

Article

IoT Botnet Anomaly Detection Using Unsupervised Deep Learning

Ioana Apostol , Marius Preda, Constantin Nila and Ion Bica 

“Ferdinand I” Military Technical Academy, 39-49 George Coșbuc Blvd, 050141 Bucharest, Romania; marius.preda@mta.ro (M.P.); constantin.nila@mta.ro (C.N.); ion.bica@mta.ro (I.B.)

* Correspondence: ioana.apostol@mta.ro

Abstract: The Internet of Things has become a cutting-edge technology that is continuously evolving in size, connectivity, and applicability. This ecosystem makes its presence felt in every aspect of our lives, along with all other emerging technologies. Unfortunately, despite the significant benefits brought by the IoT, the increased attack surface built upon it has become more critical than ever. Devices have limited resources and are not typically created with security features. Lately, a trend of botnet threats transitioning to the IoT environment has been observed, and an army of infected IoT devices can expand quickly and be used for effective attacks. Therefore, identifying proper solutions for securing IoT systems is currently an important and challenging research topic. Machine learning-based approaches are a promising alternative, allowing the identification of abnormal behaviors and the detection of attacks. This paper proposes an anomaly-based detection solution that uses unsupervised deep learning techniques to identify IoT botnet activities. An empirical evaluation of the proposed method is conducted on both balanced and unbalanced datasets to assess its threat detection capability. False-positive rate reduction and its impact on the detection system are also analyzed. Furthermore, a comparison with other unsupervised learning approaches is included. The experimental results reveal the performance of the proposed detection method.

Keywords: Internet of Things; botnet detection; deep learning; autoencoder; cybersecurity



Citation: Apostol, I.; Preda, M.; Nila, C.; Bica, I. IoT Botnet Anomaly Detection Using Unsupervised Deep Learning. *Electronics* **2021**, *10*, 1876. <https://doi.org/10.3390/electronics10161876>

Academic Editors: Emanuele Bellini, Fiammetta Marulli, Stefano Marrone and Flavio Canavero

Received: 8 June 2021

Accepted: 2 August 2021

Published: 4 August 2021

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



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The Internet of Things (IoT) has seen strong growth in recent years through widespread implementation in a plethora of sectors and industries, such as smart cities and smart homes and agriculture, transport, logistics, healthcare, and military industries, among others. Thus, our world and society are being definitively and continuously shaped by the new dimension of connected things added by the IoT [1]. Furthermore, along with the process of integrating the IoT into this new world, new technologies and research questions arise in terms of opportunities and challenges [2]. Concepts such as Supply 4.0 in the logistic sector [3], Smart Mobility in the transport sector [4], or the Internet of Battlefield Things in the military sector [5] are merely some examples of the massive proliferation of the IoT and created effects.

Moreover, integrating IoT technology with other emerging technologies such as Artificial Intelligence (using AI algorithms for data processing and analysis), Big Data (managing a huge amount of data from IoT sensors), or 5G communications (mobility and broadband communications for IoT sensors) to achieve intelligent integrated systems is the main concern of companies operating in this market sector [6]. In particular, special attention is paid to the integration of AI with IoT, a technology called the Artificial Intelligence of Things (AIoT) [7], which aims to process the data and make automatic and autonomous decisions on IoT networks, without involving the action of the human operator.

However, as the world becomes more interconnected through the Internet, along with the rapid advent of the IoT ecosystem in terms of its size, applicability, connectivity,

and complexity, the requirement to effectively secure IoT components from physical to user levels is also becoming more critical than ever. Stakeholders will be unlikely to adopt IoT technologies on a large scale unless security and privacy are provided for their networks and data [2]. The IoT has turned into a serious security concern, since many IoT systems are designed primarily for functionality, with security not being the focus. Recent cybersecurity reports [8] have confirmed that attacks on IoT environments have gained momentum because of the increasing attack surface, from the edge to the cloud of the IoT ecosystem [9]. Thus, designing and developing secure IoT systems represents a major ongoing challenge for engineers working in this field [10].

One of the major threats when we refer to IoT networks and devices is compromising IoT devices and enrolling them into IoT botnets controlled by the attackers. According to the findings presented in a Distributed Denial of Service (DDoS) report for the first quarter of 2021 [11], well-known IoT botnets such as Bashlite and Mirai remain serious DDoS threats, with hundreds of active command and control (C2) servers globally. Mitigating such threats could be very challenging because of the IoT's main features which need to be considered: heterogeneity, scalability, and limited resources (power, memory, and processor). Therefore, designing solutions that are able to detect abnormal behaviors and attacks for IoT environments has become a mainstream challenge in the IoT cybersecurity field and a hot topic for researchers.

Traditional signature-based Intrusion Detection Systems (IDS) are not efficient when applied to the IoT due to the limited computing and storage capabilities of IoT devices, the specific communication protocols used, and the high volume of data. Therefore, new advanced machine learning (ML)-based solutions are considered to be more suitable to detect and mitigate the effects of cyber-attacks and possible threats to IoT data and infrastructure [10]. These solutions imply mostly anomaly-based detection systems, which depend on classifying the network activity into normal and abnormal patterns. Such systems are more suitable for IoT environments than signature-based approaches as they do not need to check through hundreds of entries of a database for signature matching.

Our main contribution, as presented in this paper, is the proposal of an intrusion detection solution that is able to identify botnet-specific anomalies in IoT network traffic based on deep autoencoders. Using autoencoders for anomaly detection represents a current research topic, but its application is still in the early stages. Compared with other approaches [12], instead of profiling each device through a deep autoencoder, we decided to design our solution in a centralized manner for several reasons. Firstly, it is acknowledged that deep autoencoder training requires more resources than conventional autoencoders. Consequently, using a decentralized detection architecture will significantly increase the overall network resources consumption, given the fact that the number of autoencoders is determined by the number of the connected devices. Using a hybrid placement strategy might be more suitable for node-level anomaly detection profiling in IoT networks. Secondly, applying machine learning models for each device or node in an IoT environment, can elevate the network's complexity with regard to both implementation and operation (i.e., a separate model should be maintained and periodically retrained for each device).

All things considered, we conclude that the implementation of a deep learning solution at the gateway-level would be more appropriate for IoT networks. A deep autoencoder with a limited number of layers can be used at the gateway level of an IoT network without affecting its operation. However, we must note that, by using a centralized architecture, we accept the compromise of losing visibility of internal threats.

The dataset used in our experiments (Bot-IoT) was obtained by capturing the traffic between different simulated IoT devices and services (cloud) [13]. In practice, all this traffic passes through the IoT gateway, making the dataset suitable for testing gateway-level solutions.

The subsequent contributions of our work consist of evaluating various IoT traffic datasets and selecting a proper set that would not contain known botnet threats and would

not involve the high amounts of computing resources caused by a large amount of data content; designing, tuning, and testing a deep autoencoder for anomaly detection in IoT networks that could be applied on both balanced and unbalanced datasets; and improving the false-positive rate of such a solution and studying its impact in the case of both balanced and unbalanced datasets.

The rest of the paper is organized as follows. In Section 2, we present the related work on IoT botnet anomaly detection using machine learning (ML). In Section 3, we present a brief overview of the artificial neural network elements, focusing on autoencoders, which are used further in our work. In Section 4, we describe the methodology used to develop, implement, and evaluate our autoencoder-based IoT anomaly detector. We present and analyze the obtained results, along with a comparison with another approach, in Section 5, and present conclusions in Section 6.

2. Related Work

Various papers are available regarding the applicability of machine learning and Artificial Neural Network algorithms for intrusion detection in IoT ecosystems [12–29], but research in this direction is still in its infancy and requires further study and improvements. Next, we briefly present and discuss the most relevant of these papers for our work.

An interesting approach is proposed in [19], in which Hamza et al. present a solution for the multi-stage detection of attacks targeting IoT environments by using the MUD IoT device profiling tool provided by the IETF combined with machine learning techniques. The main focus of their research is on identifying the volumetric attack flows (e.g., DDoS or flooding attacks) which are not detected by the specification-based intrusion detector implemented with the help of the Software-Defined Network (SDN) controller. Furthermore, each flow is sent to an ML engine that decides if the embedded data is legitimate or not. The solution was tested on a small-dimensioned lab-setup IoT environment, and the results obtained were reasonable, but as the authors mention in their paper, the detection rate and performance parameters can be improved significantly if better tools are used for implementation (e.g., scikit-learn instead of R and Weka, as they store the training data).

A solution for classifying IoT attacks using neural networks is proposed in [16]. The authors developed, tested, and validated a system that uses Convolutional Neural Networks to detect and classify IoT attacks inside the Network Security Laboratory-Knowledge Discovery Databases (NSL-KDD) dataset.

A recent work in which Deep Learning (DL) techniques are used for IoT network intrusion detection is presented in [18], in which Ge et al. propose a model based on Feed-Forward Neural Networks (FNN) for the binary and multi-class classification of various attacks against IoT devices. As a testbed for their work, the authors used the Bot-IoT dataset [30], which was also chosen for our work. Although the proposed solution registered overall satisfying results in terms of accuracy, precision, and recall, the authors also pointed out some difficulties encountered in their evaluation and therefore some possible improvements of their solution.

The detection of IoT botnets using DL approaches is also enhanced in [15] by using a Bidirectional Long Short-Term Memory based Recurrent Neural Network (BLSTM-RNN) in conjunction with Word Embedding (WE) techniques. The authors implemented and evaluated the proposed detection model on a small-sized dataset from a lab setup network comprising IoT surveillance cameras and a Mirai botnet C2 server. From the 10 attack vectors specific to the emulated Mirai botnet, only 4 were used in their experiments, with the other vectors being considered for future research. The data pre-processing is conducted with WE, resulting in integer values of the tokenized strings contained in the captured packets. These are labeled accordingly and input into the BLSTM detection model. Although the evaluated parameters (accuracy and loss) are satisfactory, the research needs further work in terms of the dataset, attack vectors, and evaluated parameters.

An exhaustive work concerning botnet detection using DL is presented in [14]. This research presents 650 different experiments on an 83 GB dataset resulted from overlaying

existing datasets containing both legitimate and Peer-to-Peer (P2P) botnet-specific data traffic. The main objective of this work was to prove if DL can identify known botnets (including Zeus, Storm, Waledac, and ZeroAccess) network activity and substitute classical detection methods using network statistics and feature engineering. According to the authors, their results showed that all the previously mentioned methods can be minimized or substituted by using Deep Neural Networks (DNN), “feeding” them with low-level inputs, such as raw header information extracted from the packet flows. Their study also obtained a good detection accuracy with DNNs and similar results using Stacked Denoising Autoencoders (SDA). Overall, they achieved the best results with a deep feed-forward supervised neural network, registering up to 99.7% accuracy for classifying P2P-botnet traffic.

In [22], Mirsky et al. present an online network intrusion detection solution, called Kitsune, which learns to detect various attacks against the local IoT network without supervision. Kitsune’s main algorithm (KitNET) uses autoencoders to detect anomalies in the network traffic by identifying and distinguishing legitimate traffic from attacks. The proposed IDS framework contains five different modules that are named suggestively after their main functionalities: packet capture, data parser, feature extraction, feature mapping, and anomaly detection. The detection process is based on a set of three-layer autoencoders that learn the normal behavior of their subspaces, and each of them reports its reconstruction error to an output layer. The authors aim to provide a lightweight IDS that could function on a simple router, but with this setup, for some attacks, malicious packets do not reach the router on which the solution runs.

Another approach for detecting anomalies in IoT networks using DL techniques and autoencoders is presented in [12]. The authors propose a solution for detecting botnet attacks by monitoring and analyzing behavioral “snapshots” of the benign traffic from each IoT device. Their empirical evaluation is made on a testbed network of nine commercial IoT devices and the emulation of two known botnets, Mirai and Bashlite. Autoencoders corresponding to each of the IoT devices are used to learn the normal traffic features and to signal when they fail to reconstruct the benign traffic snapshots. This solution, however, may be unpractical for larger networks. The use of a separate autoencoder can make network security difficult to implement and manage, as a different model must be maintained and updated for each IoT device.

The N-BaIoT dataset [12] has been used in several research works concerning IoT botnet-anomaly detection. One of them is represented by [29], where Nomm et al. aimed to demonstrate that they can gain a reasonable detection accuracy with a reduced feature set. This solution will be more suitable for IoT environments with limited resources. In this regard, the authors considered a centralized placement strategy for their IoT botnet IDS and used a three-measurement approach for the selection of the most significant features. The authors focused more on identifying the appropriate technique for feature selection rather than evaluating their IoT botnet detection system. They obtained good results using entropy for feature selection and isolation forests with only five features. The experiments were conducted using two datasets: one balanced and one unbalanced.

The Bot-IoT dataset [31], on which our experiments are based, has also been used in several pieces of research concerning botnet attack detection. Most of them include classification-based approaches, with either binary or multi-class classification. Popoola et al. [32] proposed a detection algorithm based on the Deep Recurrent Neural Network (DRNN) for the 11-class classification of botnet attacks contained in the Bot-IoT dataset, handling the class imbalance problem using the Synthetic Minority Oversampling Technique (SMOTE). The SMOTE-DRNN model [32] achieved high performance with a very low false-positive rate. However, classification-based approaches imply supervised learning techniques that use only labeled data and, although they tend to be more accurate than unsupervised learning models, they require human intervention to appropriately label the data. Our aim is to identify a proper solution for discovering IoT network traffic anomalies without previously labeling the data.

3. Background: Autoencoder Neural Networks

Machine learning (ML) has become a widely used technology for cybersecurity solutions. It aims to “teach” computers to do what people naturally do by the use of specific algorithms.

Artificial Neural Networks (ANN) are computational algorithms used in ML. They simulate the biological nervous system made up of neurons that can receive and transmit signals through synapses. An ANN consists of layers of nodes, with each node behaving like a neuron. The first layer is the input layer, while the last layer is the output layer. Intermediate layers are called hidden layers. If there are at least three hidden layers, the ANN becomes a Deep Neural Network (DNN).

On a layer, each node performs several calculations and passes the resulting output to the nodes of the next layer. Each synapse of a node is represented by an input weight that indicates the impact that the output of the previous neuron has on the current one. Adjusting the weights in a DNN model represents the basis for how Deep Learning (DL) models are trained.

Depending on whether the labels are required or not, there are two techniques used in ML: supervised learning, which implies using labeled datasets, and unsupervised learning, where labeled data are not required in the learning process.

An autoencoder is an unsupervised learning model represented by a neural network that is trained to reconstruct the input to its output. It consists of two components: the encoder and the decoder. As presented in Figure 1, the encoder is applied to the input and the decoder is applied to the encoder’s output (code). Usually, autoencoders are not used to perfectly copy the input to its output through learning but to generate only a resemblance of the training data. They are lossy by design. In this way, they manage to learn important properties related to data.

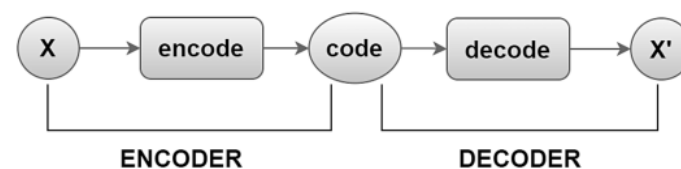


Figure 1. Components of the autoencoder.

When the two components of the autoencoder (the encoder and the decoder) consist of deep networks, it becomes a deep autoencoder. The decoder has a similar structure to the encoder, except that its layers are inverted. Formally, a deep autoencoder includes the following:

1. An input layer: a vector X , represented by the input data of dimension n , $X = (X_1, X_2, \dots, X_n)$;
2. Several hidden layers that represent multiple layers of encoding and decoding, as depicted in Figure 2, generating a non-linear representation of the input data, reconstructing the input to the output layer;
3. An output layer: the vector $X' = (X'_1, X'_2, \dots, X'_n)$. The output has the same size as the input, being the recovered version of the input data;
4. An activation function, along with weights and biases. Each neuron in a layer uses an activation function to calculate its output on a weighted sum of its input.

The activation functions used in DL models can be divided into two categories: linear functions and non-linear functions. The most used activation functions are of the non-linear kind: Sigmoid (logistic), hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU). The Sigmoid function is especially used when predicting the output as a probability, the hyperbolic tangent function is mainly used in two-class classification, and the ReLU function is the most used in almost all deep neural networks. In our work, we used the

ReLU function (2) on each hidden layer of the autoencoder, and we applied the Sigmoid function (1) on the last layer of the decoder.

$$S(x) = \frac{1}{1 + e^{-x}} \quad (1)$$

$$R(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \quad (2)$$

Weights and biases are the learnable parameters of a machine learning model. When the inputs are transmitted between neurons, the weights and biases are applied to the inputs. Weights indicate how much influence the input has on the output, while biases provide assurance that, even when all the inputs are zeros, neuron activation will still occur. Considering layer l in a deep autoencoder, we denote by $W_{ij}^{(l)}$ the weight applied on the connection from node j of layer $l - 1$ to node i of layer l , and $b_i^{(l)}$ is the bias associated with the node. The output value of neuron i from layer l is calculated as follows:

$$O_i^{(l)} = F(\sum (W_{ij}^{(l)} \cdot x_j) + b_i^{(l)}) \quad (3)$$

where F is the activation function and x_j is one input value of the neuron, obtained from the output of node j from layer $l - 1$. The output of the node i from layer l will become an input value for nodes of the next layer.

Generally, autoencoders construct an encoding map of the input data, which is further decoded, obtaining the output layer representing the recovered version of the input layer.

$$X' = D(E(X)) \quad (4)$$

Using autoencoders involves training E and D to minimize the difference between X and X' . The comparison of the produced output X' and the input X (that is expected to be produced) is achieved by a cost function that measures the error between the resulting and the expected values. In order to minimize the error (cost), the model weights are changed during training until a good mapping of inputs to outputs is created. The loss functions specific to the ANN can also be used in deep autoencoders. Often, the Mean Squared Error (MSE) is used. If the input has only binary values, then binary cross-entropy loss is preferred.

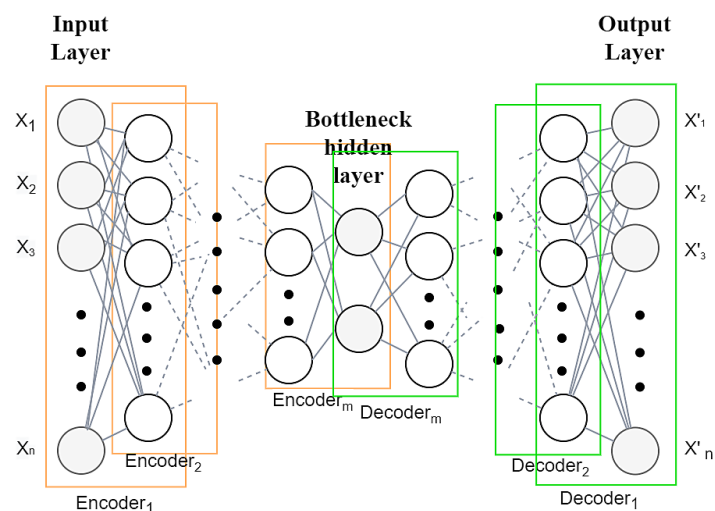


Figure 2. Deep autoencoder.

One of the applications of unsupervised ML is anomaly detection, which involves finding the outliers of a dataset. One popular detection method is represented by dimen-

sionality reduction using Principal Component Analysis (PCA), a statistical technique used for data compression and classification [33]. PCA captures linear correlations between features, but in network anomaly detection, most of the features are non-linearly correlated [34]. Autoencoders represent another approach for anomaly detection that can be used in the case of non-linear interactions and are suitable for applications involving network traffic features.

Applying depth to autoencoders reduces the computational cost of representing functions and improves the learning ability with a reduced amount of training data [35]. These acknowledgments led us to experiment with anomaly detection in IoT networks using a deep autoencoder model.

4. Methodology

In this section, we describe the methodology used to build the anomaly detector. Thus, we present the steps taken to prepare and conduct our experiments, along with the chosen architecture and parameters for the autoencoder.

4.1. Dataset

With the development of AI, datasets have become extremely popular. Since we could not rely on a laboratory setup for suitable and realistic IoT traffic generation, we decided to use one of the datasets that were made available by other researchers in the field.

First, it is important to mention that IoT botnet datasets are not available for public access in a large volume, as with other types of datasets. A. Guerra-Manzanares et al. summarize the available datasets for IoT anomaly-based IDS in [25]. In addition, they also provide their own IoT dataset extracted from a medium-sized network. For our research, we decided to focus on small-sized IoT datasets, thus minimizing the impact of the necessary data processing on our available resources.

N-BaIoT [36] is one of the datasets that we took into consideration for our work, mainly because it is based on real data traffic generated from different types of IoT and non-IoT devices. This dataset includes traffic from nine infected devices [12], but since the attack traffic was particular to Mirai and Bashlite botnets, we decided to address a more relevant dataset for our research, which includes a larger spectrum of threats, instead of some specific botnets.

Consequently, we selected for our experiments the Bot-IoT dataset [30], which is obtained from emulated network traffic between five different IoT devices and additional network devices. The dataset contains both normal and attack traffic traces. Among the attacks, different types of threats are addressed such as probing, DoS, and information theft, which are considered when implementing different botnet scenarios. The total dimension of the generated dataset is approximately 17 GB, but 5% of it was extracted and made available for the training and testing of ML models [13]. The authors of the Bot-IoT dataset also extracted a subset for selecting the “10-best features” of data that were proven to be useful to obtain higher accuracy when training models. For our work, we used this subset consisting of two CSV files: one for training, and the other for testing [31].

Feature extraction helps to improve the accuracy of unsupervised learning models, as reducing the dimension of input data lowers the need for high-power computing. A high amount of data requires more computational resources. S.Nomm and H.Bahsi [29] experimented with the impact of feature selection on the accuracy of the anomaly-based detection systems of IoT botnets and concluded that a reduced feature set would be more suitable for detecting anomalies in IoT networks. These statements enforce our motivation for choosing the dataset available in [31] since the number of generated features is not high.

The “10-best features” subset was extracted to compare the validity of the dataset through three different ML models [13]. It includes features that were selected from the initial dataset through statistical measures in which good scores were obtained for them.

We considered this dataset to be suitable for DL applications in IoT, since it has a small number of features and does not require a high computational capacity.

4.2. Experimental Setup

To implement the anomaly detector, we went through the process presented in Figure 3. All the stages of our work were conducted in Python using the TensorFlow [37] framework with Pandas and scikit-learn libraries. In terms of hardware, our work was conducted on a computer with an Intel Core i7 CPU, 16 GB memory, and an NVIDIA GeForce graphic card.

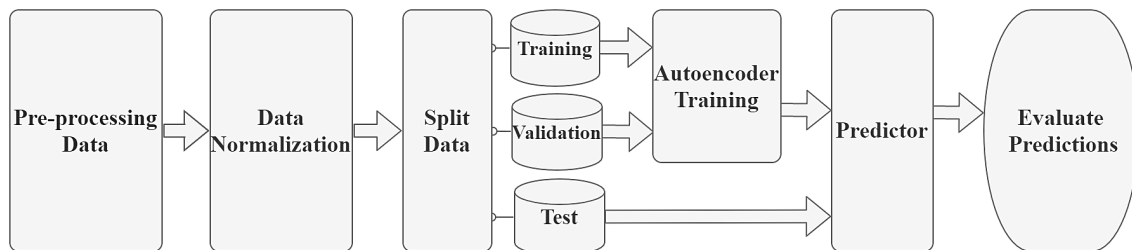


Figure 3. Experimental flow.

First, we analyzed and prepared the data from the selected dataset [30] for the data to be suitable for Python processing. Specifically, we dropped the features that were not necessary for anomaly detection (i.e., attack category and subcategory), and we applied numerical transformations on the values that were not of appropriate data types.

Data normalization was necessary to scale the values as the features obtained in the pre-processing phase had different ranges. We used the MinMaxScaler object from the scikit-learn library to rescale the values into the range [0, 1].

$$X_{sc} = \frac{X - \text{Min}_X}{\text{Max}_X - \text{Min}_X}, \quad (5)$$

where X is the network traffic feature vector, Max_X and Min_X representing the minimum and the maximum values of X .

From the “10-best features” subset of Bot-IoT data, the version available for download was divided into one set for training and one for testing. The training dataset represented 80% of the entire dataset, while the testing set comprised the remaining 20%. From the training dataset, we randomly extracted 20% to be used as validation data for the autoencoder. Therefore, from the entire downloaded data, we used 64% for training, 16% for validation, and 20% for testing. We normalized the values for all these datasets.

To avoid drawing questionable conclusions by using an unbalanced dataset, we complemented our experiments with a more balanced dataset extracted from the initial set. Thus, we tested the autoencoder in two cases:

- Case 1 : anomaly detection on the unbalanced dataset available in [12];
- Case 2: anomaly detection on a balanced dataset obtained from the initial, unbalanced set.

For the second experimental case, along with the anomaly-specific records, we extracted a similar number of records related to normal traffic. The data proportions (training, validation, testing subsets) were kept the same, with each subset having approximately the same amount of normal traffic data as that specific to attacks. In both cases, we used the same parameters to train the autoencoder and we evaluated the predictions on the test dataset.

4.3. The Autoencoder

When used to detect anomalies, an autoencoder is trained only on the normal pattern of a system. Thus, it can reconstruct normal patterns with minimum error. In case of anomalies, the reconstruction error should exceed a certain threshold.

We separated the normal traffic data from the abnormal traffic data and we trained the autoencoder on the normal traffic data only. Furthermore, when training the autoencoder, the full validation dataset was used for evaluation.

It should be noted that the dataset used in the first experimental case contained a considerable percentage of values generated by anomalies compared to those generated by normal traffic. In the second case, the dataset was almost equally composed of normal traffic data and anomaly data. The difference between the normal values and those in the case of anomalies can be observed in Figures 4 and 5.

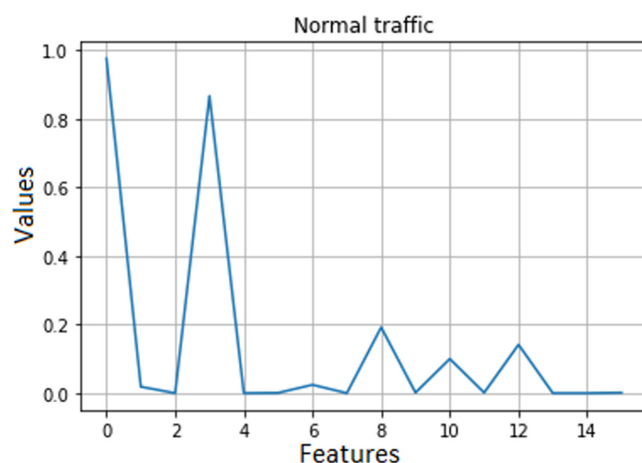


Figure 4. Values for one normal data record from the training dataset.

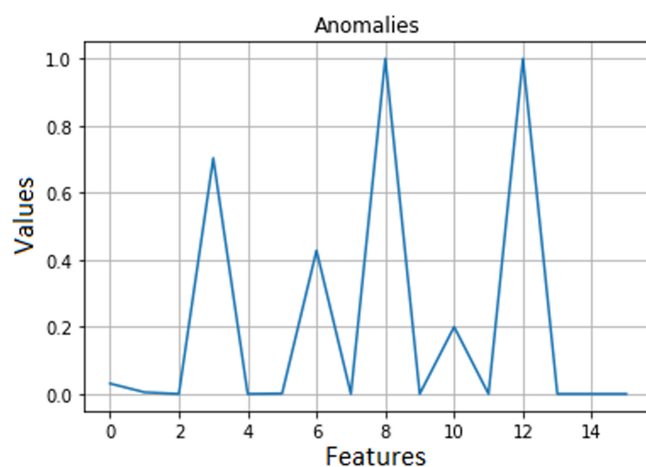


Figure 5. Values for one abnormal data record from the training dataset.

After eliminating the features considered irrelevant for our experiment, the number of inputs for our autoencoder remained 16. Therefore, the number of outputs was also 16.

Since the deep autoencoders used in [12] with hidden layers of decreasing sizes have shown good results, we implemented a similar architecture for our autoencoder. After tuning and testing the autoencoder in terms of the number of hidden layers and their neurons, we obtained good results for training loss when training on the configuration described by Algorithm 1, with 4 decreasing hidden layers for the encoder: the first having 12 neurons, the second having 8 neurons, the third having 5 neurons, and with 2 neurons in the last layer. Consequently, the decoder comprised reflected layers, in ascending order.

Algorithm 1 Autoencoder Model

```

1:  $W_i$ —matrix of weights for layer i
2:  $b_i$ —bias vector for layer i
3: procedure ENCODER( $input_{encoder}$ )
4:    $NumberOfNodes_{H_1} = 12$ 
5:    $H_1 = \text{ReLU}(input_{encoder} \cdot W_1 + b_1)$ 
6:    $NumberOfNodes_{H_2} = 8$ 
7:    $H_2 = \text{ReLU}(H_1 \cdot W_2 + b_2)$ 
8:    $NumberOfNodes_{H_3} = 5$ 
9:    $H_3 = \text{ReLU}(H_2 \cdot W_3 + b_3)$ 
10:   $NumberOfNodes_{H_4} = 2$ 
11:   $H_4 = \text{ReLU}(H_3 \cdot W_4 + b_4)$  return  $H_4$ 
12: end procedure
13: procedure DECODER( $input_{decoder}$ )
14:   $NumberOfNodes_{H_1} = 5$ 
15:   $H_1 = \text{ReLU}(input_{decoder} \cdot W_1 + b_1)$ 
16:   $NumberOfNodes_{H_2} = 8$ 
17:   $H_2 = \text{ReLU}(H_1 \cdot W_2 + b_2)$ 
18:   $NumberOfNodes_{H_3} = 12$ 
19:   $H_3 = \text{ReLU}(H_2 \cdot W_3 + b_3)$ 
20:   $NumberOfNodes_{H_4} = 16$ 
21:   $H_4 = \text{Sigmoid}(H_3 \cdot W_4 + b_4)$  return  $H_4$ 
22: end procedure
23: initialize weights and biases for ENCODER and DECODER
24: run ENCODER (X)
25: run DECODER (ENCODER)

```

In our autoencoder, ReLU was used to compute the outputs for each hidden layer. To make the learning process more stable, since we had previously normalized the input data, the Sigmoid function was applied for the last layer of the decoder.

The weights were initialized by small random numbers, thus providing asymmetry breaking, which allowed the biases to be initialized to zero. Then, weights and biases (θ) were updated for each epoch by the Adaptive Moment Estimation (Adam) [38] to minimize the error (loss function). Adam has the following configuration parameters: the learning rate (α), the exponential decay rate for the first-moment estimates (β_1), the exponential decay rate for the second-moment estimates (β_2), and a very small number that prevents division by zero (ϵ). In Tensorflow, β_1 is 0.9, β_2 is 0.999, and ϵ is 1×10^{-8} . The equations below describe how Adam optimizes the trainable parameters θ : after obtaining the gradient of the parameters (7), the first moment, m_t (8), and the second moment, v_t (9), are estimated, and then the parameters are updated (10).

$$\theta = [W, b] \quad (6)$$

$$g_t = \text{grad}(\theta_{t-1}) \quad (7)$$

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \quad (8)$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \quad (9)$$

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{m_t}{\sqrt{v_t + \epsilon}} \quad (10)$$

The error for each learning state of the model was estimated using the Mean Square Error (MSE) loss function. By applying the MSE on our model, we obtained close to zero values of the training loss parameter.

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^n (X_i - X'_i)^2, \quad (11)$$

where X_i is an input value from the dataset and X'_i is the output value of the deep autoencoder.

After evaluating of the training with different parameters, 20 epochs and a batch size of 19 samples proved to be suitable for the learning process to unfold as it should. Note that 19 is a divisor of the number of samples used for training.

As we previously stated, the autoencoder was trained using only the normal traffic data, but it was evaluated using the full validation subset in each experimental case. This helped to estimate the test error rate. In the first experimental case, the validation subset contained many more anomalies than normal traffic data, so we considered the much bigger difference between training and validation loss to be justifiable. In the second case, as the dataset was a balanced set, the gap between the training loss and the validation loss was not as large. Nevertheless, the decrease of the training loss value to nearly zero compared to the increase of the validation loss indicates the possibility of classifying normal traffic data depending on the loss range.

4.4. The Predictor

The predictor was built after observing the reconstruction error obtained when encoding and decoding normal traffic data, compared with the result obtained in case of anomalies.

Figure 6 displays the reconstruction obtained on the normal traffic data from the validation dataset, while Figure 7 shows the reconstruction performed only on the anomalies within the same dataset.

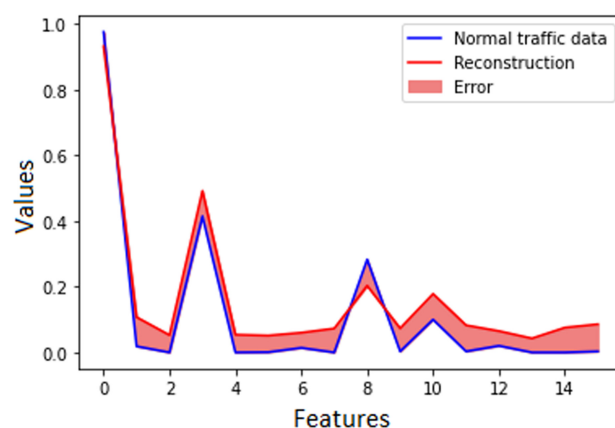


Figure 6. Reconstruction error for values on one line of the normal traffic data validation subset.

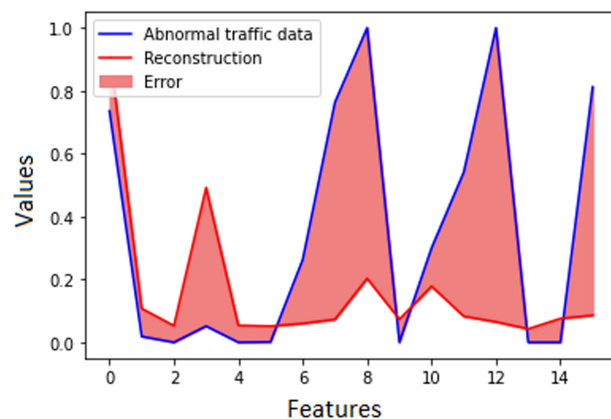


Figure 7. Reconstruction error for values on one line of the abnormal traffic data validation subset.

As the two graphics show, the reconstruction error on normal traffic data was lower than that obtained on anomalies. Therefore, we considered that if the reconstruction loss is higher than a certain threshold, anomalies are predicted.

The loss values of the autoencoder's predictions on normal traffic data were lower than the loss values obtained in abnormal traffic predictions. This can be observed in Figure 8.

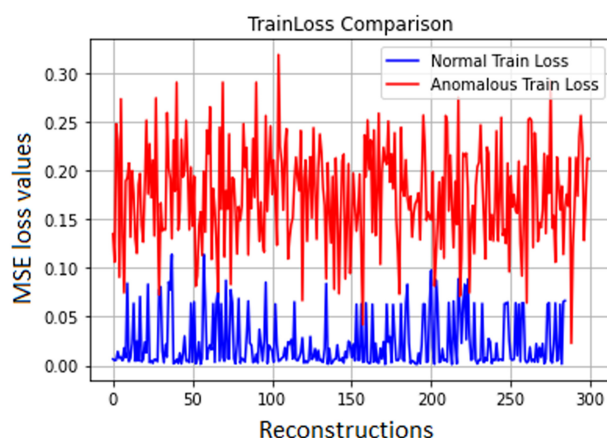


Figure 8. Graphic of the prediction loss values on normal traffic data and abnormal traffic data.

We chose a threshold to be used when predicting whether a particular sample is an anomaly or not. We first defined the threshold as one standard deviation above the mean loss obtained on the normal traffic data from the training dataset.

$$th = mean_{NrmTrLoss} + std_{NrmTrLoss} \quad (12)$$

Figure 9 shows how the threshold separates the anomalies from the normal traffic. We obtained good results when observing the predictions on the unbalanced dataset. However, after testing the same model on the balanced dataset, the need to increase the threshold became obvious because of the number of false anomalies in the predictions. We then changed the threshold value, as depicted in Equation (13), to compare the two cases.

$$th = mean_{NrmTrLoss} + 3 \cdot std_{NrmTrLoss} \quad (13)$$

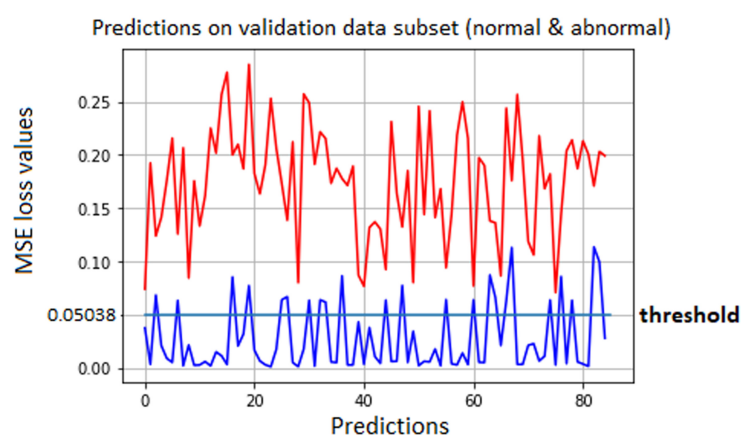


Figure 9. The threshold on the graphic of prediction loss values of both normal and abnormal traffic data.

Finally, the predictor was defined by the Algorithm 2:

Algorithm 2 The Predictor

```

1: procedure PREDICTOR(autoencoder,data,threshold)
2:   reconstructions = autoencoder.predict(data)
3:   loss = mse(reconstructions,data)
4:   if loss > threshold then
5:     return 1
6:   else
7:     return 0
8:   end if
9: end procedure

```

5. Results

In this section, we present and discuss the results obtained using an autoencoder to detect IoT botnet-specific attacks. First, we describe how the optimal network configuration and hyper-parameters were obtained. Next, we discuss the predictions of the model on the test dataset. Finally, we compare our results with other similar research that used the Bot-IoT dataset.

5.1. Training the Autoencoder

To obtain good results in training the autoencoder, we performed several experiments, varying the network configuration and the hyper-parameters. With the configuration described in the previous section, we obtained a training loss of about 0.02, both in Case 1 and in Case 2.

Regarding the autoencoder's architecture, we first performed tests on an encoder represented by three hidden layers with 12, 8, and 4 neurons, respectively. Then, we added one more hidden layer and tested it in various configurations. These included 12–8–4–2 neurons, 12–9–4–2 neurons, 12–8–5–3 neurons, and 12–8–5–2 neurons. We also experimented on a network with more hidden layers, but the results did not exceed those obtained with configuration 12–8–5–2. Each test, the decoder was the inverse of the encoder.

For Adam, we obtained good results using a learning rate of 0.01.

Twenty epochs were sufficient to obtain low values of training loss in both experimental cases. We also tested the model with a larger number of epochs, but the results of the training were not significantly improved. For example, when training the autoencoder with 30 epochs, we observed the training loss to decrease well at the beginning of the training, but after a few epochs, the training loss showed slight decreases to a value close to that obtained with 20 epochs. In the experiments performed with more epochs, training convergence was observed after 18–20 epochs.

We also tried a smaller number of epochs for training the autoencoder, but, although the training results were satisfactory in Case 1 using only 10 epochs, for Case 2, training with 20 epochs gave better results than training with 10 epochs.

In Case 1, training the autoencoder with validation showed the difference in the amount of traffic between the normal and the abnormal data subset. This is the reason why, when fitting the model in the first case, the validation loss values fell within a higher range (Figure 10). In the second experimental case, as expected, working with a balanced dataset lowered the interval of the validation loss (Figure 11).

Still, the training loss and the validation loss were expected to have a different evolution during the training process, since this was performed on normal traffic data only. This method brings with it the possibility of establishing a threshold that separates normal traffic data from the anomalies. This was achieved in both experimental cases, as the autoencoder's anomaly prediction error proved to be predominantly above the prediction error that resulted in normal traffic. Working with a more balanced dataset would have been likely to be more conclusive, but all the available datasets studied contain mostly attack-specific data.

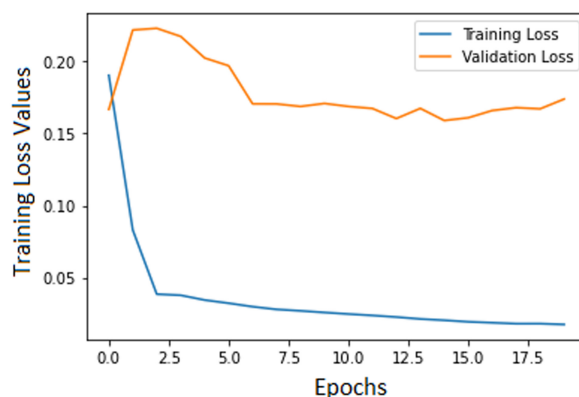


Figure 10. The difference between training and validation loss in Case 1.

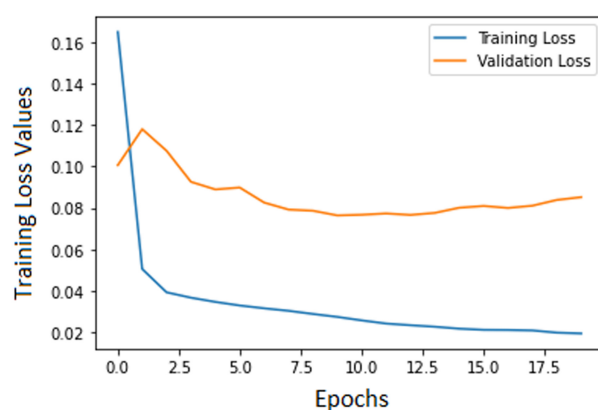


Figure 11. The difference between training and validation loss in Case 2.

5.2. Predictions

Table 1 summarizes the results of our experiments in both dataset cases. Initially, we used a threshold equal to one standard deviation above the mean loss of the autoencoder's predictions on the normal data. Then, we raised the threshold to three standard deviations above the mean loss.

Table 1. Experimental Results.

		Precision	Recall	FP(%)	FN(%)
Initial Threshold	Case 1	0.999	0.997	8.41	0.21
	Case 2	0.916	1	10.3	0
Initial Threshold	Case 1	0.999	0.967	1.8	3.26
	Case 2	0.948	1	5.94	0

In Case 1, the outputs generated by the predictor on the test data were evaluated against the test labels with a precision score of 0.99 and a recall score of 0.997.

The confusion matrix showed good results in terms of attack identification. Less than 0.5% of the attacks were missed by the predictor, while over 99% were correctly predicted as attacks. Therefore, the false-negative rate was very low.

However, the false-positive rate was higher: around 8.5% of the reported as true predictions were actually false.

In Case 2, on a balanced dataset and with the threshold set at the initial value, the predictor was evaluated as having lower precision than in the first case, but the incorrect predictions were only false attacks. Even though the training was conducted on a small

amount of normal traffic data, the precision score obtained with balanced test data was 0.916, while the recall was 1, because none of the attacks escaped detection.

The confusion matrix in this case, with a lower threshold, confirmed the no false-negative rate, but the false-positive rate was about 10%.

Since the autoencoder was trained on the same amount of normal traffic data in both cases, running the predictor from Case 2 on a dataset with mostly anomaly-specific data, such as the test dataset from Case 1, produced similar results to Case 1: a precision score of 0.99 and recall score of 0.997. Therefore, using the initial threshold proves to be effective when there is a large amount of anomaly-specific data in the input.

To reduce the prediction errors concerning the false anomalies, we increased the threshold. This resulted in improvements concerning the false-positive rate in Case 2 but reduced the performance of the model in Case 1. While in Case 2 the false-negative rate was maintained at zero and the false-positive rate decreased to approximately 6%, in Case 1, a greater value of the threshold led to more attacks being missed.

It is known that using autoencoders in anomaly detection may produce some false-positive results, and this was also observed in our experiments. The false-positive rate can be improved by setting the threshold to a higher level, but this change may become risky in situations where the anomaly detector faces many attacks during the analysis.

As a result, we conclude that a compromise in accepting a greater number of false-positive predictions must be considered, rather than risking more false-negative outputs being obtained. When it comes to threat defense approaches, missing attacks may be more harmful than over-reporting attacks.

Broadly speaking, anomaly-based systems tend to be very sensitive when it comes to the normal behavior of the network and, if there is a slight change in its normal operation, the system may consider it suspicious. This type of solution usually implies assistance in finally establishing which reported events should be treated as real threats and which should be ignored because they are false or harmless.

5.3. Comparison Benchmarks

In order to investigate the effectiveness of our intrusion detection solution, we compared our model to the previous approaches on botnet attack detection in IoT networks. The Bot-IoT dataset has been used before to evaluate the performance of several other proposed solutions. Alkadi et al. [39] also used the Bot-IoT dataset to evaluate their proposed anomaly detection system of Mixture Localization-Based Outliers (MLO) based on Gaussian Mixture Models (GMM) for fitting network data and a Local Outlier Factor (LOF) function for establishing the threshold. Gaussian Mixture models represent a form of unsupervised learning. In [39], the authors used different sample sizes in their experiments, without the need for traffic balancing. Therefore, we compared the performance of our model experiments in Case 1 with the average of the results depicted in [39], in terms of the detection rate (DR) and false-positive rate (FPR):

$$DR(Recall) = \frac{TP}{TP + FN} \quad (14)$$

$$FPR = \frac{FP}{TN + FP} \quad (15)$$

As can be observed in Table 2, both models show good performance in attack detection. When used with an increased threshold, the proposed deep autoencoder method achieves a slightly higher detection rate and a better false-positive rate than GMM. If configured with the initial lower threshold, the detection rate of our model outperforms the GMM, but the number of false alarms becomes higher, meaning that, in this case, the model flags normal traces as attacks to a greater extent than the GMM approach does.

Table 2. Performance evaluation of deep autoencoder method and GMM approach.

Method	DR	FPR(%)
GMM with LFO 4	96.57	3.85
Deep autoencoder with increased threshold	96.7	1.8
Deep autoencoder with initial threshold	99.7	8.41

Moreover, a downside of the GMM method is the fact that it requires an increased sample size for better performance. Our proposed deep autoencoder achieved good results, although the training was conducted on a small amount of normal traffic data.

6. Conclusions

Protecting against threats to an IoT network can be very challenging, especially regarding botnets, whose attacks may be quite unpredictable. A network IDS enables the detection of and response to malicious activities. Signature-based intrusion detection systems represent a common type of IDS that has proven to be effective in protecting against known threats, but in cases of unknown threats or new releases of known threats, anomaly-based solutions are preferred. Moreover, signature-based systems can be resource-consuming and therefore are not indicated in IoT environments.

This paper proposed an IoT botnet anomaly-detection solution based on a deep autoencoder model. Experimental results showed that the proposed method achieves values of 99.7 for accuracy, 0.99 for precision, and 0.99 for recall. We also analyzed an improvement of the model concerning the false-positive rate by increasing the threshold. The experiments illustrated that the proposed solution is effective and robust in detecting botnet attacks in IoT networks.

Anomaly-based IDS are known to have the disadvantage of resulting in more false-positives, and this was also highlighted in our experiments. Trying to remediate this issue proved to affect the false-negative rate of the detector, in case of a large number of anomalies in the analyzed dataset.

As long as it is not too high, the false-positive rate of anomaly-based detection systems must be accepted, since they can report even the latest threats without omission. Misdetecting an attack is more dangerous than classifying a slight change in normal behavior as an anomaly.

In addition to false alarm reporting, possible limitations of the proposed solution also include vulnerabilities to adversarial machine learning techniques and missing internal attacks that are not visible to the gateway level in IoT networks.

After proving that the detection of anomalies using DL techniques can be a solution for protecting IoT networks against botnet attacks, our future work will focus on identifying and testing the most appropriate solution of this kind, based on a more consistent training dataset. Furthermore, our research will generate better results on more balanced datasets, without significantly affecting its ability to operate in the case of anomaly-preponderant data. Moreover, we aim to complete this solution by adding a classifier based on supervised learning to further label the identified anomalies.

Author Contributions: Conceptualization, I.B., I.A. and M.P.; Methodology, I.A. and C.N.; Software: I.A.; Validation, M.P. and I.A.; Writing, I.A., M.P. and C.N.; Supervision, I.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a grant from the Romanian Ministry of Research and Innovation, CCCDI-UEFISCDI, project number PN-III-P1-1.2-PCCDI-2017-0272/Avant-garde Technology Hub for Advanced Security (ATLAS), within PNCDI III.

Data Availability Statement: The data that support the finding of this study are openly available at: <https://cloudstor.aarnet.edu.au/plus/s/umT99TnxvbpkkoE>.

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

References

1. Sundmaeker, H.; Guillemin, P.; Friess, P.; Woelfflé, S. Vision and challenges for realising the Internet of Things. *Clust. Eur. Res. Proj. Internet Things Eur. Commission* **2010**, *3*, 34–36.
2. Azmat, M.; Kummer, S.; Moura, L.T.; Gennaro, F.D.; Moser, R. Future Outlook of Highway Operations with Implementation of Innovative Technologies Like AV, CV, IoT and Big Data. *Logistics* **2019**, *3*, 15. [\[CrossRef\]](#)
3. Ahmed, S.; Kalsoom, T.; Ramzan, N.; Pervez, Z.; Azmat, M.; Zeb, B.; Rehman, M.U. Towards Supply Chain Visibility Using Internet of Things: A Dyadic Analysis Review. *Sensors* **2021**, *21*, 4158. [\[CrossRef\]](#) [\[PubMed\]](#)
4. Vaidian, I.; Azmat, M.; Kummer, S. Impact of Internet of Things on Urban Mobility. 2019. Available online: www.innovationarabia.ae/wp-content/uploads/2020/10/IA-12-Proceedings-Health-and-Environment.pdf#page=4 (accessed on 7 June 2021)
5. Kott, A.; Swami, A.; West, B.J. The Internet of Battle Things. *Computer* **2016**, *49*, 70–75. [\[CrossRef\]](#)
6. GlobalData, Aerospace and Defence T.R. Internet of Military Things. In Technical Report GDDEF-TR-S007. 2019. Available online: www.army-technology.com/wp-content/uploads/sites/3/2019/12/thematic2-researchinternet-of-military-things-in-aerospace-defense-1.pdf (accessed on 7 June 2021)
7. Ghosh, A.; Chakraborty, D.; Law, A. Artificial intelligence in Internet of things. *CAAI Trans. Intell. Technol.* **2018**, *3*, 208–218. [\[CrossRef\]](#)
8. Mukhopadhyay, S.C.; Suryadevara, N.K. Internet of Things: Challenges and Opportunities. In *Internet of Things*; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 1–17. [\[CrossRef\]](#)
9. Kaspersky. Kaspersky Security Bulletin 2020–2021. EU Statistics. Available online: www.securelist.com/kaspersky-security-bulletin-2020-2021-eu-statistics/102335/ (accessed on 27 May 2021).
10. Hussain, F.; Hussain, R.; Hassan, S.A.; Hossain, E. Machine Learning in IoT Security: Current Solutions and Future Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1686–1721. [\[CrossRef\]](#)
11. Lumen Technologies, Lumen Quarterly DDoS Report. Available online: <https://assets.lumen.com/is/content/Lumen/lumen-quarterly-ddos-report-q1-2021?Creativeid=b3ce01a2-b770-42b6-9567-8e0496e51182> (accessed on 11 May 2021).
12. Meidan, Y.; Bohadana, M.; Mathov, Y.; Mirsky, Y.; Shabtai, A.; Breitenbacher, D.; Elovici, Y. N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders. *IEEE Pervasive Comput.* **2018**, *17*, 12–22. [\[CrossRef\]](#)
13. Koroniotis, N.; Moustafa, N.; Sitnikova, E.; Turnbull, B. Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. *Future Gener. Comput. Syst.* **2019**, *100*, 779–796. [\[CrossRef\]](#)
14. Van Roosmalen, J.; Vranken, H.; van Eekelen, M. Applying deep learning on packet flows for botnet detection. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, Pau, France, 9–13 April 2018; pp. 1629–1636.
15. McDermott, C.D.; Majdani, F.; Petrovski, A.V. Botnet detection in the internet of things using deep learning approaches. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018; IEEE: Piscataway, NJ, US, 2018; pp. 1–8.
16. Abu Al-Haija, Q.; Zein-Sabatto, S. An Efficient Deep-Learning-Based Detection and Classification System for Cyber-Attacks in IoT Communication Networks. *Electronics* **2020**, *9*, 2152. [\[CrossRef\]](#)
17. Sivanathan, A.; Gharakheili, H.H.; Loi, F.; Radford, A.; Wijenayake, C.; Vishwanath, A.; Sivaraman, V. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Trans. Mob. Comput.* **2018**, *18*, 1745–1759. [\[CrossRef\]](#)
18. Ge, M.; Fu, X.; Syed, N.; Baig, Z.; Teo, G.; Robles-Kelly, A. Deep learning-based intrusion detection for IoT networks. In Proceedings of the 2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC), Kyoto, Japan, 1–3 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 256–25609.
19. Hamza, A.; Gharakheili, H.H.; Benson, T.A.; Sivaraman, V. Detecting volumetric attacks on IoT devices via sdn-based monitoring of mud activity. In Proceedings of the 2019 ACM Symposium on SDN Research, San Jose, CA, USA, 3–4 April 2019; pp. 36–48.
20. Sivanathan, A.; Gharakheili, H.H.; Sivaraman, V. Inferring IoT device types from network behavior using unsupervised clustering. In Proceedings of the 2019 IEEE 44th Conference on Local Computer Networks (LCN), Osnabrueck, Germany, 14–17 October 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 230–233.
21. Zewdie, T.G.; Girma, A. IoT Security and the Role of AI/ML to Combat Emerging Cyber Threats in Cloud Computing Environment. *Issues Inf. Syst.* **2020**, *21*, 253–256. [\[CrossRef\]](#)
22. Mirsky, Y.; Doitshman, T.; Elovici, Y.; Shabtai, A. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In Proceedings of the 2018 Network and Distributed System Security Symposium. Internet Society, San Diego, CA, USA, 18–21 February 2018; [\[CrossRef\]](#)
23. Soe, Y.N.; Feng, Y.; Santosa, P.I.; Hartanto, R.; Sakurai, K. Machine Learning-Based IoT-Botnet Attack Detection with Sequential Architecture. *Sensors* **2020**, *20*, 4372. [\[CrossRef\]](#) [\[PubMed\]](#)
24. Sivanathan, A.; Gharakheili, H.H.; Sivaraman, V. Managing IoT Cyber-Security Using Programmable Telemetry and Machine Learning. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 60–74. [\[CrossRef\]](#)
25. Guerra-Manzanares, A.; Medina-Galindo, J.; Bahsi, H.; Nömm, S. MedBIoT: Generation of an IoT Botnet Dataset in a Medium-sized IoT Network. In Proceedings of the 6th International Conference on Information Systems Security and Privacy. SCITEPRESS—Science and Technology Publications, Valletta, Malta, 25–27 February 2020; [\[CrossRef\]](#)
26. Chaabouni, N.; Mosbah, M.; Zemari, A.; Sauvignac, C.; Faruki, P. Network Intrusion Detection for IoT Security Based on Learning Techniques. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 2671–2701. [\[CrossRef\]](#)

27. Gerka, A. Searching for optimal machine learning algorithm for network traffic classification in intrusion detection system. *ITM Web Conf.* **2018**, *21*, 00027. [CrossRef]
28. Lee, S.; Kim, S.J.; Lee, J.; hee Roh, B. Supervised Learning-Based Fast, Stealthy, and Active NAT Device Identification Using Port Response Patterns. *Symmetry* **2020**, *12*, 1444. [CrossRef]
29. Nomm, S.; Bahsi, H. Unsupervised Anomaly Based Botnet Detection in IoT Networks. In Proceedings of the 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, USA, 17–20 December 2018; IEEE: Piscataway, NJ, USA, 2018; [CrossRef]
30. Koroniotis, N.; Moustafa, N. The Bot-IoT Dataset. Available online: <https://research.unsw.edu.au/projects/bot-iot-dataset> (accessed on 2 June 2021).
31. CloudStor. Bot-IoT Dataset Download Link. Available online: <https://cloudstor.aarnet.edu.au/plus/s/umT99TnxvbpkkoE> (accessed on 7 April 2021).
32. Popoola, S.I.; Adebisi, B.; Ande, R.; Hammoudeh, M.; Anoh, K.; Atayero, A.A. SMOTE-DRNN: A Deep Learning Algorithm for Botnet Detection in the Internet-of-Things Networks. *Sensors* **2021**, *21*, 2985. [CrossRef] [PubMed]
33. Fernandes, G.; Rodrigues, J.J.; Proença, M.L. Autonomous profile-based anomaly detection system using principal component analysis and flow analysis. *Appl. Soft Comput.* **2015**, *34*, 513–525. [CrossRef]
34. Chen, Z.; Yeo, C.K.; Lee, B.S.; Lau, C.T. Autoencoder-based network anomaly detection. In Proceedings of the 2018 Wireless Telecommunications Symposium (WTS), Phoenix, AZ, USA, 17–20 April 2018; IEEE: Piscataway, NJ, USA, 2018; [CrossRef]
35. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
36. Naveed, K. N-BaIoT Dataset to Detect IoT Botnet Attacks. Available online: www.kaggle.com/mkashifn/nbaiot-dataset (accessed on 7 April 2021).
37. Core, T. Tensorflow Guide. Available online: <https://www.tensorflow.org/guide> (accessed on 7 June 2021).
38. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
39. AlKadi, O.; Moustafa, N.; Turnbull, B.; Choo, K.K.R. Mixture Localization-Based Outliers Models for securing Data Migration in Cloud Centers. *IEEE Access* **2019**, *7*, 114607–114618. [CrossRef]