Deep learning for network intrusion: A hierarchical approach to reduce false alarms

Intelligent Systems with Applications 18 (2023) 200215

Contents lists available at ScienceDirect

Intelligent Systems with Applications

journal homepage: www.journals.elsevier.com/intelligent-systems-with-applications

Deep learning for network intrusion: A hierarchical approach to reduce false alarms ☆

Samuel J. Moore [a,*], Federico Cruciani [a], Chris D. Nugent [a], Shuai Zhang [a], Ian Cleland [a], Sadiq Sani [b]

[a] School of Computing, Ulster University, Shore Road, Newtownabbey, BT370QB, UK
[b] BT Research, Adastral Park, Martlesham, IP5 3RE, UK

ARTICLE INFO

Keywords:
Deep learning
Machine learning
Network intrusion
Network security

ABSTRACT

Computer networks form much of the infrastructure supporting day-to-day life in this digital age. Computer networks, however, are prone to attack and therefore require intrusion detection systems. Intrusion detection systems provide a mechanism to detect network attacks at an early stage and generate alerts. These systems, however, are far from a panacea. Rather, they tend to overwhelm their operators with alerts, which in more than 90% of cases can be false positives. As such, the problem of false positives in intrusion detection systems is a costly issue. This paper presents research to design a hierarchical network intrusion detector, using deep learning, which protects against raising vast numbers of false positives through the design and implementation of a hierarchical NIDS. This paper presents a valuable advancement in performance by reducing the occurrence of false alarms by 87.52%. The research contained in this paper presents three contributions to knowledge. The first of these is the comparison between hierarchical systems and non-hierarchical systems to understand which would yield fewer false alarms. The second contribution is the formulation of a hierarchical approach, which was able to reduce false alarms by 87.52%. Lastly, the proposed hierarchical model was deployed in a live IoT environment, exposed to genuine threats, and the performance in this environment was analysed.

1. Introduction

Networked systems are in a state of continual growth, and are the backbone of much of our modern day infrastructure and services. From smart homes, to connected health, all the way to interconnected self-driving cars, network infrastructure forms a crucial part of our everyday lives. This growth is only likely to accelerate with the emergence of the Internet of Things (IoT), meaning that billions of small devices will be equipped with the ability to communicate with eachother in a seamless manner (Čolaković & Hadžialić, 2018).

This stark increase in the amount of network traffic, combined with an increase in the amount of valuable data and services supported by networked infrastructure, leads to an increase of cyber-security threats due to the massively increased attack surface. These threats can result in severe consequences where large amounts of personal data may be leaked or stolen, such as was the case with the recent attack on Virgin Media (Kleinman, 2020). Attack consequences can be severe, beyond the initial issues of losing data or having service outages. For example, companies can be subject to large fines in the case of a data breach, such as was the case when an international airline was fined £20 million for a data breach in 2018 (Statement, 2019). Moreover, when a data breach occurs it often results in a reduction of the customer's trust in the company, thereby reducing their loyalty (Chen & Jai, 2019). The consequences of a network attack have also been known to propagate into wide areas, if the hacker is clever enough with how they deploy their exploit. Such was the case with the massive Distributed Denial of Service (DDoS) attack of 2016, where networked security cameras were leveraged as a BotNet army to eventually bring down the internet across the entire east coast of the U.S.A. (Hay Newman, 2016). The VMware Global Threat report, (VMware, 2020), indicated that the COVID-19 pandemic had led to a large increase, with 91% of respondents noting an increase in cyber-attacks. This can be attributed to many employees now working from home.

☆ This research is supported by the BTIIC (British Telecom Ireland Innovation Centre) project, funded by BT and Invest Northern Ireland.
* Corresponding author.
E-mail address: s.moore2@ulster.ac.uk (S.J. Moore).

https://doi.org/10.1016/j.iswa.2023.200215
Received 22 October 2021; Received in revised form 7 March 2023; Accepted 9 March 2023
Available online 15 March 2023

With network attacks now growing at an unprecedented rate, network security engineers are tasked with developing and deploying methods of detecting and preventing these attacks. One such way of detecting attacks on a networked system is through the deployment of a Network Intrusion System (NIDS) (Chaabouni et al., 2019), which analyses network traffic and raises alerts if an attack is thought to have occurred. Once an alert is raised, it is usually sent to an operator to process and determine the severity of the threat, meaning that operators can be tasked with dealing with very high volumes of alerts to potential threats (McElwee et al., 2017). These network attacks could be from a variety of sources, and may be varied in their characteristics. The attacks may range from reconnaissance techniques such as port scanning, all the way to Denial of Service (DoS) attacks. These varying attacks manifest themselves differently within the network traffic. For example, with a DoS attack, we may see repeated requests from a malicious attacker to connect to a service on the target system (Schuba et al., 1997). In the case of a port scan attack, we may see half-open TCP handshakes, effectively characterising an attacker's attempt to understand if the service answers the request (Hartpence & Kwasinski, 2020). The scans often occur on ports which are known to have security implications, such as default SSH or HTTP ports. These port scans may also originate from a number of addresses in the network, or be spaced out over a long period of time, while still being part of the same attack (Griffioen & Doerr, 2020). These characteristics make this particular type of attack often difficult to detect. Therefore, with a range of attack types which can often be difficult to identify, research into the development of NIDS is a growing field which continues to gain interest. Moreover, as network usage increases for critical tasks, the attacks increase in tandem, leading to an increased number of security measures being deployed. Inevitably, however, these methods are eventually circumvented by unscrupulous hackers, meaning that security engineers must develop new solutions of detecting and mitigating the new attacks. Therefore, NIDS is a field of growing and evolving research, and is the central subject of this research paper.

Of particular concern within the research area of intrusion detection is the notion of handling false alarms. Large networks may be responsible for generating extremely verbose data, which may range up to millions of events per second. This constitutes a vast sea of data, most of which is benign in nature. The challenge then arises when trying to detect malicious behaviour amid this vast set of data. As a result, any system engineered to detect these minority patterns in data must be extremely sensitive to any potential threat. This then leads to the detection systems generating a vast number of false alarms, which can easily overwhelm a cyber-security operator (Treinen & Thurimella, 2006). Moreover, often more than 90% of the alerts generated can be false, or related to non-malicious root causes (Julisch, 2003). With these challenges in mind, it is then essential that we not only build NIDS which are capable of detecting alerts in a highly sensitive manner, but also that we build NIDS which do not raise a high volume of false alarms.

This paper addresses the pervasive problem of high volumes of false alarms through the introduction of a hierarchical deep learning system, while comparing the results on false alarm reduction against a standard deep learning model. The research found that the hierarchical approach successfully reduced the number of false alarms, versus the standard deep learning approach. This research presents three valuable contributions to knowledge; firstly a comparison of hierarchical versus standard deep learning systems, and secondly a method to reduce false alarms using hierarchical models for intrusion detection. Lastly, the model designed in this experiment is deployed in a live IoT environment, and exposed to a range of genuine threats, in order to determine its effectiveness in a real world scenario.

The remainder of this paper is structured as follows; Section 2 presents the background information on NIDS along with the related work available in the research literature. Section 3 then presents the design of the 3-phase experiment conducted and presented in this paper, followed by the methodology adopted for the research in Section 4. Section 5 then presents the results of the experiment and discusses the performance of each of the models. Finally, Section 6 offers a conclusion and outlines some areas for further research.

2. Background

NIDS are an important aspect of security in networked systems. There are various different types of NIDS which are common in both research and practice. This section will review the main types of NIDS according to the research literature, and also review relevant work in the current literature.

2.1. Knowledge and data driven methods for intrusion detection

Generally, a NIDS falls into one of three main categories; Signature-based (knowledge-based), Anomaly-based (behaviour-based), and Stateful protocol analysis (specification-based) (Liao et al., 2013). More broadly, both Signature-based methods and stateful protocol analysis may be categorised as "knowledge-driven" methods, while anomaly-based methods can be categorised as "data driven" methods. With these two categorisations in mind, some important distinctions need to be considered between these two types of approaches.

A knowledge-based IDS uses a set of rules or a knowledge base which is developed by a domain expert and deployed onto a network environment. The rules may consist of empirical thresholds, exception lists of known bad hosts, lists of firmware deemed as risky by the expert, ping attempts and connections to device, to name but a few. These rules are typically static in nature, are difficult to write, and require extensive knowledge of possible attack scenarios (Hubballi & Suryanarayanan, 2014). The network traffic is observed and compared against the list of rules on a given time interval. If an exception is found, then an alert is raised stating the potential issue. These rules can range from being very generic, to being very specific and can be targeted to identify particular attack types, such as brute force or ping scan.

A data-driven IDS, on the other hand, either learns the pattern of an attack by observing some set of training data, or uses conventional statistical techniques to determine abnormal variation in the data-flow (Viinikka et al., 2009). In a statistical approach, such as with Change-point Detection, no learning is required since the model will analyse the presence of variation in the Probability Density Function given by observing a certain volume or pattern of traffic (Tartakovsky et al., 2013). On the other hand, these approaches are implemented as parametric models where the fine tuning of certain parameters cannot be performed automatically and requires human expertise. In contrast, in the supervised learning-based scenario, the IDS would require a training phase to learn which traffic patterns constitute "normal" traffic, and which patterns constitute anomalous "attack" traffic (Sultana et al., 2019). Assuming the presence of a large labelled dataset, the training process can be performed using machine learning (ML) training algorithms, with the advantage of producing a model as a result of a standard training process. This trained model, usually comprising of a set of weights and activation functions in the case of neural network based approaches, can then be saved and deployed onto a network environment. The saved model is then capable of processing input vectors comprising of live network traffic. These input vectors, when passed through the model, then lead to a classification result of the likelihood that the input vector matches a known attack behaviour. This binary class problem can then often be extrapolated into specifying a specific type of attack, such as Distributed Denial of Service (DDoS) or Port Scans (Koc et al., 2012). This would require the training and implementation of a multi-class model to classify the input vectors and produce a likelihood that the behaviour matches a known attack or that the traffic constitutes normal network behaviour.

2.2. Use of flow-based traffic data

A crucial element when adopting a data-driven approach for NIDS is related to the characteristics of the dataset, including, quality, size, and how it is representative of the conditions that the model is expected to face when it is deployed in the final environment (Ring, Wunderlich, et al., 2019). Otherwise, data-driven methods can generally be exposed to overfitting or in the worst possible case to underfitting. This data, in the context of intrusion detection, generally takes the form of network traffic flows. Generally speaking, a flow would show details of communications sent between hosts on a network. These details will include various pieces of information such as: ports, IP address, protocol, flags, and size.

There are two mainstream approaches to collecting datasets for network intrusion systems; packet-based and flow-based (Ring, Wunderlich, et al., 2019). Packet-based data shows information related to a single packet being transmitted on a network (Seth et al., 2012). This method is known to show deep levels of detail about the network traffic, and is considered the more granular of the two approaches. Flow-based traffic, on the other hand, summarises the packets sent between two hosts in a single connection in the network (Sharma et al., 2018). Therefore, one vector of flow-based traffic data corresponds to many packets. By this, some granular detail is lost in the data related to the details of individual packets.

Both packet based and flow-based data capture have their strengths and applications, and also weaknesses. The flow-based approach does lose some granularity, however, the major advantage it gains is in scalability. Indeed, it is highly impractical, perhaps infeasible, for a large network (such as an enterprise network) to collect and hold all network traffic at a packet level, due to limitations on data storage capacity (Giotis et al., 2014). Moreover, for an anomaly-based model to train on a large set of packet-based data, with the hopes of finding anomalies is a very difficult task. Anomaly detection from network traffic data, is a highly imbalanced task with the majority being the benign class. Most importantly, flow-based traffic is scalable. Enterprise networks can experience up to 500,000 network events per second, so it becomes essential that data is collected and used at a higher level than packet based. Furthermore, many network attacks tend to evolve over multiple packets. Consider, for example, a port scan; this may manifest itself as a device on a network transmitting TCP SYN flags in an attempt to check which ports return an ACK flag, thus indicating an open port. Therefore, a packet will show this behaviour with one or more of these flags activated. Nevertheless, a single port scan is not necessarily indicative of an attack occurring, because scanning for an open port is a regular behaviour of network devices. Rather, *malicious* port scans are the case when a device, or group of devices, issue many thousands of port scans to try to map out the open port space of a network, often zoning in on interesting ports such as Telnet. Moreover, hackers are often clever in how they go about this task, and can slow the port scans down so that they occur over a much longer time span (Ring et al., 2018). This slow scan scenario would make it extremely difficult to detect this attack on a packet-by-packet basis, and requires a much more high-level view of the data. Flow-based data, on the other hand, summarises this information at a much higher level. This means that a classifier, when fed a row of flow-based traffic as an input vector, has a much better picture of whether or not this flow constitutes an intrusion attempt, as opposed to looking at a single packet.

Given the limitations of packet-based network data collection, in terms of data size and scalability, combined with the advantages given for choosing a anomaly-based detection approach in terms of adaptability and scalability, the models in this paper will focus on flow and anomaly based NIDS.

2.3. Deep learning in NIDS

When it comes to selecting an approach to use for an anomaly-based IDS using flow-based traffic data, ML emerges as a potentially viable solution, considering the large volume of data in this context, whereas application of ML approaches in different contexts may be severely limited by the lack of a dataset large enough for training a large model without the risk of incurring the overfitting or underfitting phenomena. Deep neural networks tend to work well only when trained using large datasets, as is the case with network intrusion data. Deep neural networks are able to train on data, extracting representations of the data and extracting efficient features (Roy & Cheung, 2019). This learning benefit is amplified further, when considering the scenario of deep learning, where the more complex network structure allows the model to extract features which are more efficient at discriminating between the target classes. In effect, this makes the learning algorithms less dependent on feature engineering, which might allow the designer to achieve high classification accuracy without needing to know in-depth domain knowledge (Lecun et al., 2015), (Choi et al., 2019). Given the advantages discussed in this section, the models investigated in this paper will focus on deep learning approaches for network intrusion detection.

2.4. Datasets for intrusion detection

There are several datasets currently available for the task of training and evaluating intrusion detection systems. These datasets have a range of varying characteristics including timespan, types of attack, method of collection, emulation and size. As discussed previously, this experiment will only consider flow-based data, and so the dataset chosen for the experiment should be in this format. Several popular flow-based options exist and are used in other investigations, such as CIDDS-001 (Ring et al., 2017), CIDIDS-2017 (Sharafaldin et al., 2018) and UNSW-NB15 (Moustafa et al., 2019). A survey by (Ring, Wunderlich, et al., 2019) evaluated the current options for selecting a dataset for model evaluation and suggested that, while the perfect dataset cannot be found, that a dataset which is a good candidate will; span an appropriate time frame and have appropriately timed attacks. In particular, authors posit that the dataset selected should have a long enough timespan that the data can be placed into subsets, as opposed to cross-validation, because using cross-validation would potentially harm generalisation, furthermore, defining subsets such as week one and week two to split into training and testing helps address the issue of concept drift. It should also be noted, that in the case of using network flow data to train an intrusion detection model, if the data is split using a folded training, validation and testing split, then vectors corresponding to the same attack will appear in both the training and testing set. Primarily, this is because a single attack will manifest over many flow records in any given dataset. Given the conditions described here, the only dataset to the authors' knowledge which satisfies these important criteria is the CIDDS-001 dataset, and thus it has been selected for use in this experiment.

2.5. Data preparation

The CIDDS-001 Coburg Intrusion detection dataset (Ring et al., 2017), (Ring, Schlör, et al., 2019) was used in this experiment. CIDDS-001 is a fully labelled, flow-based dataset for the purposes of evaluating intrusion detection models. The data was generated through the simulation of a small business environment using OpenStack. The data consists of unidirectional NetFlow (Claise et al., 2004) data collected over the course of four weeks. In total, there are 32 million flows contained in the dataset, with 31.3 million of these flows representing OpenStack data and the remaining 0.7 million captured at the external server. The dataset contains a total of 92 attacks, with 70 of these executed inside the OpenStack environment, and 22 aimed at the external server.

Table 1
Table summarising the attacks contained in the CIDDS-001 dataset (Ring, Schlör, et al., 2019).

	OpenStack				External Server			
	Port Scan	Ping Scan	DoS	Brute Force	Port Scan	Ping Scan	DoS	Brute Force
Week 1	16	10	11	5	0	0	0	0
Week 2	8	6	7	7	2	0	0	4
Week 3	0	0	0	0	5	0	0	7
Week 4	0	0	0	0	1	0	0	3

The attacks, and which weeks they occurred on, are summarised in Table 1. This experiment focuses on using the OpenStack dataset to create a multi-class classifier for classifying all five classes in the data.

2.6. Related work

Several works have been published using the CIDDS-001 dataset, these works have been identified by searching for works citing (Ring et al., 2017). These works, their methods and their results will be discussed in this section.

He et al. (2019) developed ensemble feature selection methods aimed at improving classification accuracy within network intrusion detection. This study focused more heavily on the process of feature selection, as opposed to the classification task itself. The authors implemented four classifiers in the experiment; Decision Tree, kNN, ANN and SVM. Performance of classification was measured using these classifiers predicting whether a flow was normal, suspicious, victim, or attacker. The highest accuracy result achieved was 99.40%. Verma and Ranga (2018) also assessed the performance of machine learning classifiers on three datasets, one of which was CIDDS-001. The authors implemented the following models; Random Forest, AdaBoost, Gradient Boosted Machine, Extreme Gradient Boosting, Extremely Randomised Trees, Classification and Regression Trees and a Multi-layer Perceptron. Accuracies as high as 0.995 were recorded in the work. In both of these works, no methodology was given to describe how the dataset imbalance was treated. It is also the case in both of these studies that an imbalanced accuracy metric was used to evaluate the models, meaning that preference is given to the model guessing the much more likely normal class.

Tama and Rhee (2017) developed methods to classify network attacks in IoT networks using a deep neural network combined with three different data sampling techniques. The authors used three datasets to test their models: UNSW-NB15, CIDDS-001 and GPRS. The authors employed three approaches to validating and training the network: cross-validation (comprising of ten folds), repeated cross validation (comprising of five executions of 2-fold cross-validation), and sub-sampling (comprising of ten iterations of the dataset being split into 70% training and 30% testing). The authors validated the results using metrics of accuracy, precision, recall and False Positive Rate (FPR). Results obtained for the models on the CIDDS dataset showed an accuracy of 99.997%. Again, an imbalanced accuracy metric was used for measurement here. Moreover, the authors of this paper developed a binary classification model, and thus the results are not comparable to the multi-class classification results proposed in this paper.

Abdulhammed et al. (2018) investigated the effect of class imbalance in anomaly-based intrusion detection using the CIDDS dataset using deep and shallow learning methods. The authors employed several models through WEKA in the work: three deep neural networks (each with varying model structures), Random Forest, Voting and Stacking. In particular, the authors examined model performance both with and without class balancing techniques. The techniques employed to handle the imbalance were: up-sampling the minority class, down-sampling the majority class, spread sub-sampling, and a class balancer. The metrics used to evaluate the work were, accuracy, geometric mean, false alarm rate, detection rate, and combined metric. Accuracies of 99.99% were achieved here due to the use of an unbalanced accuracy metric. Similar to the previous paper, this paper performed binary classification.

Idhammad et al. (2018) developed a distributed intrusion detection system for cloud environments using data mining techniques. The authors used week 1 only of the CIDDS-001 dataset, and split the data into training and testing using a configuration of 60% for training and 40% for testing. The model is in a layered architecture and makes use of both Naive Bayes and Random Forest classification techniques to detect anomalies. The results were evaluated using the following metrics: accuracy, false positive rate, running time, ROC curves and AUC curves. Overall, the model achieved accuracies of 97.05%, and a false alarm rate of 0.21%. Again, an imbalanced accuracy metric was used to evaluate the model.

Nicholas et al. (2018) employed LSTM models in order to investigate the sequential relationship between flows in detecting network anomalies in flow-based traffic. The authors used the OpenStack portion of the CIDDS-001 dataset to evaluate the performance of their LSTM models. The models performed several classification tasks, which classified the data into: binary classes, a three-class scenario, a five-class scenario and a nine-class scenario. In total, four LSTM models were tested against these classification scenarios, each with a varying model structure. The ADAM loss function was used, along with gradient clipping to control exploding gradients. The authors measured performance using precision and sensitivity metrics, however, a false alarm rate was not presented. The authors found that the LSTM models were adequately able to classify most attacks, however, they struggled to differentiate between port scans and brute force attacks. The authors of Su et al. (2020) proposed a model combining a Bidirectional Long Short-term Memory (BLSTM) model with an attention mechanism. The attention mechanism was employed to screen network vectors generated by the BLSTM model, which was used to generate features for the network, as opposed to hand crafting features. The model achieved an accuracy of 84.25% when tested on the KDD dataset.

Althubiti et al. (2019) also employed a LSTM method to detect network intrusions on the CIDDS-001 dataset. The authors of this work did not classify types of attack, and only classified the flows as normal, suspicious, unknown, attacker and victim. The data was split using a 67% training and 33% testing split, and the authors noted that the accuracy for the LSTM model was 0.8483. The best false alarm rate observed in this paper was the MLP model, which achieved a FAR of 0.1325. The authors of Zhang et al. (2019) proposed a CNN-LSTM model, which combined the CNN Le-Net architecture with an additional LSTM model. When tested on the CICIDS-2017 dataset, this approach achieved an unbalanced accuracy of 99.82%, using a binary classification approach.

Bovenzi et al. (2020) proposed a hierarchical approach based upon deep auto-encoders to perform a two-stage classification. The first stage of the classification identified if in input vector was deemed to be an attack or not, while the second layer determined which type of attack was present in the case of a positive identification from the first layer. The authors were able to achieve a reduction in false alarms by 40% through adopting the hierarchical approach. Nonetheless, the proposed model was designed to operate with packet-based traffic, which does not scale well to enterprise networks, leaving a need to explore hierarchical approaches on a flow-based dataset. Auto-encoders were also used in Shone et al. (2018), where the authors developed a non-symmetric deep auto-encoder (NSDAE), for unsupervised feature learning. In this work, the auto-encoders were stacked into a deep learning hierarchy, which enabled the model to learn complex relationships between the different features of the dataset. When tested on the KDD dataset, this approach achieved an unbalanced accuracy metric of 98.81%. Auto-encoders were also employed in Al-Qatf et al. (2018). In this work, the authors proposed a self-taught learning (STL) approach based on auto-encoders, which was then combined with a Support Vector Machine (SVM) model. The authors found that the auto-encoders were able to automatically learn features effectively from the data, and the model achieved an unbalanced accuracy metric of 95.09% when tested on the

KDD dataset. These works, based on auto-encoders, present valuable research demonstrating that deep learning can successfully engineer features from network intrusion data.

The works surveyed in this Section demonstrate several important efforts in the research area of detecting intrusions in networked systems using data-driven methods. Several of the surveyed works performed binary classification, and in general these results presented lower incidences of false alarms. This suggests that perhaps a hierarchical approach which leveraged this binary classification could reduce the number of false alarms generated, while still providing a multi-class output. In general, few of these methods suggested a suitable data-preparation and training protocol which would allow the model to generalise well in the real world. Another note to make is that the research papers reviewed here did not actually deploy their models in live environments, but rather focused solely on overall classification accuracy using publicly available network intrusion datasets. While, from a training standpoint this is a viable method, without actually deploying the model in a live environment, it is not possible to say for sure if the model has been effectively developed. The risk always exists in this case that the model simply becomes an expert at detecting alerts only of the type found in the dataset, rather than being able to handle the heterogeneous and unpredictable data generated on the internet. Therefore, by validating the model in a live environment, we can begin to understand how it responds to new threats and retrain if needed. Lastly, none of these methods placed particular focus in reducing the number of false positives identified by the model. Rather, these methods were focused solely on the classification performance of the model on public datasets. With these short-comings in mind, there is a gap to address in the research field with respect to reducing false alarms, and also to prove the effectiveness of the model in a live environment. The research conducted in this paper seeks to fill both of those gaps in knowledge associated with this problem.

As such, the research in this paper sought to fill the following gaps in knowledge. First, given that deep learning appears to be an effective method for feature engineering and alert detection in IDS, an understanding is needed as to which methods are most effective. Second, the problem of false alarms is still prevalent in IDS, and only one of the works reviewed was able to report a reduction in false alarms, therefore, research should be conducted to improve this area in IDS. Lastly, none of the reported works in this section were able to deploy their models in a live environment, and as such this remains a gap to be addressed in the literature.

Therefore, given the identified gaps in the literature identified from this review, we pose the following research questions.

1. Which deep learning methods are the most effective in IDS?
2. Can deep learning be leveraged in a hierarchical fashion to develop a method to reduce the number of false alarms raised?
3. Can we deploy the proposed system in a live environment and analyse its performance when faced with genuine threats?

These research questions are answered in the remainder of this paper. In this research, a comparison is presented between common deep learning methods to understand which of them offers the best performance in terms of reducing the number of false alarms raised. Following this, the best performing model is then expanded into a hierarchical approach, which successfully reduced the false alarms versus the non-hierarchical approaches. Lastly, the hierarchical model is deployed in a live IoT environment, and the performance is analysed against real and unseen internet threats.

3. Experimental design

The aim of this experiment is to deploy data-driven deep learning models to detect network intrusion attempts on a networked system with a high degree of accuracy and special consideration for reducing false alarms and then test this model in a live environment. As mentioned earlier in this paper, false alarms are a significant problem in this area of computing, and as such the models proposed throughout this experiment are designed with intent to reduce the vast numbers of these false alarms. The experiment took place over three phases, the first phase aimed to ascertain the applicability of deep learning to the problem of network intrusion, the second phase built upon the findings of the first phase by selecting the best candidate for further research and evolving the hierarchical system from it, and the final phase demonstrated the hierarchical system's live deployment. These three phases are described in detail in the following sections.

3.1. Phase 1: initial investigation

In the initial stage of the experiment, models from three categories were selected for investigation; Convolutional Neural Network (CNN), Multi-layer Perceptron (MLP) and a hybrid CNN-LSTM network. These models were chosen because they are known to be suitable choices for the task of NIDS (Lee et al., 2018, Thakkar & Lohiya, 2020). Within each of these categories a broad range of structures and hyper-parameters were tested, to get a broad view of how each model performed. The aim of this stage of the experiment was to test a range of combinations of hyper-parameters within these models in order to determine how each model performed with respect to the task of detecting intrusions while maintaining a relatively low rate of false alarms. The models in this phase were tested using the CIDDS-001 dataset, which also serves to determine a benchmark for the model developed in phase two to compare against.

3.2. Phase 2: development of a hierarchical model for intrusion detection

Results from the first experiment were examined and analysed to see which approach would provide the potential for further investigation. The results of this analysis then lead to the development of a hierarchical model which is comprised of two separate MLP models which determine if a vector represents a threat in a filtered hierarchical fashion. This phase represents the scientific bulk of the experiment, and it is where the final model was designed and finalised. A specialised high-performance computing facility located in the British Telecoms Ireland Innovation Centre (BTIIC) at Ulster University was used to test and deploy the final model. The model was then tested in detail on the CIDDS dataset, and then the results compared to the initial benchmark set by the models from phase one.

3.3. Phase 3: deployment in real-world environment

It is often seen to be a further area for research within the cyber-security community that models developed using public data need to be validated in a real world scenario. More specifically, there is a shortage of NIDS being deployed in live IoT environments (Chaabouni et al., 2019). After the model was designed and tested against the CIDDS dataset, it was deployed in a live IoT testbed in order to investigate this crucial research area and validate that the model can generalise beyond the bounds of the public data, which is often seen to be a key determinant of a model's success. The live environment used was an Internet of Things (IoT) cyber-security testbed supported by BTIIC and located at the Ulster University in Northern Ireland. While the IoT is generally considered to be distinct from traditional enterprise networking in terms of it's application and device layer, large portions of the network and TCP/IP stack are common across the two fields. Therefore, it is a reasonable approach to employ a model trained on network flow data in this IoT context. The testbed allows the collection and analysis of network flow traffic, which was then fed into a flask API service designed for the model designed in phase two. The model was deployed here and exposed to genuine threats and bots on the internet via several open ports, including SSH, VNC, HTTP and HTTPS. This final phase of

the experiment bridges the divide that is often found between research and engineering, which is an important way to demonstrate real-world impact from the research.

4. Methodology

The research for this paper was designed and conducted in three phases, which were discussed in Section 3. Beyond the design of the phases a more specific methodology was needed to handle the data and design and test the IDS which were developed throughout the course of this research. The remainder of this Section provides details on the specific methodology employed to handle the data, and design of the deep-learning hierarchical intrusion detection system.

4.1. Handling of data imbalance

The CIDDS-001 is an extremely imbalanced dataset with the strong majority class being the "normal" or negative class in this case. The level of imbalance in datasets for classification tasks can be measured using a ρ value (Buda et al., 2018, Johnson & Khoshgoftaar, 2019), which is calculated as shown in Equation (1), where C_i represents a set of samples of class i. The OpenStack CIDDS-001 dataset is very heavily imbalanced, with the majority class (normal) totalling 28,051,906 and the minority class (brute force) totalling 4,992, thereby resulting in a ρ value of 5,619.

$$\rho = \frac{\max_i\{|C_i|\}}{\min_i\{|C_i|\}} \tag{1}$$

Various methods exist for redressing imbalanced data using sampling techniques, for example Random Under Sampling (RUS) and Random Over Sampling (ROS). These methods are generally referred to as data-level techniques, because they either generate new data samples or remove data samples from the training and validation sets. These methods have been shown to increase model performance (Burnaev et al., 2015). Nevertheless, (Johnson & Khoshgoftaar, 2019) in their survey on imbalanced deep learning found that these data-level techniques can often result in over-fitted models with increased training times. The authors also suggested that the benefits of sampling methods are likely only able to perform well on smaller datasets, and will not scale to large datasets, such as the one used in this experiment. For this reason, data-level techniques were avoided in this experiment. The imbalance was instead treated by applying class weights to the training of the models, which in the case of the deep models, modified the loss function to penalise the minority classes more heavily in the case of an incorrect prediction, as given by the weight of a class. The formula for calculating class weights is presented in Equation (2), where w represents the class weight, j represents a given class, s represents the number of data samples, and c represents the number of classes in the data.

$$w_j = \frac{s}{(c \times s_j)} \tag{2}$$

4.2. Data pre-processing

In order to convert the data into a form that could be interpreted by the deep learning models, some data pre-processing scripts were designed in Python using the sklearn library (Buitinck et al., 2013). The CIDDS-001 dataset contains a mix of text, categorical and numeric data, which is summarised in Table 2 as described in Ring et al. (2017).

The OpenStack data from CIDDS-001 is provided in the format of four CSV files, each corresponding to a week of data collected. The data was then split into files containing no more than 250,000 rows in order to allow the Python script to multi-thread the data loading onto multiple CPU cores in order to avoid slow loading times.

Once the data was loaded, features were considered with regard to how they might influence classification. Several columns were removed from the dataset because they do not provide salient information to the task of identifying network attacks. These columns were; IP address information and date of flow. Once these columns were removed, the labels were then separated from the features to enable the model to learn from the features without observing the label. Features in the datasets were then normalised through the use of one-hot encoding and standard scaling as appropriate.

4.3. Training, validation and testing

The OpenStack CIDDS-001 dataset consists of four weeks of data containing five classes. Two of these weeks contain no attacks and consist entirely of the benign class, as presented in Table 1, with the other two weeks each containing samples of all four types of attack. This creates some difficulty when it comes to training and validating the model. In general, three approaches were available to handle this. The first approach would be to use a percentage split, which would separate a given percentage of the data for training and testing. This method randomises its selection of input vectors and does not preserve the order of the data. The next method would be to use cross-fold validation (Kohavi et al., 1995), which trains and tests the model over a given amount of folds. Again, this does not preserve the order of the data. The final approach is to preserve the data in its natural order and use records in their preserved order for training, testing and validation. In detail, this would consist of taking a selection of data, generally occurring over a few days or weeks, and then using this selection of data to train the model, with a separate consecutive selection also chosen for validation. This is particularly import for CNNs and LSTMs which model locality and sequential relationships in the data respectively. The difficulty with this final approach, however, is that in the selection of data for both training and testing there needs to be a representation of each attack contained in the selection, which can often be difficult to produce. In the scenario of detecting network intrusion attempts from network flow data, this is the most optimum scenario. Principally, this is because some network intrusions evolve over the course of many input vectors, for example a DoS attack or a Brute Force attack. This holds especially true in the case of this work, given that some of the models take an input vector of more than one row, and therefore to split the records up and not preserve their order has an influence on the temporal nature of the models. For this purpose, the latter approach was selected as the training, validation and testing method in this work. The allocation of the data is summarised in Table 3.

4.4. Training & validation process

Training a neural network typically involves two key aspects; training and validation. For this purpose, two separate datasets are allocated for each component respectively and used in each epoch of training. At the outset, the weights of the model are given a random initialisation. Within each epoch, which we can represent as i, a batch of training data is selected and shuffled according to batch size k and used to update the weights, W of the model through back-propagation from the loss function. In the case of this work, each model was trained with a maximum of 500 epochs and a batch size of 256, using stochastic gradient descent (SGD) as the loss function. Each epoch allows the model to save new weights if a given metric has improved, thereby increasing the performance of the model. Various options exist here regarding the choice of the metric to be used. For example, we might choose to use the accuracy, and save the model each time this metric improves. The issue with selecting a metric like training accuracy for this process is that it would encourage the model to over-fit on the training set if the weights are saved every i^{th} epoch when the training accuracy improves. Rather, a more optimal approach to take, is to save the model only when the validation loss decreases. This approach means that during epoch i the new model weights W_i, which are determined by the training data, are only saved if a positive effect is seen on the separate

Table 2
Table describing the properties of the available attributes in the CIDDS-001 dataset (Ring, Schlör, et al., 2019).

Name	Description
Src IP	Source IP address
Src port	Source port
Dest IP	Destination IP
Dest port	Destination port
Proto	Transport protocol (e.g. ICMP, TCP, or UDP)
Date first seen	Start time flow first seen
Duration	Duration of the flow
Bytes	Number of transmitted bytes
Packets	Number of transmitted packets
Flags	OR concatenation of all TCP Flags
Class	Class label (normal, attacker, victim, suspicious or unknown)
AttackType	Type of Attack (portScan, dos, bruteForce, —)
AttackID	Unique attack ID. All flows which belong to the same attack carry the same attack id.
Attack Description	Provides additional information about the set attack parameters (e.g. the number of attempted password guesses for SSH brute force attacks)

Table 3
Table summarising the allocation of training, validation and testing data from the CIDDS-001 dataset.

	Week	No. Records	Normal	Port Scan	DoS	Ping Scan	Brute Force
Train	1	8,451,520	7,010,897	183,511	1,252,127	3,359	1,626
Validate	2	1,500,000	1,343,288	852	153,369	732	1,759
Test	2	8,810,733	7,172,041	81,555	1,553,531	1,999	1,607

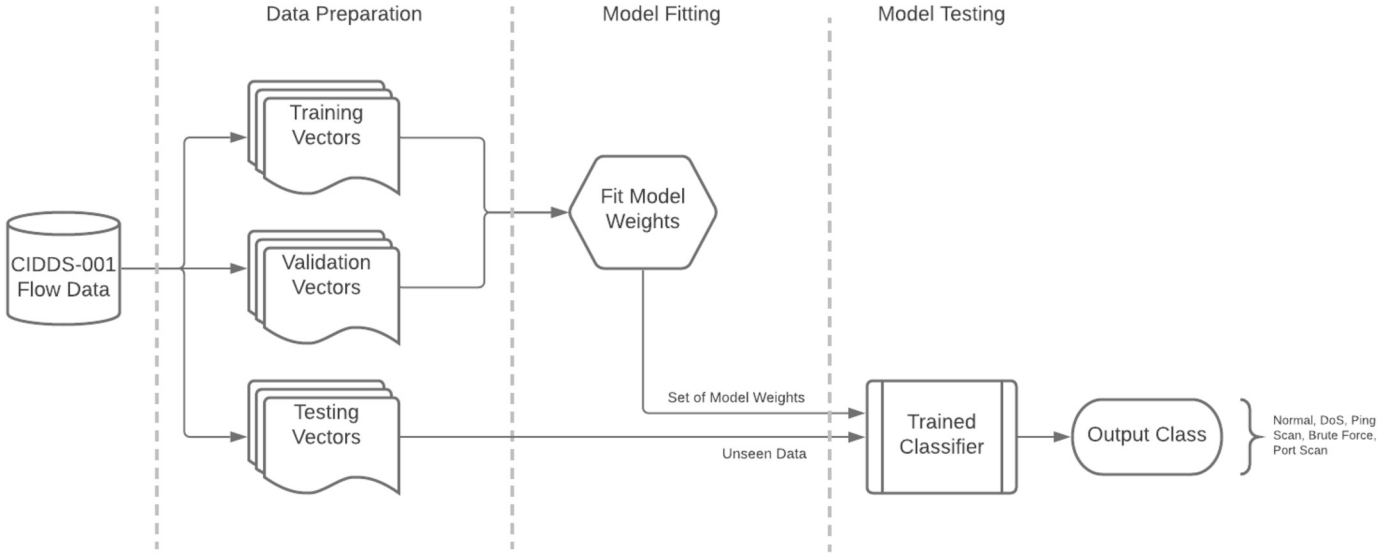

Fig. 1. Process flow depicting the data-splitting and training process employed for each model in the first phase of experimentation.

validation set which has not influenced W_i. Taking this approach protects the model from over-fitting on the training dataset and results in a model that is likely to perform better when moved into production, and as such, it has been applied to this work.

After the data was split into its respective groups for training, testing and validation, and an appropriate loss function approach was designed, each of the models were trained and deployed as depicted in Fig. 1. This same approach was deployed for all of the models used in phase one of the experiment. The model designed in phase two, however, follows a different approach in order to train the classifier which is detailed in the following section.

5. Results & discussion

The experiment was conducted in three distinct phases, as described in Section 3. The results for each phase are documented in the remainder of this Section. The primary objective of the experiment was to establish an understanding of which deep learning model performed best in the expected conditions. In particular, on this experiment, the objective was to produce a model or a system which minimized the level of false positives generated, while also maintaining a high accuracy metric. In some respects, achieving a high accuracy for a model can be a straightforward task, especially when using a fixed dataset. In general terms, if enough training and validation samples are provided during training, the classes are distinct in their nature, and the training is supervised well, then accuracy will be high. Moreover, if accuracy is not satisfactory, then it can usually be improved by simply supplying more training samples. Focusing solely, however, on improving accuracy is not always the most productive approach. Primarily, this is because taking such an approach tends to produce a model which is an expert at handling the dataset it was trained and tested on, but does not generalise well into new environments, ultimately meaning

Table 4
Table of results generated for the CNN, CNN-LSTM and MLP models developed in phase 1 of the experiment.

	Class	Precision	Recall	FPR	F-Measure	Support
CNN	Normal	0.9943	0.9474	0.0526	0.9703	1152289
	Port Scan	0.6996	0.7127	0.2873	0.7061	47760
	Brute Force	0.0111	0.5459	0.4541	0.0217	1132
	DoS	0.9977	0.9800	0.0200	0.9888	265429
	Ping Scan	0.3245	0.9221	0.0779	0.4801	1822
	Macro Avg	0.6054	0.8216	0.1784	0.6334	1468432
	Normal	0.9936	0.9452	**0.0548**	0.9688	1373195
	Port Scan	0.5351	0.6523	0.3477	0.5879	48535
CNN	Brute Force	0.0297	0.5985	0.4015	0.0566	2864
LSTM	DoS	0.9949	0.9826	0.0174	0.9887	291431
	Ping Scan	0.3039	0.9372	0.0628	0.4589	2403
	Macro Avg	**0.5714**	**0.8231**	**0.1769**	**0.6122**	**1718428**
MLP	**Normal**	0.9994	0.9584	**0.0416**	0.9785	7172011
	Port Scan	0.6734	0.9265	0.0735	0.7800	81555
	Brute Force	0.0014	0.1655	0.8345	0.0028	1607
	DoS	0.9527	0.9998	0.0002	0.9756	1553525
	Ping Scan	0.3682	0.8869	0.1131	0.5204	1999
	Macro Avg	**0.5990**	**0.7874**	**0.2126**	**0.6515**	**8810697**

that it may be of limited use outside of the scientific research community (Aleesa et al., 2019). Furthermore, accuracy can be a misleading statistic, in that it does not necessarily account for false alarms being raised. False alarms, as discussed earlier in this paper, can have a high cost in the real world, and thus reducing these false alarms should be as much a focus for designers as improving the accuracy of the model. Therefore, this experiment aimed to establish two things; a model which was accurate and raised fewer false alarms, and a model which would demonstrably perform well in an unknown and untrained environment. Therefore, the metric which is of particular interest to this work is the False Positive Rate (FPR) on the normal class. This is because by aiming to minimize this metric to the lowest possible value, we reduce the number of false alarms raised. In this case, we are not primarily concerned with the attack-type classes, but rather whether or not the system raises an alert or not for an engineer to investigate. Therefore, FPR on the normal class is the metric which we will evaluate model performance with in this work.

5.1. Phase one results

In the first phase of experimentation, three models were investigated, a CNN, CNN-LSTM and a MLP. The summarised results from the test set identified in Table 3 across all of these models are presented in Table 4, within this table the FPR of the normal class is emphasised as the metric which is targeted for optimisation. A range of model hyper-parameters were tested for each model, along with a range of model depths, in order to determine the model which was best suited for the task at hand. Determining the optimum hyper-parameters was not necessarily an objective of this experiment, and therefore is not focused upon in the Results Section. Nevertheless, a grid-search style of approach was conducted to determine these optimum hyper-parameters.

5.1.1. CNN model results

The CNN model achieved a macro f-measure of 63.34. The weighted f-measure and overall accuracy scores are considerably higher, 96.37% and 94.54% respectively, however, they are not presented in the table given that they can at times be a misleading statistic by favouring the prediction of the majority class. In such an imbalanced dataset as this, it is best to view these statistics with a critical view. The CNN achieved an f-measure of 97.03% on the normal class, which is by far the majority class. The lowest performing class was the brute force attack. This is unsurprising, because in the CIDDS-001 dataset the brute force class is the minority class by a large margin. The other lower performing class is the ping scan attack, which again is likely hampered by there being so few training samples. Finally, the CNN observed a False Positive Rate (FPR) of 5.26%, which attests to the proportion of alarms raised by the NIDS which were false.

5.1.2. CNN-LSTM model results

The CNN-LSTM model achieved similar performance to the CNN, achieving a macro f-measure of 61.22%. Weighted f-measure and accuracy for the CNN-LSTM were 94.92% and 94.27% respectively. The CNN-LSTM suffered an even more severe degradation in performance when considering the two minority classes, brute force and ping scan. Nonetheless, the CNN-LSTM still maintained a high performance on the normal class, with an f-measure of 96.88%. The normal class FPR on the CNN-LSTM was slightly worse than the CNN, scoring 5.48%. In general, the CNN-LSTM performed well, however, it underperformed when compared to the CNN.

5.1.3. MLP model performance

The MLP model improved on both the CNN and the CNN-LSTM, achieving a macro f-measure of 65.15%. The weighted f-measure for the MLP was 97.61%, and the accuracy metric was 96.21%, both of which were an improvement on the two previous models. With regard to the minority classes, the MLP was outperformed in the DoS and Brute force classes by the CNN and CNN-LSTM, nevertheless, the MLP outperformed both of these models with respect to Port Scan and Ping Scan classification. The MLP, most interestingly, outperformed both of the other models with respect to classifying the normal class, showing an f-measure of 99.94%. The FPR on the normal class, thereby the rate of false alarms, was also best on the MLP model at 4.16%.

5.1.4. Analysis & discussion

The three models did perform well in general in the task of classifying network attacks, and all three make a suitable candidate for implementation or further research. The most interesting aspect observed in this phase of the experiment was the performance of the MLP on the normal class, versus the CNN and CNN-LSTM. The MLP achieved better FPR, precision and accuracy scores than the other two models. There could be several reasons for the MLP performing better in this context, for example, it may suggest that modelling the sequential relationship in the data is perhaps not important to the task of classifying network intrusion. This may help to explain why the CNN and the CNN-LSTM both performed similarly (in the case of attacks) or slightly worse (in the case of false alarms) to the MLP, given that the MLP takes a single input vector, while the other models take multiple vectors of input. Nevertheless, this observation could form the basis of some interesting

future work in trying to determine the impact of the sequential relationships of the vectors. The superior performance of the MLP also raised an interesting question - does the MLP generally classify the normal class more accurately than other ML models? From a theoretical standpoint, it makes sense that if the model was trained and tested on only normal and abnormal class labels (thus transmuting the problem into a binary classification one) would result in better performance on these two classes, because the imbalance would be reduced on the training and testing data. Moreover, reducing the number of labels that the classifier needs to select from in the output layer also increases the chances of the model making a correct prediction against the normal class. Ultimately, a model which performs better in this class, while still performing well in the other classes, will raise fewer false alarms and therefore waste less time for the human operator. Lastly, if the MLP model can accurately classify the normal class, even when choosing from a range of 5 possible classes, would the performance be improved even further if the model only had two possible classes, normal and attack? These important questions formed the basis of the research conducted in phase two of the experiment, which is documented in the following section.

5.2. Design & implementation of a hierarchical NIDS

A hierarchical model was designed and deployed in phase two of the experiment which built upon the findings observed in phase 1. A novel model was then constructed which took advantage of the strengths observed in the MLP. Naturally, it would also be possible to engineer a hierarchical approach using the CNN or the CNN-LSTM at either stage of the hierarchy, but at this point the MLP has been selected due to its superior performance in generating a lower FPR than the other models. As such, a hierarchical framework was developed which used two MLP models to classify vectors in a two stage approach. The first stage of this approach consisted of a trained MLP making a binary classification on a vector as to whether it pertained to an attack or not. Then, if an attack was deemed likely (as given by the SoftMax probability) the vector was passed into the second stage of the model which would make a further classification on the vector as to which specific type of attack was detected.

The structure for this hierarchical NIDS framework is shown in Fig. 1. The process begins with an input vector $X = f_0, f_1 \ldots f_{12}$, where X represents a single feature vector consisting of 13 features represented by f. This feature vector X is then given as input to a 7-layer deep learning MLP model which produces output probability $p(k, j)$, which is a tuple of Softmax probability is denoted by p and the reciprocal probabilities of each output label (0 and 1, meaning normal or attack) are denoted by k and j. Next the value of p is tested against an empirically identified threshold, which was identified as 0.8. If the value of p is beneath this threshold, then the vector is labelled as normal in output label y. The test vector x is then supplied as input to the 5-class 7-layer MLP classifier. This 5-class classifier will give an output label y which corresponds to one of the attack classes. It is worth noting that even at the 5-class classification stage, it is still possible that a vector may be labelled as normal. This second step in the hierarchy was designed as an attempt of further reducing the likelihood of producing false alarms, by processing all ambiguous cases through a second model in the second layer before assigning the final classification (Fig. 2).

Given that a set of records are expected to be filtered out in the hierarchical model, it was necessary to adopt a different training protocol to prepare the MLP model at the second layer. Primarily, this is because only high-likelihood vectors will be passed into the second layer of the model. This means that the imbalance of the dataset shifts completely in the opposite direction, where the majority class is now the attack class. The process for achieving this is depicted in Fig. 3. As such, the model should be exposed in training to the conditions that it is expected to operate in when in testing. To accommodate for this, this meant passing both the training and the validation sets through the binary classifier in order to retrieve a SoftMax probability and to filter out records which fell beneath the chosen threshold. Then this filtered set, which now includes the SoftMax probability, were used to train the weights of the model in the second layer of the hierarchical framework.

5.3. Phase two results

Given the findings established from phase one, further research was conducted in the second phase of experimentation to determine if the reduction in false alarms could be enhanced by engineering a new framework which leveraged the natural qualities of the MLP in this area.

5.3.1. Binary MLP performance

One of the research questions raised after analysing the results from the first phase was to understand if the MLP's ability to correctly classify the normal class, thereby reducing false alarms, could be enhanced by turning the classification task into a binary format. As such, a 7-layer MLP was designed to perform binary classification between attacks and normal traffic using the CIDDS-001 dataset. The process for training the model adhered to the same process deployed in the first phase of experimentation, which is represented in Fig. 1. The results for this model are displayed in Table 5. A large improvement can be seen here with respect to the correct classification of the normal class, with an f-measure of 99.59% for the normal class. Crucially, a large improvement was also seen in the model's performance with respect to false alarms, with a score of 0.69% for FPR on the normal class. This recall metric is a substantial improvement compared with all of the other models from the first phase of experimentation, with the best FPR from the previous phase being 4.16% on the MLP.

5.3.2. Hierarchical NIDS performance

The need to classify into individual attack classes while still maintaining the benefits of the MLP binary classifier culminated in the design of a unique two-layer framework for intrusion detection. The design and operation of this framework were described in detail in Section 5.2. This model was then trained and tested using the sample training/validation/testing split on the CIDDS-001 dataset from the previous phases (Table 6). The training process itself, however, was different given the unique structure of the framework. This training process is also depicted and described in Section 5.2.

The hierarchical network intrusion detection system (H-NIDS) performed better than all other models tested in this experiment on the CIDDS-001 dataset, which is presented in Table 7, where the highest performance in each category across all models is highlighted in bold text. The hierarchical model outperformed the other models tested in precision, false alarms raised and f-measure. The total macro f-measure was 68.43%, which was more than 3% improved from the best model in phase one. The weighted f-measure and accuracy also improved for this model versus those in phase one, with 99.40% and 99.38% respectively. Similar struggles were found in the minority classes, but the results were comparable with the other models tested in this experiment. Most crucially, the H-NIDS model was able to maintain its large improvement in correctly classifying the normal class. The normal class f-measure was 99.64%, which is more than 4% improved than the best model from phase one. Lastly, and most importantly, the H-NIDS model managed to maintain its much-improved false alarm ratio, by achieving a FPR of 0.59% on the normal class, which was again more than 4% improved compared to the best model from phase one.

5.3.3. Comparison to related work

By way of comparison, the proposed model achieved the best performance in terms of FAR when compared against the related work. Of those works using the CIDDS-001 in a 5-class classification approach, the lowest FPR was observed in Althubiti et al. (2019), which achieved a FPR of 13.25%, while the FPR observed in this work was 0.59%. Moreover, Bovenzi et al. (2020) reported a reduction in false alarms of 40%,

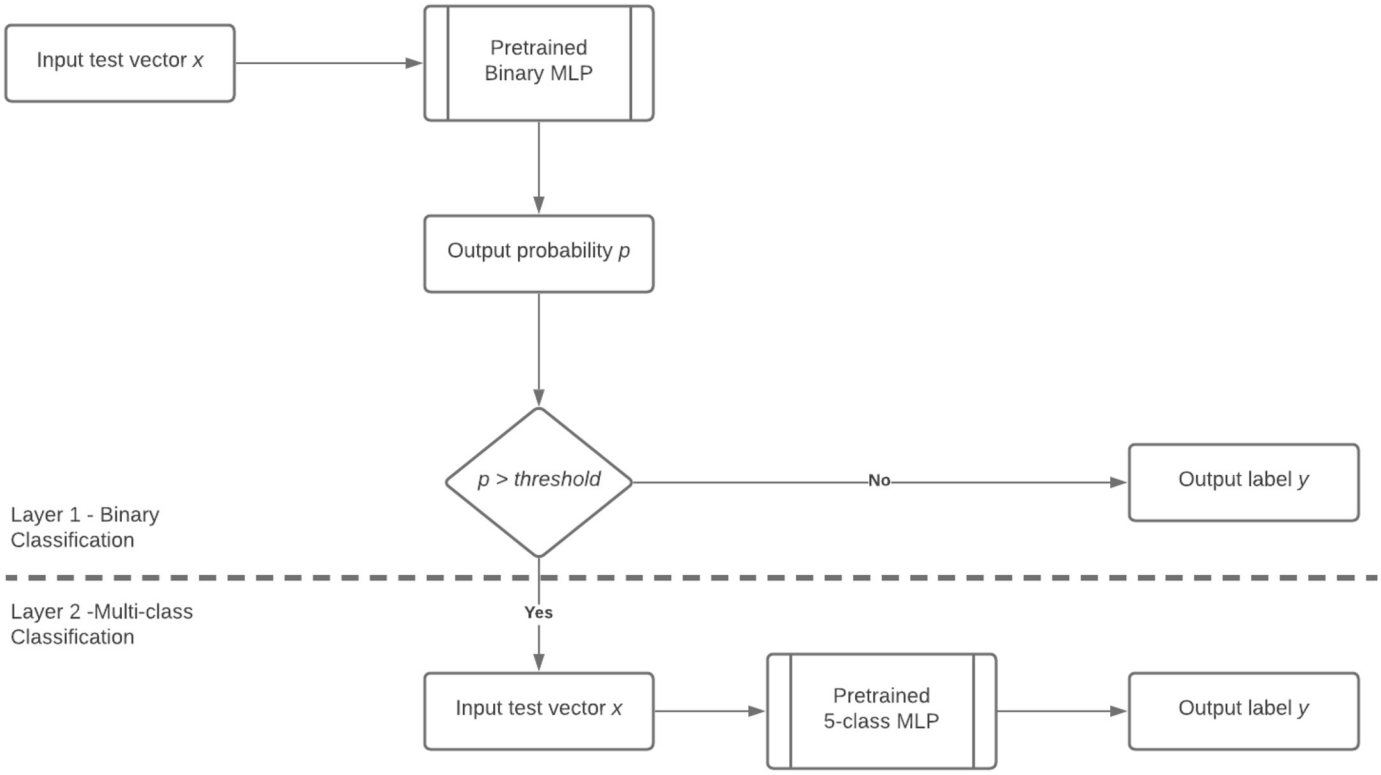

Fig. 2. Process flow for the hierarchical approach developed in phase two of experimentation.

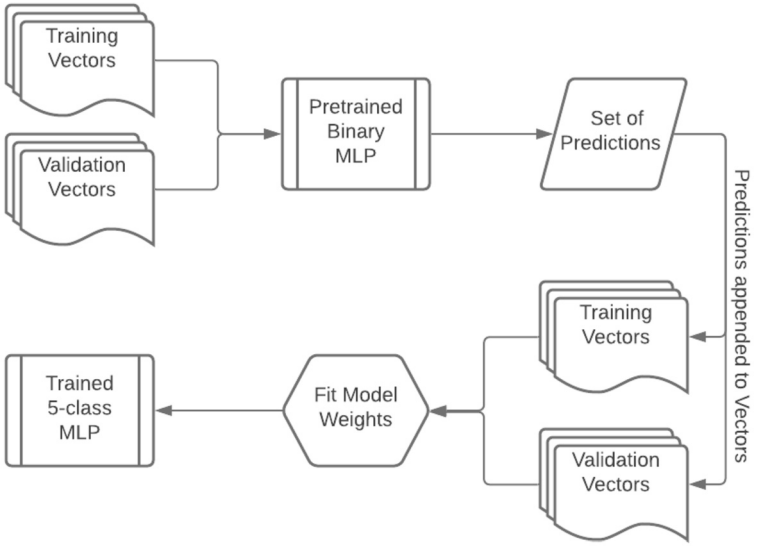

Fig. 3. Process flow describing the process of training the second layer of the hierarchical model.

Table 5
Results for the Binary 7-Layer MLP model.

Class	Precision	Recall	FPR	f1-score	Support
Normal	0.9988	0.9931	**0.0069**	0.9959	7172011
Attack	0.9705	0.9948	0.0052	0.9825	1638686
Macro Avg	0.9846	0.9939	0.0061	0.9892	8810697

Table 6
Table of results for the hierarchical NIDS on the CIDDS-001 dataset.

Class	Precision	Recall	FPR	f1-Score	Support
Normal	0.9988	0.9941	**0.0059**	0.9964	7172011
Port Scan	0.9131	0.8792	0.1208	0.8958	81555
Brute Force	0.1150	0.4661	0.5339	0.1845	1607
DoS	0.9796	0.9997	0.0003	0.9895	1553525
Ping Scan	0.3813	0.3327	0.6673	0.3553	1999
Macro Avg	**0.6776**	**0.7344**	**0.26564**	**0.6843**	**8810697**

Table 7
Table of results comparing all four models proposed in this paper for precision, recall, false alarms and f-measure. Best results in each category are highlighted in bold type.

Model	Precision	Recall	False Alarms	F1
CNN	0.6054	0.8216	0.0526	0.6334
CNN-LSTM	0.5714	**0.8231**	0.0548	0.6122
MLP	0.5990	0.7874	0.0416	0.6515
Hierarchical Model	**0.6776**	0.7344	**0.0059**	**0.6843**

while the model proposed in this research reduced the number of false alarms by 87.52%. As such, the model presented in this paper achieved a higher general performance in terms of the FPR than the approaches in the related work.

5.3.4. Analysis & discussion

Overall, in this phase of the experiment, a novel framework was established which greatly enhanced both the performance of the DL algorithm in detecting attacks, but also managed to significantly improve the models handling of the normal class, thereby reducing the number of false alarms. While the improvement of 4% might seem nominal, when we observe the numbers in detail we can understand how important this improvement is with respect to false alarms. Using the MLP model from phase one, which was the best performer in this area, a total number of 381,485 false alarms being generated by the model. By contrast, the H-NIDS only generated a total of 42,315 false alarms, which is a total reduction in false alarms of 339,170, thereby accounting for an overall decrease in false alarms of 87.52%.

5.4. Phase three - live deployment

In order to effectively test the generalisation capabilities of the model, it was deployed into a live IoT security testbed located at Ulster University in Northern Ireland. This testbed consisted of live IoT edge nodes, comprised of Raspberry Pi machines, which were located in a distributed fashion across Northern Ireland and Spain. These edge nodes had exposed services on ports 443, 80 and 22, making them a popular target for various internet crawlers and potential hackers. The edge nodes also connected a range of IoT sensors, including temperature, barometer and light sensors. The edge nodes generated a regular footprint of traffic by sending these IoT readings to a central node, located at Ulster University. The testbed collected network traffic using Tranalyzer (Burschka & Dupasquier, 2017), which was deployed on each of the Raspberry Pi devices to collect the traffic. A process was then constructed to automatically feed the traffic from Tranalyzer into the Hierarchical IDS, which was deployed on the central server node to analyse the incoming traffic collected by Tranalyzer. If the IDS detected an alert, it generated an alert record and forwarded it to a dashboard component for a cyber-security analyst to perform further analysis on. This availability of information then can assist with the development and refinement of rules to manage devices on the network. This section will analyse the effectiveness of the model in the real-world by analysing the alerts generated by the IDS, which was only trained on the CIDDS data, from real internet traffic generated by the testbed. This part of the research bridged an important divide between the theoretical component of IDS development and the actual applicability of the model in a real scenario, helping to prove the effectiveness of the technique in a non-research scenario.

A qualitative analysis was then performed on the deployed IDS to understand how accurate the alerts were that it was generating. Given that the IDS was deployed in a live and open environment, there was no labelled data. This meant that the analysis in this section was not capable of producing actual numbers of true and false positives, given that there is no ground truth available. This meant that the best way to analyse the performance was by looking at the alerts generated by the IDS and comparing them against public tools available which keep records of the behaviour of IP addresses on the internet. Furthermore, dependant upon the type of attack, we may be able to compare the IDS alert against the logs of some other reputable intrusion detectors such as fail2ban. The data analysed in this section pertains to the observations made upon a full day's deployment of the IDS on the test bed on the 2^{nd} of November 2020.

5.4.1. Comparison with Internet storm centre

The Internet Storm Centre (ISC) is a popular online tool used to keep records of the behaviour of IP addresses on the internet. The ISC monitors this behaviour by collating reports submitted by users when they note suspicious activity from a given IP address. Therefore, IP addresses which show a history of scanning or attempting to exploit other services on the internet can be assumed to be potential threats. All alerts generated by the IDS were searched on the ISC to ascertain if the alert was from a known threat or not. In all cases all of the alerts generated by the IDS on our testbed were tied back to a history of other alerts on ISC. While this is not conclusive evidence of the system's true performance in the real world, it is an important indicator that the IDS is generating realistic alerts in an environment that it was not specifically trained on. Moreover, the fact that none of the alerts generated related to a non-threat reinforces the finding in this work that the hierarchical model is effective in reducing the problematic issue of false alarms in network intrusion detection.

5.4.2. Multi-step attacks

On a deeper dive of the data, some threats when analysed were from related sources. The data for these threats is presented in Table 8. For privacy reasons the observed IP addresses were anonymised as IP A, B and C. Each of these sources generated more than one type of attack on our testbed, according to the IDS that we deployed. By analysing the types of threats detected for each IP source, we can see the pattern adheres to the typical early stages of a multi-step attack. In more detail, this means that we were able to observe that an attacker, or a bot, first performed a reconnaissance phase such as a port scan to determine services that were open, and then followed this up with a Brute Force

Table 8
Table showing potential multi-step attacks observed from the deployment of the IDS.

IP Address	Country	# of Alerts	Types of Alerts
IP_A	-	196	Port Scan, Brute Force
IP_B	md	14	Port Scan, DoS
IP_C	cn	6	Port Scan, DoS

or a DoS attempt. Again, in all cases, the analysis was corroborated by findings on the ISC.

6. Conclusion and future work

The research presented in this paper provides progress towards engineering more trustworthy and feasible intrusion detection systems which effectively handle the pervasive problem of high volumes of false alarms. This paper presented three major contributions to the research field on intrusion detection systems. Firstly, the research evaluated the efficacy of several machine learning approaches, focusing specifically on deep learning. Within this, a conclusion was drawn that a MLP model was a high performer versus the other models tested. Secondly, the research showed that adopting a hierarchical approach to intrusion detection yields higher performance in reducing false alarms, reducing the total number of false alarms by 87.52%. Lastly, this work presented the development of a novel hierarchical intrusion detection system which combined two separately trained deep MLP models to classify threats on a network, and steps towards proving the effectiveness of this model in the real world was taken by implementing the model in a live, open and real IoT network testbed. The novel MLP-based hierarchical model proposed in this research was capable of achieving high classification performance (73.44% recall), while balanced with a low FPR of 0.59%.

The work raises several viable areas for continued research. Firstly, the hierarchical model yielded impressive results in reducing the number of false alarms generated by the model, nonetheless, the classification accuracy is still an area for improvement, especially with respect to the minority class. As such, more research should be undertaken to investigate methods of improving the classification of the minority class without damaging the recall of the normal class, indeed, Multi-task Learning may prove a suitable method for enhancing the performance of this model. The results in this paper also concerned using deep learning based models, it may also be of benefit to conduct a study using approaches other than deep learning. Furthermore, the model was deployed into a live environment, which allowed the researchers to gain valuable insight into how the model would perform on data and in an environment that it had never seen before. While these results are initially very impressive, they are by no means conclusive. More research is needed in this live environment to understand how the model behaves and performs with respect to detecting known attacks and handling false alarms. One potential direction here would be to deploy it alongside another well-researched IDS to determine if they raise similar kinds of alerts. A further future study could also be conducted to develop a system based on ensembles to determine if a reduction in false alarms can be observed. The ensembles may be deployed in a hierarchical fashion. Lastly, while the model performs well in detecting the types of attacks that it was trained on, more work is required to introduce some new attack types to the model, or perhaps to splinter the model into several different versions capable of detecting these other attacks.

CRediT authorship contribution statement

Samuel J. Moore: Conceptualization, Formal analysis, Investigation, Methodology, Software, Writing – original draft. **Federico Cruciani:** Conceptualization, Software, Writing – review & editing. **Chris D. Nugent:** Conceptualization, Supervision, Writing – review & editing. **Shuai Zhang:** Conceptualization, Supervision, Writing – review & editing. **Ian Cleland:** Conceptualization, Supervision, Writing – review & editing. **Sadiq Sani:** Supervision, Writing – review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

References

Abdulhammed, R., Faezipour, M., Abuzneid, A., & AbuMallouh, A. (2018). Deep and machine learning approaches for anomaly-based intrusion detection of imbalanced network traffic. *IEEE Sensors Letters*, *3*, 1–4. https://doi.org/10.1109/lsens.2018.2879990.

Al-Qatf, M., Lasheng, Y., Al-Habib, M., & Al-Sabahi, K. (2018). Deep learning approach combining sparse autoencoder with svm for network intrusion detection. *IEEE Access*, *6*, 52843–52856.

Aleesa, A. M., Zaidan, B. B., Zaidan, A. A., & Sahar, N. M. (2019). *Review of intrusion detection systems based on deep learning techniques: Coherent taxonomy, challenges, motivations, recommendations, substantial analysis and future directions, Vol. 3*. London: Springer.

Althubiti, S. A., Jones, E. M., & Roy, K. (2019). LSTM for anomaly-based network intrusion detection. In *2018 28th international telecommunication networks and applications conference, ITNAC 2018* (pp. 1–3).

Bovenzi, G., Aceto, G., Ciuonzo, D., Persico, V., & Pescapé, A. (2020). A hierarchical hybrid intrusion detection approach in iot scenarios. In *GLOBECOM 2020 - 2020 IEEE global communications conference* (pp. 1–7).

Buda, M., Maki, A., & Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, *106*, 249–259.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., Vanderplas, J., Joly, A., Holt, B., & Varoquaux, G. (2013). API design for machine learning software: Experiences from the scikit-learn project. In *ECML PKDD workshop: Languages for data mining and machine learning* (pp. 108–122). http://arxiv.org/abs/1309.0238.

Burnaev, E., Erofeev, P., & Papanov, A. (2015). Influence of resampling on accuracy of imbalanced classification. In *Eighth international conference on machine vision (ICMV 2015), Vol. 9875*, Article 987521.

Burschka, S., & Dupasquier, B. (2017). Tranalyzer: Versatile high performance network traffic analyser. In *2016 IEEE symposium series on computational intelligence, SSCI 2016*.

Chaabouni, N., Mosbah, M., Zemmari, A., Sauvignac, C., & Faruki, P. (2019). Network intrusion detection for IoT security based on learning techniques. *IEEE Communications Surveys and Tutorials*, *21*, 2671–2701. https://doi.org/10.1109/COMST.2019.2896380.

Chen, H. S., & Jai, T. M. C. (2019). Trust fall: Data breach perceptions from loyalty and non-loyalty customers. *The Service Industries Journal*, *41*(13–14), 1–17. https://doi.org/10.1080/02642069.2019.1603296.

Choi, Y.-H., Liu, P., Shang, Z., Wang, H., Wang, Z., Zhang, L., Zhou, J., & Zou, Q. (2019). Using deep learning to solve computer security challenges: A survey. http://arxiv.org/abs/1912.05721.

Claise, B., Sadasivan, G., Valluri, V., & Djernaes, M. (2004). Cisco systems netflow services export version 9.

Čolaković, A., & Hadžialić, M. (2018). Internet of things (iot): A review of enabling technologies, challenges, and open research issues. *Computer Networks*, *144*, 17–39.

Giotis, K., Argyropoulos, C., Androulidakis, G., Kalogeras, D., & Maglaris, V. (2014). Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Computer Networks*, *62*, 122–136. https://doi.org/10.1016/j.bjp.2013.10.014.

Griffioen, H., & Doerr, C. (2020). Discovering collaboration: Unveiling slow, distributed scanners based on common header field patterns. In *Proceedings of IEEE/IFIP network operations and management symposium 2020: Management in the age of softwarization and artificial intelligence, NOMS 2020*.

Hartpence, B., & Kwasinski, A. (2020). Combating TCP port scan attacks using sequential neural networks. In *2020 international conference on computing, networking and communications, ICNC 2020* (pp. 256–260).

Hay Newman, Lily (2016). Friday's East coast Internet outage is a major DDOS attack | WIRED. https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/. (Accessed 24 January 2022).

He, W., Li, H., & Li, J. (2019). Ensemble feature selection for improving intrusion detection classification accuracy. *ACM International Conference Proceeding Series*, 28–33. https://doi.org/10.1145/3349341.3349364.

Hubballi, N., & Suryanarayanan, V. (2014). False alarm minimization techniques in signature-based intrusion detection systems: A survey. *Computer Communications*, *49*, 1–17. https://doi.org/10.1016/j.comcom.2014.04.012.

ICO (2019). Statement: Intention to fine British Airways £183.39m under GDPR for data breach. https://ico.org.uk/about-the-ico/news-and-events/news-and-blogs/2019/07/ico-announces-intention-to-fine-british-airways/. (Accessed 14 September 2021).

Idhammad, M., Afdel, K., & Belouch, M. (2018). Distributed intrusion detection system for cloud environments based on data mining techniques. In *Procedia computer science*.

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, *6*. https://doi.org/10.1186/s40537-019-0192-5.

Julisch, K. (2003). Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security*, *6*, 443–471. https://doi.org/10.1145/950191.950192.

Kleinman, Z. (2020). BBC, Virgin Media data breach affects 900,000 people. https://www.bbc.co.uk/news/business-51760510. (Accessed 14 September 2021).

Koc, L., Mazzuchi, T. A., & Sarkani, S. (2012). A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier. *Expert Systems with Applications*, *39*, 13492–13500. https://doi.org/10.1016/j.eswa.2012.07.009.

Kohavi, R., et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai, volume 14, Montreal, Canada* (pp. 1137–1145).

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444. https://doi.org/10.1038/nature14539.

Lee, B., Amaresh, S., Green, C., & Engels, D. (2018). Comparative study of deep learning models for network intrusion detection. *SMU Data Science Review*, *1*, 8.

Liao, H. J., Richard Lin, C. H., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, *36*, 16–24. https://doi.org/10.1016/j.jnca.2012.09.004.

McElwee, S., Heaton, J., Fraley, J., & Cannady, J. (2017). Deep learning for prioritizing and responding to intrusion detection alerts. In *Proceedings - IEEE military communications conference MILCOM 2017-octob* (pp. 1–5).

Moustafa, N., Hu, J., & Slay, J. (2019). A holistic review of network anomaly detection systems: A comprehensive survey. *Journal of Network and Computer Applications*, *128*, 33–55. https://doi.org/10.1016/j.jnca.2018.12.006.

Nicholas, L., Ooi, S. Y., Pang, Y. H., Hwang, S. O., & Tan, S. Y. (2018). Study of long short-term memory in flow-based network intrusion detection system. *Journal of Intelligent & Fuzzy Systems*, *35*, 5947–5957. https://doi.org/10.3233/JIFS-169836.

Ring, M., Wunderlich, S., & Ring, M. (2017). Technical Report CIDDS-001 data set.

Ring, M., Landes, D., & Hotho, A. (2018). Detection of slow port scans in flow-based network traffic. *PLoS ONE*, *13*, 1–18. https://doi.org/10.1371/journal.pone.0204507.

Ring, M., Schlör, D., Landes, D., & Hotho, A. (2019). Flow-based network traffic generation using generative adversarial networks. *Computers & Security*. https://doi.org/10.1016/j.cose.2018.12.012.

Ring, M., Wunderlich, S., Scheuring, D., Landes, D., & Hotho, A. (2019). A survey of network-based intrusion detection data sets. *Computers & Security*, *86*, 147–167. https://doi.org/10.1016/j.cose.2019.06.005.

Roy, B., & Cheung, H. (2019). A deep learning approach for intrusion detection in Internet of things using bi-directional long short-term memory recurrent neural network. In *2018 28th international telecommunication networks and applications conference, ITNAC 2018* (pp. 1–6).

Schuba, C. L., Krsul, I. V., Kuhn, M. G., Spafford, E. H., Sundaram, A., & Zamboni, D. (1997). Analysis of a denial of service attack on TCP. In *Proceedings of the IEEE computer society symposium on research in security and privacy* (pp. 208–223).

Seth, A., Singla, A. R., & Aggarwal, H. (2012). *Contemporary Computing*, *306*.

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP 2018 - proceedings of the 4th international conference on information systems security and privacy 2018-janua* (pp. 108–116).

Sharma, R., Singla, R. K., & Guleria, A. (2018). A new labeled flow-based DNS dataset for anomaly detection: PUF dataset. *Procedia Computer Science*, *132*, 1458–1466. https://doi.org/10.1016/j.procs.2018.05.079.

Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*, 41–50.

Su, T., Sun, H., Zhu, J., Wang, S., & Li Bat, Y. (2020). Deep learning methods on network intrusion detection using nsl-kdd dataset. *IEEE Access*, *8*, 29575–29585.

Sultana, N., Chilamkurti, N., Peng, W., & Alhadad, R. (2019). Survey on SDN based network intrusion detection system using machine learning approaches. *Peer-to-Peer Networking and Applications*, *12*, 493–501. https://doi.org/10.1007/s12083-017-0630-0.

Tama, B. A., & Rhee, K.-H. (2017). Attack classification analysis of IoT network via deep learning approach. *Research Briefs on Information & Communication Technology Evolution*, *3*, 1–9. https://doi.org/10.22667/ReBiCTE.2017.11.15.015.

Tartakovsky, A. G., Polunchenko, A. S., & Sokolov, G. (2013). Efficient computer network anomaly detection by changepoint detection methods. *IEEE Journal of Selected Topics in Signal Processing*, *7*, 4–11. https://doi.org/10.1109/JSTSP.2012.2233713.

Thakkar, A., & Lohiya, R. (2020). *A review on machine learning and deep learning perspectives of IDS for IoT: Recent updates, security issues, and challenges, 0123456789*. Netherlands: Springer.

Treinen, J. J., & Thurimella, R. (2006). A framework for the application of association rule mining in large intrusion detection infrastructures. *Lecture Notes in Computer Science*, *4219*, 1–18. https://doi.org/10.1007/11856214{_}1.

Verma, A., & Ranga, V. (2018). On evaluation of network intrusion detection systems: Statistical analysis of CIDDS-001 dataset using machine learning techniques. *Pertanika Journal of Science & Technology*, *26*, 1307–1332.

Viinikka, J., Debar, H., Mé, L., Lehikoinen, A., & Tarvainen, M. (2009). Processing intrusion detection alert aggregates with time series modeling. *Information Fusion*, *10*, 312–324. https://doi.org/10.1016/j.inffus.2009.01.003.

VMware (2020). Carbon black global threat report. Technical Report June, VMWare, Palo Alto, USA, Palo Alto, USA: VMWare. Accessed: 2021-9-14.

Zhang, Y., Chen, X., Jin, L., Wang, X., & Guo, D. (2019). Network intrusion detection: Based on deep hierarchical network and original flow data. *IEEE Access*, *7*, 37004–37016.