Big Data Research 36 (2024) 100446

Contents lists available at ScienceDirect

Big Data Research

journal homepage: www.elsevier.com/locate/bdr

A Real Time Deep Learning Based Approach for Detecting Network Attacks

Christian Callegari, Stefano Giordano, Michele Pagano *

Dept. of Information Engineering, University of Pisa, Via Caruso, 16, Pisa, 56126, Italy

ARTICLE INFO

Keywords:
Machine learning
Intrusion detection system
Deep neural networks

ABSTRACT

Anomaly-based Intrusion Detection is a key research topic in network security due to its ability to face unknown attacks and new security threats. For this reason, many works on the topic have been proposed in the last decade. Nonetheless, an ultimate solution, able to provide a high detection rate with an acceptable false alarm rate, has still to be identified. In the last years big research efforts have focused on the application of Deep Learning techniques to the field, but no work has been able, so far, to propose a system achieving good detection performance, while processing raw network traffic in real time. For this reason in the paper we propose an Intrusion Detection System that, leveraging on probabilistic data structures and Deep Learning techniques, is able to process in real time the traffic collected in a backbone network, offering *excellent* detection performance and low false alarm rate. Indeed, the extensive experimental tests, run to validate our system and compare different Deep Learning techniques, confirm that, with a proper parameter setting, we can achieve about 92% of detection rate, with an accuracy of 0.899. Finally, with minimal changes, the proposed system can provide some information about the kind of anomaly, although in the multi-class scenario the detection rate is slightly lower (around 86%).

1. Introduction

In recent years Internet has become the playground for providing sensitive services to an ever growing amount of end-users, most of them only partially aware of the risks deriving from information sharing on the net. In spite of the development of cryptographic primitives and their use in secure protocols, a major role in this evolutionary process will be played by Intrusion Detection Systems (IDSs), which should be able to protect legitimate users against malicious activities of any type. To also tackle unknown attacks, the potentialities of anomaly-based IDSs have been deeply investigated and different methods have been considered.

In a nutshell, an anomaly-based IDS (in contrast with a misuse-based IDS, which leverages on signatures to detect attacks) learns a reference model and uses this model to discriminate between *normal* network behavior (i.e., without attacks) and attacks. Given this working principle, the application of machine learning methods appears to be quite straightforward. Indeed, in the years, many papers have proposed to apply, among the others, clustering [1,2], SVM [3,4] and PCA [5,6].

More recently, the research focus has moved towards Deep Learning techniques and, as a result, many works (see Section 2) have proposed to apply Deep Neural Networks (DNNs) of several kinds (e.g., Multi-Layer Perceptron, Convolutional Neural Networks) to the detection of attacks in network traffic. Such systems generally offer very good performance, outperforming most of the "more classical" approaches.

Nonetheless, most of the proposed works present some common drawbacks. Indeed, as highlighted below in Section 2 (see also the more comprehensive reviews in [7,8]), almost all the systems are tested over already processed traffic traces (usually KDD [9] or its cleaned-up version NSL-KDD, in which redundant records are removed to enable the classifiers to produce an un-biased result), resulting in two main problems: first of all, they do not take into consideration the need of extracting the traffic features before actually running the attack detection phase, and secondly several features require the knowledge of the whole data-set to be computed (more details about the implications of the use of such a kind of data-sets will be given in the following). It is clear that this makes the deployment unfeasible in a real-world environment, where real time attack detection is a strict constraint. Moreover, most of the works do not discuss the rationale behind the choice of a given DNN or the settings of the different parameters.

1.1. Our contribution

As above discussed, quite a few works have already addressed the proposal of applying Deep Learning techniques to the network attack

* Corresponding author.
E-mail address: michele.pagano@unipi.it (M. Pagano).

https://doi.org/10.1016/j.bdr.2024.100446
Received 20 December 2021; Received in revised form 20 December 2023; Accepted 19 February 2024
Available online 27 February 2024
2214-5796/© 2024 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

detection problem. Nonetheless, our proposal is original from different points of view:

- We address the problem of performing network attack detection with Deep Learning techniques in real time from rough packet-level traces (namely, pcap files). Indeed, almost all of the previously proposed systems assume to work on traffic features already extracted and in most cases such features require the knowledge of the entire traffic data-set, making impossible to apply such methods in a real-world environment. In contrast, our proposal is able, by means of probabilistic data structures, to compute traffic features in real time, without the need of knowing the whole data-set
- We compare the performance of five kinds of DNNs, in terms of both computational time and detection performance indexes
- We evaluate the system performance when varying the setup of the neural networks, providing some insights on the impact of the different parameters on the system performance

This system has been originally proposed by the authors in [10]. However, the current version has been extended including two new typologies of DNNs and Multi-class classification, detailing the use of hash tables and improving the background information (namely, the related work section and the subsection about MAWI Traffic Traces). Finally, all the tests have been repeated on newer hardware (with a significant reduction of the processing time).

1.2. Paper organization

The remainder of this paper is organized as follows: Section 2 discusses the related works, while Section 3 provides an overview of the theoretical background, focusing on the description of the different DNNs tested in our work. Then, Section 4 presents the general structure of our IDS and in Section 5 we describe the experimental results. Finally, Section 6 concludes the paper with some final remarks.

2. Related work

In the recent years the application of deep learning techniques to network security has attracted a lot of research efforts. As a result, several deep learning based IDSs have been proposed in the literature, differing one from the other mainly in terms of traffic descriptors and neural network topology. In the following we briefly review some of them, not providing a full survey of the literature on the topic, but picking up some significant examples of the application of different deep learning techniques (described in Section 3) to network attack detection. For sake of readability, the contributions are divided depending on the considered kind of neural networks, and then a few comparative assessments are discussed in section 2.4.

2.1. Multi–layer perceptron (MLP)

Cannady [11] first presented an analysis of the applicability of neural networks to the identification of external attacks against a network. The data-set used in this analysis was created by a network monitor and it was composed of 10.000 connection records, including approximately 3.000 simulated attacks. Each connection was described by a vector of nine features. The neural network was an MLP of four fully connected layers with nine input nodes and two output nodes for binary classification.

Few years later, Moradi [12] compared two different MLP architectures for network intrusion detection and multiclass classification of attacks. The two compared neural networks were a three layers MLP with 35 neurons each and a two layers MLP with 45 and 35 neurons, respectively. The data-set used in this study was the DARPA'99 data-set, composed of about 20.000 connection records with 10.000 anomalies. Each connection record was described by 41 features, but only 35 were considered.

2.2. Convolutional neural network (CNN)

More recently, Vinayakumar [13] utilized a simple CNN architecture, with multiple hidden layers, and different hybrid CNNs. The data-set used was KDD, which contains five millions of connection records, each of which consisting of 41 features, labeled as either normal or attack.

Li in [14] proposed to convert the traffic features in an image to be used as inputs of the CNN models. The data-set used was KDD. Each connection vector of 41 features was converted into a 8*8 gray-scale image. Then, two different popular CNN models, ResNet-50 and GoogLeNet, were employed for classification.

Wu in [15] compared 1D and 2D CNN models and proposed another way to transform a features vector into an image. The data-set used was KDD and each connection was converted into a 11*11 image. According to the results, the accuracy was better and testing time was smaller using 2D CNN with images rather than 1D CNN with features vectors. In conclusion two different 2D CNN models, one with one hidden layer and the other with two hidden layers, were compared on KDD, showing that the two hidden layers model achieves higher accuracy.

The first work in which CNN-based system is tested against a data-set different from KDD is [16], where the authors evaluate their proposal on both KDD and the UNSW-NB15 data-set [17] (given by a combination of real and synthetic traffic). The results on the UNSW-NB15 data-set are far from being acceptable, with accuracy that is even zero for some attack classes.

CNNs are very effective in detecting DDoS attacks. For instance, CNNs from the field of Natural Language Processing have been applied in [18] to a 2-D matrix of 11 packet features and the proposed system, LUCID (Lightweight, Usable CNN in DDoS Detection), achieves a detection rate of 0.9952 with an accuracy of almost 99% on the ISCX2012 data-set.

2.3. Recurrent neural network (RNN) and variants

Differently from the other works, Yin [19] proposed an IDS based on RNN, RNN-IDS. The model was evaluated for both binary and multiclass classification using KDD data-set. Experiments showed that the proposed model performs better than the other classification algorithms in both cases.

Still based on RNN, several systems (e.g., Long-Short Term Memory RNN - LSTM, Gated recurrent units RNN - GRU) have been proposed and tested over the KDD data-set [20][21], while only a few have been tested on other data-sets: [22] uses some aggregate traffic specially generated for such a work; [23] is tested over the CIDDS-001 data-set [24] (partly emulated and partly simulated, it only contains a few samples of few attacks); and [25] makes use of a specially generated BGP data-set (quite limited and only containing routing flows) and KDD.

2.4. Comparative assessments

Some works present an empirical comparison among several kinds of DNNs. As an example, [26] performs a comparison between CNN and LSTM over the KDD data-set, while in [27] the authors compare the performance offered by Fully Connected Network (FCN), Variational AutoEncoder (VAE), and Long Short-Term Memory with Sequence to Sequence (LSTM Seq2Seq) over KDD and Kyoto Honeypot data-set [28] (which only contains attack traffic), but they don't specify how the traffic flows are processed and which features are extracted.

Finally, it is interesting the work by Know [29], where three different CNN architectures were compared using three data-sets. The three CNNs models are called shallow CNN, moderate CNN, and deep CNN, based on the depth of the network. The data-sets used are KDD, Kyoto-Honeypot and MAWILab. The experiments showed that the shallow CNN model outperforms the other CNNs on the KDD, but the three

models obtain similar results on Kyoto-Honeypot and MAWILab data-sets. This work is particularly interesting mainly because it represents the first (and only) attempt to use a data-set composed of raw traffic traces (the MAWIlab data-set), instead of data-sets given by (already processed) features vector (e.g., KDD). Nonetheless, the authors still process the traffic traces offline, without providing any hints for implementing an online anomaly detection.

However, it is important to point out that most of the recent works is tested on KDD (for more references see also the literature survey in [8]). For instance, in [30] performance of a DNN with 20 hidden layers have been evaluated on *the most used data sets, i.e., KDD, NSL-KDD, and UNSW-NB15*, while [8] combines CNN-based classification with a feature selection phase based on the conditional random field and linear correlation coefficient-based feature selection (CRF-LCFS) algorithm with an overall detection accuracy (over KDD) of 98.8%.

Hence, to the best of our knowledge, our work is the first presenting a deep learning based attack detection system able to process raw network traffic online, tested over a recent real traffic data-set.

3. Deep neural networks

Before detailing the proposed system, in this section we review some of the basics on deep learning. It is worth noting that the aim of the section is just to make easier to understand the system description (e.g., by clarifying the meaning of the different parameters that will be used in the following) and not to provide a full description of the different techniques, for which we refer the reader to [31] and references therein.

Artificial Neural Networks (ANNs) are part of machine learning, a research field that gives computers the ability *to learn* from experience and mistakes without being explicitly programmed. As the name suggests, ANNs are inspired by the brain structure and its working principles. In a nutshell, ANNs are composed of simple processing units, named neurons, organized into three kinds of layers: an input layer, one or more hidden layers, and an output layer. A neuron receives input from the nodes of the previous layer, each with an associated weight, and computes an output, given by the application of an activation function to the weighted sum of the inputs.

The working process of an ANN, independently of the typology and objective of the specific network, can be split into two main phases:

- Learning phase: the ANN is fed with a labeled data-set, so as to "learn" how it should work for a specific purpose. During this phase the weights of the network connections are computed
- Inference phase: the *trained* ANN is used to perform a specific task (i.e., classification or regression)

DNNs are a specific case of ANNs that have more than one hidden layer, as shown in Fig. 1. Due to the (potentially) high number of layers and neurons, the computational complexity of DNNs is higher wrt simple ANNs, but these networks are capable of learning high-level patterns with more complexity than shallower neural networks. Nowadays DNNs are employed for many artificial intelligence applications, such as automatic speech recognition, image processing, natural language processing, and bioinformatics [32].

There exist several typologies of DNNs, distinguished on the basis of their architecture; in the following we briefly review the five different kinds of DNNs used in this work, namely Multi–Layer Perceptron, Recurrent Neural Networks, Convolutional Neural Networks, Long-Short Term Memory Networks, and Gated Recurrent Units Networks.

3.1. Multi–layer perceptron

An MLP, the most basic kind of DNNs, is a feedforward (connections between neurons are only possible in the direction input–output) ANN that consists of an input layer, an output layer and several hidden layers. Each layer is composed of one or more nodes.

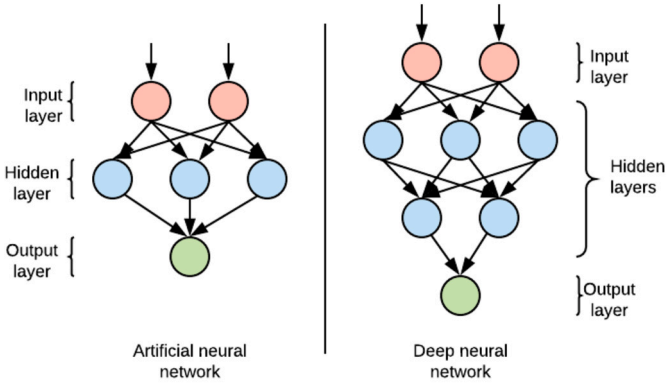

Fig. 1. Deep neural network.

The MLPs are trained in a supervised learning manner with the back-propagation algorithm, consisting of two steps: forward and backward pass. Training can be performed in several ways: stochastic training (weights are updated for each training sample within the data-set), batch training (model is only updated after all the training samples have been evaluated, using the average of losses), and mini batch training (the considered batch is smaller than the whole data-set and the weights are update after a batch has been evaluated). Finally, it is also important to define an epoch, which is one forward pass and one backward pass of all the training samples (i.e., a full training phase over all the training data-set); therefore if mini-batch training is used, one epoch terminates after $\frac{sample\_in\_dataset}{batch\_size}$ iterations.

3.2. Convolutional neural networks

A CNN is a particular MLP neural network commonly used for finding patterns in images. CNNs are composed of an input layer, one or more convolutional layers followed by a pooling layer, one or more fully connected layers (FCLs), and one output layer, as described in Fig. 2.

CNN is not a fully connected neural network, indeed each node in a layer only receives inputs from a small, local subset of input nodes or neurons of the previous layer. This subset is named *receptive field* of a neuron. In this way, each neuron still performs a two steps computation, first a dot product between weights and inputs, followed by a non linear activation function, as for generic neurons, but now the input is restricted to the receptive field, reducing the number of parameters (weights) in the network.

3.3. Recurrent neural networks

An RNN consists of an input layer, an output layer and one or more hidden layers, each composed of one or more recurrent neurons. A recurrent neuron is connected to all nodes in the previous layer, but it has also a recurrent connection that brings the output back to the node itself. Fig. 3 describes an RNN composed of a single neuron and its *unfolded* version.

The topology of RNNs has many variations. Two common types of RNN are the fully connected RNN, in which each node receives inputs from all other nodes, and the partially connected RNN, in which the recurrence is limited to the hidden layer.

3.4. Long-short term memory

LSTM ANNs are a variant of standard RNNs, where each processing unit in the hidden layer is replaced by a *memory cell*, as shown in Fig. 4.

Despite the implementation details, it is important to note that LSTMs were introduced to solve the *vanishing problem* of RNNs, due to the fact that when training RNNs using back-propagation, the gradients which are back-propagated can *vanish* (tend to zero) or *explode*

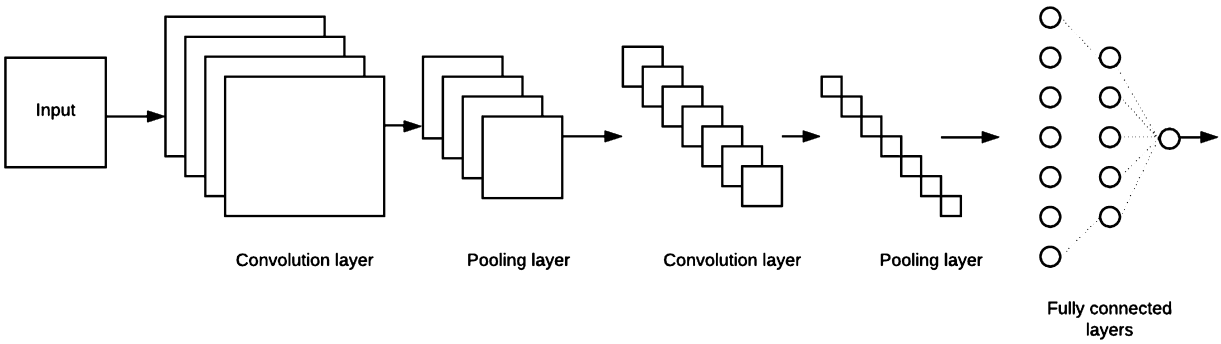

Fig. 2. Convolutional neural network.

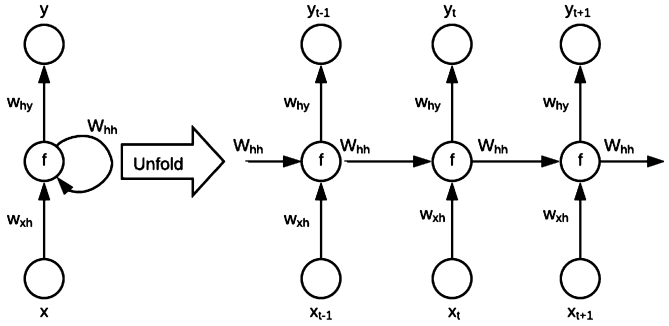

Fig. 3. Unfolded recurrent neural network.

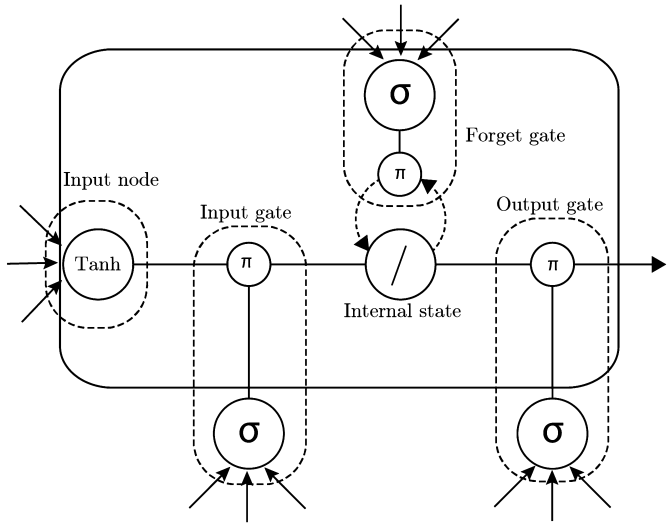

Fig. 4. LSTM memory cell.

(tend to infinity). LSTMs partially solve this issue because LSTM units allow gradients to also flow unchanged.

3.5. Gated recurrent unit

GRU ANNs are very similar to LSTMs, but have fewer parameters, as they lack an output gate in the memory cell. GRU's performance on certain tasks of polyphonic music modeling and speech signal modeling was found to be similar to that of LSTM. GRUs have been shown to exhibit even better performance on certain smaller data-sets.

4. System architecture

The proposed IDS consists of two sequential stages, Feature extraction and Attack detection, which will be detailed in the following subsections.

4.1. Feature extraction

The first processing phase, which represents the most novel contribution of our proposal, is focused on the real time extraction of the traffic features to be fed in input to the attack detection system. In a nutshell, the idea is to use a combination of hash tables, so as to be able to process a significant number of flows in real time. It is important to highlight that, as already stated in the introductory section, such a step significantly differentiates our work from the previous ones. Indeed, the latter work on already processed traffic features, which often require the knowledge of the entire traffic data-set (making the application of such methods in a real-world environment impossible).

The choice of the features (see [33] for a deeper discussion on traffic features selection, real-time extraction and IDS performance) has been done starting from the 41 KDD features (typically used in deep learning based IDSs and considered to be all of the features that can be extracted from raw traffic traces) and selecting those that can be extracted while processing traffic in (almost) real time (we will see that the traffic will be processed at the end of each time-bin). Hence, from the 41 KDD features we have selected 17 features per flow, the only ones that satisfy our constraint: the first 13 features are related to the content of packets in a flow, while the last four features are aggregated values, computed over a time-bin (note that in our experiments we have selected time-bins of two seconds, but they can be easily taken longer – for more accurate detection performance – or shorter – for being closer to real time processing). The 17 features are described in Table 1.

Let us analyze in more detail how the different features are computed. First of all, each traffic flow is inserted in a hash table, by using a hash function from a family of 4-independent hashing function [34]:

$$h(x) = \left(\left[\sum_{i=0}^{3} a_i x^i \right] mod\ p \right) mod\ m\,, \tag{1}$$

where p is a Mersenne prime number ($2^{521} - 1$ in our case), a_i are four random numbers between 0 and p, and m is equal to the dimension of the hash table.

In general, every element to be inserted in the hash table S is a couple (k, v), where the key k is a unique identifier of the flow and the value v is the partial value of a feature. In order to compute the final value of a feature, the value in the position indicated by the hash function of the key k is increased by v:

$$S[h(k)] = S[h(k)] + v\,. \tag{2}$$

Table 1
Traffic Features.

NAME	Description
SPORT	Source port of the connection
DPORT	Destination port of the connection
PROT	Protocol
NPKTS	Number of packets
NBYTES	Number of bytes
NSYN	Number of packets with the flag SYN set
NACK	Number of packets with the flag ACK set
NRST	Number of packets with the flag RST set
NFIN	Number of packets with the flag FIN set
NPUSH	Number of packets with the flag PUSH set
NURG	Number of packets with the flag URG set
NECN	Number of packets with the flag ECN set
DUR	Duration of the connection
SAMEAP1	Number of connection to same SOURCEADDR and SPORT of the current flow
SAMEAP2	number of connection to same DESTADDR and DPORT of the current flow
SAMEA1	number of connection to the same SOURCEADDR of the current flow
SAMEA2	number of connection to the same DESTADDR of the current flow

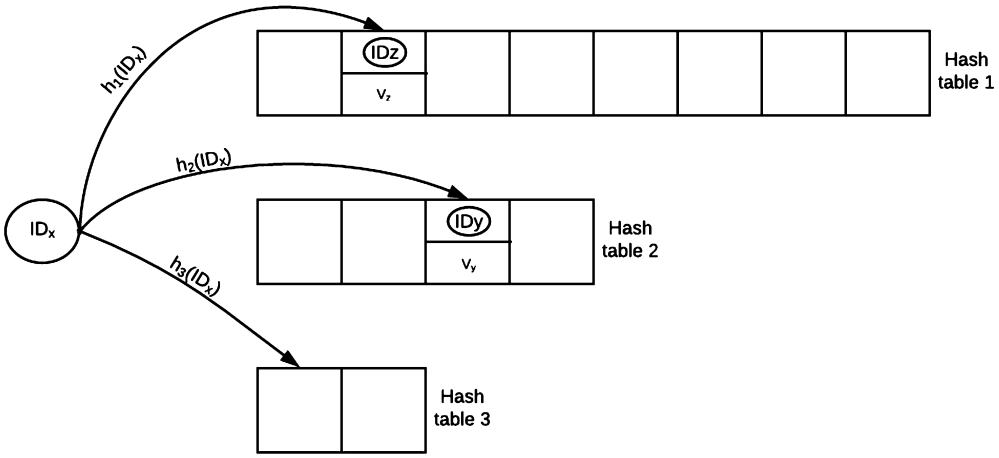

Fig. 5. Hash tables.

Note that such procedure is valid for all the features, apart from DUR (the only non-additive feature), for which we store the time-stamps of the first and last observed packets of the flow.

To solve (i.e., making it negligible) the problem of collisions (due to the use of hash functions), instead of a single hash, in our implementation we have used three distinct levels (each with an independent hash function): when the first hash is applied to a flow, we check if the corresponding entry is empty or not. If it is empty we add the value to the entry together with the flow identifier; otherwise we check the flow identifier already stored in the entry: if it corresponds to that of the current flow we increment the entry by adding v, otherwise we move to the second hash table (by applying the second hash to the flow identifier) and we iterate the procedure. The procedure is roughly sketched in Fig. 5.

Experimental results have shown that for a proper choice of the dimensions of the three hash tables (as discussed in the experimental section), the collisions become negligible. Nonetheless, in more complex cases, where this solution is not sufficient to efficiently solve the collisions problem, our system can be easily integrated with the LogLog Counting Reversible Sketch framework (presented in our previous work [35]).

In fact, the feature extraction procedure is slightly different for the first 13 features and for the aggregated ones. Regarding the first ones, the flow identifier is given by the 5-tuple (source IP address, source port, destination IP address, destination port, protocol) and the same hash table is used for all of the features. This implies that each entry of the hash table is made of an array of 14 elements: 13 for the features and one for the flow identifier. Note that these features are related to bidirectional flows, hence the flow key is in fact a concatenation of the 5-tuple, sorted according to the lower IP address.

Instead, for extracting the remaining four aggregated features we must use two additional data structures, equal in the structure to the one used for the previous features. But, in this case, the key is no longer the flow identifier: a first data structure (of three hash tables) is indexed by a concatenation of IP address and port number and is used for computing the SAMEAP1 and SAMEAP2 features, while the second one is indexed by the IP address and is used for computing the SAMEA1 and SAMEA2 features. This implies that each traffic packet is inserted into three distinct data structures.

As already stated, traffic is split into time-bins. At the end of each time-bin the extracted features are fed in input to the attack detection block. It is important to highlight that, splitting the traffic in such a way can lead to some issues with traffic flows that span over multiple time-bins. In our work, we have decided to consider those flows as a different flow in each time bin.

4.2. Attack detection

At this point, for each flow the previous block has computed 17 features; this means that each traffic flow is represented by a vector of 17 elements.

Such an element is fed in input to the actual attack detection block, where a *trained* DNN classifies the flow as either normal or attack.

In order to compare the performance of different DNNs, the modular structure of the proposed IDS permits to choose among the following architectures:

- Multi–Layer Perceptron
- Convolutional Neural Network
- Recurrent Neural Network
- Long-Short Term Memory Neural Network
- Gated Recurrent Unit Neural Network

Moreover, each of the five kinds of DNNs can be highly customized, e.g., varying the number of layers/nodes, choosing a proper activation function.

5. Performance evaluation

In this section we present the experimental tests run for evaluating our proposal and the achieved results. It is worth specifying that the proposed system has been implemented in Python3, using Keras [36], a high-level neural networks API running on top of TensorFlow [37] (CPU version, hence not optimized for GPU), and that the experiments have been run on a general purpose PC equipped with an AMD Ryzen 9 3900x at 4.2Ghz and 32 GB of RAM.

Before discussing the system performance in Sections 5.3 (Binary Classification) and 5.4 (Multi-Class Classification), in the next two subsections we describe the used data-set and the experimental methodology, respectively. Finally, a general overview of the experimental findings is provided in Section 5.5.

5.1. Data-set: MAWI traffic traces

The data-set used to evaluate our attack detection methods consists of packet traces from the MAWI (Measurement and Analysis on the WIDE Internet) archive (sample-points F), publicly available at [38]. Each trace in this database collects the traffic captured for 15 minutes in a specific day, since 2001 until nowadays, on a trans-Pacific link between Japan and the USA.

As in almost all existing databases, the key problem in testing the IDS performance is represented by a precise knowledge of the anomalies existing in the captured traffic. Such information is essential for evaluating new approaches. Although also for the MAWI archive an exact description of the attacks is not available, the data-set presents some important characteristics that make it suitable for realistic performance evaluation:

- Unlike the widely-used DARPA data-set, the network is not emulated and the traffic mixture is representative of the current mixtures of network services and applications. Moreover, the data-set is not as old as the DARPA ones, which date back to 1998 and 1999
- Unlike KDD (derived from DARPA 1998), which is the most commonly used data-set for evaluating neural network based IDS, the data-set does not contain already processed features, but raw traffic traces, making it possible to also evaluate the feature extraction capabilities of a system
- The data-set consists of real traffic captured over a real backbone network and offers a good variety of both normal and attack flows. On the contrary, in other well known data-sets, the traffic is simulated and/or only representative of a small number of applications, for example:
  - ICSX data-sets [39] are simulated and only contain a few network protocols (e.g., the most recent one, namely CSE-CIC-IDS2018, only contains https, http, smtp, pop3, imap, ssh, and ftp)
  - CTU-13 and the derived data-sets (e.g., CTU-UNB, Contagio-CTU-UNB) [40] are only significant for botnet traffic
  - RGCE (Realistic Global Cyber Environment) [41] is a cyber-range, where traffic is simulated
  - CIDDS-001 and CIDDS-002 [42] contain traffic collected in a small emulated network, where only three kinds of attacks are simulated
  - UNSW-NB15 [17] contains emulated traffic, created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS)
  - Kyoto honeypot data-set [28], begin collected in a honeypot, does not contain instances of normal traffic
- In the framework of the successive project MAWILab [43], every traffic flow is classified by means of labels, which indicate the probability (according to well-known anomaly detection algorithms) that an anomaly is present. Since these labels are available together with the traces, they can be used as a common reference for testing a new IDS

In more detail, the traces classification has been obtained combining the output of four anomaly detectors, based respectively on the Hough transform, the Gamma distribution, the Kullback-Leibler divergence and the Principal Component Analysis [44]. The anomalies are listed in an csv file for each trace, identifying them by means of traffic features as source and destination IP addresses, source port, destination port and transport protocol. Furthermore, some information about the kind of anomaly is also given, such as:

- *taxonomy*: category assigned to the anomaly using the taxonomy for backbone traffic anomalies [45]
- *heuristic*: code assigned to the anomaly based on port number, TCP flags, and ICMP code, allowing to distinguish between well-known attacks and anomalies
- *nbDetectors*: number of configurations (detector and parameter tuning) that reported the anomaly

For the experimental tests described in the following subsection, we have considered five consecutive days of traffic, collected in July 2018. Each day is roughly composed of 30 millions traffic flows and is split into two seconds time-bins (with roughly 65000 flows each).

5.2. Experimental results

First of all, the data-set has been split into two disjoint data-sets, so that training and testing are carried over different samples. The training data-set consists of the first 400 seconds of traffic collected in the first two days. Such traffic has been pre-filtered so as to have an almost equal number of normal and attack flows. All the remaining traffic, without any additional processing, is included in the Testing data-set, on which the performance of the *trained* DNNs are estimated.

As already specified, both the data-sets have been split into time-bins of two seconds (note that for sake of simplicity a flow present in two consecutive time-bins is considered as two distinct flows). Based on the average number of flows contained in each training time-bin (roughly 65000 that is rounded to 2^{16}), the number of entries of the hash tables used for the feature extraction phase have been chosen equal to $m_1 = 2^{17}$ (first level hash table), $m_2 = 2^{16}$ (second level hash table) and $m_3 = 2^{15}$ (third level hash table). Experimentally, we have verified that the number of collisions in the third level (flows that are not processed, because they cannot be stored in memory) is negligible. In any case, such flows have been considered as misclassified, hence the performance presented in the following are *worst-case* performance.

It is important to highlight that such settings, and our non-optimized Python implementation, take to an overall memory occupancy of about 600 MB, hence not representing a problem for modern devices.

Regarding the system performance, we have evaluated the results in terms of True Positives (TP), attacks correctly detected, True Negatives (TN), normal traffic correctly classified, False Positives (FP), normal traffic classified as an attack, and False Negatives (FN), undetected attacks. More specifically, we have computed three performance indexes:

- Detection Rate (DR)

$$DR = \frac{TP}{TP + FN} \tag{3}$$

- False Alarm Rate (FAR)

$$FAR = \frac{FP}{FP + TN} \tag{4}$$

- Accuracy (ACC)

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

It is worth noting that, when evaluating the system performance, these three performance indexes have been considered together with the execution time (in the following tables we will report the inference time for a two seconds time-bin).

Moreover, to comprehensively evaluate the system performance we considered two distinct scenarios: binary classification, in which the system is requested to classify a traffic flow as either normal or anomalous, and multi-class classification, where the flow can be classified as normal or as a specific attack class. It is clear that the number of input and output nodes is fixed and given by the number of features (17 in our case) and the number of classes (two and twelve for binary and multi-class classification, respectively). Finally, not that in all our experiments we used "Adam" as Optimization algorithm with a learning rate equal to 0.001 and a batch size of 32. The setting of all the other parameters will be discussed in the following two subsections.

5.3. Binary classification

A first set of tests has been run, for each considered DNN, to correctly set its parameters. For sake of simplicity, these first tests have been run on a reduced data-set, only composed of a single day of traffic, with the first 400 seconds used for training and the remaining traffic for testing. It is worth noting that, given the complexity of the parameters selection problem which cannot be exactly solved, the identified parameters values are not the optimal ones, but those that offer the *best* trade-off among DR, ACC, and FAR for the considered data-set and specific sequence of optimization steps (even if in the following we will refer to them as to the optimal parameters).

For the MLPs all the intermediate optimization steps are reported in Table 2, highlighting in bold the chosen value of each parameter. Note that in almost all the cases it is not easy to select the *best* configuration, since we must find the best trade-off between the three performance indexes (and the execution time). In more detail, we have arbitrarily chosen as starting point a network with one hidden layer, 17 nodes per layer, and one epoch for the training phase. At first we have determined the optimal number of hidden layers, running tests with a number of layers ranging from one to five.

Once fixed the number of layers to five, we have studied the system performance when varying the number of neurons selecting 17 as optimal value. Finally, the number of epochs has been considered. For this final optimization step we do not show the execution time, since the number of epochs only impacts the training time and not the inference time. From these results we have concluded that one epoch allows to achieve the best performance (indicating that more epochs probably lead to an over-fitting problem). The resulting configuration parameters for MLP are summarized in Table 3.

In an analogous way, a second set of tests has been carried out to find the optimal setting for the CNN, starting from a network with one convolutional layer, 64 filters of dimension three, and one FCL with 32 neurons.

Table 4 reports all the optimization steps, including (in order of execution): number of convolutional layers, number of filters in the two layers (as determined in the previous step), filters dimension, architecture of the fully connected network that follows the convolutional layers (namely, number of FCLs and nodes in each layer) and number of epochs.

Hence, the complete configuration of the chosen CNN is shown in Table 5.

Finally, for the three kinds of recurrent ANNs we have performed three sets of tests, similar to the MLP case (although with different optimal values), focusing on the number of hidden layers, the number of neurons in each layer and then the number of epochs. The optimization steps are reported in Tables CNN 6, 7 and 8 for RNN, LSTM and GRU,

Table 2
MLP tuning.

Parameter	Value	Time (s)	DR	ACC	FAR
Layers	1	0.31	0.571	0.908	0.064
	2	0.32	0.634	0.912	0.066
	3	0.31	0.673	0.860	0.125
	4	0.32	0.739	0.875	0.114
	5	**0.35**	**0.642**	**0.900**	**0.079**
Nodes	5	0.37	0.716	0.906	0.079
	10	0.39	0.742	0.805	0.190
	17	**0.38**	**0.742**	**0.909**	**0.077**
	20	0.41	0.624	0.915	0.061
	30	0.40	0.653	0.602	0.402
	40	0.42	0.876	0.845	0.158
Epochs	**1**		**0.709**	**0.905**	**0.079**
	2		0.717	0.891	0.094
	5		0.795	0.874	0.119
	7		0.615	0.906	0.070

Table 3
MLP parameters.

Number of hidden layers	5
Number of neurons	17
Activation function	ReLU
Loss	binary_crossentropy
Epochs	1

Table 4
CNN tuning.

Parameter	Value	Time (s)	DR	ACC	FAR
Layers	1	0.90	0.537	0.311	0.707
	2	**1.20**	**0.906**	**0.888**	**0.113**
	3	1.39	0.854	0.809	0.194
	4	1.52	0.964	0.786	0.228
	5	1.57	0.980	0.126	0.943
Filters	16	0.65	0.639	0.910	0.068
	32	0.78	0.923	0.825	0.183
	64	1.03	0.879	0.422	0.615
	128	1.25	0.956	0.868	0.139
	16,32	0.65	0.788	0.883	0.109
	32,64	0.90	0.555	0.242	0.783
	64,128	**1.26**	**0.964**	**0.852**	**0.157**
Filter size	2	1.16	0.846	0.885	0.112
	3	**1.19**	**0.923**	**0.869**	**0.135**
	4	1.23	0.965	0.778	0.237
	5	1.28	0.889	0.833	0.172
	6	1.23	0.982	0.169	0.897
FLC (Nodes)	1 (16)	1.08	0.775	0.876	0.116
	1 (32)	**1,10**	**0.937**	**0.866**	**0.140**
	1 (64)	1,13	0.941	0.845	0.162
	2 (32,16)	1,98	0.926	0.889	0.114
	2 (64,32)	2,47	0.824	0.921	0.072
Epochs	1		0.834	0.889	0.107
	2		**0.828**	**0.896**	**0.099**
	5		0.697	0.892	0.092
	7		0.940	0.895	0.109

Table 5
CNN parameters.

Number of CNN layers	2
Number of filters	64,128
Size of filters	3
Strides	1
Padding	same
Number of FCL layers	1
Number of neuron FCL	32
Activation function	ReLU
Loss	binary_crossentropy
Epochs	2

Table 6
RNN tuning.

Parameter	Value	Time (s)	DR	ACC	FAR
Layers	1	0.57	0.967	0.791	0.224
	2	0.76	0.942	0.800	0.211
	3	**0.93**	**0.933**	**0.806**	**0.204**
	4	1.11	0.964	0.400	0.645
	5	1.31	0.933	0.806	0.204
Neurons	1	0.93	0.933	0.806	0.205
	5	0.96	0.682	0.911	0.070
	10	**0.99**	**0.692**	**0.906**	**0.076**
	20	1.00	0.729	0.537	0.478
	30	0.98	0.734	0.848	0.142
	40	0.96	0.796	0.757	0.246
	50	0.96	0.709	0.864	0.123
	60	0.96	0.824	0.860	0.137
	70	0.97	0.860	0.824	0.179
	80	0.97	0.903	0.830	0.176
	90	0.98	0.725	0.464	0.558
	100	0.99	0.798	0.809	0.190
Epochs	1		0.698	0.904	0.079
	2		0.701	0.891	0.093
	5		**0.725**	**0.919**	**0.065**
	7		0.905	0.666	0.353

Table 7
LSTM tuning.

Parameter	Value	Time (s)	DR	ACC	FAR
Layers	1	0.71	0.724	0.888	0.099
	2	**1.11**	**0.613**	**0.912**	**0.064**
	3	1.47	0.561	0.917	0.054
	4	1.80	0.489	0.917	0.048
	5	2.09	0.504	0.900	0.068
Neurons	1	1.07	0.623	0.909	0.068
	5	1.21	0.724	0.262	0.775
	10	1.26	0.703	0.895	0.089
	20	1.31	0.696	0.912	0.070
	30	**1.26**	**0.715**	**0.919**	**0.065**
	40	1.25	0.701	0.898	0.086
	50	1.25	0.705	0.889	0.096
	60	1.25	0.716	0.910	0.074
	70	1.28	0.718	0.907	0.078
	80	1.27	0.716	0.898	0.088
	90	1.30	0.695	0.898	0.086
	100	1.27	0.703	0.915	0.067
Epochs	1		0.678	0.904	0.077
	2		**0.699**	**0.900**	**0.084**
	5		0.680	0.589	0.418
	7		0.684	0.823	0.166

respectively; moreover, Table 9 summarizes the corresponding optimal settings.

As a final result, in Table 10 we show the performance achieved by the system when using the five selected configurations for the MPL, RNN, CNN, LSTM, and GRU. These results have been obtained on the full data-set (five days of traffic, excluding the first 400 seconds of the first two days used for training, as already mentioned) and show that CNN are able to outperform the other DNNs, with a significantly higher DR. This comes at the cost of a relatively higher execution time (higher than MLP, RNN, and GRU, but still suitable for real time operation, considering the time-bin duration). It is worth noticing that such computation time only takes into account the inference phase of the DNN, hence the time spent in the feature extraction phase (a few seconds in our implementation for the entire data-set) is not considered. Nonetheless, considering that properly implemented hash tables are commonly used for real time traffic processing and that the inference time of DNNs can be highly reduced by using GPUs, we can conclude that the system can be deployed in a real backbone network, offering real time attack detection.

As expected, the values in Table 10 do not coincide with the corresponding ones in the previous Tables, associated with the optimal parameters for each type of DNNs calculated over the reduced data-set. It is worth noticing that in some cases (e.g., for GRU) the performance degradation is quite significant, while the differences are very small in the CNN case, highlighting the *stability* of CNNs also for the analysis of new data, a very relevant feature for the applicability of the proposed IDS to real traffic, where training can be performed only periodically.

Table 8
GRU tuning.

Parameter	Value	Time (s)	DR	ACC	FAR
Layers	1	0.64	0.728	0.901	0.085
	2	**0.94**	**0.951**	**0.798**	**0.214**
	3	1.26	0.929	0.783	0.229
	4	1.55	0.509	0.898	0.070
	5	1.76	0.536	0.895	0.076
Neurons	1	0.95	0.964	0.791	0.223
	5	1.00	0.690	0.895	0.088
	10	1.06	0.657	0.908	0.072
	20	**1.09**	**0.727**	**0.916**	**0.069**
	30	1.07	0.720	0.890	0.096
	40	1.15	0.701	0.877	0.108
	50	1.16	0.715	0.896	0.090
	60	1.15	0.729	0.902	0.084
	70	1.19	0.696	0.894	0.090
	80	1.20	0.688	0.887	0.097
	90	1.15	0.713	0.917	0.066
	100	1.11	0.711	0.902	0.082
Epochs	1		0.703	0.889	0.096
	2		0.652	0.903	0.077
	5		**0.710**	**0.904**	**0.080**
	7		0.687	0.849	0.138

Table 9
RNN, LSTM and GRU parameters.

Parameter	RNN	LSTM	GRU
Number of recurrent layers	3	2	2
Number of neurons	10	30	20
Activation function	tanh	tanh	tanh
Loss	binary_MSE	binary_MSE	binary_MSE
Epochs	5	2	5

Table 10
Performance comparison.

DNN	Time (s)	DR	ACC	FAR
MLP	0.38	0.698	0.907	0.076
RNN	0.99	0.735	0.804	0.190
CNN	1.10	0.926	0.899	0.103
LSTM	1.26	0.718	0.892	0.093
GRU	1.09	0.746	0.656	0.352

Table 11
Multi-class MLP tuning.

Parameter	Value	Time (s)	DR	ACC	FAR
Layers	1	0.30	0.624	0.918	0.058
	2	0.33	0.828	0.833	0.167
	3	**0.34**	**0.627**	**0.908**	**0.069**
	4	0.37	0.854	0.828	0.174
	5	0.36	0.688	0.882	0.103
Neurons	5	0.34	0.690	0.899	0.084
	10	0.32	0.576	0.898	0.076
	17	0.34	0.575	0.826	0.154
	20	0.34	0.663	0.915	0.064
	30	**0.35**	**0.703**	**0.919**	**0.063**
	40	0.33	0.712	0.901	0.083
Epochs	1		0.556	0.907	0.064
	2		0.691	0.918	0.063
	5		**0.646**	**0.927**	**0.050**
	7		0.747	0.873	0.117

Table 12
MLP parameters.

Number of hidden layers	3
Number of neurons	30
Activation function	ReLU
Loss	binary_crossentropy
Epochs	5

Table 13
CNN parameters.

Number of CNN layers	2
Number of filters	64,64
Size of filters	3
Strides	1
Padding	same
Number of FCL layers	1
Number of neuron FCL	32
Activation function	ReLU
Loss	binary_crossentropy
Epochs	2

Table 14
RNN, LSTM and GRU parameters.

Parameter	RNN	LSTM	GRU
Number of recurrent layers	2	3	2
Number of neurons	30	50	40
Activation function	tanh	tanh	tanh
Loss	categorical_crossentropy	categorical_crossentropy	categorical_crossentropy
Epochs	5	5	2

5.4. Multi-class classification

Analogously to the binary case, also for the multi-class classification scenario, before actually evaluating the system performance over the whole data-set, we have carried out a first set of tests, for each considered DNN, to correctly set its parameters. These first tests have been run on the same reduced data-set (composed of a single day of traffic, with the first 400 seconds used for training and the remaining traffic for testing) and the same initial settings, as in the binary case.

For sake of brevity, we show the intermediate optimization steps only for MLP (see Table 11), and only report the final configurations for all the considered DNNs (see Tables from 12 to 14). It is important to note that the values reported in the tables for the performance indexes are the weighted average over the different classification classes.

As for the binary case, we present the performance achieved over the full data-set (five days of traffic, with the first 400 seconds of the first two days used for training). Differently from the binary case, not only are the multi-class classifiers able to distinguish between normal and anomalous traffic, but they are also able to indicate the attack class. In more detail, the classifiers can label a traffic flow as belonging to either the normal traffic class (indicated as class number 0) or one of the 11 attack classes represented in the MAWI data-set (Table 15 report the different attack types and the corresponding classification classes).

Tables from 16 to 20 represent the confusion matrices obtained when using the different DNNs. As a general observation, it is worth noting that there are some attack classes (namely, classes seven and eight) that are never outputted by the system (with a few exceptions in case of GRU) and some (i.e., classes one and two) that are rarely outputted. These results clearly indicate that the system reliability is not constant over all of the attack types. Nonetheless, after having analyzed the MAWI data-set, we can conclude that such behavior is most probably due to the fact that some attack classes (as the ones previously mentioned) are scarcely present in the collected traffic and so it is hard to train the DNNs.

Table 15
Attack classes.

Class number	Type of Attack
1	Unknown
2	Other
3	HTTP
4	Multi. Points
5	Alpha Flow
6	IPv6 Tunneling
7	Port Scan
8	Network Scan ICMP
9	Network Scan TCP
10	Network Scan UDP
11	DoS

Table 16
MLP confusion matrix.

A/P	0	1	2	3	4	5	6	7	8	9	10	11
0	98306861	0	0	3278362	2042010	659225	203218	0	0	3119270	322067	2
1	1819	0	0	217	410	33	0	0	0	126	0	0
2	3100	0	0	633	1867	1253	5	0	0	167	4	0
3	979833	0	4	1767713	600964	113696	3639	0	0	61509	43137	1
4	1212425	0	1	298083	1652573	78116	2609	0	0	400639	26077	3
5	66866	0	0	4421	15145	18378	2	0	0	10215	10338	0
6	17658	0	0	2004	1085	1112	63886	0	0	834	401	0
7	99	0	0	846	30	7	0	0	0	34	4	0
8	1785	0	0	568	1758	946	3	0	0	132	0	0
9	486181	0	0	138230	191760	12786	3649	0	0	308611	41449	0
10	89305	0	0	63169	40303	137	1	0	0	364	22611	0
11	11715	0	0	770	1306	1043	43	0	0	1882	423	0

Table 17
CNN confusion matrix.

A/P	0	1	2	3	4	5	6	7	8	9	10	11
0	98410410	3	0	6032714	3324365	1975	0	0	0	59386	102162	0
1	1137	0	0	1411	57	0	0	0	0	0	0	0
2	2920	0	0	2133	1976	0	0	0	0	0	0	0
3	542895	4	0	2576228	447983	76	0	0	0	2889	421	0
4	1288878	9	4	810112	1494278	378	0	0	0	7083	69776	8
5	45348	12	1	5510	74067	380	0	0	0	26	20	1
6	78427	0	0	2657	4987	0	0	0	0	909	0	0
7	21	0	0	876	123	0	0	0	0	0	0	0
8	2455	0	0	1653	1084	0	0	0	0	0	0	0
9	500477	0	0	509665	126294	0	0	0	0	25472	20758	0
10	42653	0	0	51759	27238	0	0	0	0	0	94240	0
11	12452	0	0	2408	2301	0	0	0	0	21	0	0

Table 18
RNN confusion matrix.

A/P	0	1	2	3	4	5	6	7	8	9	10	11
0	90163252	0	0	7019033	6801345	375126	111757	0	0	1123822	1533870	802810
1	1756	0	0	391	430	0	0	0	0	0	24	4
2	226	0	0	1654	3411	0	0	0	0	831	154	753
3	767870	0	0	2169251	317274	133965	493	0	0	155312	17387	8944
4	984102	0	1	534076	1542582	50638	2292	0	0	230916	217570	108349
5	72136	0	0	27107	8314	16753	16	0	0	69	650	320
6	17445	0	0	14078	911	1855	49046	0	0	1797	688	1160
7	182	0	0	715	66	2	0	0	0	42	12	1
8	176	0	0	1400	2458	67	0	0	0	338	93	660
9	339426	0	0	298006	119155	2717	3410	0	0	306497	89614	23841
10	119387	0	0	66694	291	0	0	0	0	248	29270	0
11	7971	0	0	3507	1990	456	0	0	0	367	789	2102

Finally, Table 21 reports the overall performance achieved by the system. It is interesting to notice that, as for the binary case, the best results are obtained by the CNNs, which are able to outperform the other DNNs, offering a higher DR with still an acceptable FAR. As for the previous case, this comes at the cost of a higher execution time. In general, we can observe that the performance is slightly worse than in the binary case, but this can be easily justified simply observing that, in multi-class classification, an attack classified as an attack, but of the wrong class, determines a degradation of the performance, while it is considered correct in the binary case.

Table 19
LSTM confusion matrix.

A/P	0	1	2	3	4	5	6	7	8	9	10	11
0	83615245	60133	4855	2644073	2709914	785994	947064	0	0	201919	171873	16789945
1	1939	0	0	208	428	2	27	0	0	1	0	0
2	2376	0	0	1034	3604	0	10	0	0	5	0	0
3	1649505	22	229	1456558	378612	53974	8051	0	0	7995	8537	7013
4	1507202	3644	372	410716	1570558	111258	9780	0	0	48631	6038	2327
5	81560	6	72	779	9125	24463	10	0	0	8716	508	126
6	13887	295	24	2808	1343	4067	61950	0	0	2518	62	26
7	753	0	0	84	181	1	1	0	0	0	0	0
8	2873	0	0	814	1500	1	1	0	0	3	0	0
9	587930	188	17	251357	260704	28306	11917	0	0	30441	11793	13
10	157151	0	0	15801	149	0	97	0	0	8	42684	0
11	12236	9	8	389	1608	2534	270	0	0	52	74	2

Table 20
GRU confusion matrix.

A/P	0	1	2	3	4	5	6	7	8	9	10	11
0	97477261	0	392	4304230	3631916	1541923	183777	47	62	616414	126517	48476
1	1794	0	0	220	539	1	1	0	0	50	0	0
2	3481	0	0	49	3064	330	0	0	0	105	0	0
3	1294736	0	4	1569148	425779	148740	52949	0	68	75536	1402	2134
4	1515935	0	10	217187	1511559	192566	7314	57	6	114654	108683	2555
5	65725	0	0	1989	1864	55308	3	0	0	280	196	0
6	36710	0	0	5161	635	1631	40849	0	0	1992	0	2
7	321	0	0	466	71	6	0	0	0	3	153	0
8	2531	0	0	108	2256	209	3	0	0	85	0	0
9	554961	0	0	122295	115880	20252	445	0	0	330419	37823	591
10	119193	0	0	34748	173	1422	0	0	0	59	60295	0
11	13459	0	0	575	1383	1531	80	0	0	103	42	9

Table 21
Performance comparison.

DNN	Time (s)	DR	ACC	FAR
MLP	0.35	0.780	0.914	0.075
RNN	0.79	0.642	0.918	0.059
CNN	2.00	0.859	0.897	0.099
LSTM	1.72	0.723	0.903	0.083
GRU	1.04	0.626	0.845	0.137

5.5. Results discussion

The system is, in general, able to attain excellent performance, with the best case achieved by the use of CNN for the binary classification, which offer a detection rate of over 92% with a false alarm rate of about 10%.

As a general consideration, it is worth highlighting that the system performance, in most cases, are not significantly impacted by small changes in the parameter choice. This seems to indicate that such methods are quite robust. Instead, even if not shown in the presented results for sake of brevity, the tests have highlighted that the performance are strongly dependent on a properly chosen training data-set, which must be significant (in dimension) and well-balanced (in terms of number of samples) among the different output classes. This consideration is even more valid for the multi-class case, in which (as demonstrated by the confusion matrices) classes that are not frequent in the training data-set are often misclassified by the system.

From the computational point of view, all the considered DNN configurations have an inference time that is suitable for real-time operation, already in our implementation. As specified above, the indicated times do not take into account the time spent in the feature extraction phase (a few seconds in our implementation for the entire testing data-set), but, considering that properly implemented hash tables are commonly used for real time traffic processing and that the inference time of DNNs can be highly reduced by using GPUs, we can conclude that the system can be deployed in a real backbone network, offering real time attack detection.

As a final remark, it is important to highlight that on one hand these results are far from being as good as those presented in the literature (when testing the system over KDD), but on the other hand the performance are aligned (and even better) with those presented by [29] (the only work in the literature that tests the system over the MAWILab data-set), where accuracy is always lower than 0.85. This confirms the importance of choosing a realistic data-set to evaluate a novel IDS, as the use of old and synthetic data-sets (e.g., KDD) can lead to highly too optimistic results.

6. Conclusions

In this paper we proposed a deep learning based approach for network attack detection. Differently from most of the works in the literature, our system is able to directly work on raw traffic traces in real time (even in the worst case the classification results are obtained before new traffic data are available as input), first performing a feature extraction phase and then executing a DNN-based classification phase. Moreover, we have performed a comparison among the most widely used DNNs (namely MLP, RNN, CNN, LSTM and GRU).

The implemented system has been tested over traffic traces collected in the MAWILab archive. Our system has shown its effectiveness in detecting network attacks, while maintaining an *acceptable* false alarm rate, achieving, through a proper choice of the system parameters, 92.6% of detection rate, with an accuracy of 0.899.

Finally, the proposed approach has been extended to perform multi-class classification, providing in this way some information about the kind of anomaly. As before, the best results are provided by the CNNs with a detection rate of around 86% and the same accuracy of the binary case.

CRediT authorship contribution statement

Christian Callegari: Conceptualization, Data curation, Methodology, Project administration, Supervision, Writing – original draft. **Stefano Giordano:** Conceptualization, Methodology, Project administra-

tion, Resources. **Michele Pagano:** Conceptualization, Methodology, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data are publicly available on the site indicated in the paper

Acknowledgement

The authors would like to thank Elena Bucchianieri and Francesco Castiglione for their activities in support of the presented work. This work was partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the FoReLab project (Departments of Excellence) and by the University of Pisa in the framework of the PRA_2022_64 “hOlistic Sustainable Management of distributed softWARE systems (OSMWARE)” project.

References

[1] A. Little, X. Mountrouidou, D. Moseley, Spectral clustering technique for classifying network attacks, in: 2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2016, pp. 406–411.
[2] E. Nikolova, V. Jecheva, Applications of clustering methods to anomaly-based intrusion detection systems, in: 2015 8th International Conference on Database Theory and Application (DTA), 2015, pp. 37–41.
[3] V.V. Kumari, P.R.K. Varma, A semi-supervised intrusion detection system using active learning svm and fuzzy c-means clustering, in: 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2017, pp. 481–485.
[4] V. Justin, N. Marathe, N. Dongre, Hybrid ids using svm classifier for detecting dos attack in manet application, in: 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2017, pp. 775–778.
[5] G. Yan, An improved ids model based on pca and fcm algorithms, in: 2016 International Conference on Smart City and Systems Engineering (ICSCSE), 2016, pp. 330–333.
[6] C. Callegari, L. Donatini, S. Giordano, M. Pagano, Improving stability of pca-based network anomaly detection by means of kernel-pca, International Journal of Computational Science and Engineering 16 (1) (2018) 9–16, https://doi.org/10.1504/IJCSE.2018.089573.
[7] M. Aljnidi, M.S. Desouki, Big data analysis and distributed deep learning for next-generation intrusion detection system optimization, Journal of Big Data 6 (88) (2019) 1–18, https://doi.org/10.1186/s40537-019-0248-6.
[8] B. Riyaz, S. Ganapathy, A deep learning approach for effective intrusion detection in wireless networks using cnn, Soft Computing 24 (2020) 17265–17278, https://doi.org/10.1007/s00500-020-05017-0.
[9] KDD Cup 1999 Data, http://kdd.ics.uci.edu/databases/kddcup99.
[10] C. Callegari, E. Bucchianeri, S. Giordano, M. Pagano, Real time attack detection with deep learning, in: 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), 2019, pp. 1–5.
[11] J. Cannady, Artificial neural networks for misuse detection, in: National Information Systems Security Conference, Vol. 26, Baltimore, 1998.
[12] M. Moradi, M. Zulkernine, A neural network based system for intrusion detection and classification of attacks, in: IEEE International Conference on Advances in Intelligent Systems, 2004.
[13] R. Vinayakumar, K. Soman, P. Poornachandran, Applying deep learning approaches for network traffic prediction, in: Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on, IEEE, 2017, pp. 2353–2358.
[14] Z. Li, Z. Qin, K. Huang, X. Yang, S. Ye, Intrusion detection using convolutional neural networks for representation learning, in: ICONIP, 2017.
[15] K. Wu, Z. Chen, W. Li, A novel intrusion detection model for a massive network using convolutional neural networks, IEEE Access 6 (2018) 50850–50859.
[16] S. Potluri, S. Ahmed, C. Diedrich, Convolutional neural networks for multi-class intrusion detection system, in: A. Groza, R. Prasath (Eds.), Mining Intelligence and Knowledge Exploration, Springer International Publishing, Cham, 2018, pp. 225–238.
[17] N. Moustafa, J. Slay, Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set), in: 2015 Military Communications and Information Systems Conference (MilCIS), 2015, pp. 1–6.
[18] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del-Rincón, D. Siracusa Lucid, A practical, lightweight deep learning solution for ddos attack detection, IEEE Transactions on Network and Service Management 17 (2) (2020) 876–889.
[19] C. Yin, Y. Zhu, J. long Fei, X. He, A deep learning approach for intrusion detection using recurrent neural networks, IEEE Access 5 (2017) 21954–21961.
[20] T.A. Tang, L. Mhamdi, D. McLernon, S.A.R. Zaidi, M. Ghogho, Deep recurrent neural network for intrusion detection in sdn-based networks, in: 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), 2018, pp. 202–206.
[21] S.K. Dey, M.M. Rahman, Flow based anomaly detection in software defined networking: a deep learning approach with feature selection method, in: 2018 4th International Conference on Electrical Engineering and Information Communication Technology (iCEEiCT), 2018, pp. 630–635.
[22] G. Qin, Y. Chen, Y. Lin, Anomaly detection using lstm in ip networks, in: 2018 Sixth International Conference on Advanced Cloud and Big Data (CBD), 2018, pp. 334–337.
[23] S.A. Althubiti, E.M. Jones, K. Roy, Lstm for anomaly-based network intrusion detection, in: 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), 2018, pp. 1–3.
[24] M. Ring, S. Wunderlich, D. Gruedl, D. Landes, A. Hotho, Flow-based benchmark data sets for intrusion detection, in: Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS), ACPI, 2017.
[25] Z. Li, A.L.G. Rios, G. Xu, L. Trajković, Machine learning techniques for classifying network anomalies and intrusions, in: 2019 IEEE International Symposium on Circuits and Systems (ISCAS), 2019, pp. 1–5.
[26] S. Naseer, Y. Saleem, S. Khalid, M.K. Bashir, J. Han, M.M. Iqbal, K. Han, Enhanced network anomaly detection based on deep neural networks, IEEE Access 6 (2018) 48231–48246, https://doi.org/10.1109/ACCESS.2018.2863036.
[27] R.K. Malaiya, D. Kwon, J. Kim, S.C. Suh, H. Kim, I. Kim, An empirical evaluation of deep learning for network anomaly detection, in: 2018 International Conference on Computing, Networking and Communications (ICNC), 2018, pp. 893–898.
[28] Traffic Data from Kyoto University’s Honeypots, http://www.takakura.com/Kyoto_data/.
[29] D. Kwon, K. Natarajan, S.C. Suh, H. Kim, J. Kim, An empirical study on network anomaly detection using convolutional neural networks, in: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2018, pp. 1595–1598.
[30] S. Choudhary, N. Kesswani, Analysis of kdd-cup’99, nsl-kdd and unsw-nb15 datasets using deep learning in iot, in: International Conference on Computational Intelligence and Data Science, Procedia Computer Science 167 (2020) 1561–1573, https://doi.org/10.1016/j.procs.2020.03.367, https://www.sciencedirect.com/science/article/pii/S1877050920308334.
[31] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
[32] V. Sze, Y.-H. Chen, T.-J. Yang, J. Emer, Efficient processing of deep neural networks: a tutorial and survey, arXiv:1703.09039, Comment: Based on tutorial on DNN Hardware at eyeriss.mit.edu/tutorial.html http://arxiv.org/abs/1703.09039, 2017.
[33] C. Callegari, S. Giordano, M. Pagano, On the proper choice of datasets and traffic features for real-time anomaly detection, Journal of Physics. Conference Series 2091 (1) (2021) 012001, https://doi.org/10.1088/1742-6596/2091/1/012001, https://dx.doi.org/10.1088/1742-6596/2091/1/012001.
[34] J.L. Carter, M.N. Wegman, Universal classes of hash functions (extended abstract), in: Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, STOC ’77, ACM, New York, NY, USA, 1977, pp. 106–112, http://doi.acm.org/10.1145/800105.803400.
[35] C. Callegari, A. Di Pietro, S. Giordano, T. Pepe, G. Procissi, The loglog counting reversible sketch: a distributed architecture for detecting anomalies in backbone networks, in: Communications (ICC), 2012 IEEE International Conference on, 2012, pp. 1287–1291.
[36] Keras: the Python deep learning library, https://keras.io/.
[37] TensorFlow Home Page, https://www.tensorflow.org/.
[38] MAWI Working Group Traffic Archive, http://mawi.wide.ad.jp/mawi/.
[39] CSE-CIC-IDS2018 on AWS, https://www.unb.ca/cic/datasets/ids-2018.html.
[40] The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic, https://www.stratosphereips.org/datasets-ctu13.
[41] Realistic Global Cyber Environment (RGCE), https://jyvsectec.fi/cyber-range/overview/.
[42] Coburg Intrusion Detection Data Sets, https://www.hs-coburg.de/forschung-kooperation/forschungsprojekte-oeffentlich/informationstechnologie/cidds-coburg-intrusion-detection-data-sets.html.
[43] MAWILab, http://www.fukuda-lab.org/mawilab/.
[44] R. Fontugne, P. Borgnat, P. Abry, K. Fukuda, MAWILab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking, in: ACM CoNEXT, 2010.
[45] MAWI Attacks Taxonomy, http://www.fukuda-lab.org/mawilab/documentation.html#taxonomy.