labiche, Y.; Briand, L.; Couture, M. Automatic evaluation of intrusion detection systems. In Proceedings of the 2006 22nd Annual Computer Security Applications Conference (ACSAC'06), Miami Beach, FL, USA, 11–15 December 2006; pp. 361–370.
4. McHugh, J. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.* **2000**, *3*, 262–294. [CrossRef]
5. A Realistic Cyberdefense Dataset (CSE-CIC-IDS2018). Available online: <https://registry.opendata.aws/cse-cic-ids2018/> (accessed on 19 March 2022).
6. Thapa, N.; Liu, Z.; Kc, D.B.; Gokaraju, B.; Roy, K. Comparison of machine learning and deep learning models for network intrusion detection systems. *Future Internet* **2020**, *12*, 167. [CrossRef]
7. Ferreira, P.; Antunes, M. Benchmarking behaviour-Based Intrusion Detection Systems with Bio-inspired Algorithms. In Proceedings of the Security in Computing and Communications: 8th International Symposium, SSCC 2020, Chennai, India, 14–17 October 2020; Revised Selected Papers; Springer Nature: Singapore, 2021; Volume 1364, p. 152.
8. Leevy, J.L.; Khoshgoftar, T.M. A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data. *J. Big Data* **2020**, *7*, 1–19. [CrossRef]
9. Ferrag, M.A.; Maglaras, L.; Moschogiannis, S.; Janicke, H. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *J. Inf. Secur. Appl.* **2020**, *50*, 102419. [CrossRef]
10. Basnet, R.B.; Shash, R.; Johnson, C.; Walgren, L.; Doleck, T. Towards Detecting and Classifying Network Intrusion Traffic Using Deep Learning Frameworks. *J. Internet Serv. Inf. Secur.* **2019**, *9*, 1–17.
11. Le, T.T.H.; Kim, Y.; Kim, H. Network intrusion detection based on novel feature selection model and various recurrent neural networks. *Appl. Sci.* **2019**, *9*, 1392. [CrossRef]

12. Pooja, T.; Shrinivasacharya, P. Evaluating neural networks using Bi-Directional LSTM for network IDS (intrusion detection systems) in cyber security. *Glob. Transitions Proc.* **2021**, *2*, 448–454.
13. Ding, Y.; Zhai, Y. Intrusion detection system for NSL-KDD dataset using convolutional neural networks. In Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence, Shenzhen, China, 8–10 December 2018; pp. 81–85.
14. Fuchsberger, A. Intrusion detection systems and intrusion prevention systems. *Inf. Secur. Tech. Rep.* **2005**, *10*, 134–139. [[CrossRef](#)]
15. Hindy, H.; Brosset, D.; Bayne, E.; Seeam, A.; Tachtatzis, C.; Atkinson, R.; Bellekens, X. A taxonomy and survey of intrusion detection system design techniques, network threats and datasets. *arXiv* **2018**, arXiv:1806.03517v1.
16. Snort—Network Intrusion Detection & Prevention System. Available online: <https://www.snort.org/> (accessed on 19 March 2022).
17. Mishra, P.; Varadharajan, V.; Tupakula, U.; Pilli, E.S. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Commun. Surv. Tutor.* **2018**, *21*, 686–728. [[CrossRef](#)]
18. Alsoufi, M.A.; Razak, S.; Siraj, M.M.; Nafea, I.; Ghaleb, F.A.; Saeed, F.; Nasser, M. Anomaly-based intrusion detection systems in iot using deep learning: A systematic literature review. *Appl. Sci.* **2021**, *11*, 8383. [[CrossRef](#)]
19. Mirza, A.H.; Cosan, S. Computer network intrusion detection using sequential LSTM neural networks autoencoders. In Proceedings of the 2018 26th signal processing and communications applications conference (SIU), Izmir, Turkey, 2–5 May 2018; pp. 1–4.
20. Sherstinsky, A. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Phys. D Nonlinear Phenom.* **2020**, *404*, 132306. [[CrossRef](#)]
21. Susilo, B.; Sari, R.F. Intrusion detection in IoT networks using deep learning algorithm. *Information* **2020**, *11*, 279. [[CrossRef](#)]
22. Patterson, J.; Gibson, A. *Deep Learning: A practitioner's Approach*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2017.
23. Kim, J.; Kim, J.; Kim, H.; Shim, M.; Choi, E. CNN-based network intrusion detection against denial-of-service attacks. *Electronics* **2020**, *9*, 916. [[CrossRef](#)]
24. Chastikova, V.; Sotnikov, V. Method of analyzing computer traffic based on recurrent neural networks. *J. Phys. Conf. Ser.* **2019**, *1353*, 12133. [[CrossRef](#)]
25. Lin, P.; Ye, K.; Xu, C.Z. Dynamic network anomaly detection system by using deep learning techniques. In *Cloud Computing—CLOUD 2019, Proceedings of the International Conference on Cloud Computing, San Diego, CA, USA, 25–30 June 2019*; Springer: Cham, Switzerland, 2019; pp. 161–176.
26. Kwon, D.; Kim, H.; Kim, J.; Suh, S.C.; Kim, I.; Kim, K.J. A survey of deep-learning-based network anomaly detection. *Clust. Comput.* **2019**, *22*, 949–961. [[CrossRef](#)]
27. Pinaya, W.H.L.; Vieira, S.; Garcia-Dias, R.; Mechelli, A. Autoencoders. In *Machine Learning*; Elsevier: Amsterdam, The Netherlands, 2020; pp. 193–208.
28. Varma, P.R.K.; Kumari, V.V.; Kumar, S.S. A survey of feature selection techniques in intrusion detection system: A soft computing perspective. In *Progress in Computing, Analytics and Networking*; Springer: Singapore, 2018; pp. 785–793.
29. Uddin, M.P.; Mamun, M.A.; Hossain, M.A. PCA-based feature reduction for hyperspectral remote sensing image classification. *IETE Tech. Rev.* **2021**, *38*, 377–396. [[CrossRef](#)]
30. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **2018**, *1*, 108–116.
31. Muraleedharan, N.; Janet, B. A deep learning based HTTP slow DoS classification approach using flow data. *ICT Express* **2021**, *7*, 210–214.
32. Patator—Penetration Testing Tools. Available online: <https://en.kali.tools/?p=147> (accessed on 19 March 2022).
33. DVWA—Damn Vulnerable Web Application. Available online: <https://dvwa.co.uk/> (accessed on 19 March 2022).
34. Shah, M.; Ahmed, S.; Saeed, K.; Junaid, M.; Khan, H.; Rehman, A.U. Penetration testing active reconnaissance phase—optimized port scanning with nmap tool. In Proceedings of the 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, Pakistan, 30–31 January 2019; pp. 1–6.
35. Kompougias, O.; Papadopoulos, D.; Mantas, E.; Litke, A.; Papadakis, N.; Paraschos, D.; Kourtis, A.; Xylouris, G. IoT Botnet Detection on Flow Data using Autoencoders. In Proceedings of the 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), Athens, Greece, 7–10 September 2021; pp. 506–511.
36. Nagpal, B.; Sharma, P.; Chauhan, N.; Panesar, A. DDoS tools: Classification, analysis and comparison. In Proceedings of the 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 11–13 March 2015; pp. 342–346.
37. Orange Data Mining—Data Mining. Available online: <https://orangedatamining.com/> (accessed on 19 March 2022).
38. Keras: The Python Deep Learning API. Available online: <https://keras.io/> (accessed on 19 March 2022).
39. Tensorflow. Available online: <https://tensorflow.org/> (accessed on 19 March 2022).
40. Scikit-Learn: Machine Learning in Python: Scikit-Learn 1.0.1. Available online: <https://scikit-learn.org/> (accessed on 19 March 2022).
41. Matplotlib—Visualization with Python. Available online: <https://matplotlib.org/> (accessed on 19 March 2022).
42. D'hooge, L.; Wauters, T.; Volckaert, B.; De Turck, F. Inter-dataset generalization strength of supervised machine learning methods for intrusion detection. *J. Inf. Secur. Appl.* **2020**, *54*, 102564. [[CrossRef](#)]

43. Catillo, M.; Rak, M.; Villano, U. 2l-zed-ids: A two-level anomaly detector for multiple attack classes. In *Artificial Intelligence and Network Applications—WAINA 2020, Proceedings of the Workshops of the International Conference on Advanced Information Networking and Applications, Caserta, Italy, 15–17 April 2020*; Springer: Cham, Switzerland, 2020; pp. 687–696.
44. Huancayo Ramos, K.S.; Sotelo Monge, M.A.; Maestre Vidal, J. Benchmark-based reference model for evaluating botnet detection tools driven by traffic-flow analytics. *Sensors* **2020**, *20*, 4501. [[CrossRef](#)] [[PubMed](#)]
45. Fitni, Q.R.S.; Ramli, K. Implementation of ensemble learning and feature selection for performance improvements in anomaly-based intrusion detection systems. In *Proceedings of the 2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT), Bali, Indonesia, 7–8 July 2020*; pp. 118–124.