Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 9054336, 14 pages
https://doi.org/10.1155/2021/9054336

WILEY | Hindawi

Research Article

Effective Anomaly Detection Using Deep Learning in IoT Systems

Lerina Aversano[1], **Mario Luca Bernardi**[1], **Marta Cimitile**[2], **Riccardo Pecori**[1] and **Luca Veltri**[3]

[1]*University of Sannio, Benevento, BN, Italy*
[2]*Unitelma Sapienza University, Rome, RM, Italy*
[3]*University of Parma, Parma, PR, Italy*

Correspondence should be addressed to Marta Cimitile; marta.cimitile@unitelmasapienza.it

Received 5 August 2021; Accepted 21 September 2021; Published 23 October 2021

Academic Editor: Zhihan Lv

Anomaly detection in network traffic is a hot and ongoing research theme especially when concerning IoT devices, which are quickly spreading throughout various situations of people's life and, at the same time, prone to be attacked through different weak points. In this paper, we tackle the emerging anomaly detection problem in IoT, by integrating five different datasets of abnormal IoT traffic and evaluating them with a deep learning approach capable of identifying both normal and malicious IoT traffic as well as different types of anomalies. The large integrated dataset is aimed at providing a realistic and still missing benchmark for IoT normal and abnormal traffic, with data coming from different IoT scenarios. Moreover, the deep learning approach has been enriched through a proper hyperparameter optimization phase, a feature reduction phase by using an autoencoder neural network, and a study of the robustness of the best considered deep neural networks in situations affected by Gaussian noise over some of the considered features. The obtained results demonstrate the effectiveness of the created IoT dataset for anomaly detection using deep learning techniques, also in a noisy scenario.

1. Introduction

The pervasive spreading of the IoT paradigm in many aspects of our lives is becoming more and more an emerging reality [1]; however, its huge and widespread development implies also critical security issues [2–4], given that this particular Internet traffic is much more variegated and pervasive and comes from many sources such as industrial machines during their maintenance, driverless cars for their safe driving and positioning on the road, health sensors measuring important vital signs of the body of people, and smart home devices that try to automate daily housework. Indeed, the large ongoing usage of IoT devices can foster novel and emerging malicious manipulations and can have deep implications on the security and the robustness of the whole Internet. For example, the Mirai malware [5] launched a severe distributed Denial of Service (DoS) attack by gaining control over several zombified IoT bots [6] and revealed the utmost need for secure authentication mechanisms [7] and of apt traffic classification and identification techniques [8]. As a consequence, many emerging IoT applications require more and more security and protection mechanisms, which often entail accurate classification of network traffic for the early detection of anomalies and attacks as well as the enforcement of suitable and viable countermeasures. Hence, the timely detection of IoT-specific anomalous traffic is an ongoing and emerging hot topic, but the current techniques published in the relevant literature so far, including those employing artificial neural networks, have the following shortcomings [9]: (i) they do not present accurate preprocessing and optimization phases, (ii) they are grounded only on local and ad hoc traffic datasets coming from one single network scenario, (iii) they often do not rely on IoT traffic, and (iv) they seldom tackle

the data dimensionality reduction problem rigorously. Basing on the above considerations, we optimize a deep learning approach to perform anomaly detection and attack classification of IoT traffic over an integrated dataset. This study significantly extends the research proposed in [8] with the following novel contributions:

(i) The integration of further IoT traffic instances to obtain a larger and more multifaceted integrated dataset with even more proper IoT attack types from different real IoT scenarios;

(ii) The analysis and optimization of different deep neural networks able to obtain very high classification accuracy, both in the binary and the multiclassification context, over the IoT dataset we built;

(iii) The identification of the minimum set of features allowing the optimized deep neural networks to achieve the best results. For data dimensionality reduction, the autoencoder proposed in [10] is used and compared with an alternative approach;

(iv) The verification of the robustness of the optimized deep neural networks in a scenario that inherently adds Gaussian noise to an increasing percentage of features.

The remainder of the article is structured as follows: in Section 2, the background about Internet traffic classification is summarized. In Section 3, some recently published articles regarding deep learning techniques for IoT anomaly detection are reviewed. Section 4 presents the used deep learning model we employed. Section 5 describes how the integrated dataset was built as well as the experimental settings. Section 6 shows the obtained results comparing the performance of the optimized deep neural networks in both a normal and a noisy scenario, as well as the outcomes of the performed feature reduction. Finally, Section 7 concludes the paper.

2. Background

In the context of anomaly detection, network traffic is usually seen as a sum of bidirectional flows. Each flow is formed of an ordered sequence of packets, exchanged between two endpoints, and it is normally identified by the source and destination IP addresses, the protocol number, and possible upper-layer identifiers. In case of a flow between two transport layer endpoints (e.g., TCP or UDP entities), it is uniquely identified by the following: *source IP address*, *destination IP address*, *transport protocol*, *source port*, and *destination port*. A flow is composed of two unidirectional subflows (from source to destination and vice versa) identified by interchanging source and destination addresses and the corresponding transport ports. Internet traffic traces can be captured on a network interface using standard network sniffers like tcpdump (http://www.tcpdump.org/) and Wireshark (http://www.wireshark.org/) or at the user space by using network virtualization mechanisms like in [11]. These tools allow the gathering and analysis of Internet traffic packets belonging to different flows, both in an offline (reading of .pcap files) and in an online (live capture of the packets) scenario. In the latter case, the capture takes place for the packets flowing across the particular node the sniffer is installed in. Regarding a classification purpose, all Internet traffic classification techniques can be framed in the following categories [12] [13]: session-based, content-based, and statistical approaches. The first ones rely on the knowledge of the so-called "well-known" ports, assigned to already defined services and protocols by the Internet Assigned Numbers Authority (IANA) (https://www.iana.org/). Conversely, the second one performs an exhaustive analysis of the packets' payload to look for particular signatures of transport and application protocols. Finally, the third ones are those employed in this paper and they take advantage of concepts and methods from statistics and information theory, as well as artificial intelligence to perform the required identification. Differently from the previous ones, these techniques do not require known packet signatures or any information on the application content; conversely, they perform identification just based on "external" traffic characteristics, like packet sizes and timing information, forming the set of input features of the classification mechanism. Further characterization of the possible techniques may refer to the granularity of the performed traffic classification. In particular, the following two classification types could be considered:

(i) *fine-grained* classification, whose aim is the detection of the particular *application protocol* generating a certain flow; [14]

(ii) *coarse-grained* classification, whose focus is the identification of a larger subset of protocols (e.g., web surfing, mailing and file transfer), and not of a particular protocol.

As regards the traffic features that can be used for the classification, the most used refer to transfer-based, time-based, and protocol-based characteristics of the packets [15]. Moreover, network packets are usually considered as belonging to unidirectional or bidirectional flows between two endpoints. The features describing a flow can be extracted from multiple levels of abstraction [16]. Indeed, these features may regard:

(i) a single packet and its intrinsic characteristics, such as the sequential position in a flow, the time distance from the previous and the following packet, the size in bytes, etc.

(ii) summarizing metrics of both the whole flow and its constituting subflows, such as total duration, overall volume in bytes, mean value, and standard deviation thereof.

As we will point out later, in the experiments we described in this paper, we considered both a binary classification, by distinguishing benign traffic from anomalous flows, and a more fine-grained multiclassification, by detecting different typologies of attack flows. As for the feature model, described in detail in Section 4.1, we considered features related to the

summarizing metrics of a bidirectional flow and its two unidirectional forward and backward subflows.

3. Related Work

Applications in the Internet of Things are becoming pervasive in many domains around the world (e.g., smart buildings, fleet management, and smart agriculture). However, this leads to many security threats. In the literature, several studies are focusing on the use of artificial intelligence techniques for anomaly detection in IoT scenarios. In [17], an approach exploiting a conditional autoencoder for anomaly detection in IoT environments is studied. The proposed method allows the retrieval of missing features as well as feature reconstruction in case of incomplete data. The NSL-KDD (https://www.unb.ca/cic/datasets/nsl.html) dataset was used. The obtained results highlight that the method improves classification performance and is less complex compared with other unsupervised approaches. An analysis of wireless network threats is proposed in [18], where the authors use an anomaly detection system to classify attacks in IEEE 802.11 networks. The proposed network adopts a Stacked Autoencoder, built by stacking multiple layers of sparse autoencoders. A dataset generated from an emulator was used for testing, and the obtained results with a 2-layer neural network report an accuracy of 98.668% in a multinomial classification (4 types of attacks are identified). Fog Computing principles were adopted in [19] to detect intrusions in IoT environments. Specifically, the authors propose the use of edge devices provided with detection abilities and adopt a deep learning network to detect intrusion attacks. The NSL-KDD dataset was used for the experiments considering 123 features, achieving 98.27% of accuracy for 4-class detection, which is improved by increasing the number of fog nodes. Another approach for intrusion detection is the AdaBoost ensemble method proposed in [20]. It exploits artificial neural networks, decision trees, and Naive Bayes classifiers to mitigate botnet attacks in different protocols (DNS, HTTP, and MQTT) utilized in IoT networks. The study considered 36 features extracted from two different datasets, but the authors adopted a feature selection process as well. The best accuracy obtained for the binary classification is 98.97%. In [21], the authors propose a hybrid and scalable Dense Neural Network framework for the real-time monitoring of network traffic and host-level events, in order to identify possible attacks. They consider multiclassification for detecting different attacks and binary classification to identify anomalies in the traffic. The evaluation is performed on different public datasets, and the obtained results were compared with traditional machine learning algorithms. The best achieved overall accuracy for the multiclassifier changes for the two used datasets (87.3% and 93.57%, respectively). In [22], a Hybrid Neural Network approach is proposed and evaluated on two datasets. Similar to [21], the model was tested for multiclassification and binary classification. The adopted features refer to different types of traffic flow. The best accuracy value for the binary classification is 99.58%, while in the multiclass assessment is 99.61%.

More recently, in [23] a system for attack detection that interlinks development and operations frameworks is proposed. Specifically, a deep convolutional neural network architecture is used, with an optimization of the activation functions, the filters, and the filter sizes. The experimental results indicate that the proposed algorithm, for application under the GAF-GYT attack, achieves higher accuracy than the compared methods.

The analysis of the literature highlights that the performance of the existing approaches strongly depends on the adopted dataset and experimental settings. Therefore, in this study, we evaluate our proposed approach on a large and integrated dataset regarding IoT scenarios. Moreover, our evaluation also includes the assessment of the classification performance using different network configurations (hyperparameters permutations, feature selection, and noisy features). The obtained performance is, generally, higher than other similar approaches.

4. Proposed Approach

The proposed approach is summarized in Figure 1. Once the complete feature set has been extracted, it is reduced by using an autoencoder neural network (as an alternative, also Principal Component Analysis (PCA) is evaluated). The reduced feature set is used as input to the classifier to perform both a multinomial and binary classification. The detailed description of the extracted feature set, the data reduction step, and the classification step are reported in the following.

4.1. Feature Model. Starting from the raw traffic flows, 70 features were extracted for each flow by using the CICFlowMeter tool (https://github.com/ahlashkari/CICFlowMeter) [24]. Each flow has an initiator, that is, the entity that sent the first packet to the other entity, and the responder, that is, the other entity. Forward packets are the packets sent from the initiator to the responder, while backward packets are the packets from the responder to the initiator. The features are summarized in the following:

(i) *General features of the flow* (5): duration of the whole flow, that is, the interval between the first and the last packet; the total number of forward packets; the total number of bytes sent forward; the total number of backward packets; and the total number of bytes sent backward

(ii) *Features related to packet sizes* (14): minimum, maximum, mean, standard deviation, and variance of the size of flow packets; minimum, maximum, mean, and standard deviation of the size of the forward packets; minimum, maximum, mean, and standard deviation of the size of the backward packets; and backward to forward byte ratio, that is, the total number of forward bytes divided by the total number of backward bytes

(iii) *Packet and byte rates* (4): byte rate, computed as the total number of bytes divided by the duration; packet rate, that is, the total number of packets

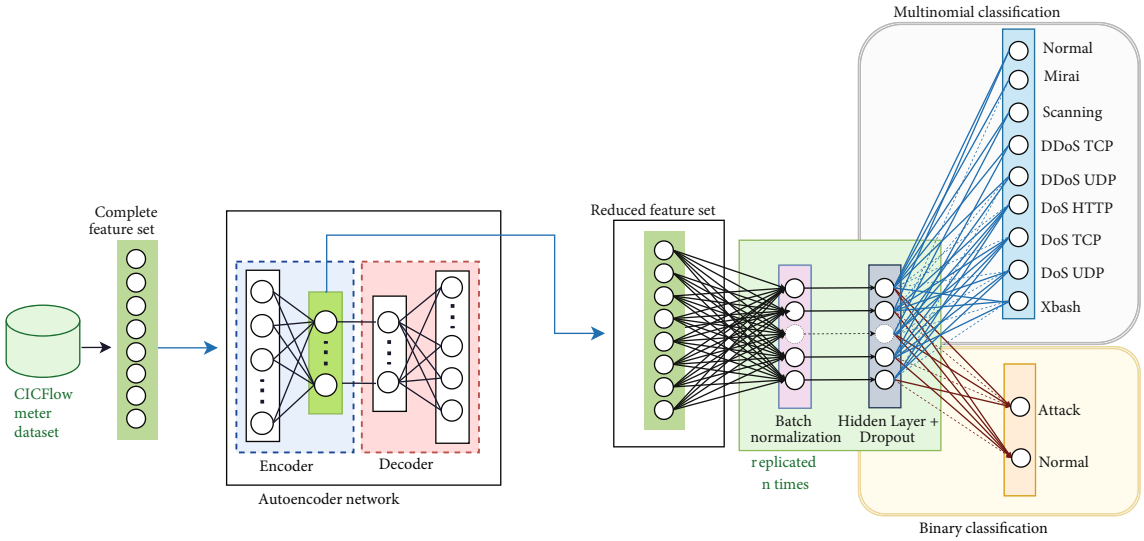

Figure 1: An overview of the proposed approach.

divided by the duration; forward packet rate; and backward packet rate

(iv) *Packet interarrival times* (14): minimum, maximum, mean, and standard deviation of packet interarrival time for all flow packets; minimum, maximum, mean, and standard deviation of the interarrival time of forward packets; minimum, maximum, mean, and standard deviation of the interarrival time of backward packets; the sum of all interarrival times of forward packets; and the sum of all interarrival times of backward packets

(v) *Features related to packet headers* (14): the number of forward TCP [25] PSH packets, that is, forward packets with PSH flag on; the number of forward TCP URG packets; the number of backward TCP PSH packets; the number of backward TCP URG packets; the total number of TCP FIN packets; the number of TCP SYN packets; the number of TCP RST packets; the number of TCP ACK packets; the number of TCP CWR packets; and the number of ECE packets

(vi) *Features related to packet payloads* (11): the average payload size among all flow packets; the average payload size in forward direction; the average payload size in backward direction; the average number of packets and average number of bytes in a burst of forward packets, formed of almost consecutive packets; the average number of packets and average number of bytes in a burst of backward packets; the number of bytes in the initial burst of forward packets; the number of bytes in the initial burst of backward packets; the number of forward packets with payload; and the minimum payload size of a forward packet

(vii) *Flow state features* (8): minimum, maximum, mean, and standard deviation of active time and of idle time

4.2. Dimensionality Reduction. Dimensionality reduction is a method to decrease the complexity of a model and avoid overfitting. In this work, we adopt a dimensionality reduction approach based on encoding/decoding neural networks (i.e., autoencoders) comparing it with a standard baseline approach like PCA [26].

An autoencoder (AE) is a neural network architecture designed to learn new features. Autoencoders have been originally formulated to initialize neural network weights [27, 28] and continued to satisfy that goal for some time. Across the latest years, other purposes for AEs have been developing and other methods for training and regularization of neural networks have replaced AEs [29, 30]. Consequently, AEs moved from supporting neural networks training to different purposes. A distinguishing aspect of the AE training process is that it can be performed in an unsupervised way, i.e., the model class labels can be ignored. Alternatively, it elicits useful knowledge from each case by applying to its feature vector several transformations that force constraints on the permissible representations. Next, the initial feature representation is linked to a novel feature space by a set of transformations, and the autoencoder quality is assessed by looking at the correctness of the reconstructed data. The computed error enables iteratively adjusting the weights until the requested performance is met. AEs are themselves neural networks with a single hidden layer at least and are comprised of two main parts: an encoder subnet and a decoder subnet. These two subnets are linked by a coding layer [31] compressing input data and are normally symmetric in layer configurations to each other, particularly if they are realized as fully connected neural networks. In the bottom center of Figure 1, the architecture of a typical AE is depicted. Essentially, an AE is a composition of the following:

(i) An encoding map F which projects inputs over a distinct feature space

(ii) A decoding map G working with inverted logic

The primary purpose of the AE is to gain as much knowledge as possible of the initial input to minimize the distance between its original inputs and the outputs reconstructed from the coding layer:

$$\min_{\phi} \sum_{i \in I} \lambda\left(i, G_{\phi}\left(F_{\phi}(i)\right)\right), \tag{1}$$

where ϕ is the full set of trainable parameters of the AE (e.g., weights and biases) and I is the set of all input instances. The distance function λ used in the loss function is usually either the cross-entropy or the mean squared error. In this work, we adopted the latter, defined as follows:

$$\lambda\left(i, i'\right) = -i * \log\left(i'\right) + (1 - i) * \log\left(1 - i'\right), \tag{2}$$

where $*$ is the element-wise product. All the other operations are executed element-wise. For a cross-entropy loss, variables are modeled by a Bernoulli distribution, normalizing input values in the [0,1] interval. Last layer units can use a sigmoid activation function. The AE proposed in this work is an adaption of the concrete AE used in [10]. The number of units in the selector layers increases with the requested level of feature compression. This layer picks a stochastic linear combination of input features during training, reaching a subset of features by the conclusion of the training step. The decoder part, which serves as the regeneration function, is a neural network whose architecture can be sized by looking at the dataset extent and complexity. The AS as proposed in [10] uses a temperature parameter T of the encoding layer handled by a simulated annealing process that forces it to reach zero at the end of the training obtaining a discrete feature selection instead of feature reduction. In our variant, the parameter is managed to keep the layer temperature low: this allows the generation of combinations of a reduced number of features (sparse autoencoder) using a thin single-layer encoding subnet and a generic n-layer decoder that can be adequately sized by looking at the dataset size itself.

4.3. Classification. The main deep neural network of the classifier we used is a feedforward neural network architecture, whose main layers are depicted in Figure 1 and are described in detail in the following:

(i) *Input layer*: this is the entry-level of the whole neural network, composed of several nodes equal to the number of features in the considered dataset;

(ii) *Batch normalization layer*: this layer, added before each dense hidden layer, is employed to enhance the training phase of the neural network itself, given that it augments the velocity of the training and allows the adoption of higher learning rates and the saturation of possible nonlinearities. This, in turn, usually permits a higher accuracy on both validation and test sets, because of a stable gradient propagation inside the deep neural network itself; [32]

(iii) *Hidden layers*: these are a variable number of dense layers constituted of artificial perceptrons (MLP [33]) which output a weighted sum of their inputs, passed through a proper activation function. The overall neural network is made up of at least five fully connected (dense) layers of perceptrons;

(iv) *Dropout layer*: this layer is tightly coupled with the aforementioned one and immediately following it. Indeed, we replicated different times the triple batch normalization layer-dense hidden layer-dropout layer. The dropout layer allows the prevention of overfitting by turning off randomly several neurons in the coupled dense layer, following a Bernoulli probability distribution function;

(v) *Output layer*: it provides the final classification and is made of several nodes equal to the number of classes. In Figure 1, two different output layers are shown, the binary one and the multiclassification one, but only one of them is considered in each experiment. Indeed, it is a simple dense layer with a softmax as an activation function.

In case of binary classification, only two traffic classes are considered: *Normal* and *Attack*. Instead, in the case of multiclassification, the neural network is used to distinguish between *Normal* traffic and eight specific types of attacks that have been considered, that are,

(i) *Scanning*: activity aimed at scanning a network for discovering active hosts and open ports and for identifying possible vulnerable active services;

(ii) *TCP DoS*: DoS attacks based on the TCP protocol, usually consisting in a SYN flood attack that exploits the initial TCP three-way handshake procedure, trying to saturate the processing resource of the victim;

(iii) *UDP DoS*: DoS attacks where UDP packets are sent to a targeted node to overload the processing capability of the node itself;

(iv) *TCP DDoS*: TCP DoS attack performed by a distributed attacker like a botnet or a TCP SYN-ACK reflection attack, where the attacker sends spoofed SYN packets to several TCP servers, using as source IP address the victim IP address;

(v) *UDP DDoS*: UDP reflection attack or distributed attack performed by a botnet;

(vi) *HTTP DoS*: DDoS attacks in which an HTTP server is flooded by HTTP requests making the server unable to respond to normal requests; like the previous ones, it is based on the fact that either the resource required by the target to respond is larger than the resource used by the attacker or

the attacker has much more resources (e.g., the attacker uses a botnet and/or the server is a constrained device);

(vii) *Mirai*: specific Distributed Denial of Service (DDoS) attack performed by a malware that mainly targets consumer IoT devices such as home WiFi routers and IP webcams and tries to install a copy of the malware and transforming the node into a zombie of a larger botnet;

(viii) *Xbash*: malware that spreads by attacking weak passwords and unpatched vulnerabilities; it targets Windows and Linux-based systems and combines cryptomining, ransomware, botnet, and self-propagation capabilities.

5. Evaluation

This section describes the used integrated dataset along with the procedures followed for its construction and balancing and presents the considered evaluation settings as well as the considered neural network parameters.

5.1. Dataset Construction. The literature review demonstrates that the research about anomaly detection mainly uses ad hoc datasets, employed to assess specific malicious traffic. Indeed, the main well-known limitations of the available datasets reside in the fact that:

(i) they are small and not suitable to be exploited by deep learning techniques, which require a certain amount of training samples;

(ii) they contain only a limited number of attacks or are built in such a way that it is difficult to detach the abnormal flows from the normal ones;

(iii) they are often built with traffic from the same networking environment, wherein packets and traffic flows manifest the same behaviors and patterns across the considered network attributes.

Taking inspiration from these drawbacks, the construction of a large integrated dataset was made. Specifically, five different IoT subdatasets, with different types of attacks, were integrated.

The integration procedure followed the subsequent steps:

(i) Selection of the datasets;

(ii) Dataset transformation;

(iii) Dataset labeling checking;

(iv) Final dataset combination.

The first phase regarded finding recent datasets in the IoT domain. Moreover, the selected datasets (D1, D2, D3, D4, and D5) had to entail a sufficient number of instances for both normal and malicious traffic.

Dataset D1 (https://ieee-dataport.org/open-access/iot-network-intrusion-dataset) was published in September 2019. Its traffic comes from two typical smart home devices (i.e., SKT NUGU and EZVIZ Wi-Fi Camera), and from some laptops and smartphones, present in the same wireless network. For this dataset, we considered normal traffic and two types of malicious flows: Mirai traffic and scanning traffic. As regards Mirai traffic flows, the packets are modified to appear as originated from an IoT device. Conversely, scanning flows, which include both "OS Scan" and "Service Scan" attacks, contain packets simulated using Nmap.

Dataset D2 (https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot_iot.php) was generated in the Cyber Range Lab of UNSW of Canberra. For this dataset, normal traffic, as well as various types of malicious flows (i.e., Scanning, DDoS TCP, DDoS UDP, DoS HTTP, DoS TCP, and DoS UDP), is considered.

As regards dataset D3 (https://iotanalytics.unsw.edu.au/iottraces.html), it is described in [34]. For this dataset, only normal traffic, coming from various devices, is considered.

For what concerns dataset D4 (https://www.stratosphereips.org/datasets-iot), it mainly involves malicious traffic, obtained at the Stratosphere IPS laboratory of the Czech Technical University in 2018 and 2019. For this dataset, only abnormal traffic of type Mirai and Xbash is considered.

Concerning dataset D5 (https://github.com/tjcruz-dei/ICS_PCAPS), it is derived from a small automation testbed using MODBUS/TCP for research in the context of cybersecurity in Industrial Control Systems. The testbed emulates a cyberphysical system process controlled by a SCADA system using both MODBUS and TCP protocols.

The second phase concerned the creation of .csv files starting from raw .pcap files. This activity was performed by using the CICFlowMeter tool.

The third phase concerned the labeling of the flow instances. For datasets D3 and D4, they were processed by a proper Python script to assign the correct label to each flow. This phase produces both a binary dataset containing only the labels "Normal" and "Attack" and the multiclass dataset, composed of nine different classes, namely, "Normal," "Mirai," "Scanning," "DDoS TCP," "DDoS UDP," "DoS HTTP," "DoS TCP," "DoS UDP," and "Xbash." The scanning class merges "OS Scan" and "Service Scan" attacks because often "OS Scan" attacks involve also the scanning of well-known service ports.

The overall statistics of both the binary and the multiclass dataset are summarized in Tables 1 and 2, respectively. The whole integrated dataset is freely available (all data used for this research were provided as a supplementary material (available here)) and comprises a total of 421,530 flow instances in the binary version and 213,210 in the multiclass version. The binary version of the dataset is almost perfectly balanced, whereas the multiclass version is slightly unbalanced as regards the "Scanning" class and rather balanced for all the other classes.

5.2. Evaluation Settings. The evaluation of the proposed approach is made, firstly, by performing both binary and

Table 1: Statistics of the integrated binary dataset.

Class\dataset	D1	D2	D3	D4	D5	Integrated
Normal	496	9,171	198,320	0	0	207,987
Attack	21,322	160,000	0	26,668	5,556	213,543
Total instances	21,818	169,171	198,320	26,668	5,556	421,530

Table 2: Statistics of the integrated multiclass dataset.

Class\dataset	D1	D2	D3	D4	D5	Integrated
Normal	496	9,171	10,000	0	0	19,667
Mirai	18,623	0	0	4,000	0	22,623
Scanning	2,699	40,000	0	0	0	42,699
DDoS TCP	0	20,000	0	0	5,556	25,556
DDoS UDP	0	20,000	0	0	0	20,000
DoS HTTP	0	20,000	0	0	0	20,000
DoS TCP	0	20,000	0	0	0	20,000
DoS UDP	0	20,000	0	0	0	20,000
Xbash	0	0	0	22,665	0	22,665
Total instances	21,818	149,171	10,000	26,665	5,556	213,210

multinomial classifications on the complete feature set extracted from the integrated dataset. Successively, a hyperparameter optimization is performed and the performance of the classifiers with different hyperparameter combinations is evaluated. The best hyperparameter combination is finally considered to evaluate and compare the classification results obtained on a more reduced set of features (the considered feature numbers are 60, 50, 35, and 25). The feature reduction is performed using two different autoencoder networks (3 layers and 9 layers), with the alternative PCA approach also used as a comparison. Finally, we compare the previously obtained results with those coming from a noisy scenario, wherein up to 40% of the features may be affected by Gaussian noise.

The classification is performed by using a deep neural network based on MLP. The validation set is the 20% of the training set, which is 90% of the whole dataset considered in each experiment.

The hyperparameter optimization [35] is performed with a Sequential Bayesian Model-based Optimization (SBMO) approach, implemented using the Tree Parzen Estimator (TPE) algorithm as defined in [36].

The hyperparameters considered in the optimization are reported in Table 3 and described in the following:

(i) *Network size*: we considered two possible sizes of the DNN (small and medium), named after the number of nodes per layer. A small-sized network contains a maximum of 1.5 mln learning parameters, while a medium one is composed of several parameters between 1.5 mln and 7 mln;

(ii) *Activation function*: we considered DNN configurations with only the well-known and widely adopted ReLU as an activation function and with a mix of ReLU and Swish, a novel activation function with promising results in recent studies [37]. This choice is because ReLU suffers from the "dead" unit problem, i.e., during the training phase, some ReLU units always output the same value for any input, with no role in discriminating between inputs. This takes place when the network learns a large negative bias term for its weights during the training step. Whenever a ReLU unit arrives at this state, it is not easy to be recovered in the future, because the gradient function at 0 is still 0, thereby SGD will not alter the weights. Although some variants of ReLU, e.g., "Leaky" ReLU, with a small positive gradient for negative inputs, try to tackle this issue and provide a recovery possibility, we chose to introduce Swish since it does not suffer from the dead neuron problem and faces better the vanishing gradient issue;

Table 3: Optimized hyperparameters and ranges.

Hyperparameters	Ranges
Batch size	{256, 512}
Network size	{Small, medium}
Activation functions	{ReLU, Swish}
Dropout	In range [0.1, 0.2]
Optimization algorithm	{SGD, RMSProp, Nadam}
Learning rate	[3, 11]

(iii) *Learning rate*: it ranged from 3 to 11, normalized for the selected optimizer. For example, when the SGD optimizer is used, the range was from 0.03 to 0.11;

(iv) *Number of layers*: the number of considered hidden layers, which was varied from 5 to 9;

Table 4: Best validation accuracy on the binary dataset for the top permutations.

Permutation	Network size	Activation functions	Layers	Bach size	Dropout	Optimization algorithm	Learning rate	Val accuracy
Best	Small	All ReLUs	8	512	0.15	Nadam	9	0.9989
P1	Medium	All ReLUs	6	256	0.15	SGD	9	0.9986
P2	Small	ReLUs and Swish	6	256	0.15	SGD	9	0.9985

(v) *Batch size*: it is the number of training samples used in one iteration of update of the DNN internal parameters. Values greater than 512 usually make the training phase rather unstable and the final accuracy not satisfactory; thus, we considered two batch sizes, namely, 256 and 512;

(vi) *Optimization algorithm*: we evaluated some of the most used optimization algorithms, i.e., Stochastic Gradient Descent (SGD) [38], RmsProp [39], and Nadam [39]. Moreover, in all experiments, SGD was integrated with Nesterov Accelerated Gradient (NAG) correction, thus avoiding excessive changes in the parameter space [40], while its momentum was set to 0.12 and its decay to 10^{-6};

(vii) *Dropout rate*: the considered dropout rates belong to the interval [0.1,0.2] with a step of 0.05;

(viii) *Number of training epochs*: it is the number of times the training set is presented to the DNN and is set to 100 for the validation phase.

The classifier's performance is evaluated by using four well-known metrics: accuracy, validation accuracy, loss, and validation loss. Accuracy is an overall metric and is computed as the ratio of the sum of true positives and true negatives to the total number of samples. The accuracy is computed over the training set, while the validation accuracy is calculated on the validation dataset. The loss implies how poorly or well a model behaves after each iteration of optimization and, similarly to the accuracy, is computed on both the training and the validation set.

Moreover, to analyze results on the test set, we also consider the weighted Precision, Recall, *F*-measure, and the complete confusion matrix.

Precision is evaluated as the part of samples that truly belong to a given attack (or normal flow) among all those which were assigned to it by the classifier. The recall is the proportion of samples assigned to a given attack (or normal flow), among all the samples that truly belong to the attack (or normal traffic) itself. The *F*-measure is the weighted harmonic mean of precision and recall.

The DNN classifiers are developed by using Python language, with a particular focus on TensorFlow (https://www.tensorflow.org/), an open platform for deep learning tasks from Google, coupled with Keras (https://keras.io/), an open-source library working on a higher level than TensorFlow itself. For the hyperparameter optimization, we took advantage of Talos (https://autonom.io/), a hyperparameter tuning library specifically developed to be used with Keras. To carry out the various experiments, we employed an Intel Core i9 9940X 10th gen server, equipped with 4 NVIDIA Tesla T4 GPUs and 64 GB of RAM.

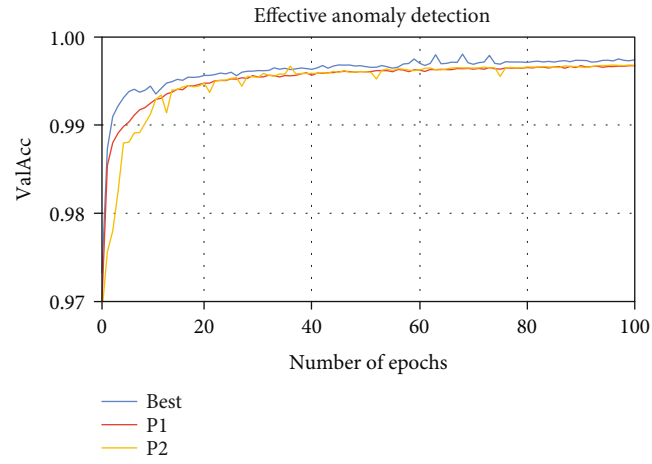

Figure 2: Validation accuracy versus the number of epochs for the binary classification for the best performing permutations of the hyperparameters.

6. Results and Discussion

In this section, we present the results of the experiments described in the previous section.

6.1. Classification Performance. Herein, the performance of the classifiers using the complete feature set is discussed. As regards the binary classification, the best validation accuracy value is 0.9989, reached at epoch 73 in the best hyperparameter permutation. In Table 4, we present the values of the hyperparameters for the best permutation and those of the other two permutations (P_1 and P_2) performing very close to the best one. As one can see, the configuration achieving the top validation accuracy considers 8 layers and all ReLU functions in a small DNN configuration. The learning rate of the best permutation is 9, while the chosen optimizer is Nadam, and the batch size is 512. Conversely, the other two considered permutations use SGD as an optimizer and 256 as the batch size and are endowed with 6 hidden layers as well as a variable network size and activation function map. In Figure 2, we show the trend of the validation accuracy versus the number of epochs across a 10-fold cross-validation process on the binary dataset for all the aforementioned permutations. It can be seen that a sort of saturation trend is reached just after the 20th epoch for all the three curves, even if some oscillations are still present after epoch 60, especially for the best permutation and permutation P_2. Differently, permutation P_1 exhibits a much

Table 5: Best validation accuracy on the multiclassification dataset for the top permutations.

Permutation	Network size	Activation functions	Layers	Bach size	Dropout	Optimization algorithm	Learning rate	Val accuracy
Best	Medium	All ReLUs	8	512	0.15	RMSProp	10	0.994
P1	Small	ReLUs and Swish	5	256	0.15	SGD	9	0.990
P2	Medium	ReLUs and Swish	9	256	0.15	Nadam	9	0.990

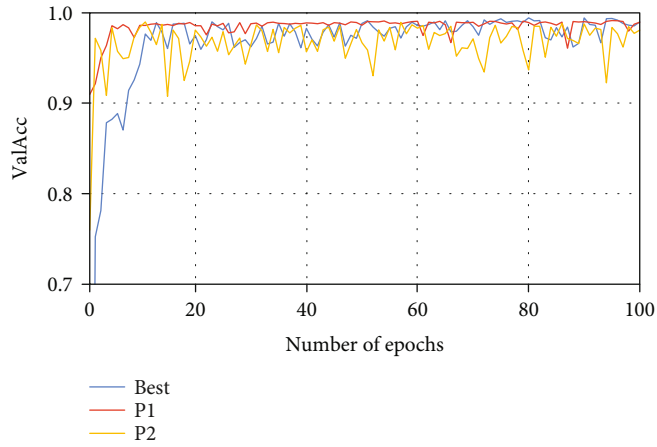

Figure 3: Validation accuracy versus the number of epochs for the multinomial classification for the best performing permutations of the hyperparameters.

smoother trend and its curve is usually placed between the other two.

For what concerns the multinomial classification, the best validation accuracy value is 0.994, reached at the 94th epoch. The top-performing permutations are shown in Table 5. Similar to the binary case, the configuration achieving the top target result entails 8 layers, all ReLU functions, and a batch size of 512. Differently from the binary case, a medium DNN configuration is considered and the learning rate is 10, while the optimizer is RMSProp. The other two high-performing permutations are characterized by an activation function map with both ReLU and Swish functions, a learning rate equal to 9, and a batch size of 256. In Figure 3, the trend of the validation accuracy versus the number of epochs across a 10-fold cross-validation process on the multiclass dataset, for the aforementioned permutations, is shown. It can be seen that all the permutations reach a sort of saturation, in this case, almost at epoch 10, and that, differently from the binary case, the curves start at a much lower point and tend to oscillate more till the 100th epoch. This behavior, as well as the smaller top value for the validation accuracy, can be motivated by the greater inherent difficulty in discriminating more than 2 classes. Moreover, in this case, permutation P_1 is the most stable and smoothest, even if not reaching the best validation accuracy values.

It is worth noting that the best-achieved results, on the validation accuracy, are better than those obtained in the work in [8] for the binary dataset and similar in the case of the multiclass dataset, respectively. However, in the multinomial classification task, in this work, a higher number of malicious traffic kinds (i.e., eight attacks instead of four) are considered. Additionally, to improve the reliability of the assessment, the validation is performed with a 10-fold cross-validation rather than a 5-fold one, obtaining more trustworthy results.

Finally, in Figure 4, we report the confusion matrix for the multiclassification case under the best permutation. As it can be easily inferred, the optimized DNN detects perfectly all DDoS or DoS attacks, except for the HTTP DoS attack, confused only in one case with a Scanning attack and in another one with the Xbash attack. Scanning attacks are recognized almost perfectly as well, with less than 1% of misclassified sample flows and mainly confused with Mirai and normal traffic. Normal traffic and Xbash attacks experience about 2% of not correctly classified flows, with mistakes in the classification mainly focused on malware attacks as well as onto normal traffic, respectively. Finally, Mirai flows are the worst to be correctly detected, by exhibiting, even in the best hyperparameter configuration, misclassification in 3% of the cases, with HTTP DoS attacks as the most confused class. Besides the confirmed riskiness of Mirai attacks, the confusion matrix highlights also that the majority of the classification mistakes regard normal traffic or a very common type of traffic, i.e., HTTP flows.

6.2. Classification Performance with Feature Reduction. In this subsection, we present the classification results we obtained with a varying number of features.

Indeed, we performed various elaborations on the initial integrated dataset, mainly focused on reducing the number of attributes through an autoencoder. The results obtained using two different autoencoder networks (9 layers and 3 layers) are evaluated and compared with those obtained by using a PCA reduction approach. The new elaborated

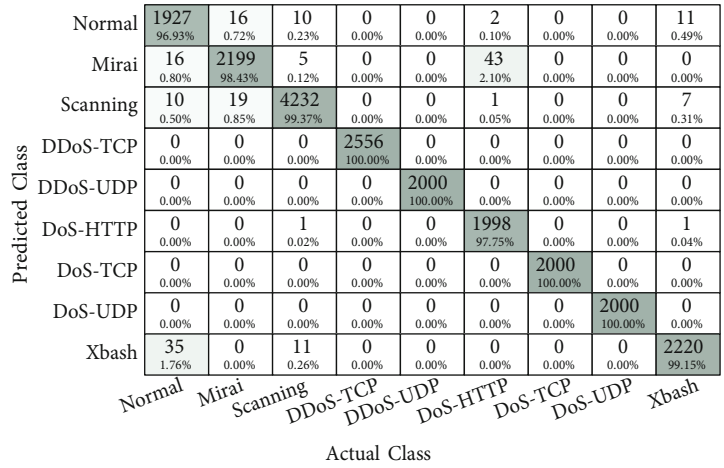

Figure 4: Confusion matrix for the best performing multinomial classifier.

Table 6: Performance of the binary classifier varying the number of features through PCA and autoencoders, respectively.

Features	PCA		AE 3 layers		AE 9 layers	
	Accuracy	F1	Accuracy	F1	Accuracy	F1
70	0.993	0.993	0.993	0.993	0.993	0.993
60	0.991	0.991	0.960	0.960	0.991	0.991
50	0.992	0.992	0.964	0.964	0.994	0.994
35	0.994	0.994	0.976	0.976	0.994	0.994
25	0.992	0.992	0.960	0.960	0.993	0.993

datasets, both for the binary and the multiclassification task, contain 60, 50, 35, and 25 features, respectively.

For the binary classification, the accuracy and F-measure (F1) for a different number of features obtained, respectively, using PCA, an autoencoder with 3 (AE 3layers), and 9 layers (AE 9layers) are reported in Table 6. The accuracy is quite stable across the reduction of the number of features for both PCA and AE 9layers, and it achieves a top value of 0.994, whenever 35 features. This may indicate that about half of the initially 70 features could be cut or merged into other more significant ones, without perturbing the overall performance at all.

In Figure 5, we show the trend of the F-measure on the binary classifier for a varying number of features and the three considered modes to perform feature reduction. The performance of PCA and AutoEnc 9layers is very similar, whereas the performance of AutoEnc 3layers is much worse. In all cases, it is clear that only a small decrease in the number of features leads to worse average results, both in terms of absolute values and in terms of trustworthiness (greater standard deviation, values not shown).

The results for the multinomial classification are shown in Table 7. Different from the binary case, the performance changes across the considered evaluation metrics because the dataset is rather unbalanced. Besides, it can be inferred, from the obtained outcomes, that F-measure reaches the best values when 50 features are considered in the case of PCA and with 60 features in the case of AE 9layers, thus

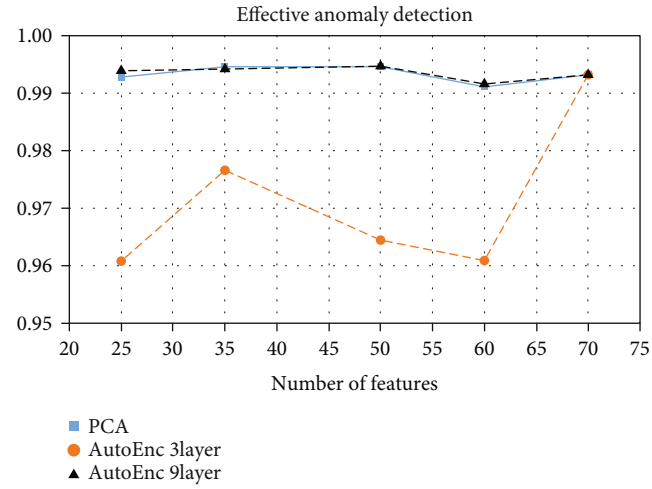

Figure 5: F-measure on the binary classifier versus a variable number of features obtained with PCA and autoencoder.

Table 7: Performance of the multinomial classifier varying the number of features through PCA and autoencoders, respectively.

Features	PCA		AE 3 layers		AE 9 layers	
	Accuracy	F1	Accuracy	F1	Accuracy	F1
70	0.960	0.961	0.960	0.961	0.960	0.961
60	0.978	0.978	0.917	0.919	0.989	0.989
50	0.983	0.983	0.901	0.900	0.979	0.980
35	0.980	0.980	0.912	0.912	0.976	0.977
25	0.930	0.931	0.903	0.904	0.928	0.928

demonstrating that, for the multiclassification, more features can be necessary to perform good discrimination compared to the binary case, but also confirming that the original 70 features are too many and may introduce unwanted noise. Notwithstanding, the performance is quite stable across the reduction of the number of features, dropping heavily only when considering 25 features.

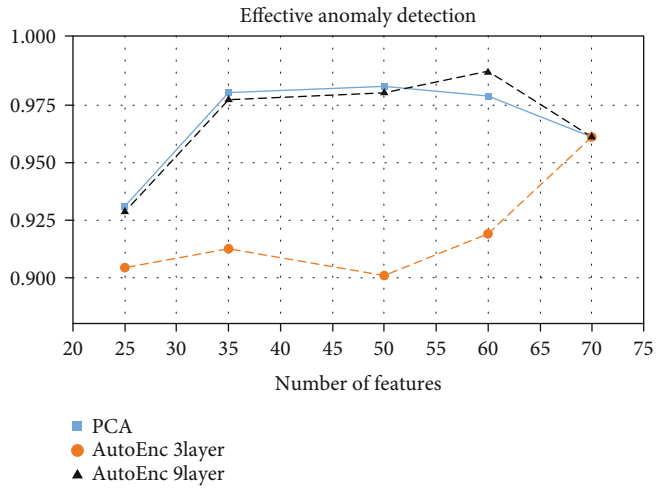

Figure 6: F-measure on the multinomial classifier versus a variable number of features obtained with PCA and autoencoder.

Table 8: Performance of the binary and multinomial classifier for a varying number of features created through the 9-layer AE and with 5 and 10 random noisy features.

Features	Binary classifier		Multinomial classifier			
	$Accuracy_5$	$Accuracy_{10}$	$Accuracy_5$	$Accuracy_5$	$Accuracy_{10}$	$Accuracy_{10}$
70	0.990	0.992	0.977	0.977	0.973	0.974
60	0.991	0.994	0.983	0.983	0.986	0.986
50	0.994	0.993	0.984	0.984	0.982	0.982
35	0.994	0.992	0.983	0.983	0.984	0.984
25	0.987	0.994	0.953	0.953	0.928	0.927

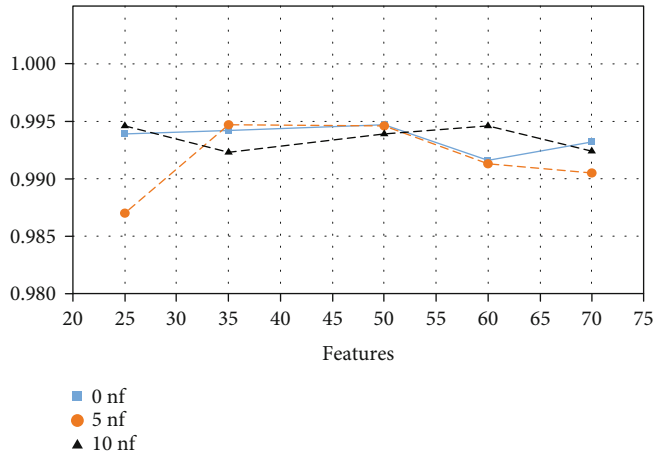

Figure 7: Accuracy on the binary dataset versus a variable number of features and noisy features (nf) for the 9-layer AE.

Figure 6 shows the trend of the F-measure for a varying number of features obtained with PCA and autoencoder approaches, respectively. The curve of AutoEnc 3layer is the worst one as in the binary case, while the curves of PCA and AutoEnc 9layer are quite superimposable, except for the 60 feature case. Indeed, differently from the binary case, it is clear that only a small decrease in the number of features leads to better average results, both in terms of absolute values and in terms of trustworthiness (smaller standard deviation, values not shown).

6.3. Classification Performance in a Noisy Scenario. In this section, we discuss the performance of the classifier in a scenario wherein some features of the dataset under examination are corrupted by Gaussian noise with zero mean and 0.1 standard deviation (applied right after the min–max normalization). The analyses regard both the original integrated dataset, sporting 70 features, and the datasets with a reduced number of features. Moreover, we report only the results obtained by using the autoencoder with 9 layers because this achieves similar results as PCA, and in one case, it is better.

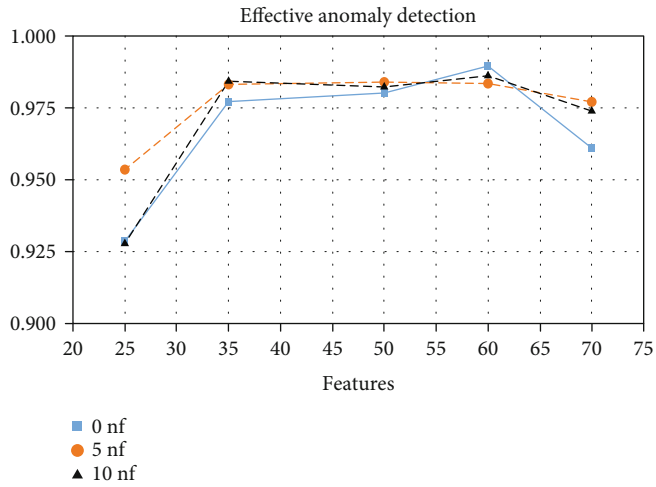

Figure 8: *F*-measure on the multiclass dataset with a variable number of features and noisy features (nf) for the 9-layer AE.

The number of features affected by Gaussian noise is 5 and 10, selected randomly across those available in each considered dataset, resulting in a percentage ranging from a minimum of 7% for the 70-feature dataset to a maximum of 40% for the 25-feature dataset.

The second and the third columns of Table 8 report the values of the binary classifier accuracy when the number of noisy features is 5 and 10 (Accuracy_5 and Accuracy_{10}), respectively.

In Figure 7, we plot the relative curves. The figure shows that, generally, the no-noise curve has an intermediate trend compared to the two noisy curves, which are intertwined. However, the curves are all very similar and inside the standard deviation range of the no-noisy curve (values not shown for the sake of readability); thus, the optimized DNN results are quite robust, in the binary case, against the introduced noise on the features, even with 40% of noisy features.

For what concerns the multinomial classification, the rightmost columns of Table 8 report the classifier performance with 5 and 10 noisy features. As discussed in the previous section, since the dataset is rather unbalanced, in this case, both *F*-measure and accuracy values are reported. In Figure 8, we show the *F*-measure curves for the multinomial classifier in a noisy scenario and with a variable number of features. As it can be seen, the no-noise curve exhibits a behavior very similar to that of the 10-noisy-feature curve. In all cases, the various points reside in the confidence interval of the standard deviation of the other curves (values not shown for the sake of readability), indicating a quite robust trend against this type of noise. Moreover, in the case of 60 features, the best performing configuration for the multiclass dataset with 9-layer AE, the original curve, i.e., the one with no noise, is all the same as the best performing one.

7. Conclusions

This paper proposes a DL-based approach for anomaly detection in IoT scenarios. We introduced a DNN architecture and a feature model composed of 70 features to perform anomaly detection identifying the type of attack from network traffic. The approach also includes a feature reduction step performed by using an autoencoder and a hyperparameter optimization analysis. To perform our experiments, we created a novel integrated dataset from IoT public traffic traces of different nature.

The obtained results show good performance in all the analyzed scenarios. For the binary classification, the best accuracy is obtained when all the features are used (0.9989 for the top hyperparameter permutation). Moreover, when feature reduction is performed, the classifier performance is quite stable (changing the features number by using a PCA and a 9-layer autoencoder, we always obtain accuracy greater than or equal to 0.992, provided that the number of features is greater than 35).

For the multinomial classifier, we observed that the 70 considered features are too many and that fewer features can lead to better and more trustworthy outcomes. However, in this case, the best accuracy (0.989) is obtained when the number of features is 60 using the 9-layer autoencoder for feature reduction. This feature reduction approach always ensures better performance when the number of features is between 60 and 35.

Finally, the robustness of the optimized DNN architecture in a noisy scenario involving some of the considered features is evaluated. We also show that the addition of Gaussian noise up to 40% of the considered features does not affect too much the performance, especially for the binary case.

As future work, we will focus on a more detailed feature selection approach, to find out explicitly the most relevant features and not only their number, on the integration of other IoT traffic datasets with more attack types, as well as on the testing of different DL network architectures.

Data Availability

The data used to support the findings of this study are included within the supplementary information file(s).

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Supplementary Materials

The integrated datasets, used for the binary and multinomial classifications and described in Section 5.1, respectively, are available as supplementary material. *(Supplementary Materials)*

References

[1] H. Tahaei, F. Afifi, A. Asemi, F. Zaki, and N. B. Anuar, "The rise of traffic classification in IoT networks: a survey," *Journal of Network and Computer Applications*, vol. 154, p. 102538, 2020.

[2] G. Acampora, M. L. Bernardi, M. Cimitile, G. Tortora, and A. Vitiello, "A fuzzy clustering-based approach to study malware phylogeny," in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Rio de Janeiro, Brazil, 2019.

[3] M. Bernardi, M. Cimitile, F. Martinelli, and F. Mercaldo, "Keystroke analysis for user identification using deep neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*, Budapest, Hungary, 2019a.

[4] M. L. Bernardi, M. Cimitile, D. Distante, F. Martinelli, and F. Mercaldo, "Dynamic malware detection and phylogeny analysis using process mining," *International Journal of Information Security*, vol. 18, no. 3, pp. 257–284, 2019.

[5] G. Perrone, M. Vecchio, R. Pecori, and R. Giaffreda, "The day after Mirai: a survey on MQTT security solutions after the largest cyber–attack carried out through an Army of IoT devices," in *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security*, pp. 246–253, Porto, Portugal, 2017.

[6] M. L. Bernardi, M. Cimitile, F. Martinelli, and F. Mercaldo, "Game bot detection in online role player game through behavioural features," in *Proceedings of the 12th International Conference on Software Technologies*, pp. 50–60, SciTePress, Madrid, Spain, 2017.

[7] M. Calabretta, R. Pecori, and L. Veltri, "A token-based protocol for securing MQTT communications," in *2018 26th international conference on software, telecommunications and computer networks (SoftCOM)*, Split, Croatia, 2018.

[8] R. Pecori, A. Tayebi, A. Vannucci, and L. Veltri, "Iot attack detection with deep learning analysis," in *2020 International Joint Conference on Neural Networks (IJCNN)*, Glasgow, UK, 2020.

[9] L. Aversano, M. L. Bernardi, M. Cimitile, and R. Pecori, "A systematic review on deep learning approaches for IoT security," *Computer Science Review*, vol. 40, article 100389, 2021.

[10] M. F. Balın, A. Abid, and J. Zou, "Concrete autoencoders: differentiable feature selection and reconstruction," *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., , pp. 444–453, PMLR, Long Beach, California, USA, 2019.

[11] L. Veltri, L. Davoli, R. Pecori, A. Vannucci, and F. Zanichelli, "Nemo: a flexible and highly scalable network emulator," *SoftwareX*, vol. 10, article 100248, 2019.

[12] R. Pecori and L. Veltri, "A statistical blind technique for recognition of internet traffic with dependence enforcement," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, Nicosia, Cyprus, 2014.

[13] C. L. Chowdhary, M. Mittal, P. A. Pattanaik, and Z. Marszalek, "An efficient segmentation and classification system in medical images using intuitionist possibilistic fuzzy C-mean clustering and fuzzy SVM algorithm," *Sensors*, vol. 20, no. 14, p. 3903, 2020.

[14] P. Ducange, G. Mannará, F. Marcelloni, R. Pecori, and M. Vecchio, "A novel approach for internet traffic classification based on multi-objective evolutionary fuzzy classifiers," in *2017 IEEE international conference on fuzzy systems (FUZZ-IEEE)*, Naples, Italy, 2017.

[15] J. Camacho, P. Padilla, P. Garcia-Teodoro, and J. Diaz-Verdejo, "A generalizable dynamic flow pairing method for traffic classification," *Computer Networks*, vol. 57, no. 14, pp. 2718–2732, 2013.

[16] Y. Wang, Y. Xiang, J. Zhang, W. Zhou, and B. Xie, "Internet traffic clustering with side information," *Journal of Computer and System Sciences*, vol. 80, no. 5, pp. 1021–1036, 2014.

[17] Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in IoT," *Sensors*, vol. 17, no. 9, p. 1967, 2017.

[18] V. L. L. Thing, "IEEE 802.11 network anomaly detection and attack classification: a deep learning approach," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, San Francisco, CA, USA, 2017.

[19] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for internet of things," *Future Generation Computer Systems*, vol. 82, pp. 761–768, 2018.

[20] N. Moustafa, B. Turnbull, and K. R. Choo, "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4815–4830, 2019.

[21] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41525–41550, 2019.

[22] C. Ma, X. Du, and L. Cao, "Analysis of multi-types of flow features based on hybrid neural network for improving network anomaly detection," *IEEE Access*, vol. 7, pp. 148363–148380, 2019.

[23] S. Sarma, "Optimally configured deep convolutional neural network for attack detection in internet of things: impact of algorithm of the innovative gunner," *Wireless Personal Communications*, vol. 118, no. 1, pp. 239–260, 2021.

[24] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Tor traffic using time based features," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, pp. 253–262, SciTePress, Porto, Portugal, 2017.

[25] Information Sciences Institute, USC, *Transmission control protocol*, RFC 793. RFC Editor, 1981, https://www.rfc-editor.org/rfc/rfc793.txt.

[26] I. Jolliffe, "Principal component analysis," in *International Encyclopedia of Statistical Science*, pp. 1094–1096, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2011.

[27] D. H. Ballard, "Modular learning in neural networks," *Proceedings of the sixth National Conference on artificial intelligence-volume 1*, , pp. 279–284, AAAI press, 1987.

[28] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[29] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudik, Eds., , pp. 315–323, JMLR Workshop and Conference Proceedings, Fort Lauderdale, FL, USA, 2011.

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[31] D. Charte, F. Charte, S. Garcaa, M. J. del Jesus, and F. Herrera, "A practical tutorial on autoencoders for nonlinear feature fusion: taxonomy, models, software and guidelines," *Information Fusion*, vol. 44, pp. 78–96, 2018.

[32] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," *proc. 32nd Int. Conf.On machine learning-Vol. 37*, pp. 448–456, 2015, http://jmlr.org/.

[33] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)–a review of applications in the atmospheric sciences," *Atmospheric Environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[34] A. Sivanathan, H. H. Gharakheili, F. Loi et al., "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Transactions onMobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2019.

[35] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural Computation*, vol. 12, no. 8, pp. 1889–1900, 2000.

[36] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, pp. 2546–2554, Curran Associates Inc., USA, 2011.

[37] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, http://arxiv.org/abs/1710.05941.

[38] T. Schaul, I. Antonoglou, and D. Silver, "Unit tests for stochastic optimization," 2013, http://arxiv.org/abs/1312.6055.

[39] Y. Wang, J. Liu, J. Misic, V. B. Misic, S. Lv, and X. Chang, "Assessing optimizer impact on DNN model sensitivity to adversarial examples," *IEEE Access*, vol. 7, pp. 152766–152776, 2019.

[40] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, pp. 1139–1147, PMLR, 2013, http://jmlr.org/.