electronics

Article

Deep Learning Model Transposition for Network Intrusion Detection Systems

João Figueiredo [1,*], Carlos Serrão [1,*] and Ana Maria de Almeida [2,*]

1 Information Sciences, Technologies and Architecture Research Center (ISTAR), Instituto Universitário de Lisboa (ISCTE-IUL), 1600-189 Lisboa, Portugal
2 CISUC—Center for Informatics and Systems of the University of Coimbra, 3004-531 Coimbra, Portugal
* Correspondence: jpmpf@iscte-iul.pt (J.F.); carlos.serrao@iscte-iul.pt (C.S.); ana.almeida@iscte-iul.pt (A.M.d.A.)

Abstract: Companies seek to promote a swift digitalization of their business processes and new disruptive features to gain an advantage over their competitors. This often results in a wider attack surface that may be exposed to exploitation from adversaries. As budgets are thin, one of the most popular security solutions CISOs choose to invest in is Network-based Intrusion Detection Systems (NIDS). As anomaly-based NIDS work over a baseline of normal and expected activity, one of the key areas of development is the training of deep learning classification models robust enough so that, given a different network context, the system is still capable of high rate accuracy for intrusion detection. In this study, we propose an anomaly-based NIDS using a deep learning stacked-LSTM model with a novel pre-processing technique that gives it context-free features and outperforms most related works, obtaining over 99% accuracy over the CICIDS2017 dataset. This system can also be applied to different environments without losing its accuracy due to its basis on context-free features. Moreover, using synthetic network attacks, it has been shown that this NIDS approach can detect specific categories of attacks.

Keywords: network intrusion detection system (NIDS); intrusion detection; anomaly detection; deep learning (DL); long short-term memory (LSTM)

check for updates

Citation: Figueiredo, J.; Serrão, C.; de Almeida, A.M. Deep Learning Model Transposition for Network Intrusion Detection Systems. *Electronics* **2023**, *12*, 293. https://doi.org/10.3390/electronics12020293

Academic Editor: Harald Vranken

Received: 30 November 2022
Revised: 2 January 2023
Accepted: 3 January 2023
Published: 6 January 2023

1. Introduction

Companies sometimes overlook security in the design of their products and services in the name of faster releases in the production of their new features, which results in a wider attack surface exposed to adversaries. Consequently, and thanks to new regulatory and compliance requirements, legal obligations and governmental pressure, more and more companies are starting to invest in cybersecurity [1]. Intrusion Detection and Prevention Systems are one of such investments. An Intrusion Detection System (IDS) [2] is a security system that actively monitors either hosts (as a Host Intrusion Detection System, HIDS) or networks (as a Network Intrusion Detection System, NIDS) and detects malicious activity. Typically, they use two detection methods: signature-based and anomaly-based [3].

The signature-based detection method is the most common detection method used by IDSs. This detection approach searches for Indicators of Compromise (IOC) in the host or the network. They are as efficient as the signatures they can recognize and need to be constantly updated to protect companies against intrusions performed via new attacks.

On the other hand, an anomaly-based detection method is based on heuristics and rules. Instead of looking for specific signatures or patterns, it starts by measuring and defining a baseline of expected activity in the host or network. After establishing the baseline, it looks for outliers. Since anomaly-based IDSs are trained to recognize normal activity and behaviour of their respective hosts or networks, they don't need to be constantly updated to fulfil their purpose. This makes them especially relevant in detecting intrusions performed by adversaries that might be exploiting unknown or new (0-day) vulnerabilities or in legacy, outdated or non-updated systems that lack the prevention capabilities to defend them against known vulnerabilities. However, they're plagued by the high number of false-positive detection rates [4].

Regarding vulnerabilities, there has been a shift in how cyber-attackers attack and compromise systems. Some time ago, the major danger originated from unpatched vulnerabilities. Today, main threats arise from 0-day vulnerabilities exploitation [5]. This is important in the context of this work because organizations need to have intelligent ways to detect and deter unknown attacks.

In the early days, most anomaly-based NIDS would work using a strict mathematical model and flag any deviation as an attack [3]. With the recent availability of new and better machine learning models, such as artificial neural networks, more and more anomaly-based IDS solutions started to adopt machine learning as their classification mechanisms [6–8].

In this study, we build an anomaly-based deep learning network intrusion detection system (DL NIDS), with the objective that given a robust pre-processed dataset and context-free features, the DL NIDS can be used in different environments without losing its accuracy for intrusion detection. Our key contributions are:

1. A novel pre-processing technique that improves the detection performance of an anomaly-based deep learning network intrusion detection system in a real-world network environment;
2. A stacked Long Short-Term Memory (LSTM) model configuration that, when paired with our pre-processing technique, can, according to our background study, achieve the highest recorded accuracy on the CICIDS2017 dataset;
3. An analysis, via a case study, of the impact of implementing this anomaly-based DL NIDS in a real-world network environment.

2. Related Work

In this section, we present a summary of a review of relevant literature that was performed using the terms: "intrusion detection", "network intrusion detection", "anomaly detection", "deep learning", and "long short-term memory".

2.1. Machine Learning, Deep Learning and Intrusion Detection

Since the inception of the first IDS in 1980 by James P. Anderson [2] for the National Security Agency (NSA), a great deal of research has been used on its evolution to match the current reality in terms of network infrastructures. This first model strictly used signature-based detection and was designed to help system administrators review audit trails. In 1986, Dorothy E. Denning et al. [3] proposed an evolution of James P. Anderson's IDS model using both signature-based detection and anomaly-based detection methods. The authors proposed several statistical approaches for anomaly detection, but most implementations used a Markov Chain model. A few years later, in 1992, and with the increasing prominence of malware usage on information systems, Wang et al. [6] proposed an anomaly-based detection method for IDS using a one-class Support Vector Machine (SVM) in conjunction with a Markov Chain model.

In the following years, most works focused on advancing anomaly-based detection capabilities, but most models still used statistical approaches. In 2009, Chih-Fong Tsai et al. [9] proposed an IDS model using a hybrid learning model called triangle area nearest neighbours (TANN), which is composed of three stages: (1) extraction of cluster centres with a k-means algorithm, (2) new data formulation by triangle areas, and (3) application of a KNN classifier on the data. Using this model, the authors achieved a 97.0% detection accuracy rate.

Deep learning (DL) is a sub-field of Machine Learning (ML) that models the learning process using multiple layers of neurons forming a heavier artificial neural network (ANN). In fact, the word "deep" refers to the huge number of layers through which the data is transformed, and it was coined by Rina Dechter [10] in 1986. While several researchers have improved DL over the following years, it was only by 2012 that it started to be implemented to solve various tasks. This happened mostly due to three reasons:

1. Advances to the algorithms, which enabled them to be used on a broader range of domains;
2. Increased availability of larger datasets to train the models;
3. Increased computing power, which enabled improved usage of DL systems. One key development for this was made by NVIDIA's release of a new graphical processing unit (GPU) in 2009, which was used to increase the speed of DL systems a hundred times [11].

As the scene was set for further research in how to apply DL to more domains, not long after, we started to see a new variant of anomaly-based detection methods for IDS. One of the first integration of an ANN for anomaly-based NIDS was published in 2008 by Jimmy Shum et al. [7]. In 2011, Xianjin Fang et al. [8] integrated an ANN in the popular open-source IDS "Snort". To achieve this integration, Xianjin Fang et al. developed a custom detection plug-in for the IDS detection engine that, depending on the ANN model classification, could generate an alert file for the system's administrator. In 2015, Jamal Esmaily et al. [12] proposed an IDS model working with a hybrid approach, using both a Multilayer Perceptron (MLP) and a Decision Tree (DT) for classification, that achieved 99.7% detection accuracy on the KDD CUP 99 dataset [13]. It consisted of a two-phase model: in the first phase, the dataset was trained on both MLP and DT models and outputted a binary classification of either "attack" or "normal", after which the resulting classifications of both models were processed by another MLP network that outputted the final result (again, either "attack" or "normal").

In a comprehensive study of different ML approaches for intrusion detection, Kunal et al. [14] surveyed 19 different models by different authors to evaluate which one presented the best outcome. While not being able to identify a correlation between the different aspects of each model and their outcome, the author did document several possible approaches for further work.

Emmanuel Alalade et al. [15] proposed an IDS for Smart Homes using an artificial immune system and extreme learning hybrid approach (AIS-ELM). The AIS was used for input optimization and pre-processing, and ELM for classification. In the authors' framework, the resulting action was a notification to an arbitrary email to react locally to an attack.

In 2018, Nikita Lyamin et al. [16] documented the challenges of using DL for intrusion detection in a VANET, which is a mobile ad hoc network applied to the context of vehicles. Despite acknowledging that existing protocols in vehicle-to-vehicle communications integrated into a DL model are capable of detecting a DoS attack, they concluded that it would be difficult to train such a model because there were not enough available data to train it and, even if there were, most DL methods required annotated datasets about a Denial of Service (DoS) class. On the same note, focusing on a specific kind of network attack, Irfan Sofi et al. [17] explored the capabilities of four different ML algorithms (Naïve Bayes, DT, MLP and SVM) for the detection and analysis of modern types of Distributed Denial of Service (DDoS) attacks in 2017. The results showed the MLP model achieving the highest accuracy rate of the four tested models, with a 98.9% detection accuracy rate [17]. The authors found that no public datasets were available for training and testing a model against modern DDoS attacks. Thus, they recorded their own dataset in a controlled environment. While this enabled the training and testing of a model, the volume of records was low: the dataset has approximately 75,000 records, and nearly 67,000 are "normal" records. In comparison, the KDD CUP 99 dataset, still widely used by research for IDS models, has 5,000,000 records.

Still focused on DDoS attack detection, Merlin Dennis et al. [18] proposed two approaches for detecting these attacks in a software-defined network (SDN). The first was a statistical approach that implemented entropy computation and flow statistical analysis. The second was an ML approach that used a Random Forest classifier for detection. In the end, the first approach obtained a 92% accuracy, and the second achieved 97.7% accuracy. The authors also compared their ML approach against similar ones and accredited their success to using the random forest classifier, with fast computing times and resistance to noisy data, and implementing weighted voting instead of the more standard majority voting. Lingfeng Yang et al. [19] also proposed an SDN framework capable of identifying and defending the network against DDoS attacks. For this, the authors defined a framework consisting of 3 parts: the traffic collection module, the DDoS attack identification module, and the flow table delivery module. The DDoS attack identification module resorted to an SVM network to identify the attacks, and the model was trained with the KDD CUP 99 dataset, obtaining a 99% accuracy. Abdulsalam Alzahrani et al. [20], also working on SDN, went in a different direction and proposed an IDS as part of the SDN controller to detect pure intrusions on the network, instead of detecting DDoS attacks. After evaluating and comparing different ML algorithms, the authors concluded that the extra gradient boosting tree technique, XGBoost, was the more accurate one, showing a 95.55% accuracy.

In a different approach, Pablo Rivas et al. [21] leveraged data from a honeypot network to train two models, an SVM and a CNN, to be able to detect DDoS attacks. The work published by the authors followed the Marist College's effort to develop, deploy and maintain a HoneyNet—a network of honeypots—called Peitho (named after the Greek god). This network can then attract malicious actors that promptly try to attack it, sometimes even trying to use 0-day vulnerabilities. Peitho's network data is then fed to an SVM model and a CNN model to compare the different accuracy results. The CNN outperformed the SVM, achieving results of 100% accuracy in identifying DDoS attacks after processing eight network packets and achieving 90% accuracy after just two network packets. Another example of an effective ML model for DDoS detection was proposed by Jian Zhang et al. [22] with an approach of a gradient decision boosting tree to evaluate and optimize feature selection in a closed loop continuously.

2.2. Advanced Anomaly-Based Intrusion Detection Using Long Short-Term Memory Networks

As researchers compared several different ML algorithms for intrusion detection, one model started to stand out: the long short-term memory network (LSTM) model. Proposed in 1995 by Hochreiter et al. as a subset of recursive neural networks (RNN) [23], the name LSTM refers to the analogy that while a standard RNN has both long-term and short-term memory, the LSTM architecture provides a short-term memory for RNNs that can last several timesteps, thus the name long short-term memory. The authors published the LSTM technique as a proposal to solve the vanishing gradient problem, which happens when training an ANN with gradient-based learning methods and backpropagation [24]. The neural network's weights receive an update proportionate to the partial derivative of the error function concerning the current weight during each training iteration. The issue is that the gradient may become increasingly small in some situations, thereby preventing the weight from changing its value. This could prevent the neural network from being further trained in the worst-case scenario. LSTM intends to fight this problem by allowing gradients to flow unchanged while backpropagating. The result is a model that is capable of handling long-term dependencies. LSTMs were a step forward for what we could accomplish with RNNs. Xu et al. [25] used an LSTM model to create descriptions for images. In 2015, Google [26] started using LSTM models for its speech recognition services. More famously, OpenAI and DeepMind, two of the most prominent organizations for artificial intelligence research, started implementing LSTM models to beat humans in the video games Dota 2 and Starcraft II, respectively.

Following the ever-bigger trend of using LSTMs in DL problems, researchers started documenting its effects on intrusion detection. To research the trends in ML and DL algorithms applied to intrusion detection, M. Stampar et al. [27] gathered and measured the number of publications on intrusion and detection, and artificial intelligence/machine learning during the period from 2010 to 2014. They found that the most common models were: ANN, Genetic algorithms, Fuzzy logic, KNN, Naïve Bayes, Decision Tree, Random

Forest, SVM and Self-organizing map. The results revealed that the most investigated model was by far the ANN, with close to 1500 publications per year, presenting almost double the publications of the following one, the SVM, which shows close to 800 publications per year. In 2018, Ali Mirza et al. [28] investigated different kinds of LSTM networks for intrusion detection, such as vanilla LSTM, Bi-LSTM (which is a bidirectional LSTM network) and Deep Auto LSTM (which the authors define as a network with five stacks of LSTMs). Omar Alsyaibani et al. [29] also proposed a bidirectional LSTM network for intrusion detection. The authors iterated over 24 cycles different training parameters to find the most suitable configuration and trained the model for 100 epochs on the CICIDS2017 dataset, reaching an F1-Score of 97.75%. After 1000 epochs, the model reached a 98.37% F1-Score. Sara Althubiti et al. [30] experimented with a different dataset, the CIDDS-001, using a single LSTM layer model for intrusion detection and achieved an unusually lower accuracy of 84.83%.

Jorge Meira [31] compared four different intrusion detection techniques over the two most used datasets at the time: the KDD CUP 99 and the ISCX 2012 IDS. The techniques used were: Isolation forest, the Equal frequency with z-score and autoencoder, Z-score with nearest-neighbour and z-score with K-means. While on the KDD CUP 99 dataset, none of the techniques performed particularly better than the others, in the ISCX 2012 IDS dataset, the Z-score with nearest-neighbour performed significantly better than the competition.

In 2019, Thi-Thu-Huong Le et al. [32] proposed a novel IDS framework to reduce the high false-positives rates that plagued anomaly-based IDSs. The framework proposed by the authors included a hybrid sequence forward selection algorithm with a decision tree to generate the best feature subset from the original feature set. Those features were used to train several models of RNNs, such as a traditional RNN, an LSTM and a GRU. The models were trained with the NSL-KDD 2010 IDS dataset and the ISCX 2012 IDS dataset. In the end, the RNN model showing the best performance was the LSTM, achieving an accuracy of 92% in the NSL-KDD dataset and 97.5% in the ISCX dataset.

One of the biggest challenges of a NIDS is processing the sheer dimension of a typical high network traffic dataset. Liang Zhang et al. [33] proposed, in 2020, an improved LSTM IDS model based on a two-phase DL algorithm, where the first phase uses quantum particle swarm optimization (QPSO) for feature selection and dimensionality reduction, and the second phase uses an LSTM network for classification and therefore, intrusion detection. Running it over the KDD CUP 99 dataset, the model's QPSO reduced the 41 features to 24 features, resulting in an upgrade of the LSTM from 83.47% to 97.79%, and an F1-Score of 96.65%. The authors state that the proposed model can be used in real network environments for intrusion detection.

In 2020, Arunavo Dey [34] proposed a CNN-LSTM hybrid model for intrusion detection and trained it on the CICIDS2018 dataset. The author's model outperformed similar works and achieved a 99.98% accuracy. The model used four different layers: a CNN layer, that performed the convolution operation on the input, an LSTM layer, an attention layer and finally, a dense layer. Amutha et al. [35] proposed an IDS model that used a single LSTM layer for classification and trained it with the UNSWNB18IDS dataset [36]. The authors achieved an accuracy of 99.96%. Lokesh Karanam et al. [37] proposed another IDS model consisting of a CNN for feature selection and an LSTM for sequence analysis and classification. Thanks to the reduced dimensionality from the CNN feature selection, the computation time for the KDD CUP 99 dataset was reduced to one-tenth of previous works while maintaining a 99.6% accuracy. Another hybrid approach to intrusion detection using LSTM was proposed by Bhushan Deore et al. [38] in 2022. The author's model also consisted in a CNN for feature selection but, for the detection, the authors used a chimp-chicken swarm optimization-based deep LSTM. The results demonstrated that the proposed model outperformed other works models using vanilla CNNs and RNNs.

As recent network intrusion datasets are composed of high-dimensionality data, feature selection is a promising and important research direction to better implement ML and DL models in real-world deployments of anomaly-based NIDS. Joowha Lee et al. [39] propose a NIDS using a Deep Sparse Autoencoder that uses DL as a pre-processing mechanism. The authors used the autoencoder to identify the optimal model, reconstruct the data, and then use the resulting dataset with an RF model with lower dimensionality. Using this architecture, the authors reduce the training and testing time of the RF model by approximately half. To reduce computational complexity, Mohammadnoor Injadat et al. [40] propose a novel multi-stage optimized ML NIDS with an information gain-based feature selection methodology, which results in a reduction of 74% of the required training sample and 50% of the required feature size, while still maintaining the same accuracy performance as before the reductions for the CICIDS2017 and the UNSWNB2015 datasets. Di Mauro et al. [41] review several recent works that implement ML techniques for network intrusion detection and perform several analyses such as feature correlation, time complexity, and performance. The authors find that using the appropriate feature selection algorithms can reduce the dataset size up to 95% for single-class datasets and up to 92% for multi-class datasets and can reduce close to 95% of the computational time required to train the model. Paired with these computational complexity reductions, in the KDDCUP99 dataset, the accuracy slightly drop from 99.93% to 99.71%.

In 2021, Jitti Abraham et al. [42] published a review of the state-of-the-art for intrusion detection and prevention in networks using ML and DL, concluding that, while an ever-growing number of works are being published in this area, there is an absence of real-world network traffic analysis, and thus, the real impact of applying these models in a real-world scenario.

One of the challenges associated with using ML and DL models for real-world applications is the space and time complexity. Models with a large number of features can quickly exhaust system memory, as most ML and DL frameworks use exclusively the system memory for training. RNNs and their subclasses, such as the LSTM, can be negatively impacted by the added memory requirements, which effectively adds more computational complexity. Mario di Mauro et al. [43] reviewed thirteen of the most prominent neural-based techniques relevant to intrusion detection, evaluated the current datasets used for intrusion detection and performed some experimental analysis, which includes single-class vs multi-class datasets time complexity analysis. In this analysis, the authors find that, as usual, single-class datasets result in lower errors than multi-class datasets. The authors also find that DL models are up to two orders of magnitude slower than other techniques due to the fully connected neurons between each layer. Shi Dong et al. [44] propose a semi-supervised Double Deep Q-Network optimization method for network anomaly detection and perform a time complexity analysis of the proposal against frequently used ML and DL architectures. Although the training and prediction times are similar to previous works, the authors find that it is low enough for real-world deployment and usage. Charlotte Pelletier et al. [45] compare Random Forests to RNNs and CNNs for image time series and conclude that RNNs might not be suited for large-scale datasets as they present a higher time complexity.

3. Methodology

Following the introduction and aligned with the previous summary of the related literature, we decided on developing NIDS using a stacked LSTM architecture—DL NIDS. This approach was then trained and tested on a synthetic emulation of real-world attacks for a final evaluation set-up. The developed code was made publicly available in the GitHub repository indicated in the Data Availability Statement section.

3.1. Building a Deep Learning Network-Based Intrusion Detection System (DL NIDS)

Our Deep Learning Network-based Intrusion Detection System (DL NIDS) architecture is depicted in Figure 1. All components of the architecture were implemented using the Python programming language.

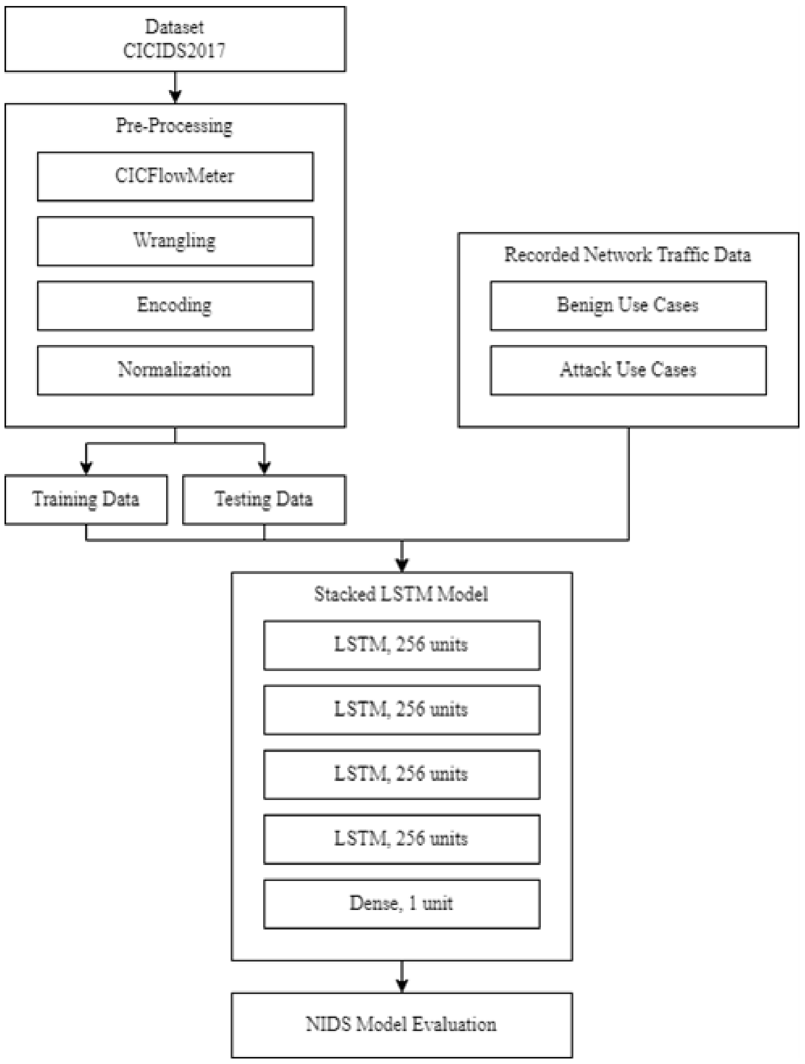


Figure 1. Anomaly-based DL NIDS arhictecture.

The following sections will detail each aspect of this system's architecture.

3.2. Training Dataset

Aligned with the study of the related literature, we choose the Intrusion Detection Evaluation Dataset CICIDS2017 [46] from the Canadian Institute for Cybersecurity (CIC) as the training dataset. The CICIDS2017 was built to emulate up-to-date attacks at the date of its recording since most of the datasets available then dated before 2000. Therefore, the resemblance to modern types of network traffic and ports used in today's network communications is low. Nevertheless, we have set up a more actual test, which will be detailed in Section 3.5.

The CICIDS2017 dataset was captured from 9 AM, Monday, 3 July 2017, to 5 PM Friday, 7 July 2017, for a total of five days. To simulate realistic network traffic, the network used was based on the user behaviour of 25 persons using systems for several applications based on HTTP, HTTPS, FTP, SSH and SMTP protocols. The attacks performed in the recording include Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attacks, Infiltration, Botnet and DDoS, as shown in Table 1.

Table 1. CICIDS2017 dataset contents.

Day of Recording	Network Activity Recorded
Monday	Benign
Tuesday	Benign, FTP-Patator, SSH-Patapor
Wednesday	Benign, DoS GoldenEye, DoS Hulk, DoS slowhttptest, DoS slowloris, Heartbleed
Thursday	Benign, Web Attack Brute Force, Web Attack SQL Injection, Web Attack XSS, Infiltration
Friday	Benign, Bot, PortScan, DDoS

The CICIDS2017 is thus a 15-class dataset with 83 features where each flow is labelled as benign or as one of the fourteen different attack classes shown in the right column of Table 1. While a typical network traffic recording has less than 20 features, such as those recorded by default from typical network capture software (e.g., Wireshark [47]), the CICIDS2017 is much more feature rich. The reason is that the dataset authors used a network traffic analysis tool called CICFlowMeter [48], which we will further detail in the next section.

For our purpose, we converted the dataset to a binary labelled dataset, where each instance can be classified as 'benign' or as an 'attack'. The final dataset has 2,830,540 instances, where 2,359,087 (83.34%) are classified as benign, and 471,453 (16.66%) are classified as attacks.

3.3. Pre-Processing

To build an adequate NIDS, we must prepare the dataset before using it for model training and testing. This is an important step of every successful ML and DL model and can greatly impact the overall performance of the models. For this purpose, we use several tools and techniques further described below.

3.3.1. Cicflowmeter

CICFlowMeter [48] is a network traffic analyzer developed by CIC, the same organization that authors the CICIDS2017 dataset. The tool receives as input either a PCAP file or performs analysis of real-time data from a network adapter. After analysis, the tool processes the captured network packets converts them into flows and outputs the results to a CSV file with 83 features. This tool is crucial for our model, as it will be used to train and test the model and pre-process the real-world network traffic captured.

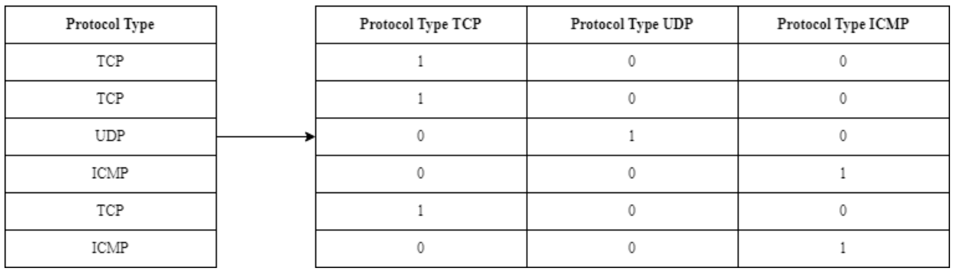
One important aspect of the CICFlowMeter tool is that the resulting CSV file is not a 1-to-1 mapping to the originally recorded network packets but is instead a conversion of individual network packets to network flows. While network packets comprise individual data segments with a unidirectional attribute, network flows define a sequence of the network packets. With network flows, we can better understand the flow of network communications between two endpoints, resulting in a better candidate for intrusion detection.

The CICIDS2017 dataset already has the resulting CSV files for processing each of the five recorded days of network traffic through CICFlowMeter. Although we didn't have to run CICFlowMeter, the resulting CSV files had a few encoding and whitespace inconsistencies, which had to be clean.

3.3.2. Data Wrangling and Encoding

After cleaning the inconsistencies, we merged the CSV files into a single pandas data frame for data wrangling. Afterwards, the rows that presented NaN "values" and the "FlowID" feature, as it is simply a primary key for each flow, were removed. The rows were then sorted by timestamp to guarantee that the network flows are ordered according to real deployment, after which the timestamp feature was dropped as it won't be needed anymore.

Encoding data is the process of transforming some input to numbers, usually in a way that is reversible and allows the translation between the resulting output and the original input. However, we must be careful when encoding since categorical features can either be nominal or ordinal, the difference being that ordinal features comprise a finite set of discrete values with a ranked ordering between values. On the other hand, nominal features present no relationship between values other than being different (e.g., port 22, port 80, port 443). If we simply convert every category to a different number, we force an ordinal encoding, resulting in a dataset feature with a natural ordering between categories that can be misleading for the training of the ML model. As such, we employed a binarization technique called one-hot encoding. Using one-hot encoding, we convert each possible category from a specific feature into a new binary feature of its own, with the value one (1) meaning that it belongs to this category and zero (0) otherwise, as shown in Figure 2.



Protocol Type
TCP
TCP
UDP
ICMP
TCP
ICMP

Protocol Type TCP	Protocol Type UDP	Protocol Type ICMP
1	0	0
1	0	0
0	1	0
0	0	1
1	0	0
0	0	1

Figure 2. One-Hot Encoding explanation.

One-hot encoding was used to encode every categorical feature: Protocol, Destination, Source, and a new Port feature: since the dataset has both the source and destination ports, we merged them into a single feature, preserving the port that maps the flow to a single network application. While having both ports is relevant from a manual network analysis perspective, these won't be useful for an ML model because one of the ports is always a dynamic port typically attributed to the Network Address Translation (NAT) process of network routing. These dynamic ports usually belong to the higher range of ports—from 49,152 to 65,535. The lower port number belongs to a specific network application such as HTTP (port 80), HTTPS (port 443) or SSH (port 22). As one-hot encoding increases the dimensionality of a dataset, we cannot transpose every single port. As such, we opted to perform one-hot encoding for every flow with a port from 0 to 4096 as illustrated in Algorithm 1. For every flow after port 4096, we classified it as 4096. While this means that the NIDS won't be able to distinguish different ports over 4096, we still encompass the most frequently used ports in both benign and attack network use cases (e.g., RDP, SSH, FTP, DNS).

Algorithm 1 Port number conversion

```
for $Row$ in $Dataset$ do
    $sp \leftarrow SourcePort$
    $dp \leftarrow DestinationPort$
    if $sp <= dp$ then
        $Port = sp$
    else
        $Port = dp$
    end if
    if $Port >= 4096$ then
        $Port = 4096$
    end if
    $Row \leftarrow Row + Port$
end for
```

One of the key parts of our work is the choice of pre-processing of the source and destination features. Originally, these features had an IP address. While the IP address served as an identifier for each system in the network, it's hard to translate into a feature for ML. Two of the most common approaches to solve this problem are: (a) removing these features altogether or (b) using a dictionary to translate the IP addresses to a number, which can be reversed in the end to identify a malicious IP address. While both techniques have been proven to achieve high accurate ratings, they're limited for model transpositions since:

1. removing the features altogether removes precious context, such as the general network location and results in the incapacity of answering questions like: is the traffic coming from inside the network? Or is it coming from outside? Using RDP from an internal system to another internal system is a very different situation from using RDP from outside the network to the inside.
2. using a dictionary effectively maps individual systems and can detect situations such as: is all traffic coming from the same IP address? Is this individual IP address trying to reach all of the networks consequently? However, the problem is that while this is a valid method for intrusion detection in the same network context as the trained model, if we transpose this model to a different network, it will have a high rate of misclassifications as different networks can have very different traffic coming from the same translated IP address. Additionally, an IP address (translated or not) is effectively a categorical feature, which means that encoding it correctly would result in a dimensionally increased dataset.

We opted to implement a trade-off between the two most common options—we converted each IP address to a binary feature of either Internal, whenever the IP address belongs to a private address space, or External, otherwise (Algorithm 2). As both Internal and External are universal features of every network flow, using this method provides more network context than simply removing the source and destination features and enables an efficient transition of the model to different networks since this feature is context-independent. This feature was also encoded with one-hot encoding.

Algorithm 2 IP address conversion

```
for $Row$ in $Dataset$ do
    if $IP$ starts with "192.168." or "172.16." or "10." then
        $IP = Internal$
    else
        $IP = External$
    end if
end for
```

3.3.3. Data Scaling

Another important aspect of pre-processing is feature scaling. Since the range of values for some of the numerical ordinal features vary widely, we performed a min-max scaling technique on all of the non-categorical features. This is commonly known as the normalization of values and scales all values to a fixed range between 0 and 1 using formula (1):

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (1)$$

where x_{max} and x_{min} are the maximum and the minimum values for feature X, respectively.

Lastly, we split the dataset using scikit's train_test_split function [49], which randomly distributes the dataset into two smaller datasets: a training dataset, which is used to fit the model, and a testing dataset, which is used for unbiased model evaluation and tuning. We dedicate 75% of the complete dataset for the training dataset and 25% for the testing dataset. Both training and testing datasets have the same proportion of benign to attack ratio as the original dataset (83.34% benign, 16.66% attack).

3.4. A Stacked LSTM Model for NIDS

Following our background study, we propose using a stacked LSTM model for the DL NIDS. The LSTM is a subset of the RNN family of models and is best suited for classifying temporal-dependent data, such as the data from a network intrusion problem. An LSTM unit consists of a cell, an input gate, an output gate and a forget gate. We can see the composition of an LSTM cell in Figure 3.

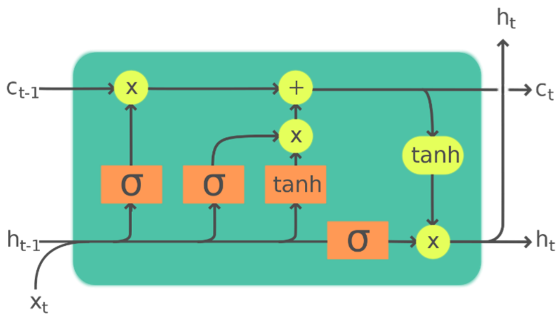


Figure 3. Depiction of an LSTM unit.

Using $c_0 = 0$ and $h_0 = 0$ as the initial values, the equations for an LSTM unit are:

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right) \tag{2}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{3}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{4}$$
$$\widetilde{c_t} = \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{5}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c_t} \tag{6}$$
$$h_t = o_t \odot \sigma_h(c_t) \tag{7}$$

where the subscript t indexes the timestep, $x_t \in R^d$ is the input vector to the LSTM unit; $f_t \in (0,1)^h$ is the forget gate's activation vector; $i_t \in (0,1)^h$ is the input and update gate's activation vector; $o_t \in (0,1)^h$ is the output gate's activation vector; $h_t \in (-1,1)^h$ is a hidden state vector or output vector of the LSTM unit; $\widetilde{c_t} \in (-1,1)^h$ is the cell input activation vector; $c_t \in R^h$ is the cell state vector; $W \in R^{h.d}, U \in R^{h.h}, b \in R^h$ are weight matrices and bias vector parameters learned during the training phase, where d represents the number of input features, and h represents the number of hidden units.

One of the biggest challenges in training a large model is overfitting the training data, resulting in the model learning statistical noise. When the model is applied to different environments, it can result in poor performance. As such, we use the dropout technique as a preventive measure to reduce this phenomenon. The dropout technique is a regularization method that randomly drops nodes from the input and the hidden layers in the model. The model attempts to filter out the statistical noise by dropping these nodes and learning only the defining characteristics. With the dropout, the forward propagation equation changes to:

$$r_j^{(l)} \sim Bernoulli(p) \tag{8}$$
$$\widetilde{y^{(l)}} = r^{(l)} \cdot y^{(l)} \tag{9}$$
$$z_i^{(l+1)} = w_i^{(l+1)} \widetilde{y^{(l)}} + b_i^{(l+1)} \tag{10}$$
$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right) \tag{11}$$

where z represents the vector of output from layer $(l+1)$ before activation, y represents the vector of outputs from layer $(l+1)$, w represents the weight of layer $l+1$ and b represents the bias of the layer $l+1$.

In our model, we implement a dropout layer after each of the LSTM's layers with a rate of 0.1, which means that 10% of the input units will be dropped for each LSTM layer. The final model topology is defined by the code in Table 2.

Table 2. Stacked LSTM model topology.

Layer (Type)	Output Shape	Parameters
LSTM	(None, 1, 256)	1,248,256
Dropout	(None, 1, 256)	0
LSTM	(None, 1, 256)	525,312
Dropout	(None, 1, 256)	0
LSTM	(None, 1, 256)	525,312
Dropout	(None, 1, 256)	0
LSTM	(None, 256)	525,312
Dropout	(None, 256)	0
Dense	(None, 1)	257
Activation	(None, 1)	0

The loss function is the function that computes the distance between the current output of the algorithm and the desired one. To have the best possible model, we want this distance to be minimised, which means trying to get the output of the function as close as possible to zero. There are many possible loss functions to use in a model, but as our problem is a binary classification one, we use the cross-entropy function, also known as the log loss function. The equation for the binary cross-entropy loss is:

$$-\sum_{n=i}^{c} t_i log(f(s_i)) + (1-t_i) log(1-f(s_i)) \tag{12}$$

where s_i represents the inputs and their weights, f represents the activation function, t represents the target predictions and i represents the class to predict.

While the loss function expresses how close the model is to the desired output, we can optimize the learning process of the model with an optimization algorithm and then update them in the model. These algorithms find the best possible value for the weights that minimize errors when mapping inputs to outputs. For our purposes, we use the Adam optimization algorithm [50] from Diederik Kingma et al., which is an extension to the stochastic gradient descent and has been seeing significant adoption since 2015. The main difference between the Adam algorithm and the classical stochastic gradient descent is that, while the latter maintains a single learning rate for all weight updates, Adam can dynamically update the learning rate using adaptive gradient algorithms, and the root means square propagation, whose update equations are:

$$v_t = \beta_1 \cdot v_{t-1} - (1-\beta_1) \cdot g_t \tag{13}$$

$$s_t = \beta_2 \cdot v_{t-1} - (1-\beta_2) \cdot g_t^2 \tag{14}$$

$$\Delta w_t = -\eta \frac{v_t}{\sqrt{s_t + \epsilon}} \cdot g_t \tag{15}$$

$$w_{t+1} = w_t + \Delta w_t \tag{16}$$

where w represents each parameter, η represents the initial learning rate, g_t represents the gradient at time t along w, v_t represents the exponential average of gradients along w, s_t represents the exponential average of squares of gradients along w, and β_1 and β_2 are hyperparameters.

In the end, we need an activation layer to define how the weighted sum of the previous layers results in a prediction, which needs to be chosen and adapted concerning the problem at hand. Again, since we face a binary classification problem, the recommended activation function is the sigmoid activation function, whose equation is:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{17}$$

where x represents the input of the sigmoid function and e is the Euler number.

The model is trained by running the fit function for 25 epochs with a batch size of 128.

3.5. Real-World Data Collection

After the model's training, we must evaluate if the NIDS can accurately identify intrusions when confronted with real-world data outside the training and testing dataset. Most NIDS receive network flows via integration with either a firewall or a network sniffing tool in a real network environment. But, for academic purposes, most authors opt to use pre-recorded network traffic data, usually in the format of a PCAP file. In our work, we had the opportunity to choose either of them thanks to the CICFlowMeter tool because while the main purpose of the tool is to extract features from network traffic data in a PCAP format, the tool's authors also included the capability of extracting these features in realtime by sniffing the network data of a chosen network adapter. Additionally, the tool is open-source, and its code is publicly available on GitHub, which means anyone can directly link these two pieces in the future.

3.6. Real-World Network Traffic Data Recording

To evaluate the model against real-world network traffic data, we need to define:

1. What to record?
2. Where to record?
3. How to record?

Regarding what to record, we opted to record several different benign and malicious network use cases, as described in Table 3. These use cases range from pure white noise network traffic (to establish a baseline of the network traffic data) to synthetic network attacks. By analyzing the classification given by the NIDS to all of these different use cases, we can understand if the NIDS can accurately predict if there is a network intrusion ongoing or not.

While the benign use cases were recorded by performing the use cases themselves, all of the malicious use cases were generated with the help of a malicious network traffic generator tool called "flightsim" [51], which is an open-source tool made by Alphasoc [52] capable of generating several different kinds of synthetic network attacks to evaluate if security teams are capable of detecting and responding to them.

As to where to record, we chose to capture the network traffic in a desktop system inside of a controlled network environment. While we could use a more complex capturing scenario, such as using port mirroring in a switch, with our pre-processing, the results should still be accurate and proportionally similar to the use of the model in a larger network.

Finally, regarding how to record, we use the widely popular network capture tool "Wireshark" [47]. We captured all traffic from the network-connected adapter without any filter while running all of the use cases and exported the results to a PCAP file.

The resulting PCAP file acts as raw data that will be used to evaluate the proposed DL NIDS approach. For that, the previously described pre-processing will be used to convert and extract the same features used for modelling. As such, we run all of the PCAP files through the CICFlowMeter tool to extract the new features and export them to a CSV file. Afterwards, the same pre-processing techniques were applied, such as converting internal IP addresses to just internal, dropping the unnecessary timestamp feature, using the same scaling factor and weights from the training phase, and encoding the categorical features.

Table 3. Recorded network use cases.

Use Case	Type of Use Case	Number of Packets	Number of Flows	Description
White noise	Benign	1069	108	Background network traffic from the recorded system with the default background services from Windows 10
Internet browsing	Benign	66,688	322	General internet browsing usage consisting of YouTube, Google Spreadsheets and Google Search
WhatsApp	Benign	1020	64	Message and images exchange via the WhatsApp Windows desktop client
Spotify	Benign	36,189	373	Music streaming via the Spotify Windows desktop client
C2	Attack	111	23	DNS and IP traffic to a random list of known C2 destinations
DGA	Attack	398	37	Domain generation algorithm traffic using random labels and top-level domains
Imposter	Attack	62	12	DNS traffic to a list of imposter domains
Miner	Attack	118	22	Stratum mining protocol traffic to known cryptomining pools
Scan	Attack	263	13	Port scan of random RFC 5737 addresses using common TCP ports
Sink	Attack	131	26	Connection to known sinkholed destinations run by security researchers
Spambot	Attack	173	47	Resolve and connection to random Internet SMTP servers to simulate a spam bot
Tunnel DNS	Attack	216	62	DNS tunneling requests to *.sandbox.alphasoc.xyz
Tunnel ICMP	Attack	209	11	ICMP tunneling traffic to an Internet service operated by AlphaSOC

4. Results and Discussion

Aligned with our objectives, we performed two different experiments: the first test of the performance of the proposed NIDS model employing our novel pre-processing techniques in the CICIDS2017 dataset, and the second is the evaluation of the new NIDS model using real-world network traffic data.

4.1. Results—CICIDS2017

We used the evaluation metrics for the first experiment: accuracy, precision, recall, and F1-Score. These metrics depend on the number of true positives (TP), correctly predicted as an attack, true negatives (TN), correctly predicted as benign, false positives (FP) and false negatives (FN).

The hardware used for the experimentation consisted of an AMD Ryzen 5 3600X 6-core processor with 12 virtual CPUs and 3.8GHz, 32GB RAM and an NVIDIA RTX 3070 GPU. As for time complexity, the phase of pre-processing took 147 s for the 2,671,674 flows, while the training of the model using a 75–25% split for train and test, respectively, ran over

25 epochs, taking 13,325 s (3 h, 42 min, and 5 s) for the 2,824,449 parameters of our model configuration as described in the previous chapter.

Accuracy is defined as the total correctly classified samples divided by the total number of samples:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{18}$$

Precision is defined as the actual attack correct predictions divided by the total number of positively classified samples:

$$Precision = \frac{TP}{TP + FP} \tag{19}$$

Recall, also known as the sensitivity or true positive rate, is defined as the number of true positives divided by the total number of true positives and false negatives and is given by:

$$Recall = \frac{TP}{TP + FN} \tag{20}$$

The F1-Score is a weighted average of the precision and recall metrics. We use the most common harmonic mean relation for the F1 measure, as given by the Equation (21):

$$F1\text{-}Score = 2\frac{Precision \times Recall}{Precision + Recall} \tag{21}$$

The results of the test are depicted by the confusion matrix in Figure 4.

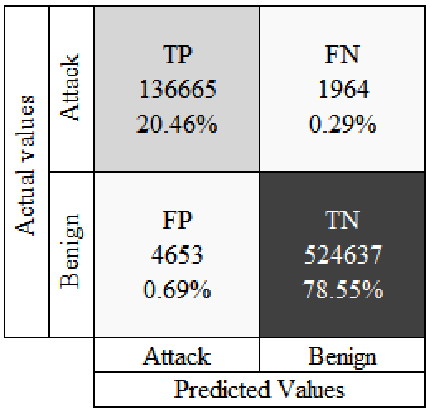


Figure 4. Confusion matrix with our results.

Transposing the results in an evaluation metrics table, we can observe (Table 4) that the proposed DL NIDS model can accurately classify network flows as either attack or benign. In fact, the Precision for attack classification is 96.71% and for benign classification is 99.63%.

Table 4. Evaluation metrics for the CICIDS2017.

	Precision	**Recall**	**F1-Score**
Attack	0.9671	0.9858	0.9764
Benign	0.9963	0.9912	0.9937

The accuracy of the proposed model is 99.01% (with a weighted average of 99.02%, that is, the average accounted for the contribution of each class as weighted by the number of examples of that class). These results show that our model outperforms related works using the CICIDS2017 dataset, as observed in Table 5.

Table 5. Comparison with similar studies.

Authors	Model Architecture	Accuracy	Precision	Recall	F1-Score
Figueiredo et al. (this study)	Stacked LSTM	99.01%	96.71%	98.58%	97.64%
Zhang et al. [33]	Single LSTM layer	97.79%	N/A	N/A	96.95%
Alsyaibani et al. [29]	Bidirectional LSTM	97.72%	N/A	N/A	97.66%
Ferrag et al. [53]	Unspecified	97.28%	N/A	N/A	N/A
Nayyar et al. [54]	Single LSTM layer	96.70%	N/A	N/A	N/A
Kim et al. [55]	CNN-LSTM	93.00%	86.47%	76.83%	81.36%
Alin et al. [56]	CNN	N/A	N/A	N/A	95.00%

4.2. Real-World Network Traffic Data Evaluation

To evaluate our NIDS efficacy with real-world network traffic data, we can't use the same evaluation metrics as the ones used for the CICIDS2017 dataset, as real-world data does not come labelled. As such, we look at the number of network flows that have been classified as malicious for each of the recorded use cases. The results are further detailed in Table 6.

Table 6. Real-world network data captured. Note that attacks are labelled using *.

Use Case	# of Benign Classifications	# of Attack Classifications	Percentage of Attack Classifications
White noise	107	1	0.93%
Internet browsing	320	2	0.62%
WhatsApp	64	0	0.00%
Spotify	372	1	0.27%
C2 *	23	0	0.00%
DGA *	37	0	0.00%
Imposter *	12	0	0.00%
Miner *	22	0	0.00%
Scan *	12	1	7.69%
Sink *	26	0	0.00%
Spambot *	40	7	14.89%
Tunnel DNS *	61	1	1.61%
Tunnel ICMP *	11	0	0.00%

While these results were subpar in comparison with the results from the CICIDS2017 dataset, we must remember that we are using synthetically generated network intrusion attacks instead of performing real network intrusion attacks, which contributes to a higher probability of false negative rates.

As stated in the introduction and methodology, a DL NIDS focuses not on IOCs but on heuristics and rules. A typical network intrusion is not achieved via a single network flow, and as such, the NIDS is not expected to immediately return a binary classification that is based on the analysis of several network flows. To be properly implemented in a production environment, a NIDS must output, at any given time, the probability that an intrusion might occur.

The best approach is using a ratio of benign-classified and attack-classified network flows. These ratio results can be observed in the last column of Table 6. Starting with the benign use cases (use cases without *), we can observe that in each case less than 1% of network flows were classified as an attack. In contrast, although some attacks have not been spotted, Scan, Spambot and TunnelDNS present ratios above the previously established benchmark from the benign use cases. We can interpret these results in two ways:

1. The proposed NIDS can correctly detect some categories of attacks, as the resulting output of some of the categories of attack is clearly over a the previously established benchmark established by comparing the benign use cases.
2. This NIDS can't detect the remaining categories of attacks since it outputted 0% in most of them.

We believe that the NIDS didn't perform as well with this real-world data as with the CICIDS2017 is mainly due to the recorded use cases present in each of the datasets.

While both datasets record several different types of network intrusion use cases, they don't overlap on most of them. As such, we trained our model to detect the specific network intrusion attacks used in CICIDS2017, but as our real-world dataset was recorded with different kinds of network intrusion attacks, the model cannot detect them accurately.

The exception to this rule is the similar attacks in both datasets – specifically, the denial-of-service attacks. As seen in Table 6, our NIDS accurately detected intrusions in the two recorded use cases that share similar behaviour to a denial of service attack (Spambot and Scan).

5. Conclusions and Future Works

Our work reviewed the current state of the art in anomaly-based intrusion detection systems supported by deep learning architectures. We concluded that since its first iteration in 1980, there had been a lot of research in advancing its heuristics capabilities. In the last decade, with the uprising of machine learning and deep learning techniques enabled by newer and more powerful hardware, we have seen improvements in these models and a reduction in false positive rates. Some of the more recent works reach over 90% accuracy rates, which enables them for production, although arguably for the purpose of detection but not for automated prevention.

We propose a novel pre-processing technique that can obtain context-free features for deploying a high-accuracy trained anomaly-based DL NIDS. This proposal pairs this pre-processing technique with a stacked LSTM architecture DL model as indicated for temporal context-related problems such as intrusion detection.

Regarding the accuracy of a DL model for intrusion detection, we conclude that with the usage of our model, we were able to reach 99.01% accuracy, which, according to our literature review, makes it the most accurate model using the CICIDS2017 dataset. This is an accurate enough model to consider adding automated prevention capabilities without disturbing the well-being of the network in which it is implemented. We also conclude that with the proposed pre-processing methodology, we can transpose a labelled dataset-trained DL NIDS model to a different network and detect similar intrusion use cases as the ones used for training.

Using our proposed DL NIDS model in the CICIDS2017 dataset, we obtained a 99.01% accuracy. Concerning the application of the trained DL NIDS model with real-world network data, we find that it can detect the network intrusion use cases of denial-of-service attacks similar to those used for the original training. As such, it is expected that if the model is integrated into an autoML production test, it will be able to adjust and improve its precision.

In this work, we were limited to using a small network environment rather than what one could expect to be used in a typical corporate network setup. We also used synthetically generated network intrusion attacks instead of performing real network intrusion attacks, which did contribute to higher false negative rates. Another limitation we faced is the port number encoding, as while the usage of one-hot encoding will contribute to an even more accurate model, it is susceptible to dimensionality problems and will need a very high amount of system memory.

As our DL NIDS was only capable of detecting these previously known network intrusion use cases, it's not suitable for a fire-and-forget implementation in a real network as the sole intrusion detection mechanism. To improve our model, future works should consider training the model with more network intrusion use cases using an existing framework of attacks, such as the MITRE ATT&CK [57]. Future works should also consider training the model using a multi-class approach, as it might improve accuracy and better adapt to future automated intrusion-blocking mechanisms. On the other hand, we should consider the usage of IPv6 and its attributes when learning to detect intrusion, as while it is still not widely used in internal corporate networks, it might be relevant in the future.

Author Contributions: Conceptualization, J.F., C.S. and A.M.d.A.; methodology, J.F.; software, J.F.; validation, C.S. and A.M.d.A.; formal analysis, C.S. and A.M.d.A.; investigation, J.F.; writing—original draft preparation, J.F.; writing—review and editing, J.F., C.S. and A.M.d.A.; supervision, C.S. and A.M.d.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: CICIDS2017 Dataset: https://www.unb.ca/cic/datasets/ids-2017.html (accessed on 15 January 2022). GitHub Repository: https://github.com/ImJoaoPedro/Deep-Learning-Model-Transposition-for-Network-Intrusion-Detection-Systems (accessed on 20 December 2022).

Acknowledgments: We would like to thank the research environment provided by the Information Sciences, Technologies, and Architecture Research Center (ISTAR), supported by Fundação para a Ciência e a Tecnologia (FCT), Portugal, under projects UIDB/04466/2020 and UIDP/04466/2020.

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

References

1. Company, M. Cybersecurity Trends: Looking over the Horizon. Available online: https://www.mckinsey.com/capabilities/risk-and-resilience/our-insights/cybersecurity/cybersecurity-trends-looking-over-the-horizon (accessed on 1 October 2020).
2. Anderson, J.P. Computer Security Threat Monitoring and Surveillance. In *Technical Report James P Anderson Co Fort Washington Pa*; 1980; p. 56. Available online: https://docslib.org/doc/2332250/computer-security-threat-monitoring-and-surveillance (accessed on 1 October 2022). [CrossRef]
3. Denning, D.E. An intrusion-detection model. In Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, USA, 7–9 April 1986; pp. 118–131. [CrossRef]
4. Jallad, A.; Data, J.B.; Jallad, K.A.; Aljnidi, M.; Desouki, M.S. Anomaly detection optimization using big data and deep learning to reduce false—Positive. *J. Big Data* **2020**, *7*, 68. [CrossRef]
5. Roumani, Y. Patching zero-day vulnerabilities: An empirical analysis. *J. Cybersecur.* **2021**, *7*, tyab023. [CrossRef]
6. Wang, Y.; Wong, J.; Miner, A. Anomaly intrusion detection using one class SVM. In Proceedings of the Fifth Annual IEEE System, Man and Cybernetics Information Assurance Workshop, SMC, West Point, NY, USA, 10–11 June 2004; pp. 358–364. [CrossRef]
7. Shum, J.; Malki, H.A. Network intrusion detection system using neural networks. In Proceedings of the 4th International Conference on Natural Computation, ICNC 2008, Jinan, China, 18–20 October 2008; pp. 242–246. [CrossRef]
8. Fang, X.; Liu, L. Integrating artificial intelligence into Snort IDS. In Proceedings of the 2011 3rd International Workshop on Intelligent Systems and Applications, ISA 2011, Wuhan, China, 28–29 May 2011. [CrossRef]
9. Tsai, C.F.; Lin, C.Y. A triangle area based nearest neighbors approach to intrusion detection. *Pattern Recognit.* **2010**, *43*, 222–229. [CrossRef]
10. Dechter, R. Learning While Searching in Constraint-Satisfaction-Problems. *Aaai* **1986**, 178–185.
11. NVIDIA CEO Bets Big on Deep Learning. Available online: https://venturebeat.com/business/nvidia-ceo-bets-big-on-deep-learning-and-vr/ (accessed on 1 October 2022).
12. Esmaily, J.; Moradinezhad, R.; Ghasemi, J. Intrusion detection system based on Multi-Layer Perceptron Neural Networks and Decision Tree. In Proceedings of the 2015 7th Conference on Information and Knowledge Technology, IKT 2015, Urmia, Iran, 26–28 May 2015; pp. 1–5. [CrossRef]
13. KDD CUP 99. Available online: https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data (accessed on 1 January 2022).
14. Kunal; Dua, M. Machine Learning Approach to IDS: A Comprehensive Review. In Proceedings of the 3rd International Conference on Electronics and Communication and Aerospace Technology, ICECA 2019, Coimbatore, India, 12–14 June 2019; pp. 117–121. [CrossRef]
15. Alalade, E.D. Intrusion Detection System in Smart Home Network Using Artificial Immune System and Extreme Learning Machine Hybrid Approach. In Proceedings of the IEEE World Forum on Internet of Things, WF-IoT 2020—Symposium Proceedings, New Orleans, LA, USA, 2–16 June 2020; pp. 20–21. [CrossRef]
16. Lyamin, N.; Kleyko, D.; Delooz, Q.; Vinel, A. AI-Based Malicious Network Traffic Detection in VANETs. *IEEE Net.* **2018**, *32*, 15–21. [CrossRef]
17. Sofi, I.; Mahajan, A.; Mansotra, V. Machine Learning Techniques used for the Detection and Analysis of Modern Types of DDoS Attacks. *Int. Res. J. Eng. Technol. (IRJET)* **2017**, *4*, 1085–1092.
18. Dennis, M.J.R.; Li, X. Machine-Learning and Statistical Methods for DDoS Attack Detection and Defense System in Software Defined Networks. Master's Thesis, Ryerson University, Toronto, ON, Canada, 2018.
19. Yang, L.; Zhao, H. DDoS attack identification and defense using SDN based on machine learning method. In Proceedings of the 2018 15th International Symposium on Pervasive Systems, Algorithms and Networks, I-SPAN 2018, Yichang, China, 16–18 October 2018; pp. 174–178. [CrossRef]
20. Alzahrani, A.O.; Alenazi, M.J.F. Designing a network intrusion detection system based on machine learning for software defined networks. *Future Internet* **2021**, *13*, 111. [CrossRef]

21. Rivas, P.; Decusatis, C.; Oakley, M.; Antaki, A.; Blaskey, N.; Lafalce, S.; Stone, S. Machine Learning for DDoS Attack Classification Using Hive Plots. In Proceedings of the 2019 IEEE 10th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2019, New York, NY, USA, 10–12 October 2019; pp. 401–407. [CrossRef]
22. Zhang, J.; Liang, Q.; Jiang, R.; Li, X. A Feature Analysis Based Identifying Scheme Using GBDT for DDoS with Multiple Attack Vectors. *Appl. Sci.* **2019**, *9*, 4633. [CrossRef]
23. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
24. Mitchell, T.M. *Machine Learning*; McGraw-Hill: New York, NY, USA, 1997; Volume 1.
25. Xu, K.; Ba, J.L.; Kiros, R.; Cho, K.; Courville, A.; Salakhutdinov, R.; Zemel, R.S.; Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6–11 July 2015; Volume 3, pp. 2048–2057.
26. Danko, Z. Neon Prescription. Or Rather, New Transcription for Google Voice. Available online: https://blog.google/products/google-voice/neon-prescription-or-rather-new/ (accessed on 1 October 2020).
27. Stampar, M.; Fertalj, K. Artificial intelligence in network intrusion detection. In Proceedings of the 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2015, Opatija, Croatia, 25–29 May 2015; pp. 1318–1323. [CrossRef]
28. Mirza, A.H.; Cosan, S. Computer network intrusion detection using sequential LSTM Neural Networks autoencoders. In Proceedings of the 26th IEEE Signal Processing and Communications Applications Conference, SIU 2018, Izmir, Turkey, 2–5 May 2018; pp. 1–4. [CrossRef]
29. Alsyaibani, O.M.A.; Utami, E.; Hartanto, A.D. An Intrusion Detection System Model Based on Bidirectional LSTM. In Proceedings of the 3rd International Conference on Cybernetics and Intelligent Systems, ICORIS 2021, Makasar, Indonesia, 25–26 October 2021. [CrossRef]
30. Althubiti, S.A.; Jones, E.M.; Roy, K. LSTM for Anomaly-Based Network Intrusion Detection. In Proceedings of the 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), Sydney, NSW, Australia, 21–23 November 2018; pp. 20–22. [CrossRef]
31. Meira, J. Comparative Results with Unsupervised Techniques in Cyber Attack Novelty Detection. *Proceedings* **2018**, *2*, 1191. [CrossRef]
32. Le, T.T.H.; Kim, Y.; Kim, H. Network intrusion detection based on novel feature selection model and various recurrent neural networks. *Appl. Sci.* **2019**, *9*, 1392. [CrossRef]
33. Zhang, L.; Yan, H.; Zhu, Q. An Improved LSTM Network Intrusion Detection Method. In Proceedings of the 2020 IEEE 6th International Conference on Computer and Communications, ICCC 2020, Chengdu, China, 11–14 December 2020; pp. 1765–1769. [CrossRef]
34. Dey, A. Deep IDS : A deep learning approach for Intrusion detection based on IDS 2018. In Proceedings of the 2020 13th International Conference on Developments in eSystems Engineering (DeSE), Liverpool, UK, 14–17 December 2020; pp. 19–20. [CrossRef]
35. Amutha, S.; Kavitha, R.; Srinivasan, S.; Kavitha, M. Secure network intrusion detection system using NID-RNN based Deep Learning. In Proceedings of the IEEE International Conference on Advances in Computing, Communication and Applied Informatics, ACCAI 2022, Chennai, India, 28–29 January 2022; pp. 1–5. [CrossRef]
36. UNSWNB18IDS. Available online: https://research.unsw.edu.au/projects/unsw-nb15-dataset (accessed on 1 October 2022).
37. Karanam, L.; Pattanaik, K.K.; Aldmour, R. Intrusion Detection Mechanism for Large Scale Networks using CNN-LSTM. In Proceedings of the International Conference on Developments in eSystems Engineering, DeSE, Liverpool, UK, 14–17 December 2020; pp. 323–328. [CrossRef]
38. Deore, B.; Bhosale, S. Hybrid Optimization Enabled Robust CNN-LSTM Technique for Network Intrusion Detection. *IEEE Access* **2022**, *10*, 65611–65622. [CrossRef]
39. Lee, J.; Pak, J.G.; Lee, M. Network Intrusion Detection System using Feature Extraction based on Deep Sparse Autoencoder. In Proceedings of the 2020 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 21–23 October 2020; pp. 1282–1287. [CrossRef]
40. Injadat, M.; Moubayed, A.; Nassif, A.B.; Shami, A. Multi-Stage Optimized Machine Learning Framework for Network Intrusion Detection. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 1803–1816. [CrossRef]
41. Di Mauro, M.; Galatro, G.; Fortino, G.; Liotta, A. Supervised feature selection techniques in network intrusion detection: A critical review. *Eng. Appl. Artif. Intell.* **2021**, *101*, 104216. [CrossRef]
42. Abraham, J.A.; Bindu, V.R. Intrusion Detection and Prevention in Networks Using Machine Learning and Deep Learning Approaches: A Review. In Proceedings of the 2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation, ICAECA 2021, Coimbatore, India, 8–9 October 2021; pp. 2–5. [CrossRef]
43. Mauro, M.D.; Galatro, G.; Liotta, A. Experimental Review of Neural-Based Approaches for Network Intrusion Management. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 2480–2495. [CrossRef]
44. Dong, S.; Xia, Y.; Peng, T. Network Abnormal Traffic Detection Model Based on Semi-Supervised Deep Reinforcement Learning. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 4197–4212. [CrossRef]

45. Pelletier, C.; Webb, G.I.; Petitjean, F. Deep Learning For The Classification Of Sentinel-2 Image Time Series. In Proceedings of the IGARSS 2019—2019 IEEE International Geoscience and Remote Sensing Symposium, Yokohama, Japan, 28 July 2019–2 August 2019. pp. 461–464.
46. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **2018**, *1*, 108–116. [CrossRef]
47. Wireshark. Available online: https://www.wireshark.org/ (accessed on 1 October 2022).
48. CIC. CICFlowMeter. Available online: https://www.unb.ca/cic/research/applications.html#CICFlowMeter (accessed on 1 October 2022).
49. Learn, S.K. train_test_split. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (accessed on 1 October 2022).
50. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980. [CrossRef]
51. Flightsim. Available online: https://github.com/alphasoc/flightsim (accessed on 1 October 2022).
52. AlphaSOC. Available online: https://alphasoc.com// (accessed on 1 October 2022).
53. Ferrag, M.A.; Maglaras, L.A.; Janicke, H.; Smith, R. Deep Learning Techniques for Cyber Security Intrusion Detection: A Detailed Analysis. In Proceedings of the 6th International Symposium for ICS & SCADA Cyber Security Research 2019 (ICS-CSR), Bucharest, Romania, 10–12 September 2019; pp. 126–136. [CrossRef]
54. Nayyar, S.; Arora, S.; Singh, M. Detection System. In *Computer Science and Communications Dictionary*; Springer: New York, NY, USA, 2000; p. 393. [CrossRef]
55. Kim, A.; Park, M.; Lee, D.H. AI-IDS: Application of Deep Learning to Real-Time Web Intrusion Detection. *IEEE Access* **2020**, *8*, 70245–70261. [CrossRef]
56. Alin, F.; Chemchem, A.; Nolot, F.; Flauzac, O. Towards a Hierarchical Deep Learning. In *Machine Learning for Networking*; Springer: Cham, Switzerland, 2020. [CrossRef]
57. MITRE ATT&CK. Available online: https://attack.mitre.org/ (accessed on 1 October 2022).