Data Balancing and Hyper-parameter Optimization for Machine Learning Algorithms for Secure IoT Networks

by

Omar Elghalhoud

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis contains some sections that are based on co-authored materials by Prof. Kshirasagar Naik, Dr. Marzia Zaman, Nishith Goel, and Ricardo Manzano S. [19], [20]. I am the primary contributor and first author of these co-authored materials.

Abstract

Nowadays, many industries rely on Machine Learning (ML) algorithms and their ability to learn from existing data to make inferences about new unlabeled data. Applying ML algorithms to the network security domain is not new. However, without proper data pre-processing and proper optimization of the hyper-parameters (HPs) of ML algorithms, these algorithms might not achieve their full potential. Furthermore, attacks on network infrastructures come in a variety of forms and at different frequencies. Cyber-security experts often require the help of an automated process that filters and classifies attacks. To apply specific preventive measures for securing networks, the classification of the attack type is key. Many ML models have been proposed as a base for Network Intrusion Detection (NID) systems. However, their performance varies based on multiple factors. For instance, an ML model fitted on a highly imbalanced dataset may be biased toward over-represented attack types. On the other hand, paying attention only to the ML model's performance in the minority classes can negatively affect its performance in the majority classes or overall performance. This research proposes a framework that applies pre-processing steps, including data balancing, and utilizes optimization techniques to tune the HPs of random forest, gradient boosting machine, and deep neural networks. The conducted experiments in this research provide a performance comparison between three different optimization algorithms: Tree-structured Parzen Estimator (TPE), Bayesian Optimization and Hyperband (BOHB), and Particle Swarm Optimization (PSO). The research results show that through data balancing and optimization of the HPs and architecture of deep neural networks, their performance can improve significantly: false alarm rate of 0% and only 1.79% using the BoT-IoT and the ToN-IoT benchmark datasets, respectively.

To address the issue of imbalanced datasets, this research gives a data balancing algorithm and compares its performance to other existing approaches that use: Random Over-Sampling (ROS), Synthetic Minority Oversampling TEchnique (SMOTE), Adaptive Synthetic Sampling (ADASYN), and Generative Adversarial Networks (GAN). The data balancing algorithm is combined with Convolutional Neural Networks (CNN) to extract spatial features and classify the different attack types. Using the NSL-KDD and the BoT-IoT datasets for benchmarking, the proposed system achieves high performance in the minority classes: recall scores of 70.50% and 72.08% on the User to Root (U2R) and Remote to Local (R2L) attack classes of the NSL-KDD dataset, respectively, while maintaining an overall False Alarm Rate (FAR) of 6.50% and a recall of 90.46% on the binary classification task. The proposed system scores a weighted average F1-Score of 99.45% on the multi-class classification task using the BoT-IoT dataset.

Acknowledgements

Reaching the end of a two-year journey of research and study, I do not think I would have made it without the support of my family, colleagues, and mentors.

I would like to express my gratitude to my supervisor Dr. Naik for all his support, patience, and guidance throughout my journey. Also, I would like to thank Dr. Marzia Zaman for her valuable inputs. Thanks go to all my lab teammates and my friends. Special thanks go to my friend Abdurhman Albasir for the support he provided me with.

Dedication

This is dedicated to my parents, and siblings who supported me though out my life.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACGAN Auxiliary Classifier Generative Adversarial 53, 54

ADASYN Adaptive Synthetic Sampling iv, 48, 51–53, 57, 61

ANN artificial neural network 7

ARP Address Resolution Protocol 14

AUC Area Under The Curve 27, 30–32, 35, 36, 40

BiLSTM Bi-directional Long Short-Term Memory 48

BOHB Bayesian Optimization and Hyperband iv, 18, 24, 25, 27, 30, 31, 35, 41, 43

CNN Convolutional Neural Networks iv, 3, 47, 48, 50, 54, 56, 58, 59, 61, 63

CPU Central Processing Unit 13, 44

CTGAN Conditional Tabular Generative Adversarial Network 53

daddr destination IP address 9

DDoS distributed denial of service 10, 11

DL deep learning 2, 3, 14, 16, 47, 48, 53, 58, 61, 63

DNN Deep Neural Network 7, 16, 17, 25, 30–32, 34–41, 43, 48, 61–63

DNNs Deep Neural Networks 8, 15, 17, 41, 44

DNS Domain Name System 8, 14

DoS Denial of Service 1, 4, 9–11, 14, 46, 52, 55, 61

dport destination port number 9

DT Decision Tree 7

ENN Edited Nearest Neighbour 52

FAR False Alarm Rate iv, 1, 7, 27, 32, 34, 36, 39, 45, 50, 58, 62

FNR false negative rate 7, 29, 34, 40

FPR False Positive Rate 35, 36

FTP File Transfer Protocol 8

GAN Generative Adversarial Networks iv, 47, 48, 53, 54

GBM Gradient Boosting Machine 7, 8, 14, 15, 25, 34–37, 43

GP Gaussian Process 17, 18

HB Hyperband 18

HP Hyper-parameter 2, 3, 6, 7, 16, 17, 19, 27, 30, 34, 44

HPO Hyper-parameter optimization 5, 17, 18, 24, 25, 27, 30, 31, 41, 43, 44

HPs hyper-parameters iv, 2–6, 15–18, 20, 21, 24, 30, 31, 34–38, 41, 43

HTTP hypertext transfer protocol 4, 8, 12, 54

ICMP Internet Control Message Protocol 20

ICVAE Improved Conditional Variational Auto-Encoder 48, 62, 63

IoT Internet of Things 1, 4, 7–9, 44–46

IP Internet Protocol 7, 9, 13, 20

IR Imbalance Ratio 54

KNN K-Nearest Neighbor 51

LAN Local Area Network 8

LR learning rate 6

MCC Matthews Correlation Coefficient 36, 37

MITM Man-in-The-Middle 11

ML Machine Learning iv, 1–6, 20, 45–51, 54–57, 63

MQTT Message Queuing Telemetry Transport 8, 11, 13

NB Naive Bayes 7

NID Network Intrusion Detection iv, 1–3, 7, 36, 45–48, 50, 52, 54–56, 58, 62, 63

NPV Negative Predictive Value 37

OS Operating System 13

OSS one-side selection 48, 50

OvO One versus One 48

PSO Particle Swarm Optimization iv, 24, 25, 30, 31

R2L Remote to Local iv, 48, 52, 59, 61

RELU Rectified Linear Unit 58

RF Random Forest 6–8, 14, 15, 25, 30, 34–36, 39, 40

ROC Receiver Operating Characteristics 27

ROS Random Over-Sampling iv, 48, 51–54, 56, 57, 61

RUS Random Under-Sampling 49, 56

saddr source IP address 9

SAVAER Supervised Adversarial Variational Auto-Encoder With Regularization 48, 61–63

SDN Software-Defined Network 11

SMAC Sequential Model-based Algorithm Configuration 15

SMOTE Synthetic Minority Oversampling TEchnique iv, 2, 46–48, 50–54, 57, 61

sport source port number 9

SSH Secure Shell 8

SSL Secure Sockets Layer 12

SVM Support Vector Machine rate 6, 50

T-SNE T-distribution Stochastic Neighborhood Embedding 20

TACGAN Tabular Auxiliary Classifier Generative Adversarial Networks 54

TCP transmission control protocol 4, 20

TPE Tree-structured Parzen Estimator iv, 18, 24, 27, 30, 31, 43, 44

TSE-IDS Two-Stage Classifier Ensemble for Intelligent Anomaly-Based Intrusion Detection System 63

TSODE Transient Search Optimization based on Differential Evolution 47, 48, 58

U2R User to Root iv, 48, 51, 54, 55, 59, 61

UDP user datagram protocol 4, 20

VAE Variational Auto-Encoders 47

VMs Virtual Machines 8, 9, 11

WAN Wireless Area Network 8

WGAN Wasserstein Generative Adversarial Networks 54

WGAN-GP Wasserstein Generative Adversarial Network-Gradient Penalty 48, 53, 54

WGAN-IDR Wasserstein Generative Adversarial Networks with Improved Deep Regret 54

XSS cross-site scripting 11, 14

Chapter 1

Introduction

Cloud computing and Internet of Things (IoT) technologies, and the generations of wireless technologies are advancing expeditiously [70]. With the help of these advanced technologies, millions of users and devices are interconnected. This creates more opportunities for cyber-security attackers to target more victims. Securing users' information and protecting the IoT devices is crucial for the continuation of the communication process [22]. Knowing that some of the targeted systems may have a strong NID system, cyber-security attackers use reformed attack methods. Therefore, a well-performing NID system must be able to distinguish new attacks even if it has not seen any or many of them [27].

Many ML based NID systems have been introduced recently [70]. However, while implementing such systems, ML engineers have to address several issues. For instance, fitting models on an imbalanced dataset may result in a high FAR on the minority classes [35, 10]. Figure 1.1 shows a system model for a secure IoT network where an IoT cloud server trains an ML model for detecting network attacks and updates it frequently when a new form of attack emerges. While the system allows legitimate users to subscribe and connect to IoT services securely, it blocks abnormal traffic and prevents attackers from harming legitimate users or the IoT infrastructure.

It may be noted in Figure 1.1 that the attempts of "Hacker 2" to bring the network down through Denial of Service (DoS) attacks or to phish for user information keep failing since the ML model has seen many of these attacks in the training phase. Consequently, the hacker disguises the DoS attack in a new form. In such a scenario, the NID system should be able to generalize well, flag the new form of attack, and through an effective balancing strategy, the system should use this incident to update and enhance the performance of its applied ML model.

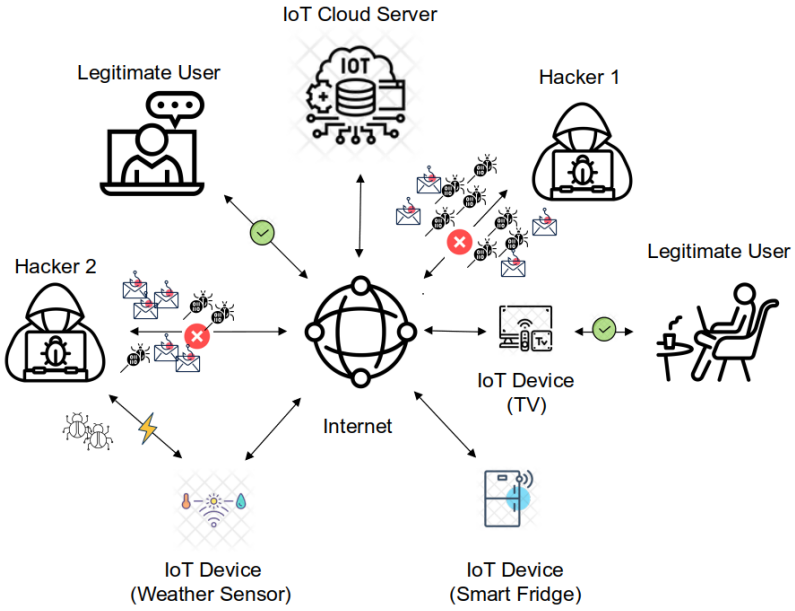


Figure 1.1: System model

To tackle the data imbalance issue, researchers have incorporated different balancing techniques into their NID systems. The simplest and most popular of these is random oversampling where randomly selected samples of the minority classes are replicated [16]. Other techniques involve the creation of new synthesized samples. For instance, SMOTE creates a new sample out of **k** neighbor samples [77]. Since the introduction of SMOTE, other variants of it have been proposed. For example, Borderline-SMOTE is a variant of SMOTE where any sample from the minority class that has more neighbors from the majority class, is declared as a borderline sample and used to generate new samples[27]. This technique improves the classification process as samples near the borderline are more likely to be classified incorrectly [27].

Hyper-parameter (HP) tuning is an important part of training ML and deep learning (DL) models [39]. It can help not only improve the accuracy of ML models, but in some cases, it can help build a model that is efficient and fast to train and use to infer. For instance, if used to optimize the architecture of a neural network, it can reduce unnecessary connections even layers that add costs to the computation process. Reducing the number of epochs required for the algorithm to run is another example of saving resources. Without HP tuning, data scientists would miss out on enhancing their models, as settling with the default HPs may not achieve the best results [69]. Furthermore, manually setting up

the HPs could be a tedious process and it is inefficient for large search spaces [69]. HP tuning can help not only improve the inference ability of ML models but also help build models that are efficient and faster while training and predicting [74]. When applying ML algorithms in the domain of network security, it is crucial to deploy a high-performing model. Security experts are not only unable to handle threat detection manually, but also they are unable to double-check many of the false alarms that applied ML models raise [25]. This highlights the importance of training robust models that minimize such occurrences of false alarms.

This research makes the following contributions:

- It gives an algorithm for data balancing to enhance the performance of ML/DL algorithms. The algorithm applies a dynamic approach that considers both under-sampling and over-sampling in the balancing process. In this thesis, a comprehensive comparison is conducted between the proposed NID system's performance and other systems [71, 70, 27] that utilize different data balancing techniques.
- It shows the importance of HP optimization and how tuning the HPs can significantly improve the performance of ML/DL algorithms.
- It proposes an NID system that combines the strength of CNN for feature extraction with a balancing technique for improving the system's performance on the minority classes.

Two highly imbalanced datasets are used to validate the aforementioned contributions: the NSL-KDD dataset [63] and the BoT-IoT dataset [38]. The proposed NID system achieves accuracies of 85.49% and 99.99% on the NSL-KDD and the BoT-IoT datasets, respectively.

Chapter 2

Hyber-parameter optimization[1]

2.1 Introduction

The widespread use of IoT infrastructures gives attackers more opportunities to exploit it [38]. Securing IoT devices is important to prevent attackers from tampering with telemetry data or gaining access that should not be authorized. Figure 2.1 shows a system model where telemetry devices subscribe to an IoT cloud server that gathers and processes received data to train ML models that are able to flag attack traffics. Attacks should also be detectable on the edge layer; this can be achieved by deploying lightweight ML models built on the IoT cloud server and installed on the IoT edge devices. For example, the smart fridge in Figure 2.1 is able to prevent the attacker from overloading it with traffic that could render it useless for legitimate users. This type of attack is referred to as DoS attack, and it can be categorized further into three types depending on the protocol used: the user datagram protocol (UDP), the transmission control protocol (TCP), and the hypertext transfer protocol (HTTP).

In the domain of anomaly detection, any enhancement in performance is very appreciated, as one less false positive can protect a company from a security breach that can cause havoc to the company. Some learning models are sensitive to their HPs and some do not have default HPs which is the case in deep learning where neural networks learn the

[1]This is an Accepted Manuscript of an article published by the Association for Computing Machinery in the proceedings of the 18th ACM International Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '22) on 24 October 2022, available online:https://doi.org/10.1145/3551661.3561364. Omar Elghalhoud, Kshirasagar Naik, Marzia Zaman, and Nishith Goel. "Data Balancing and Hyperparameter Optimization for Machine Learning Algorithms for Secure IoT Networks"

deep representation of the data through the proper setup of the network architecture which makes the use of an Hyper-parameter optimization (HPO) algorithm a must, in order to get the most out of these neural networks.

ML algorithms can be categorized into two groups: non-parametric and parametric algorithms. The former group includes statistical models that would have no parameters of their own and do not make many assumptions about the data distribution, while the latter group includes models that make assumptions about the data distribution and use parameters which are updated or set by the model itself during the training process such as the mean of the data [69]. These parameters should not be confused with the HPs of an ML algorithm. A hyper-parameter of a learning algorithm is a parameter that helps control some aspects of the learning process. Different learning algorithms have different HPs [74]. ML algorithms are designed to model available data, so they can be used to make predictions on unseen data. However, these ML algorithms can be used and applied at times in different domains, hence many of these ML algorithms have a set of HPs that are designed in such a way to hold a varying value in order to address variations in datasets and to make the learning process relevant to the problem at hand [6, 32]. For example, in a random forest model the number of features to consider when looking for the best split in a tree within the forest can be and should be set differently depending

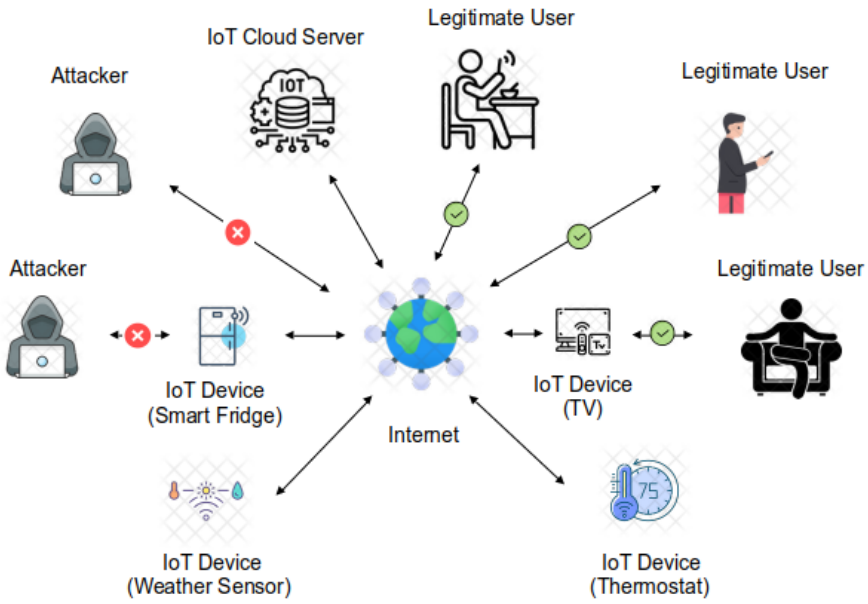


Figure 2.1: System model

on the dataset and the problem the random forest model is trying to fit [74, 65]. The HPs of a learning algorithm are set and controlled by the user applying the algorithm to the problem and their role varies from one learning algorithm to another as it depends on the learning process itself. For example, in neural networks, batch size HP will dictate after how many processed samples the model weights are updated [74]. In another model, such as random forest, maximum depth HP will control the branching process within a tree such that it does not exceed a certain level when the learning algorithm is applied to a problem [69]. Although default HP values set by field experts for different ML can be a good starting point or at least better than a random guess [74], they do not provide consistent performance from one domain to another and from a dataset to another. Once set, most of the HPs are fixed throughout the training process, however, there are quite few that are periodically updated while the model is fitting the data, for example, the learning rate (LR) in neural networks can be set to decay as the learning process progresses, and in such cases another HP is introduced that controls this update (i.e., LR decay) [74]. Yang et al. [69] showed that in all experiments they have conducted to study the effect of HP tuning, the optimized HP of ML algorithms, including Random Forest (RF) and Support Vector Machine rate (SVM), enabled the models to achieve better performance than using the default HPs. Hyper-parameter tuning can be thought of as an optimization problem where trying multiple combinations of the algorithm's HPs can yield a global optimum (for example, in ML classification models it can represent the classification accuracy or recall). Tuning the HPs can have impacts not only on the performance of the learning algorithm, but it can also impact the execution time, memory, and computation resources required by the algorithm to train and fit the data [69]. For example, SVM algorithm can use a simple linear kernel which will work well for solving easy problems, however, for complicated problems a more sophisticated kernel, such as a polynomial or a radial basis function, may be required to have a better fit on the data. These sophisticated kernels may have a negative impact on the computational resources and the timing required for fitting the data and predicting classes. Therefore, using the same value of HPs for all problems is not a good practice and it may lead to poor model efficiency and performance.

This chapter demonstrates the following contributions of this thesis:

- It gives an algorithm for data balancing to enhance the performance of ML/DL algorithms. The algorithm applies a dynamic approach that considers both under-sampling and over-sampling in the balancing process.
- It shows the importance of HP optimization and how it can significantly improve the performance of ML/DL algorithms.

To validate the aforementioned contributions, the BoT-IoT and the ToN-IoT datasets are used for benchmarking. The optimized model achieves a recall of 100% on the BoT-IoT dataset and a recall of 99.96% on the ToN-IoT dataset.

The rest of this chapter is structured as follows; in Section 2.2 related work is reviewed with more focus on the use of the BoT-IoT or the ToN-IoT datasets as benchmarks to evaluate NID systems proposed by researchers in this field. Next, both datasets used in this research are introduced in Section 2.3. The proposed methodology is presented in Section 2.7. In Section 2.8, the results of the conducted experiments are discussed. Finally, in Section 2.9, the conclusion is given and future work is outlined.

2.2 Literature review & Related work

Many network intrusion detection (NID) systems have been introduced by researchers in this field. Jamal et al. [33] introduced an artificial neural network (ANN) that achieves an accuracy of 94%; however, its false negative rate (FNR) is 11%. Moustafa [50] compared the performance of four ML/DL algorithms: RF, Gradient Boosting Machine (GBM), Naive Bayes (NB) and Deep Neural Network (DNN). He used the H20.ai package which provides HP optimization through random search or grid search. The reported results were encouraging and all of the used algorithms, except NB algorithm, performed well. However, the high performance (accuracy: 100%, recall: 100%, and false alarm rate: 0%) can be attributed to the use of highly correlated features such as Internet Protocol (IP) addresses and timestamps which the author [50] recommends not to use. Nevertheless, the primary focus of Moustafa's work [50] was on the introduction of an IoT dataset that can be used as a benchmark for ML algorithms.

Sarhan et al. [58] focused their work on feature selection, to show that ML/DL models can get a performance boost if those were to be fitted on a reduced set of features. Their RF and DNN models achieved recall values of 97.51% and 97.49%, respectively. Kumar et al. [41] proposed an ensemble model that combines three learners: Decision Tree (DT), NB and RF models. While their model achieved a recall of 99.98%, it achieved 5.59% FAR and 90.55% precision. Sarhan et al. [57] introduced new versions of some of the popular NID datasets such as the CSE-CIC-IDS2018, the BoT-IoT, and the ToN-IoT datasets. They have used a unified technique (NetFlow) to generate enhanced features extracted out of raw packets. Their work is quite important as working with the same set of input features, will allow for benchmarking and cross testing of ML models on different datasets [57]. Thaseen et al. [64] deployed a modified under-sampling technique to address the issue of class imbalance in the ToN-IoT dataset. This technique removes redundant samples

after measuring their similarities with other samples using Cosine similarity, City-block, and Euclidean distance. Pawlicki et al. [54] compared multiple sampling techniques for balancing the CICIDS2017 [59] dataset. They showed that random under-sampling has a similar performance, if not a superior performance, compared to the other tested sampling techniques such as Near Miss, and Cluster Centroids [54].

2.3 Review of NIDs Datasets

There are several datasets available in the literature that simulate network traffic. However, most of them lack heterogeneity and do not resemble the recent sophisticated forms of cyber-attacks [50]. Therefore, the BoT-IoT [38] and the ToN-IoT [50] benchmark datasets have been chosen to test the performance of RF, GBM, and Deep Neural Networks (DNNs) for detecting attacks on IoT networks.

2.3.1 The BoT-IoT dataset

This dataset has 72 million records and consists of 43 features. It was designed at the Research Cyber Range lab of the University of New South Wales (UNSW), Canberra. The testbed used to create this dataset included several Virtual Machines (VMs) that were connected to Local Area Network (LAN) and Wireless Area Network (WAN) interfaces in the cluster and were linked to the Internet through the PFSense machine [38]. The Ubuntu server was one of the targeted machines by the generated attacks. In the Ubuntu server, several services had been deployed such as Domain Name System (DNS), email, File Transfer Protocol (FTP), HTTP, and Secure Shell (SSH) servers, along with simulated IoT services. The network consisted of four attacking machines, an Ubuntu server, an Ubuntu mobile, Windows 7, Metasploitable, and an Ubuntu Tap machine. The Ostinato tool was used to generate the normal traffic, and the tshark tool was used to capture the raw packet volume that goes through the built network.

The authors of this dataset have used Node-red tool to simulate various IoT sensors [38]. The dataset's testbed mimicked IoT sensors, such as, temperature, pressure, and humidity sensors through JavaScripts which the authors developed themselves where Message Queuing Telemetry Transport (MQTT) messages were sent to the brokers. The MQTT broker resides in the Ubuntu server side where devices can publish data to it under a topic and it allows clients to connect and fetch information from a topic of a device they wish to interact with. The simulated IoT devices and services were connected to both the Ubuntu

server and an Amazon Web Services (AWS) IoT hub. The simulated IoT devices included a weather station, a smart fridge, a remotely activated garage door, a smart thermostat, and motion-activated lights. The network traffic was captured and logged in pcap files which subsequently were processed by Argus tool to extract initial features, out of which the authors generated more features to capture the statistics of flow groups in a relatively small sliding time-window of 100 connections, inside of which, patterns of several attacks can be discovered [38]. Examples of the generated features include a total Number of bytes per source IP, the average rate per protocol per source IP, and the number of inbound connections per destination IP. The packet filtering firewall was used to facilitate the labeling process as it enables monitoring of incoming and outgoing network packets, from which the attack and normal traffic can be distinguished based on specific source and destination IP addresses associated with attack and normal platforms. They also maintained periodically normal connections between the VMs by executing normal functions of the services installed on the Ubuntu server. The authors scheduled different types of attacks to run at different times, with normal background traffic being constantly generated. There were 43 features extracted, among which, source IP address (saddr), source port number (sport), destination IP address (daddr), and destination port number (dport). These features are considered network flow identifiers, as they hold information that is capable of uniquely identifying a flow at any given time and assisting in the labeling process.

The attack types included in this dataset were probing attacks, DoS attacks, and information theft attacks. Each of these attack categories consisted of some sub-categories, such as, keylogging, and data theft for the information theft attack category, Table 2.1 shows the distribution of the different simulated traffic scenarios and highlights how much imbalanced this dataset is. More details on these attack types can be found in section 2.4.

Table 2.1: Traffic distribution of the whole BoT-IoT dataset

Traffic Type			**No. of samples**	**Total**
Attack	DoS	UDP	1981230	3668045
		TCP	1593180	
		HTTP	2474	
	Reconnaissance	Service_Scan	73168	
		OS_Fingerprint	17914	
	Theft	Keylogging	73	
		Data_Exfiltration	6	
Normal				477

Although the testbed used to create this dataset represented a realistic network en-

vironment, the dataset did not contain heterogeneous data sources and did not include telemetry data of IoT services that could assist to establish IoT security systems.

This dataset consists of 8 traffic types: 7 different attack types and normal traffic. However, some of these traffic types are included more than others. Figure 2.2 highlights how much this dataset is imbalanced. For instance, more than 97% of the records represent the DoS and distributed denial of service (DDoS) attacks, while normal traffic constitutes only 0.01% of the overall dataset.

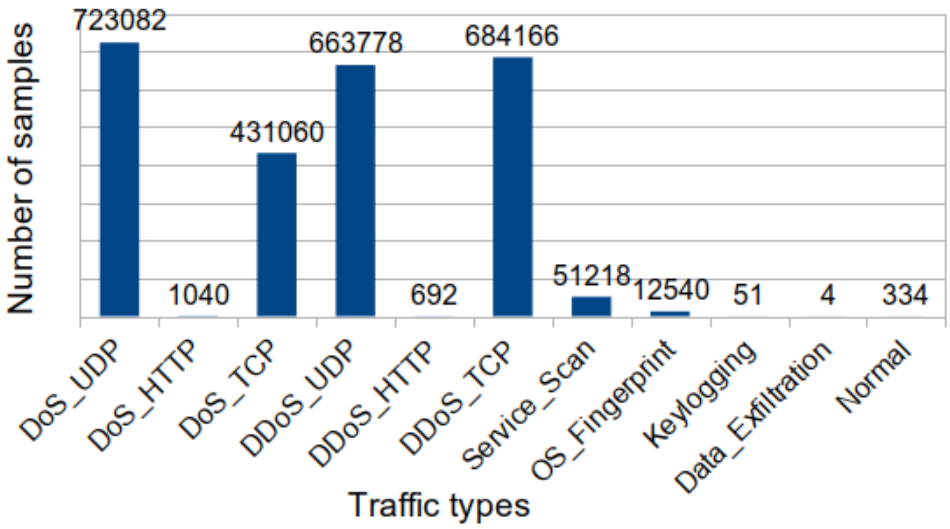


Figure 2.2: Network traffic distribution of the BoT-IoT training dataset

2.3.2 The ToN-IoT dataset

This dataset was designed at the IoT Lab of the UNSW Canberra Cyber, the School of Engineering and Information technology (SEIT). It consists of telemetry data captured through simulated IoT and IIoT sensors [50]. Also, it includes operating system datasets: Windows 7, Windows 10, Ubuntu 14.04, and Ubuntu 18.04 [50]. The biggest dataset in the ToN-IoT dataset is the network traffic dataset. The testbed was developed to capture and simulate interactions between the IoT devices in three network layers, the edge layer where the physical devices lie, for example, a smart TV, the fog layer where virtual machines and their services were set up, and the cloud layer which included cloud services, such as,

Table 2.2: Traffic distribution on the ToN-IoT dataset

Traffic Type		No. of samples
Attack	scanning	20000
	dos	20000
	injection	20000
	ddos	20000
	password	20000
	xss	20000
	ransomware	20000
	backdoor	20000
	mitm	1043
Normal		300000

a vulnerable website that simulates injection hacking events. This dataset includes nine attack events: scanning, backdoor, ransomware, DoS, DDoS, Man-in-The-Middle (MITM), data injection, cross-site scripting (XSS) and password violations, Table 2.2 shows the distribution of the different simulated traffic scenarios. It may be noted from Table 2.2 that most of the attack scenarios of the ToN-IoT network dataset are balanced except for MITM attacks.

The testbed utilized NSX-VMware data center platform as Software-Defined Network (SDN) solution. To enable multi-domain, hybrid physical and VM deployments, the testbed benefited from the vCloud platform which allowed for the replacement of expensive hardware devices with software-based network devices implemented as VMs. There were five main systems included in the fog nodes that contained 17 VMs for launching normal and attack traffic and regulating the traffic of IoT/IIoT sensors between the edge and cloud layers [50]. These VMs included a middleware server in which the IoT/IIoT sensors were simulated through deployed scripts. Node-red and Mosquitee MQTT broker tools were installed in this server for publishing and subscribing to telemetry data of IoT/IIoT data. Seven IoT/IIoT sensors were simulated: weather, smart garage door, smart fridge, smart TCP/IP Modbus, GPS tracker, motion-enabled light, and smart thermostat. Figure 2.3 demonstrates the different simulated traffic scenarios and shows an example of the imbalance in these telemetry datasets.

Different loggers were applied to log the data generated by the different sources [50]. The authors used netsniff-ng tool to capture the entire network packets in pcap formats [50]. The ToN-IoT network dataset has 44 features, which can be grouped into four categories. The first category holds connection attributes, such as, protocol types, the second

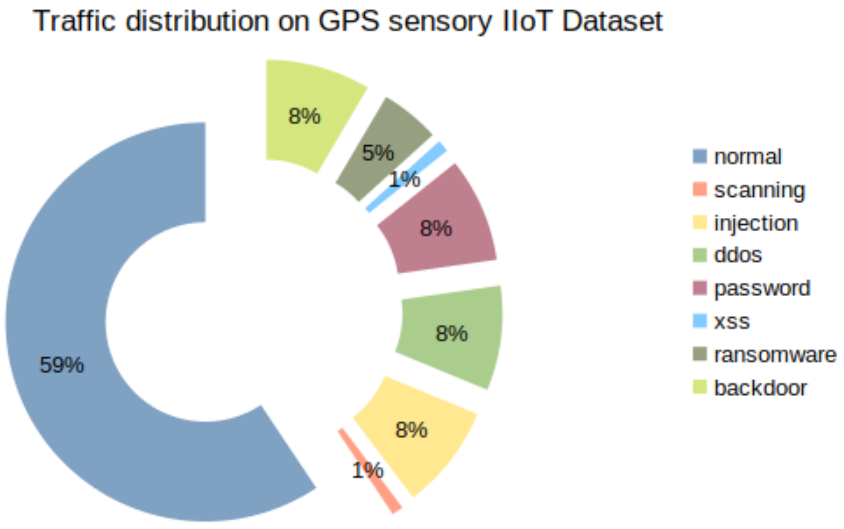


Figure 2.3: Distribution of the different simulated traffic scenarios on the GPS sensor (ToN-IoT dataset)

category consisted of statistical features about the flow identifiers, such as, the number of destination packets that are estimated from destination systems, the third category includes information about high-level services, such as, HTTP and Secure Sockets Layer (SSL) attributes. The last category is the one that indicates if any abnormalities occurred while data has been transmitted.

2.4 Review of attack scenarios in IoT datasets

The most important aspect of intrusion detection systems is to identify attack traffic from normal traffic. However, if the attack type and intrusion technique is not identified, the network and IoT devices are still at risk of being compromised by other attackers or if the attacker hides his footprint in a different form. Therefore, classifying the specific type of attack can be quite beneficial for cyber-security analysts to employ the appropriate defensive mechanisms.

The following are some of the common attack types:

2.4.1 Scanning attacks

This type of attack tries to gather information about victims such as finding active IP addresses and open ports. It is also referred to as probing or fingerprinting. The attack is performed by generating network traffic that targets the system while recording the responses, and comparing them with known responses. This allows attackers to make inferences about underlying services and Operating System (OS). In BoT-IoT testbed [38], Hping3 and Nmap tools were used to perform OS and port scanning. In ToN-IoT testbed [50], Nessus and Nmap tools were used from the offensive systems against the targeted systems such as Public MQTT broker.

2.4.2 Denial of Service (DoS) attacks

The main aim of this type of attack is to disrupt services rendering them inaccessible to legitimate users. Attackers generate a huge volume of network traffic to crash the target machine by depleting its Central Processing Unit (CPU) and memory resources.

2.4.3 Information theft attacks

The aim of this attack is to gain access to sensitive data by means like keylogging. In BoT-IoT testbed [38] Metasploit framework was used to perform these attacks. In the ToN-IoT [50] dataset, the backdoor attack type carries similar characteristics to this type of attack.

2.4.4 Ransomware attacks

The goal of this attack is to prevent normal users from accessing systems or services by encrypting the targeted systems until the owners of the system pay a ransom. In ToN-IoT testbed [50], the Metasploit framework was used to launch this type of attack.

2.4.5 Injection attacks

This type of attack inserts some fake malicious input data from clients to applications to expose, damage data, or make the application execute malicious commands not intended by its original developers. This type of attack can be used as a means to exhaust the

targeted system's resources such that it denies service to legitimate users, which makes it evolve into a DoS attack.

2.4.6 Cross-site Scripting (XSS) attacks

The XSS attack happens when an attacker employs a web application to transmit malicious code, generally in the form of a browser-side script, to different end-users [50]. In the ToN-IoT dataset, XSSer toolkit was used to hack web applications in the testbed [50].

2.4.7 Password cracking attacks

In this attack, the attacker uses brute force to guess possible combinations of a targeted password until the correct password is discovered. In the ToN-IoT dataset, hydra and cewl tool-kits were used to launch this type of attack [50].

2.4.8 Man-In-The-Middle (MITM) attacks

In this type of attack, the attacker interrupts an existing conversation or data transfer and inserts themselves between users and applications pretending to be the legitimate participant which allows the attacker to control the conversation and be able to intercept confidential data. There are many forms of this attack such as Address Resolution Protocol (ARP) Cache Poisoning, DNS Cache Poisoning, HTTPS Spoofing, Eavesdropping, or Session Hijacking.

2.5 Review of some ML algorithms

In this section, three learning algorithms are outlined: two ML algorithms (RF and GBM) and one DL algorithm.

Random Forest (RF): This algorithm utilizes multiple decision trees where each tree uses different randomly selected features to split nodes which makes RF robust with respect to noise and over-fitting [8], [76]. RF uses tree classifiers where the generalization error depends on the overall performance of each tree on its own and the correspondence between them [8]. The combined effort of all trees would have much better attack detection and classification ability than an individual tree [29].

Using a random selection of features to split nodes makes it robust with respect to noise [8] and over-fitting [76]. RF algorithm is a popular algorithm that has been used not only as a classifier but also as an algorithm for selecting and figuring out which features are of more importance. Also, RF has been used as means to build optimization algorithms that are used for hyper-parameter tuning such as Sequential Model-based Algorithm Configuration (SMAC). Not only RF algorithm randomize the selection of features subset to be used within a tree, but it also applies randomization to the training set to be used in the individual trees [76]. Table 2.3 summarizes the important HPs for RF algorithm and highlights their importance. Table 2.4 lists the HPs and the search space defined for the RF optimization experiments conducted in this research.

Table 2.3: HPs of Random Forest algorithm

HP	Type	Description	Importance level
min_samples_leaf	int or float	The minimal number of data points required in order to create a leaf	Very High
max_features	int, float, 'auto' or 'None'	Fraction of random features sampled per node.	Very High
bootstrap	boolean (true or false)	Whether to train on bootstrap samples or on the full train set	Medium
split criterion	entropy, gini	Function to determine the quality of a possible split	Low
...	...	...	...

Table 2.4: RF HP Search space

HP	**Search range**
bootstrap	[True, False]
criterion	[gini, entropy]
max_features	[5, n_features]
min_samples_leaf	[0.000001, 0.001]
n_estimators	[5, 250]

Gradient Boosting Machine (GBM): This model integrates many weak learners to form a stronger model which boosts the overall performance [42]. Rather than building these weak learners (e.g., decision trees) independently as in RF, GBM builds them sequentially [50] where each new tree accounts and tries to improve on the errors of the trees built before it. Some of the HPs of GBM algorithm are similar to RF algorithm; however, it has its own unique HPs, such as the learning rate which controls the speed of the learning process and how stable should it be. Table 2.5 lists the HPs and the search space defined for the GBM optimization experiments conducted in this research.

Deep Learning (DL): Unlike ML algorithms that rely on the raw input features fed to it, DNNs learn new nad more useful representations out of the raw input features of

Table 2.5: GBM HP Search space

HP	**Search range**
subsample	[0.6, 1.0]
learning_rate	[0.05, 3.0]
max_depth	[3, 50]
max_features	[1, n_features]
min_samples_leaf	[0.000001, 0.001]
n_estimators	[5, 250]

the data [2]. Such a process makes the normal traffic representation stand out from the representations of other traffic types, which simplifies threat detection. DL has been used for image and natural language processing. However, it can be used to extract new features out of the input data as in Auto-encoders. It also can be used to solve classification tasks where the last layer has a number of neurons that matches the number of classes to be distinguished. Figuring out the right architecture for the DNN is not a straightforward process and involves a lot of trials and errors. Although there are best practices one can follow to get the most out of deep learning, a great deal of attention is required while tuning its hyper-parameters. To measure how well the optimization algorithms cope with larger search spaces of different HP types, a search space that includes 8 HPs (4 of which were categorical) was defined. Here are some of the DL hyper-parameters and what they contribute to the learning process.

- Learning rate: The learning rate controls how much to change the model in response to the estimated error each time the model weights are updated. Goodfellow et al. [24] describe it to be perhaps the most important hyper-parameter and if time allows for tuning only one hyper-parameter, the learning rate should be that HP .
- Activation function: This HP defines how the weighted sum of the input is transformed into an output from a node or nodes in a specific layer of the network.
- Optimizer: The optimizer represents the algorithm or method used to minimize an error function (loss function) or to maximize the efficiency of production.
- Batch size: The batch size determines the number of records (rows, if the data is in a structured form), that are parsed by the model before its weights are updated. [39].
- Number of epochs: This HP defines the number of times that the learning algorithm will work through the entire training dataset. One epoch means that each sample

in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, an epoch that has one batch is called the batch gradient descent learning algorithm [9]. When the number of epochs is too small, the model will not be able to reach a satisfactory performance. However, when it is too large, the model will be able to reach a satisfactory performance but it will waste too much time on training each candidate for many extra epochs. The number of epochs depends on the size of the training set and should be tuned by slowly increasing its value until validation accuracy starts to decrease, which indicates that the model started over-fitting the data [69].

Table 2.6 lists the search space defined in this research for tuning the HPs of DNNs. In the DNN experiments, the number of epochs was set to 90 and the early stopping technique has been applied to avoid over-fitting. Drop out was included as an HP, and when it was enabled, a drop out ratio of 0.2 was used.

Table 2.6: DNN HP Search space

HP	**Search range**
learning_rate	$\log_{10}[10^{-6}, 10^{-3}]$
batch_size	$\log_2[2^5, 2^8]$
optimizer	[sgd, adam, RMSprop]
drop_out	[Enable, Disable]
Batch_Norm	[Enable, Disable]
Hidden Layers	[5, 20]
Neurons	[200, 600]
activation_func	[relu, sigmoid, tanh]

2.6 Review of some HP optimization algorithms

This section outlines and explains the mechanism of three different HPO algorithms: a model-based algorithm, a multi-fidelity algorithm, and a heuristic algorithm.

Tree-structured Parzen Estimator (TPE): This optimization algorithm looks at the search space as a tree-structured configuration. This allows it to support different HP types (i.e., discrete, continuous, and categorical types). It is considered as one of the best-performing Bayesian optimization algorithms as it reduces the cubic computation time of Gaussian Process (GP) which is the most popular Bayesian optimization algorithm [21].

While GP models the conditional probability $p(y|x)$, TPE uses a kernel density estimator to model two non-parametric densities [5]:

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^*, \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{2.1}$$

where y^* is set by the algorithm to be some quantile γ of the previously observed loss values, x represents a configuration in the search space, $l(x)$ represents the density in which the corresponding loss y is less than y^*, and $g(x)$ represents the density in which the corresponding loss y is greater or equal than y^*. γ is a hyper-parameter itself that controls the trade-off between exploration and exploitation [5]. TPE samples new candidate configurations by maximizing the ratio $l(x)/g(x)$ favoring points with high probability under $l(x)$ and low probability under $g(x)$ [5].

Bayesian Optimization and Hyperband (BOHB): Whenever the search space is large and the objective function evaluation is too expensive to run, it becomes difficult for an HPO algorithm to converge. Therefore, a multi-fidelity approach is a better choice in such cases where more resources are only given to promising areas. Hyperband (HB) [43] offers a solution to this problem, however, due to its randomness, it is slower to converge. BOHB [21] replaces the random selection process in HB with a model-based search (TPE) which makes the model learn from previously drawn and evaluated samples. BOHB controls how many configurations within an iteration of its process can advance to the next iteration where more budget is allowed through η which is a hyper-parameter of the algorithm itself.

Particle Swarm Optimization (PSO): This algorithm is inspired by the way birds search for food [76]. Particles are placed randomly in the search space where each particle's location represents a solution to the optimization problem. Each particle looks for a better location to move to through its own best knowledge (best location it found so far) and through the shared global knowledge of the swarm, [7]. Equations (2.2) and (2.3) show how the velocity of a given particle is calculated and how its next location is updated, respectively [39]:

$$v_{t+1} = v_t + \theta_1 * r_1 * (x_{lbest} - x_t) + \theta_2 * r_2 * (x_{gbest} - x_t) \tag{2.2}$$

$$x_{t+1} = x_t + v_{t+1} \tag{2.3}$$

where θ_1 and θ_2 are HPs that control the local exploitation and the global exploration aspects of the algorithm. r_1 and r_1 are randomly generated numbers between 0 and 1. For a given particle in a given generation (t), v_t and v_{t+1} represent the current and updated velocity of the particle, respectively, x_t and x_{t+1} represent the current and updated location

of the particle, respectively, x_{lbest} represents the personal best location of the particle, and x_{gbest} represents the global best location of the swarm.

2.7 Proposed methodology

In this Section, a framework to develop a high-performing learning model is presented. As depicted in Figure 2.4, the framework includes the following steps: categorical feature transformation, data balancing, and normalization which are discussed in Section 2.7.1, then the setup of HP optimization algorithms and ML/DL training and testing are discussed.

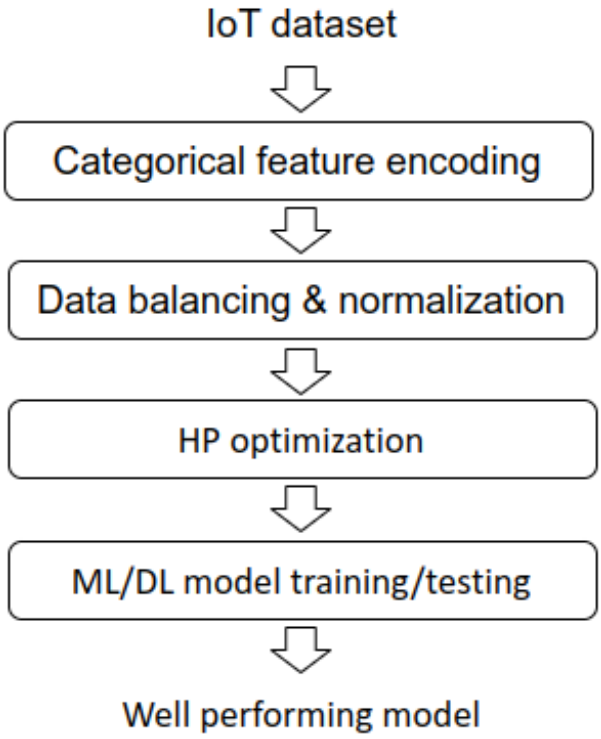


Figure 2.4: The proposed framework

2.7.1 Data Pre-processing

Two datasets have been used in this research: the BoT-IoT dataset and the ToN-IoT dataset. The publicly available 5% portion of the BoT-IoT dataset was used in the experiments conducted in this research. This dataset was designed for ML benchmarking and contains around 3.6 million records. For the ToN-IoT dataset, the "Train_Test_datasets" was used for optimizing and finding the effective HPs setup of multiple ML models, the dataset consisted of more than half a million records.

Feature transformation

Both datasets contained categorical features, such features require encoding, so one hot encoding has been applied. One hot encoding transforms each distinct value in a categorical feature to an independent feature on its own that holds 0 or 1 depending on the value of the original categorical feature. Such encoding ensures that different values of the categorical features are treated with the same magnitude rather than assigning different numbers to different categorical values which might mislead some ML algorithms into assuming certain ranks among the categorical values. Examples of these categorical features in the BoT-IoT dataset are 'state' (also 'state_number'), 'flags' (also 'flags_number'), and 'proto' (also 'proto_number'). In the ToN-IoT dataset, examples of categorical features are 'conn_state', 'http_method', and 'serivce'. The outcome of transforming a categorical feature such as 'proto' in the ToN-IoT would result in three features per distinct value which are Internet Control Message Protocol (ICMP), TCP, and UDP.

Although Moustafa [50] used the IP addresses and ports in the trained ML models. He has recommended excluding such attributes. In this work, these attributes along with the timestamps have been excluded to make the job of the ML models harder while fitting the data. This step was necessary especially when it is the case that the authors of the dataset used static IP addresses to launch the attacks, and the logged IP addresses of the offensive systems along with their timestamps were utilized to accurately label the data. In real-world scenarios, IP addresses and ports of anomalous nodes change dynamically from network to network, and relying on them limits the capabilities of the algorithm [37]. Therefore, their removal would help create robust models that are able to generalize well. In a similar research, the authors of [57] mentioned that they have excluded the flow identifiers such as IDs, source/destination IP, ports, timestamps, and start/end time to avoid learning bias towards attacking and victim end nodes.

To evaluate the features used in each dataset and to get an insight into how separable the different traffic types are, T-distribution Stochastic Neighborhood Embedding (T-SNE)

has been applied to map these Botnet datasets into a two-dimensional (2D) space where insights about clusters of different traffic types can be visualized. Figures 2.5 and 2.6 demonstrate how different attacks are spread across the problem space and show how some traffic types are fairly separable and how some are intersecting. For example, normal traffic in the Bot-IoT does not seem to intersect much with attack traffics. However, there are some normal traffic data points in the ToN-IoT dataset that are mixed within other attack clusters. This may attribute to the binary classification task in the ToN-IoT dataset being a little bit harder than the one in the BoT-IoT. This is one of the reasons why the focus of this thesis was on tuning the HPs of models used in the ToN-IoT.

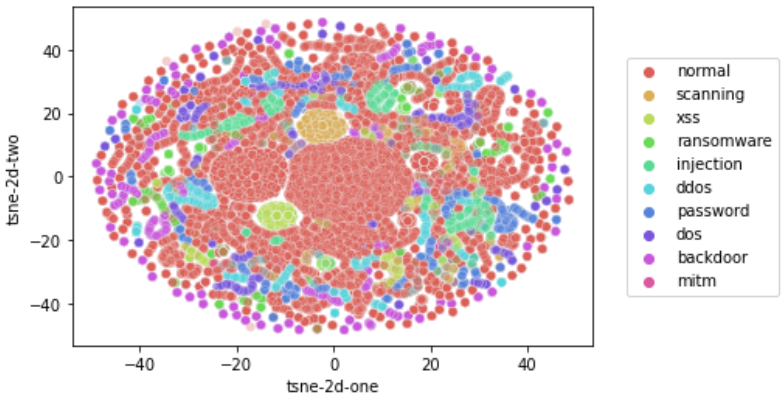


Figure 2.5: Embedding of the ToN-IoT dataset using T-SNE

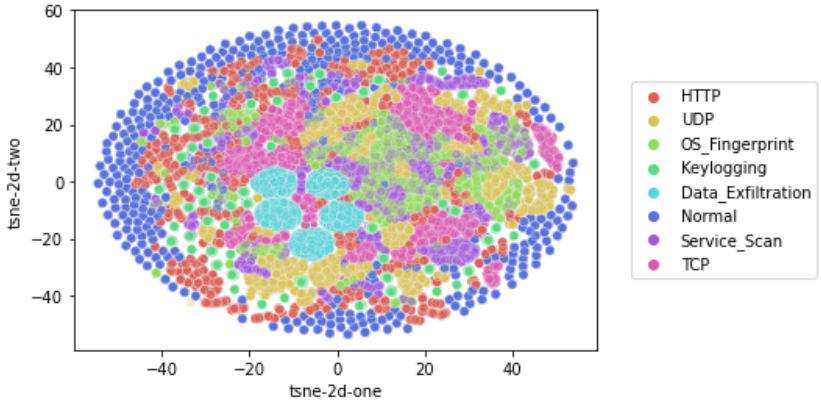


Figure 2.6: Embedding of the BoT-IoT dataset using T-SNE

To avoid over-fitting, cross-validation was applied and highly correlated features with

class labels, such as, IP addresses, ports, and timestamps were excluded.

Data Balancing

A balancing technique is essential to enable ML/DL algorithms to classify different attack types without being biased towards highly represented attacks. For instance, the BoT-IoT dataset has more than 10^6 DoS attack records, while it only has 73 keylogging attack records. Figure 2.2 demonstrates this high data imbalance. This thesis proposes Algorithm 1 as a process to bring some balance into IoT datasets. The algorithm iterates through the different traffic types (T) present in the input dataset, and at each iteration, if the number of samples of the provided reference traffic type is less than the number of samples of the traffic type associated with the iteration, it means that the traffic associated with the iteration is over-represented in comparison to the reference traffic. Therefore, the algorithm randomly under-samples that traffic making it similar to the reference traffic in terms of the number of samples (line 8). Otherwise, the traffic type associated with the iteration would be less represented and the algorithm over-samples it (lines 14 - 17). This aspect of the algorithm is controlled by ($prop$) parameter which represents the targeted new proportionality. Finally, since the new training dataset has been formed of sequentially added traffic records, the algorithm shuffles it (line 20). If the dataset is large, it is better to set the reference traffic (ref) to one of the attack types that are neither under-represented nor over-represented. Otherwise, it should be set to the most representative traffic. In this research, the Service Scan traffic type has been used as a reference traffic to balance the BoT-IoT dataset with ($prop$) set to 1.0 (see Fig. 3.2.), resulting in a total of 561,940 samples compared to a total of around 2.5×10^6 samples in the imbalanced training BoT-IoT dataset. However, when using the GPS telemetry dataset, which is a part of the ToN-IoT dataset, the normal traffic has been the optimal reference traffic, and ($prop$) was tuned to a value of 0.3. This highlights the importance of checking different target ratios since this dataset has a different imbalance ratio compared to the BoT-IoT dataset. Figure 2.7 shows the network distribution traffic of the BoT-IoT training dataset after applying the algorithm. For the ToN-IoT GPS telemetry dataset, the normal traffic accounts for 59% of the whole dataset records while scanning accounts only for 1%, Table 2.7 shows the number of traffic records in the GPS ToN-IoT dataset before and after balancing.

It is important that this process is only applied to the training set while maintaining the original traffic distribution in the validation and the testing sets, especially if they represent or mimic real distributions in real environments.

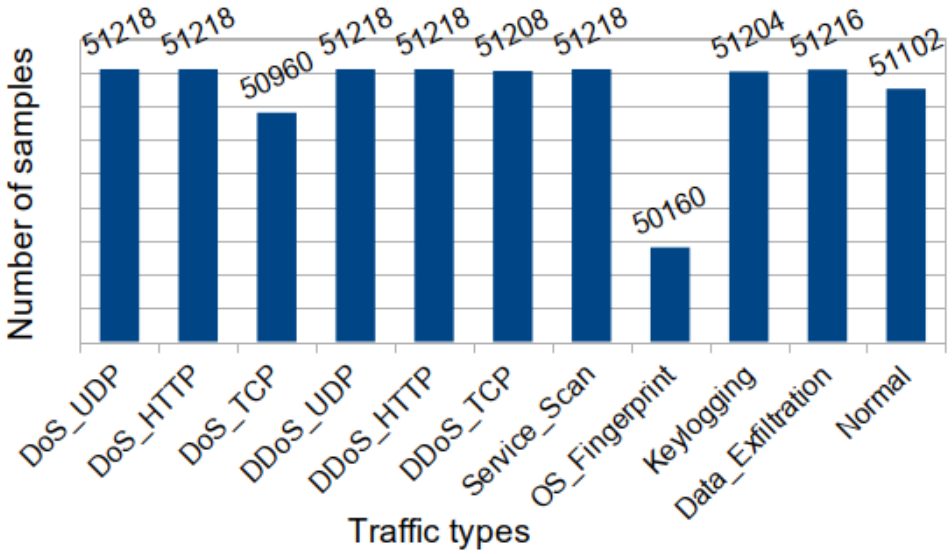


Figure 2.7: More balanced training BoT-IoT dataset

Algorithm 1 Data balancing algorithm

INPUT: Imbalanced Training Dataset (D_{train}), Reference traffic to use for balancing (ref), Proportion to add ($prop$).
OUTPUT: More Balanced Training Dataset (D_b)

procedure BALANCE_IOT_DATASET($D_{train}, prop, ref$)
  $D_b \leftarrow \emptyset$
  $no\_rec \leftarrow size(D_{train}[type == ref])$
  for each $t \in \mathcal{T}$ **do** ▷ $\mathcal{T}$ is a set of the traffic types present in D_{train}
    $D_t \leftarrow D_{train}[type = t]$
    $scale \leftarrow no\_rec/size(D_t)$
    if $scale <= 1.0$ **then**
      $D_{scaled} \leftarrow sample\_records(D_t, scale)$
      $D_b \leftarrow D_b \cup D_{scaled}$
    else
      $rc \leftarrow int(scale * prop)$ ▷ rc is a replication counter
      if $rc = 0$ **then** $rc \leftarrow 1$
      end if
      while $rc \neq 0$ **do**
        $D_b \leftarrow D_b \cup D_t$
        $rc \leftarrow rc - 1$
      end while
    end if
  end for
  $D_b \leftarrow shuffle(D_b)$
  return D_b
end procedure

Table 2.7: The GPS ToN-IoT dataset traffic distribution before and after balancing

Traffic type	Whole dataset	Imbalanced Train set	More Balanced Train set
normal	35000	28000	28000
scanning	550	440	8360
injection	5000	4000	8000
ddos	5000	4000	8000
password	5000	4000	8000
xss	577	462	8316
ransomware	2833	2266	6798
backdoor	5000	4000	8000

Data Normalization

Some learning algorithms fail to fit training data if a proper normalization is not performed. In DL, for example, input features with higher values could be treated as more important than other features with small values and this can slow the learning process. Therefore, in this step, input features are transformed to be in the same scale. In this research, Robust Scaling [47] was used to normalize the data. Equation (2.4) demonstrates how this technique can be applied.

$$x^{'} = \frac{x - median(X)}{Q_3 - Q_1} \tag{2.4}$$

where X is the input dataset and $x^{'}$ is the normalized value of $x \in X$. Q_3 and Q_1 represent the 75^{th} quantile and 25^{th} quantile, respectively.

Set up of the HP optimization algorithms used

Three different HPO algorithms have been used in this research. The selection decision was made to include three different optimization techniques: a model-based approach, a multi-fidelity approach, and a heuristic approach. TPE [5] was chosen as a model-based HPO . Its ability to handle categorical HPs more efficiently makes it a suitable choice for this research. BOHB [21] was selected due to its multi-fidelity approach, and PSO [7] was selected because of its heuristic nature.

Tree-structured Parzen Estimator (TPE): In this research, the publicly available implementation for TPE algorithm (hyperopt) [34] has been used with the default setup.

Bayesian Optimization and Hyperband (BOHB): In this research, the implementation made available by the original authors of the BOHB algorithm [21] was used. BOHB controls how many configurations within an iteration of its process can advance to the next iteration where more budget is allowed through η which is a hyper-parameter of the algorithm itself. Falkner et al. [21], when introducing BOHB, showed that it is insensitive to η after experimenting with different values. Therefore, η has been set to 3 (the default value).

For RF and GBM HP tuning a minimum budget of 1 and a maximum budget of 81 were used. A budget of 81 means the use of the whole training dataset (81% of the whole dataset where the rest 19% used for testing), while a budget equal to 27 means the use of 27% of the training dataset. For DNN, the maximum budget was set to 9. The budget for deep learning models incorporated both the training dataset size and the early stopping process. Table 2.8 shows the different setups for the different budgets. The main reason behind using a lower maximum budget in the DNN experiments compared to the one used in RF experiments is that deep learning requires more training data in order to yield good results, and on the contrary ML algorithms such as RF, do not require too much data to build a decent model. Furthermore, different budgets for the Deep learning optimization task were considered. Using a maximum budget of 9 per the defined scheme in table 2.8 provided a good correlation between the budgets. Figure 2.8 shows an example of this correlation metric. Since different trials can have different budgets, and to make fair comparisons with the other HPO algorithms used in this research, BOHB was allowed to run for 200 full function evaluations. To do so, the number of iterations of BOHB algorithm was set to 49 for the two ML algorithms (RF & GBM) while it was set to 75 for DNN since a different budget scheme has been used for that.

Budget	Train Set size	No EPOCHs	Improvement Patience Percentage
1	22.68%	10	40%
3	37.26%	30	30%
9	81%	90	10%

Table 2.8: DNN-BOHB setups for different budgets

Particle Swarm Optimization (PSO): Optunity [15], which is an optimization package that includes an implementation of PSO, was used in this work with θ_1 set to 1.5 and θ_2 set to 2.0, which are the default values chosen for these parameters by the developers of the optunity package. Bouktif et al. [7] concluded these values as well after trying some combinations. The number of particles was set to 8 and the algorithm has been allowed to

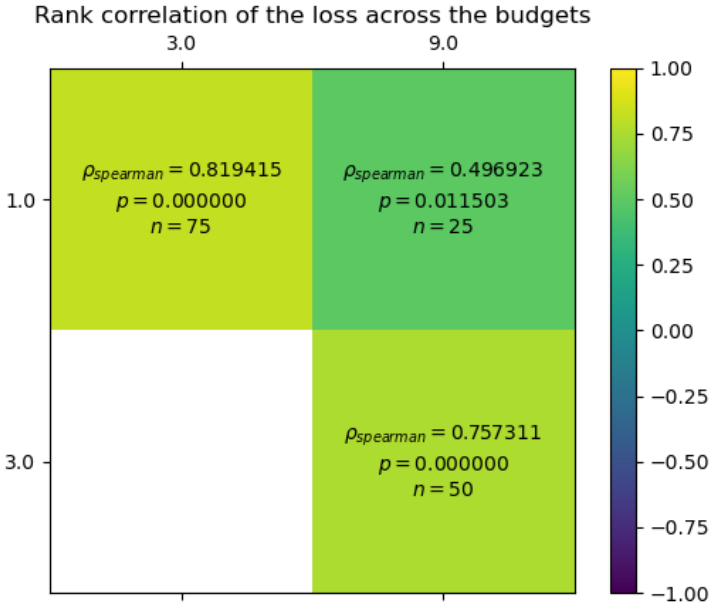


Figure 2.8: DNN-BOHB Rank correlation across budgets

run for 25 generations, so the total number of trials matches the number of trials allowed for the TPE and the BOHB algorithms.

2.7.2 Performance Evaluation Criteria

The metric used as an objective function for binary classification optimization was Area Under The Curve (AUC) Receiver Operating Characteristics (ROC) curve. It takes into consideration both the recall and the FAR. The optimization problem was transformed to resemble a minimization problem. Equation (2.5) demonstrates how the optimization problem was defined.

$$x^* = argmin_{x \in X} \log\left(1 - AUC(x)\right) \tag{2.5}$$

where x^* represents the best configuration of hyper-parameters that achieve the highest AUC value, and x represents a candidate point among all points X in the search space suggested by the HPO algorithm in different iterations. It is difficult to distinguish between HP configurations that result in very high similar performance that is nearly 0 [14]. Therefore the log transformation was applied in Eq. (2.5). Due to the class imbalance in the datasets used in this research, the weighted F1-Score was used for optimizing multi-class classifiers. This metric calculates the weighted harmonic mean of precision and recall.

Evaluating the performance of ML models using only a single metric can be misleading. Therefore, this research examines and reports the performance based on multiple metrics. Here is a list of the considered performance metrics.

- Area Under the Curve (AUC): This metric measures the classification ability of the entire sample and the balance of classified samples simultaneously [76], it represents the area under the curve of receiver operating characteristic which is built using the false positive rate as the x-axis and the true positive rate (recall) as the y-axis, a value of 1.0 means that the classifier gets all normal traffic and all attack traffic correctly.

- Accuracy: Accuracy is defined as the fraction of correctly classified records from the total number of records [39]. Equation (2.6) shows how it is calculated.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.6}$$

  where TP represents the number of samples correctly predicted as positive (i.e., attack), FP represents the number of samples incorrectly predicted as positive, TN represents the number of samples correctly predicted as negative (i.e., Normal traffic) and FN represents the number of samples incorrectly predicted as negative.

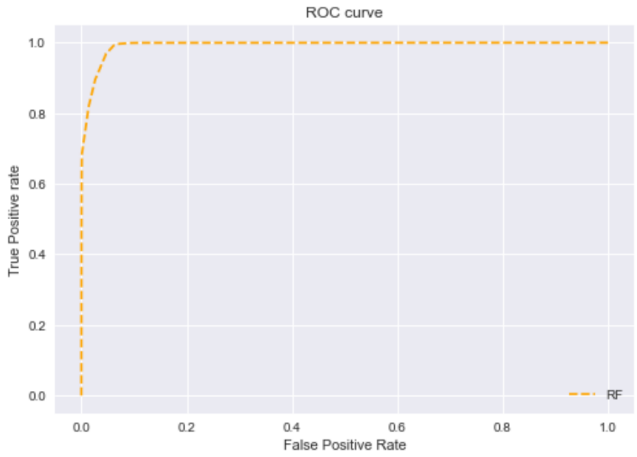


Figure 2.9: Example that shows the AUC after training an RF model

- Precision: This metric measures how well the classifier identifies attacks only, therefore it is calculated only using the true and false positives without minding the true and false negatives. It is also known as positive predictive value (PPV) and it can be calculated using equation (2.7) or equation (2.8)

$$Precision = \frac{TP}{TP + FP} \tag{2.7}$$

$$Precision = 1 - FDR \tag{2.8}$$

  where FDR is the false detection rate that is calculated as in equation (2.9)

$$FDR = \frac{FP}{FP + TP} \tag{2.9}$$

- Recall: Unlike precision, this metric measures how well the classifier identifies all attacks, hence it includes the false negatives that should be identified as positives since they represent attack traffic. This metric is the same as the true positive rate and some researchers refer to it as HIT rate or sensitivity. Equation (2.10) shows how recall can be calculated.

$$Recall = TPR = \frac{TP}{TP + FN} \tag{2.10}$$

where TPR is the true positive rate. It is also equal to (1 - FNR) where FNR is the false negative rate and can be calculated through equation (2.11).

$$FNR = \frac{FN}{TP + FN} \tag{2.11}$$

- F1-Score: F1-Score calculates the harmonic mean of precision and recall equally weighted. Equation (2.12) shows how recall is calculated. Since it is based on the harmonic mean, this metric was favored in our multi-classifications tasks as it applies more penalty whenever precision or recall is low compared to the geometric mean which applies less penalty. Due to the class imbalance in the datasets used in this research, the weighted F1-Score was used. Although data balancing was applied and considered, it was only applied to the training dataset while validation and testing datasets were kept imbalanced to see how the trained models deal with more realistic scenarios.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.12}$$

- Specificity This metric is also known as the true negative rate or selectivity. It measures the ratio of negative samples that are correctly predicted over the entire sample predicted as negative, which evaluates how precisely the model predicts in terms of negative samples [76]. Equation (2.13) shows how specificity is calculated.

$$Specificity = TNR = \frac{TN}{TN + FP} \tag{2.13}$$

- Negative Predictive Value (NPV): Unlike precision which focuses on attacks, this metric focuses on normal traffic only. NPV is calculated using equation (2.14) or equation (2.15).

$$NPV = \frac{TN}{TN + FN} \tag{2.14}$$

$$NPV = 1 - FOR \tag{2.15}$$

where FOR is the false omission rate that is calculated using equation (2.16).

$$FOR = \frac{FN}{TN + FN} \tag{2.16}$$

2.8 Results and discussions

This section outlines the results of optimizing the HPs of the RF model and the DNN model using the ToN-IoT dataset. The HP search space has been defined in section 2.5.

2.8.1 Optimization results of the ML/DL models using the ToN-IoT Dataset

Three HPO algorithms have been applied to tune the RF and the DNN models: TPE, BOHB, and PSO. These HPO algorithms were discussed in Section 2.6. Each of these three HPO algorithms has been allowed 200 full objective function evaluations to optimize the HPs of both the RF and the DNN algorithms. To ensure that the results are not due to random chance, this process has been repeated 5 times where a different seed has been used each time.

Results of optimizing the HPs of the RF algorithm: An AUC value of 0.9927 has been achieved in one of the PSO runs, which was slightly better than the best AUC achieved when training an RF model with its default HPs. The three HPO algorithms provided consistent performance with an AUC mean of 0.9924 from the different runs and a small standard deviation of 0.0002 for both TPE and PSO while the standard deviation of BOHB runs was 0.0001. However, the convergence time was not consistent. Table 2.9 demonstrates the AUC value of each of the 5 independent runs per HPO algorithm and table 2.10 shows at which trial each algorithm converged per independent run along with the mean and standard deviation of these 5 independent runs. Based on these results, a value of around 140 trials is recommended for this type of problem.

Table 2.9: AUC results of 5 independent runs of three different HPO algorithms to tune the HPs of RF model (SD: Standard deviation)

HPO Algorithm	Run #1	Run #2	Run #3	Run #4	Run #5	Mean	SD
TPE	0.9922	0.9923	0.9923	0.9926	0.9923	0.9924	0.0002
BOHB	0.9924	0.9925	0.9924	0.9925	0.9923	0.9924	**0.0001**
PSO	0.9924	0.9923	0.9924	**0.9927**	0.9921	0.9924	0.0002
Default HPs	0.9924	0.9923	0.9923	0.9926	0.9921	0.9923	0.0002

Table 2.10: Trial# at which the HPO converged for RF algorithm HPs optimization (SD: Standard deviation)

HPO Algorithm	Run #1	Run #2	Run #3	Run #4	Run #5	Mean	SD
TPE	146	25	174	151	158	130.80	60.08
BOHB	140	20	97	180	53	**98.00**	64.42
PSO	30	176	81	189	44	104.00	74.19

Results of optimizing the HPs of DNNs: Optimizing the architecture and HPs of Deep Neural networks is a tedious task. After applying the three HPO algorithms to tune the DNN model given the search space defined in section 2.5, an AUC value of 0.988 has been achieved in one of the TPE runs. Table 2.11 demonstrates how close the performance of the three HPO algorithms, and Table 2.12 shows at which trial each algorithm converged per independent run along with the mean and standard deviation of the 5 independent runs.

Table 2.11: AUC results of 5 independent runs of three different HPO algorithms to tune the HPs of the DNN model (SD: Standard deviation)

HPO Algorithm	Run #1	Run #2	Run #3	Run #4	Run #5	Mean	SD
TPE	0.9889	0.9889	0.9887	**0.9891**	0.9884	**0.9888**	**0.0003**
BOHB	0.9850	0.9886	**0.9891**	0.9890	0.9890	0.9881	0.0018
PSO	0.9872	0.9886	0.9854	0.9888	**0.9892**	0.9878	0.0016

Table 2.12: Trial# at which the HPO converged for DNN model's HPs optimization (SD: Standard deviation)

HPO Algorithm	Run #1	Run #2	Run #3	Run #4	Run #5	Mean	SD
TPE	195	114	23	159	141	126.40	64.86
BOHB	188	137	138	173	70	133.25	48.42
PSO	77	170	191	59	176	134.60	61.61

Figure 2.10 shows a convergence comparison from one of the five independent runs to tune the HPs of DNN. As the optimization process progressed, the three HPO algorithms converged to a similar AUC value around 0.988 with a slight advantage in favor of TPE. The AUC mean of the five independent runs per HPO algorithm was similar and around 0.988, and the standard deviation of BOHB and PSO runs was 0.001 while TPE had a lower standard deviation of 0.0003.

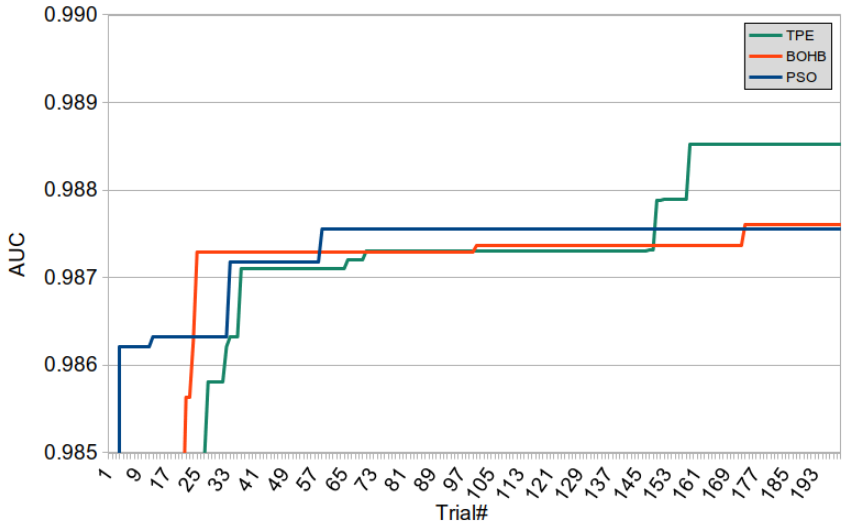
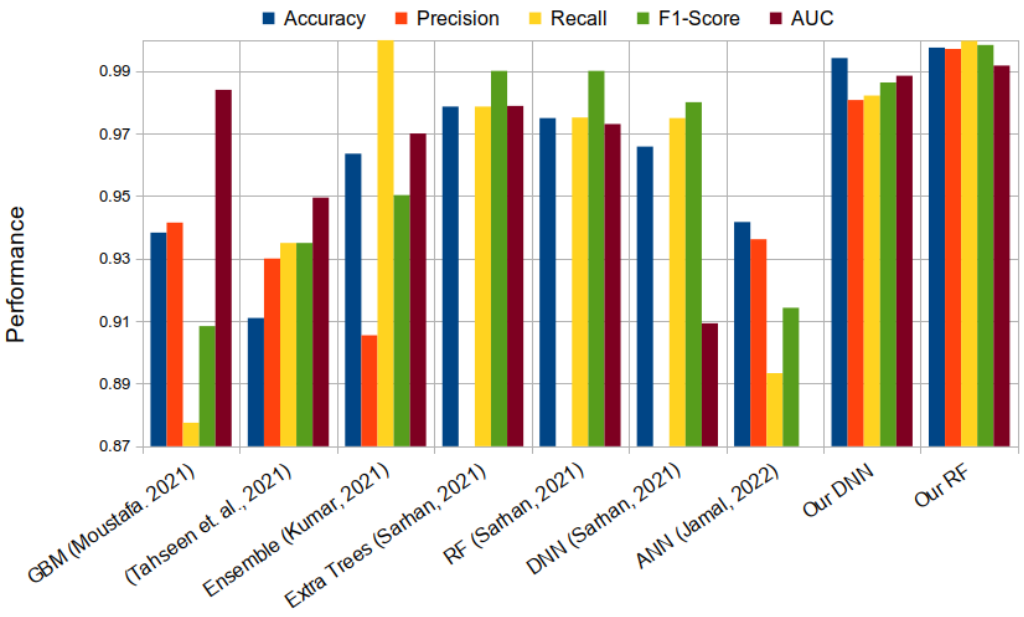


Figure 2.10: Convergence comparison of three HPO algorithms used to optimize DNNs

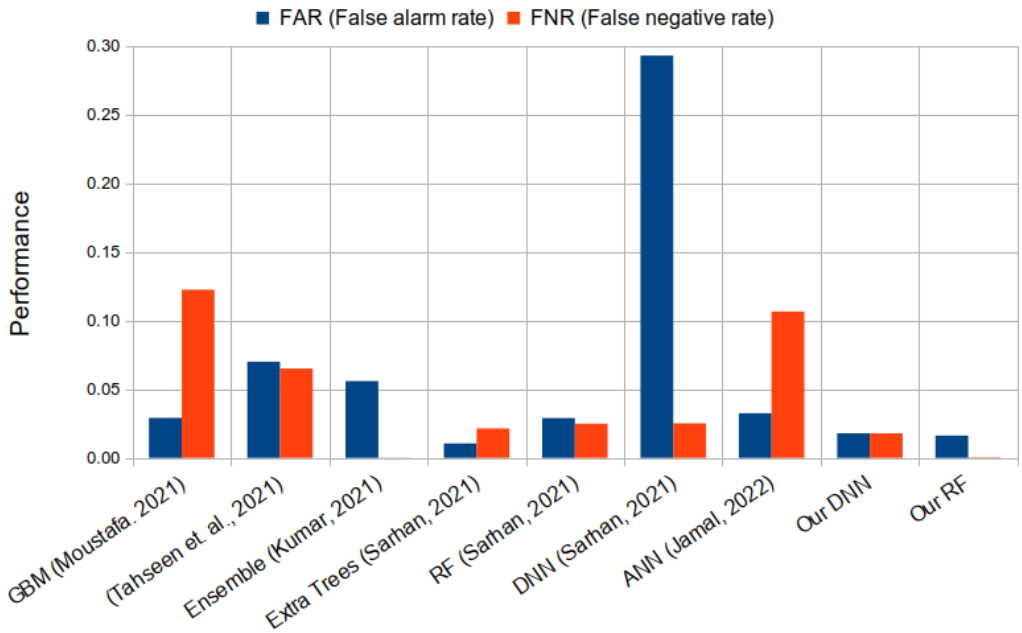
ML/DL model performance comparisons with other models reported in the literature on the ToN-IoT network dataset:

Here, more evaluation metrics of the best optimized models are reported and discussed. Then a comparison with other models reported in the literature is outlined.

Figure 2.11 demonstrates how the results of the proposed framework are more balanced and better than the ones reported in the literature. Some of the compared references did not report all metrics, hence the missing columns in Fig. 2.11. It may be noted that most models in the literature achieved high accuracy, above 90%. However, the results of the proposed framework are consistent under different performance metrics. The DNN model introduced by Sarhan et al. [58] scored an AUC value of 0.91 and an undesirably high FAR of 29%, while the proposed DNN model in this research scores an AUC value of 0.99 and only 1.79% FAR, which can be attributed to the optimization effect. Furthermore, the ToN-IoT dataset may require a deeper neural network: the DNN model in [58] included only 3 hidden layers compared to 10 hidden layers used in the optimized DNN model in this research. Although the ensemble model introduced by Kumar et al. [41] scores a high recall of 99.98%, the model performances in other metrics were not good: precision and FAR were 90.55% and 5.59%, respectively.



(a) Multiple performance metrics comparison



(b) False alarm rate and false negative rate comparison

Figure 2.11: ML/DL performance comparison using the ToN-IoT network dataset

The optimized RF model in this research achieves an overall superior performance: it achieves a similar recall of 99.96%, while its precision and FAR were 99.70% and 1.62%, respectively. Thaseen et al. [64] used an under-sampling technique to balance the data to achieve an accuracy of 91.10%. Comparing this with the results of this research's proposed framework where a dynamic data balancing approach in Algorithm 1 is used, it may be noted that the proposed framework achieves much better accuracy: 99.74%. Moreover, the framework introduced by Thaseen et al. [64] has an undesirably high FAR of 7%. The GBM model introduced in [50] and the ANN introduced in [33] had poor FNR results (more than 10%) compared to the FNR results of this research's optimized GBM (0.46%) and DNN (1.79%) models. The optimal HP values resulted from the optimization experiments conducted in this research using the ToN-IoT dataset to tune the HPs of RF, GBM and DNN models are listed in tables 2.13, 2.14 and 2.15, respectively.

Table 2.13: RF optimal HPs

HP	**Optimal Value**
bootstrap	False
criterion	entropy
max_features	27
min_samples_leaf	1.30×10^{-6}
n_estimators	10

Table 2.14: GBM optimal HPs

HP	**Optimal Value**
subsample	0.65
learning_rate	0.16
max_depth	23
max_features	99
min_samples_leaf	5.67×10^{-6}
n_estimators	223

Performance results of the ML/DL models on the binary classification task using the ToN-IoT network dataset

In this section, more evaluation metrics of the optimized models discussed in the previous section are reported. The results reflect the performance of the ML/DL models after

Table 2.15: DNN optimal HPs

HP	**Optimal Value**
learning_rate	1.00×10^{-6}
batch_size	128
optimizer	RMSprop
drop_out	Disabled
Batch_Norm	Enabled
Hidden Layers	10
Neurons	420
activation_func	tanh

applying cross-validation. Table 2.16 shows the results on the binary classification for the three models: RF, GBM & DNN. It may be noted from the table that the GBM model has a slight edge over the two other models. It achieves the lowest False Positive Rate (FPR) : 0.0113, and the highest AUC value: 0.9937. However, in terms of accuracy and F1-Score the RF model achieves better results. Its accuracy and F1-Score are 99.51% and 99.62%, respectively.

Performance results of the ML/DL models on the multi-class classification task using the ToN-IoT network dataset

For multi-class classification of the different attack types using the ToN-IoT dataset, an accuracy of 98% and an overall weighted average of 98% for recall, precision, and F1-Score have been achieved. Optimizing the HPs of the RF model was less time-consuming; however, optimizing the GBM model and the DNN model took more time. Therefore, based on the previous results of optimizing these models on the binary classification task, only RF and GBM HPs were optimized for the multi-class classification task. Nonetheless, for the DNN model, the optimized HPs on the binary classification task have been used to measure the performance on the multi-class classification task. Table 2.17 shows that the three models achieve close results although GBM model was the best among the three. Table 2.18 shows the confusion matrix for the different attacks where the correct predictions are on the diagonal. For the multi-class classifications of the different attack types of the ToN-IoT dataset, BOHB has been used to tune the HPs of a GBM model. Table 2.19 shows the classification report for each attack type as one versus the rest.

Table 2.16: Performance of the optimized ML & DL models on the binary classification task using the ToN-IoT network dataset

Metric \ ML Model	RF	GBM	DNN
AUC	0.9936	**0.9937**	0.9893
FPR	0.0116	**0.0113**	0.0116
FNR	**0.0012**	**0.0012**	0.0099
F1_Score	**0.9962**	0.9857	0.9918
MCC	0.9893	**0.9895**	0.9772
Accuracy	**0.9951**	0.9855	0.9895
Precision_PPV	0.9936	**0.9938**	0.9936
Recall_Sensitivity	**0.9988**	**0.9988**	0.9901
Specificity	0.9884	**0.9887**	0.9884
NPV	**0.9978**	**0.9978**	0.9823
G_Mean	0.9936	**0.9937**	0.9893
BA	0.9936	**0.9937**	0.9893
FM	0.9962	**0.9963**	0.9918

2.8.2 Optimization results of the ML/DL models using the BoT-IoT Dataset

After trying the default HPs of the RF algorithm on the BoT-IoT dataset and getting very high performance (100% accuracy and 0% FAR), a different approach to the one conducted using the ToN-IoT dataset was followed. Instead of optimizing the HPs on the BoT-IoT dataset, cross testing has been performed where the optimized values of the HPs tuned on the ToN-IoT were used to fit ML/DL models on the BoT-IoT dataset. The results using this approach have been compared to the results of using the default HPs of the same models. Table 2.20 shows the results of this experiment noting that for the DNN model only one model is reported since there are no default values of its HPs that can be used. For the RF algorithm, the results were identical which may indicate its robustness and less sensitivity to the choice of HPs for this type of NID datasets. However, the GBM model with HPs optimized on the ToN-IoT dataset achieved better performance than the GBM model with the default HPs. For instance, the Matthews Correlation Coefficient (MCC) metric of the former was 98.95% while the latter scored only 95.57%. Although the former model achieved better performance for some metrics such as AUC, FPR, and the geometric

Table 2.17: Performance of the optimized RF, GBM and DNN models on the multi-class classification task using the ToN-IoT network dataset

Metric \ ML Model	RF	GBM	DNN
Accuracy	0.9808	**0.9855**	0.9772
Weighted Avg. Precision	0.9817	**0.9864**	0.9782
Weighted Avg. Recall	0.9808	**0.9855**	0.9772
Weighted Avg. F1-Score	0.9811	**0.9857**	0.9774

Table 2.18: The confusion matrix after optimizing the GBM model on the multi-class classification task using BOHB and the ToN-IoT network dataset

Attack type	backdoor	ddos	dos	injection	mitm	normal	password	ransomware	scanning	xss
backdoor	**7598**	0	0	0	0	2	0	0	0	0
ddos	0	**7476**	0	13	3	8	0	89	0	11
dos	0	0	**7570**	0	8	9	2	1	10	0
injection	0	26	1	**7501**	1	2	0	5	2	62
mitm	0	0	0	0	**11081**	0	0	17	0	0
normal	1	15	4	6	2	**39449**	26	368	0	29
password	0	4	0	0	10	3	**7502**	75	4	2
ransomware	0	0	0	0	0	58	0	**7390**	0	152
scanning	0	0	8	0	2	0	2	1	**7587**	0
xss	0	13	0	45	0	5	2	512	0	**7023**

mean (G_Mean), this difference was not high. On the other hand, for the metrics where the latter model performed better, the difference was high, for example, the reported MCC values of GBM model with the default HPs and the GBM model that used a tuned HPs were 95.57% and 98.95% respectively. Another example is the reported Negative Predictive Value (NPV) values, these were 91.35% and 98.95%, respectively. It may be argued that the latter model is more balanced and superior to the former model which is supported by the number of false negatives and true positives; the latter model only missed one attack while the former model missed nine attacks. Furthermore, this table is a perfect example of how important to look at multiple metrics when comparing ML models. Surprisingly, the DNN model performed much better on the BoT-IoT dataset than the one applied to the ToN-IoT dataset, even though it was only tuned on the ToN-IoT dataset. This

Table 2.19: Multi-class classification results of the GBM-BOHB optimized model using the ToN-IoT network dataset

	Precision	**Recall**	**F1-score**	Support
backdoor	1.00	1.00	1.00	7600
ddos	0.99	0.98	0.99	7600
dos	1.00	1.00	1.00	7600
injection	0.99	0.99	0.99	7600
mitm	1.00	1.00	1.00	11098
normal	1.00	0.99	0.99	39900
password	1.00	0.99	0.99	7600
ransomware	0.87	0.97	0.92	7600
scanning	1.00	1.00	1.00	7600
xss	0.96	0.92	0.94	7600
accuracy			0.99	111798
macro avg	0.98	0.98	0.98	111798
weighted avg	0.99	0.99	0.99	111798

may be attributed to the similar features that these two datasets share. Figures 2.5 and 2.6 demonstrate how normal traffic is more separable in the BoT-IoT dataset, therefore this may be attributed to the better performance of the DNN model in this dataset. The DNN model managed to find deep representation and embedded features that make normal traffic easily distinguishable from attack traffic.

Table 2.20: Binary classification results using the BoT-IoT network dataset

Learning Model	**RF**		**GBM**		**DNN**	**Best value**	**Winner**
Metric	Default HPs	HPs tuned on ToN-IoT	Default HPs	HPs tuned on ToN-IoT	HPs tuned on ToN-IoT		
TN	**95**	**95**	**95**	94	**95**	95	RF, GBM default, DNN
FP	**0**	**0**	**0**	1	**0**	0	RF, GBM, DNN
FN	**1**	**1**	9	**1**	9	1	RF, GBM optimized
TP	**733609**	**733609**	733601	**733609**	733601	733609	RF, GBM optimized
AUC	**1.0000**	**1.0000**	1.0000	0.9947	1.0000	0.999999318438953	RF
FPR	**0.0000**	**0.0000**	**0.0000**	0.0105	**0.0000**	0	RF, GBM default, DNN
FNR	**0.0000**	**0.0000**	0.0000	**0.0000**	0.0000	1.36312209484604E-06	RF, GBM optimized
F1_Score	**1.0000**	**1.0000**	1.0000	1.0000	1.0000	0.999999318438488	RF
MCC	**0.9948**	**0.9948**	0.9557	0.9895	0.9557	0.994777354190588	RF
Accuracy	**1.0000**	**1.0000**	1.0000	1.0000	1.0000	0.999998637054402	RF
Precision_PPV	**1.0000**	**1.0000**	**1.0000**	1.0000	**1.0000**	1	RF, GBM default, DNN
Recall_Sensitivity	**1.0000**	**1.0000**	1.0000	**1.0000**	1.0000	0.999998636877905	RF, GBM optimized
Specificity	**1.0000**	**1.0000**	**1.0000**	0.9895	**1.0000**	1	RF, GBM default, DNN
NPV	**0.9896**	**0.9896**	0.9135	0.9895	0.9135	0.989583333333333	RF
G_Mean	**1.0000**	**1.0000**	1.0000	0.9947	1.0000	0.99999931843872	RF
BA	**1.0000**	**1.0000**	1.0000	0.9947	1.0000	0.999999318438953	RF
FM	**1.0000**	**1.0000**	1.0000	1.0000	1.0000	0.99999931843872	RF

(a) Macro average

(b) Weighted average

Figure 2.12: Data balancing effect on the GPS ToN-IoT dataset for classifying different traffic types

2.8.3 Results of the proposed data balancing algorithm

In this Section, the effect of applying the proposed data balancing Algorithm 1 is discussed. Since the network dataset part of the ToN-IoT was not severely imbalanced as the telemetry datasets, no significant improvement in the performance has been captured using the ToN-IoT network dataset. However, when using the ToN-IoT telemetry datasets, the performance increases after applying Algorithm 1. For instance, the overall macro F1-Score of the DNN model used for multi-class classification improves from 71% to 74% on the GPS ToN-IoT dataset. Figure 2.12 demonstrates the difference in the performance of both macro and weighted averages of multiple performance metrics. As depicted in Figure 2.12 (a), the overall macro average precision drops from 76% to 72%; however, the overall macro average recall significantly increases from 69% to 82%, reflecting a higher overall macro average F1-Score. Figure 2.12 (b) shows that both the overall weighted average precision and F1-Score increased without compromising the overall weighted average recall of the DNN model. It is worth noting that the RF algorithm was less sensitive to the imbalance in these datasets: balancing the GPS ToN-IoT dataset only improved the RF model overall macro average F1-Score slightly from 77% to 78% and the overall weighted average F1-Score from 89.02% to 89.48%. This result further demonstrates the RF algorithm's robustness for this type of datasets. On the binary classification task, the RF and the DNN models had a similar performance with or without balancing the GPS ToN-IoT dataset. The RF model scored an F1-Score of 96% and the DNN model scored an F1-Score of 95%.

Since the BoT-IoT has a much higher imbalance ratio, the effect of applying the data balancing algorithm while building a DNN model was more apparent. The FAR of the

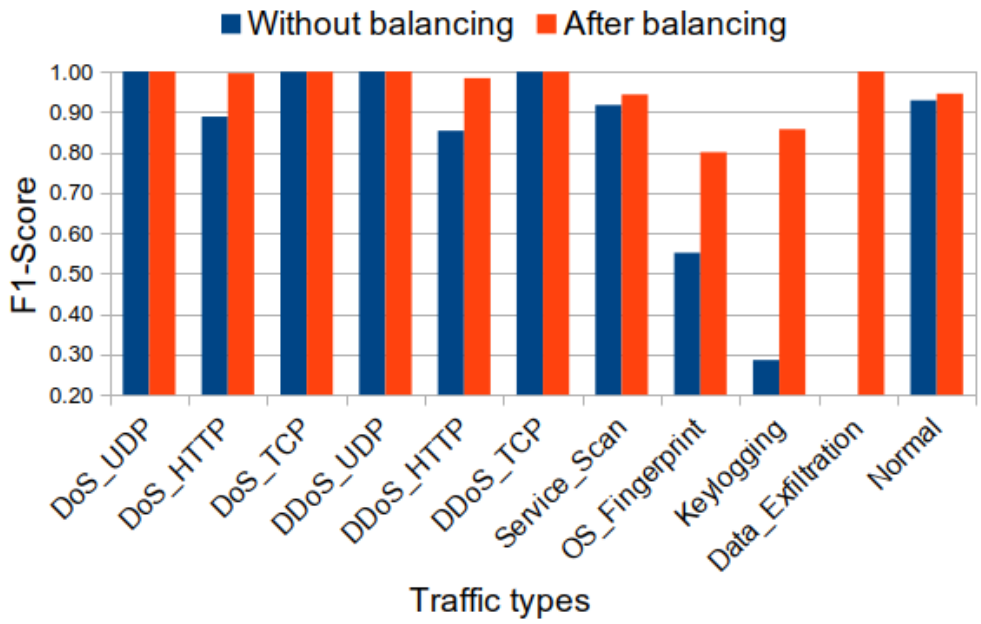


Figure 2.13: Data balancing effect on classifying different traffic types of the BoT-IoT dataset

DNN model has been significantly improved from 30% to 0% while maintaining 0% FNR. Without data balancing, the AUC value was 0.85; however, it jumps to 1.00 when Algorithm 1 is applied while conducting binary classification. Figure 2.13 highlights the performance improvement to the DNN model used for multi-class classification. It is clear that the data balancing algorithm maintained the DNN model performance on the over-represented attacks such as DoS UDP attacks, whereas it enabled the deep neural network to capture the deep representation of the other under-represented attacks resulting in a much better multi-class classifier. For instance, the F1-Score of keylogging attack improved significantly from 29% to 86%.

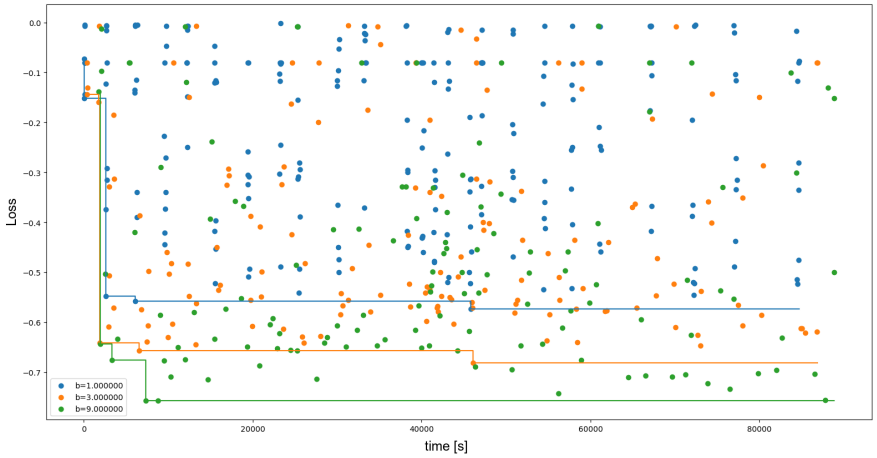
In contrast to the results of using the RF algorithm for the GPS TON-IoT dataset, Algorithm 1 enhances the performance of the RF multi-class classifier used for the BoT-IoT dataset: It improved the F1-Score of keylogging attack from 89% to 100% while maintaining 100% F1-Score in the other attacks.

2.8.4 The effect of narrowing the search space on the HP optimization process

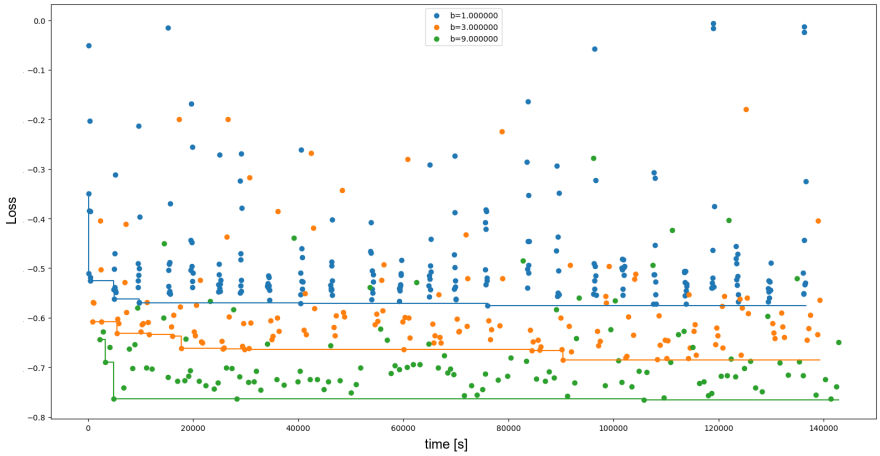
In order to measure the performance of an HPO algorithm, it may be important to study and compare the time and the optimized result related to the set objective function when it is run on a smaller set of HPs. Since training a DNN model can be a lengthy process, this part of the research conducts an experiment on tuning DNN for classifying different attacks on the ToN-IoT GPS telemetry dataset. In this experiment, BOHB algorithm was run twice. One time to tune the 8 HPs of a DNN model as defined in Table 2.6, and another time to tune only the learning rate, the number of neurons, and the number of hidden layers. Table 2.21 and Figures 2.14 (a) and (b) highlights the results of this comparison. On the one hand, the table demonstrates a slight advantage of the optimized model where only 3 HPs were tuned. This might be attributed to the fact that more hidden layers and neurons per layer were tried with different learning rates. However, this led to more consumption of time as DNNs with different architectures would need different training times; the larger the network the more resources it requires, especially, if the associated learning rate was small. On the other hand, it may be noted that BOHB when used to tune more HPs (8 as defined in Table 2.6) was still able to converge and did not require higher number of full function evaluations. In both runs, only 200 evaluations were allowed. Moreover, as depicted in Figure 2.14 (a) BOHB when used to tune 8 HPs has taken less time to converge to a similar value achieved in the other run which used to tune only 3 HPs including the number of hidden layers.

Table 2.21: Performance comparison of optimizing DNN model using BOHB

Metric	8 HPs	Only 3 HPs
Weighted F1_Score	82.52	**82.87**
Macro F1_Score	79.66	**79.98**
Accuracy	**82.83**	83.29
AUC	93.66	**94.40**
FPR	6.94	**4.62**
FNR	**5.75**	6.57

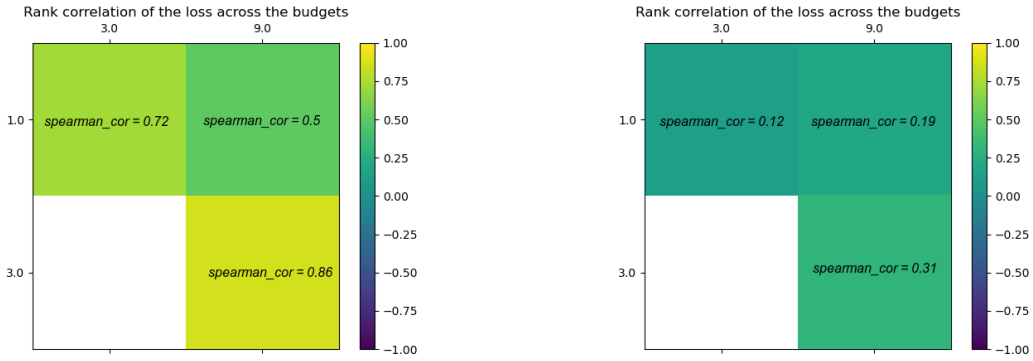


(a) Losses for different budgets over time while tuning 8 HPs



(b) Losses for different budgets over time while tuning 3 HPs

Figure 2.14: Losses for different budgets over time



(a) Rank correlation of the loss across budgets tuning 8 HPs

(b) Rank correlation of the loss across budgets tuning 3 HPs

Figure 2.15: Rank correlation of the loss across budgets

It may be noted that limiting the search space to the most important HPs only, such as, the learning rate, allowed the HPO to try more combinations. However, this does not reflect much better performance. In fact, tuning 8 HPs of the DNN model resulted in a slightly better performance when factoring in the time of convergence. The optimized overall F1-Score on the multi-class classification task when 8 and 3 HPs were tuned was 82.52% and 82.87%, respectively. Furthermore, Figures 2.15(a) and (b) demonstrate that using the larger search space allowed the HPO to have better correlation for the defined budgets for BOHB algorithm.

2.8.5 The effect of allowing the HPOs to run more full function evaluations on performance

Setting up the allowed number of full function evaluations for the HPO algorithm may affect its convergence performance. As discussed in Section 2.8.1, the three HPO algorithms required at least 140 full function evaluations to converge. However, the question here is, would the HPO algorithm converge to a better result if it was allowed to run more full function evaluations? To answer this question, TPE has been re-run with 1200 full function evaluations to tune the HPs of a GBM model using the ToN-IoT GPS telemetry dataset. Figure 2.16 highlights the progress of the TPE algorithm when allowed more full function evaluations. At 200^{th} trial, the optimal F1-Score, which was used as an objective function, was 89.55%. However, after allowing more time and more function evaluations, the TPE

algorithm converged to a better F1-Score of 90%. Comparing this with the difference in performance from the 1st trial to the 200th trial, it can be noted that the difference was small. The TPE algorithm started with F1-Score of 88.32% at the early iterations, by the time it reached the 200th trial, it have improved by 1.23%. On the other hand, the enhancement seen from the 200th trial to 586th trial was only around 0.45%. In addition, more than half of the allowed full function evaluations (around 600 trials), which are expensive and may translate to days of CPU time, did not yield better performance. As depicted in Fig. 2.16. from the 586th trial forward, TPE algorithm could not converge any further. This experiment is very important and may help other researchers on setting up an acceptable number of trials to use while using an HPO algorithm.

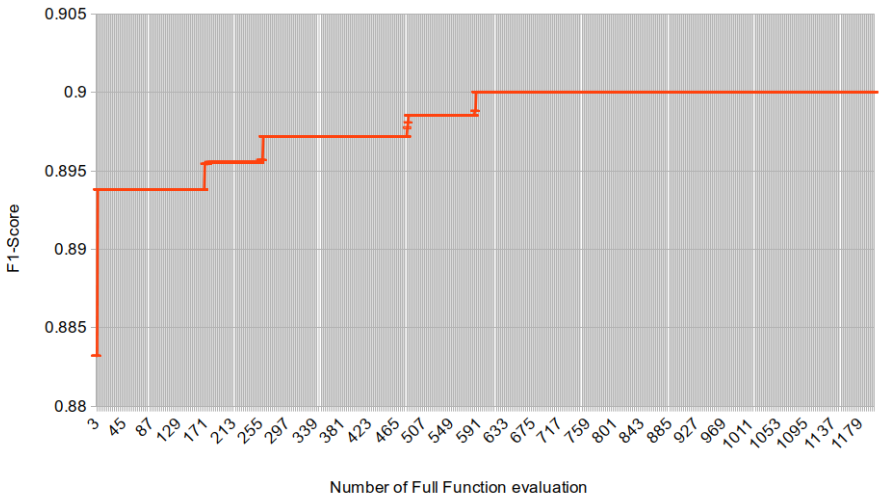


Figure 2.16: Convergence of BOHB when allowed more full function evaluations

2.9 Summary

To evaluate the performance of the ML/DL algorithms, in this chapter, two broad approaches were presented: data pre-processing and HPO techniques. This chapter demonstrates that through HP optimization and by defining a large HP search space, the learning algorithms can achieve better performance. Moreover, a data balancing technique to handle the imbalance in some IoT datasets was presented. The algorithm can help train DNNs that can distinguish different attack types more accurately regardless of their representation proportionality within the original dataset.

Chapter 3

Data Balancing and CNN based Network Intrusion Detection System[1]

3.1 Introduction

Cloud computing, IoT technologies, and the generations of wireless technologies are advancing expeditiously [70]. With the help of these advanced technologies, millions of users and devices are interconnected. This creates more opportunities for cyber-security attackers to target more victims. Securing users' information and protecting the IoT devices is crucial for the continuation of the communication process [22]. Knowing that some of the targeted systems may have a strong NID system, cyber-security attackers use reformed attack methods. Therefore, a well-performing NID system must be able to distinguish new attacks even if it has not seen any or many of them [27].

Many ML based NID systems have been introduced recently [70]. However, while implementing such systems, ML engineers have to address several issues. For instance, fitting models on an imbalanced dataset may result in a high FAR on the minority classes [35, 10]. Figure 3.1 shows a system model for a secure IoT network where an IoT cloud server trains an ML model for detecting network attacks and updates it frequently when a new form of attack emerges. While the system allows legitimate users to subscribe and

[1]The contents of this chapter have been incorporated within a paper that has been submitted and accepted for publication. Omar Elghalhoud, Kshirasagar Naik, Marzia Zaman, and Ricardo Manzano S, "Data Balancing and CNN based Network Intrusion Detection System" Submitted to IEEE Wireless Communications and Networking Conference (WCNC), 2023. Submission date Sep. 30, 2022. Acceptance date Dec. 9, 2022.

connect to IoT services securely, it blocks abnormal traffic and prevents attackers from harming legitimate users or the IoT infrastructure.

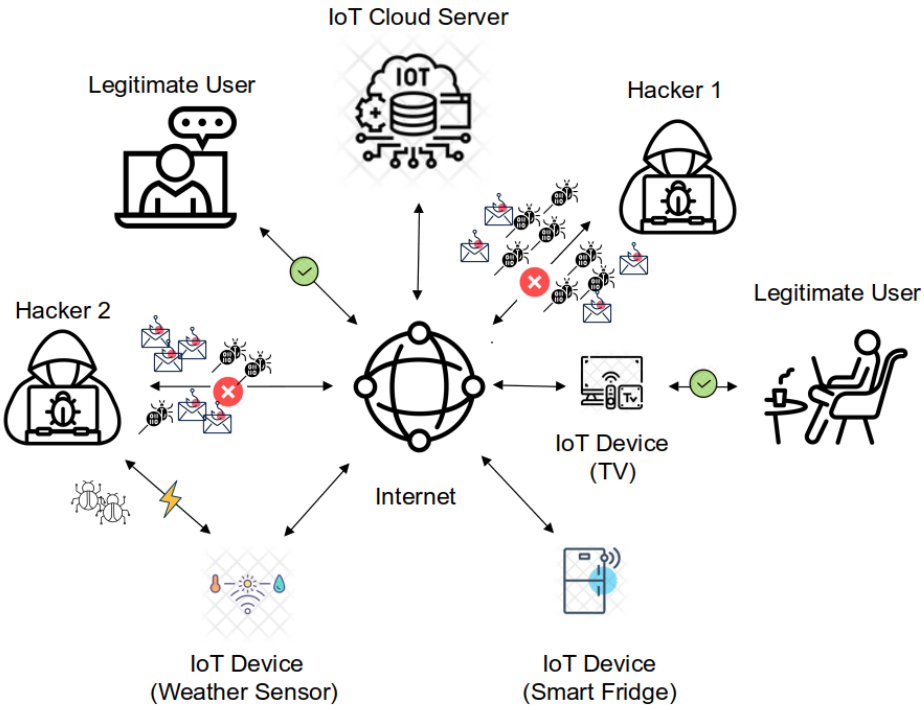


Figure 3.1: System model

It may be noted in Figure 3.1 that the attempts of "Hacker 2" to bring the network down through DoS attacks or to phish for user information keep failing since the deployed ML model has seen many of these attacks in the training phase. Consequently, the hacker disguises the DoS attack in a new form. In such a scenario, the NID system should be able to generalize well, flag the new form of attack, and through an effective balancing strategy, the system should use this incident to update and enhance the performance of its applied ML model.

To tackle the data imbalance issue, researchers have incorporated different balancing techniques into their NID systems. The simplest and most popular of these is random oversampling where randomly selected samples of the minority classes are replicated [16]. Other techniques involve the creation of new synthesized samples. For instance, SMOTE creates a new sample out of **k** neighbor samples [77]. Since the introduction of SMOTE,

other variants of it have been proposed. For example, Borderline-SMOTE is a variant of SMOTE where any sample from the minority class that has more neighbors from the majority class is declared as a borderline sample and used to generate new samples[27]. This technique improves the classification process as samples near the borderline are more likely to be classified incorrectly [27].

Another challenge that ML engineers face is working with raw input features of the network traffic data. Without proper feature extraction, the performance of the end model can be poor [35]. Classical ML algorithms can provide good performance on some NID datasets [46, 62]. However, it becomes difficult for these algorithms to classify the different attack types on complicated datasets with many features [16, 22] leading researchers to explore the DL realm more. DL extracts new data representations from the input features making different attack types more distinguishable. For instance, Variational Auto-Encoders (VAE) and GAN are examples of DL algorithms that are able to generate new samples after learning the deep representations of hidden input features [71, 70].

This chapter demonstrates the following contributions of this thesis:

- In this chapter, an NID system that combines the strength of CNN for feature extraction with a balancing technique for improving the system's performance on the minority classes is proposed.
- This chapter shows that data balancing affects the performance of ML models immensely, and compares the performance of the proposed NID system with other systems [71, 70, 27] that utilize other data balancing techniques.

To validate the aforementioned contributions, two highly imbalanced datasets: the NSL-KDD dataset [63] and the BoT-IoT dataset [38] were used. The proposed NID system achieves accuracies of 85.49% and 99.99% on the NSL-KDD and the BoT-IoT datasets, respectively.

3.2 Literature review & Related work

Although many NID systems have been proposed by researchers in this field, not all of them address the class imbalance in the data which can lead to poor performance in the minority classes. Fatani et al. [22] introduced a method based on CNN for feature extraction and Transient Search Optimization based on Differential Evolution (TSODE) for feature selection. Their method scores a good weighted F1-Score of 76.52% with the NSL-KDD

Test+ dataset. However, since their method focuses mainly on feature selection and does not address class imbalances, its performance on the minority classes is poor. The F1-Score of TSODE [22] in the U2R and R2L classes are 0.00% and 9.50%, respectively. Divekar et al. [17] have examined the class imbalance issue on multiple NID datasets, including the NSL-KDD dataset. They have compared the performance of different ML algorithms before and after data balancing using the SMOTE method. Their neural network's performance on the U2R and the R2L minority classes of the NSL-KDD Test+ dataset jumps from 5% and 3% to 18% and 31%, respectively. Gupta et al. [27] proposed a model that works in two stages. In the first stage, normal traffic is classified, then in the second stage, different attack types are classified in a One versus One (OvO) manner. Furthermore, their proposed model (LIO-IDS) [27] handles the class imbalance issue by using techniques, such as, ROS and SMOTE. The weighted F1-Score of their model reaches 80% on the NSL-KDD Test+ dataset. Jiang et al. [35] proposed an NID system that addresses the class imbalance issue via oversampling the minority classes using SMOTE while using one-side selection (OSS) for reducing noise samples in the majority class. For feature extraction, they have developed a deep hierarchical network model based on CNN and Bi-directional Long Short-Term Memory (BiLSTM) network [35]. Their system achieves a classification accuracy of 83.58% on the NSL-KDD Test+ dataset.

While classical data balancing techniques, such as ROS and SMOTE, may enhance the performance of ML algorithms in the minority classes, DL techniques for generating new samples have shown much promise to attract researchers' attentions. Yang et al. [70] came up with an Improved Conditional Variational Auto-Encoder (ICVAE), which they have used to maximize the contrast between attack types and to generate new samples. They have used DNN model as a multi-class classifier. Their proposed NID system (ICVAE-DNN) achieves an accuracy of 85.97% and a recall of 44.41% in the R2L minority class of the NSL-KDD Test+ dataset. Their proposed model performed better than the other techniques they have tried: SMOTE, ADASYN and ROS. Another NID system that is based on DNN was introduced by Yang et al. [71]. They have tackled the data imbalance issue through integrating Wasserstein Generative Adversarial Network-Gradient Penalty (WGAN-GP) with Supervised Adversarial Variational Auto-Encoder With Regularization (SAVAER) [71]. Even though GAN based models, proposed in references [31] and [13], have shown promising results, training these networks is not an easy task. Yang et al. [71] used WGAN-GP rather than the classical GAN as it allows for more stable training of the network. In Section 3.6, a performance comparison of these different techniques is conducted. The following Section studies data sampling methods and examines them in more detail.

3.3 Review of data sampling methods

This section outlines the existing sampling approaches that are used for data balancing. These methods can be classified into under-sampling approaches, over-sampling approaches, and a combination of both under-sampling and over-sampling.

3.3.1 Under-sampling techniques

Under-sampling is used to balance datasets by reducing the number of samples from the majority class. Selecting which samples to keep from the majority class will affect the performance of ML algorithms. Hence, many under-sampling were introduced by researchers.

Random Under-Sampling (RUS)

Random Under-Sampling (RUS) selects the samples to be kept from the majority class randomly. Therefore, this technique requires doing this process multiple times in order to make sure that such random selection is sufficient for the dataset at hand. Furthermore, when using RUS on a highly imbalanced dataset where the imbalance ratio is very high and the majority class consists of many samples compared to the minority class, picking samples to be a representative of the majority class randomly can be satisfactory, especially, if the process includes over-sampling of the minority class. RUS can help reduce the overall training time as the process of under-sampling is done randomly rather than relying on some modeling or heavy calculations.

Near Miss Under-Sampling

Near miss method works on cleaning the boundary border by calculating the distance between the majority samples and the minority samples [75]. There are three types of Near Miss: The first selects samples from the majority class that have the smallest average distance to the k closest samples of the minority class, the second type of Near Miss selects samples from the majority class that have the smallest average distance to the k furthest samples of the minority class, and the third selects samples from the majority class that have the smallest distance to each sample from the minority class [75].

Condensed Nearest Neighbor Under-Sampling

Condensed Nearest Neighbor Under-Sampling reduces the majority class through eliminating samples that are away from the decision boundary [28]. It maintains data samples from the majority class that are more similar [56].

Belgrana et al. [4] showed that using Condensed Nearest Neighbor Under-Sampling helps minimize the FAR in an efficient manner. However, while their method scores a low FAR of 2.25% on the NSL-KDD dataset, their method does not perform well on the recall metric: it achieves 73.5%.

Tomek-Links Under-Sampling

TOMEK [1] suggested changing the condensed nearest neighbor method so it does not remove samples randomly. Instead, Tomek proposed to target samples that are close to the decision boundary when reducing the majority class samples [1]. Two data samples are declared to be Tomek-Link samples if they belong to different classes and each of them is the closest sample to the other [64].

Mbow et al. [48] made use of Tomek-Link under-sampling to clean the CICIDS2017 dataset from overlying samples. They have combined Tomek-Links under-sampling with SMOTE for the over-sampling process to achieve a high-performing ML model. Their CNN model reaches an overall F1-Score of 98.98% using the CICIDS2017 dataset [48] and F1-Score of 100% on one of the lowest represented attacks (DoS Slowhttptest) of the same dataset. Using the NSL-KDD dataset, the F1-Score of the NID system introduced by MBOW et al. [48] on the R2L class was 62% and the overall F1-Score was 99.27%. It may be noted that the high performance of their system on the NSL-KDD dataset can be attributed to the test dataset being used is not the conventional NSL-KDD Test+ dataset [48].

One-Sided Selection Under-Sampling (OSS)

OSS is a combination of two under-sampling techniques [40, 73]. It relies on the Condensed Nearest Neighbor under-sampling technique to remove some of the majority samples that are away from the decision boundary using nearest neighbor rule; however, it removes noisy samples on the decision boundary as well by identifying and eliminating Tomek-Links [73].

Velarde-Alvarado et al. [66] compared the performance of multiple ML approaches to handle the data imbalance issue. They showed that SVM combined with OSS under-

sampling method outperforms other sampling methods, such as, SMOTE, and cost-sensitive learning methods, such as, weighted decision trees.

Cluster Centroids Under-Sampling

Cluster centroids under-sampling method bundles samples that are close to each other into K clusters and assigns a cluster centroid for each of these collections. Then based on the targeted new ratio of the majority sample to the minority samples, a sufficient number of samples belonging to the majority class are withdrawn from each cluster and removed [72]. Parsaei et al. [53] used SMOTE for over-sampling and combined it with Cluster Centroids and Nearest Neighbor [45] for the classification achieving an accuracy of 55.91% on the U2R attack type of the NSL-KDD dataset. Silva et al. [60] tried multiple under-sampling techniques to balance CICIDS2017 dataset and concluded that for distance-based ML algorithms, cluster centroid under-sampling was the best choice. The K-Nearest Neighbor (KNN) model they fitted on the CICIDS2017 dataset after using cluster centroid under-sampling method achieved 99% on the accuracy, recall, and F1-Score metrics.

3.3.2 Over-sampling techniques

Over-sampling is used to balance datasets by introducing new samples to imbalanced datasets. The new samples are generated to increase the number of samples of the minority class. They can be replicated samples out of the original samples as in ROS, or they can be generated out of the original samples after adding some noise as in SMOTE. The following techniques are the main existing over-sampling methods.

Random Over-Sampling (ROS)

ROS selects samples from the minority classes to replicate randomly relying on the conditional density estimate of the data [31, 49]. It does not produce any noise to the original samples [49]; however, due to its randomness, a good practice while applying this method is to repeat the process a few times to ensure that the randomness selection is sufficient for the given dataset.

Liu et al. [46] compared ROS to other over-sampling methods: SMOTE and ADASYN using the NSL-KDD dataset, the UNSW-NB15 dataset, and the CICIDS2017 dataset. The authors showed that the performance of these methods is very close. The accuracy achieved by the applied ML algorithm after balancing the CICIDS2017 dataset using ROS, SMOTE,

and ADASYN was 99.9%, 99.9%, and 99.91%, respectively. Gupta et al. [27] used ROS among other over-sampling techniques in the second layer of their two-layer NID system. They have conducted experiments to compare the performance of over-sampling combined with random forest and bagging methods. Using the CICIDS2017 dataset as a benchmark, they show that the used over-sampling methods achieve similar performance [27]. When combining bagging with ROS and SMOTE, both score 100% detection rate for all classes except for the DoS class where both score 74% and Patator class where ROS scores 100% and SMOTE scores 92% [27].

Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE generates new data samples from the minority classes by introducing some noise to the original samples [11]. The process of creating synthesized samples out of a minority sample involves the use of the k nearest samples to this minority sample where a new sample will be along the line of the original minority sample and its k nearest neighbors [11]. The exact position in this line is determined randomly. However, SMOTE may cause the classification performance to degrade if the majority class and minority class are very close [31].

Zhang et al. [78] used SMOTE to over-sample the minority classes of the NSL-KDD dataset achieving better performance compared to using the original dataset without balancing. JIANG et al [35] used SMOTE to balance two datasets: the NSL-KDD dataset and the UNSW-NB15 dataset [51]. They have managed to improve the detection rate of their NID system from 76.84% to 83.76% after applying a hybrid sampling method that included SMOTE for over-sampling the minority classes. To eliminate the noise from the majority classes, they have used OSS [78]. Similar to this, Zhang et al. [77] used SMOTE for oversampling; however, for under-sampling and reducing the noise from the samples belonging to the majority classes, they used Edited Nearest Neighbour (ENN) method. The combination of these two sampling methods allowed the authors to achieve an accuracy of 83.31% using the NSL-KDD dataset. Furthermore, they have managed to improve the detection rate of the R2L class from 12.54% to 40.31% [77].

Adaptive Synthetic Sampling (ADASYN)

ADASYN Sampling method modifies the imbalanced dataset such that the decision boundary is changed to adapt for learning the difficult samples to classify [30]. ADASYN focuses on generating new synthesized samples in low-density areas of the minority samples' distribution [12, 16]. After determining which minority samples to use for over-sampling, the

process of creating synthesized samples in ADASYN is similar to the SMOTE's process where the new sample would lie along the line between the original minority sample and its nearest neighbor [12]. Unlike SMOTE the noise introduced by ADASYN is not high [12].

Liu et al [46] applied ADASYN to over-sample the minority classes within the NSL-KDD dataset, the UNSW-NB15 dataset, and the CICIDS2017 dataset. Using Friedman and Nemenyi test [23], the authors have compared the performance of ADASYN with other over-sampling methods: SMOTE, and ROS. They found that these three oversampling algorithms almost have the same performance. Comparing ADASYN to SMOTE, the authors show that the average ordinal difference is just 1.17 [46]. Similarly, CHEN [12] conducted a comparison between SMOTE and ADASYN for detecting network intrusions using the CICIDS2017 dataset. The reported results show that the performance of both sampling methods is very close under multiple metrics. For instance, both methods scored an AUC value around 0.997 [12]. He et al. [30] when introducing ADASYN, they have compared its performance with SMOTE using five benchmarking datasets. ADASYN won four times over SMOTE on the recall metric.

Generative Adversarial Networks (GAN)

GAN are deep learning technique that is used mainly for generating new synthesized data samples to train DL models on an enhanced dataset to avoid over-fitting and to make DL models generalize well. Training a GAN model consists of training two models: a generator and a discriminator [31, 13, 71]. The former is the one responsible for generating new samples and the latter is used to determine the quality of the generated samples [31, 71]. The ideal situation, when training such models, is when the discriminator is unable to distinguish the fake data samples from the original data samples [31, 13]. Vanilla GAN model trains on the given data to learn its distribution without paying much attention to the labels associated with different samples [31]. Furthermore, training vanilla GAN is a tough task as it does not converge easily [71, 67]. Therefore, variants of GAN were introduced to account for these issues. For instance, Conditional Tabular Generative Adversarial Network (CTGAN) [68] and Auxiliary Classifier Generative Adversarial (ACGAN) [52] allow for sampling from a specific attack type within the dataset after training the GAN on the whole dataset. Another variant of GAN is WGAN-GP [26] which allows for steady network training and reduces the convergence time [71, 67].

Andresini et al. [3] trained an ACGAN to use it for balancing the Canadian Institute for Cyber-Security Intrusion Detection System 2017 (CICIDS2017) dataset. They have

managed to achieve an F1-Score of 98.71%. The high performance of their proposed system is attributed to combining ACGAN with two-dimensional CNN. Using only the CNN, the F1-Score reached only 89% [3]. Chkirbene et al. [13] highlighted the importance of optimizing the GAN in order to achieve higher performance. They have managed to increase the detection rate of the U2R attack type on the NSL-KDD dataset from around 20% to 57%. Also, they have achieved an accuracy of 86%. Ding et al. [16] extended the ACGAN model to enable its use for tabular data introducing Tabular Auxiliary Classifier Generative Adversarial Networks (TACGAN) model. They have included two extra loss functions to penalize any deviation between the original data and the generated data [16]. For validating their approach, they have compared its performance against other GAN based models. Using the CICIDS2017 dataset for benchmarking, TACGAN achieved a higher F1-Score of 95.81% compared to 93.34% and 92.30% achieved by ACGAN and Wasserstein Generative Adversarial Networks (WGAN), respectively. Their model performance was competitive with the IGAN-IDS proposed by Huang et al. [31]. TACGAN achieved a slightly better F1-Score of 95.81% compared to an F1-Score of 95.48% achieved by the IGAN-IDS. Wang et al. [67] utilized WGAN to build an NID system based on CNN. Their NID system achieved an F1-Score of 97.78% using the CICIDS2017 dataset. While the experiments conducted by Wang et al. [67] show the effectiveness of their system. The authors highlighted that ROS scored a better F1-Score than vanilla GAN. The F1-Score of ROS was 96.26% while the vanilla GAN achieved an F1-Score of 94.19%. Furthermore, the same trend may be noted in the performance on specific attacks. WGAN scored better than vanilla GAN and the other methods, such as, ROS and SMOTE. However, ROS was better than vanilla GAN. On the slow HTTP test attack type, ROS achieved an F1-Score around 94% while vanilla GAN achieved an F1-Score around 88%. Chapaneri et al. [10] introduced a regularized version of WGAN. Their method Wasserstein Generative Adversarial Networks with Improved Deep Regret (WGAN-IDR) uses a one-sided penalty rather than the two-sided penalty that is used in WGAN-GP [26]. Using the CICIDS2017 dataset for benchmarking, they have compared their method to other data balancing techniques, such as, ROS, SMOTE, and WGAN-GP. Their method achieved an F1-Score of 98% which is better than the F1-Score achieved by the other methods.

3.4 Benchmarking Dataset

There are several datasets in the literature that simulate network traffic. However, to evaluate how an ML algorithm copes with high class imbalance, the NSL-KDD has been chosen as a benchmarking dataset. The NSL-KDD dataset has an Imbalance Ratio (IR) of

1295:1. Figure 3.2 highlights this high class imbalance, where U2R attack traffic constitutes only 0.04% of the whole training dataset, while DoS traffic constitutes more than 36% of the training dataset.

The NSL-KDD dataset is an improved version of the KDD'99 dataset. The improvements introduced in this dataset can be listed as:

- More than 75% of the training and testing data samples in the KDD'99 dataset were identified as redundant samples, and, therefore removed [63].
- Easily predicted data samples were removed from the KDD'99 dataset.
- The NSL-KDD dataset includes another challenging test dataset (Test-21 dataset) where 21 ML algorithms have been applied and data samples that were predicted correctly by all the 21 ML algorithms were removed.

The NSL-KDD training dataset's probability distribution differs from the test datasets' probability distribution [63], which makes the NSL-KDD dataset a more challenging dataset for evaluating the generalizability of the proposed NID system. Moreover, to test the robustness of this research's approach, it has been validated using the BoT-IoT dataset [38] as well.

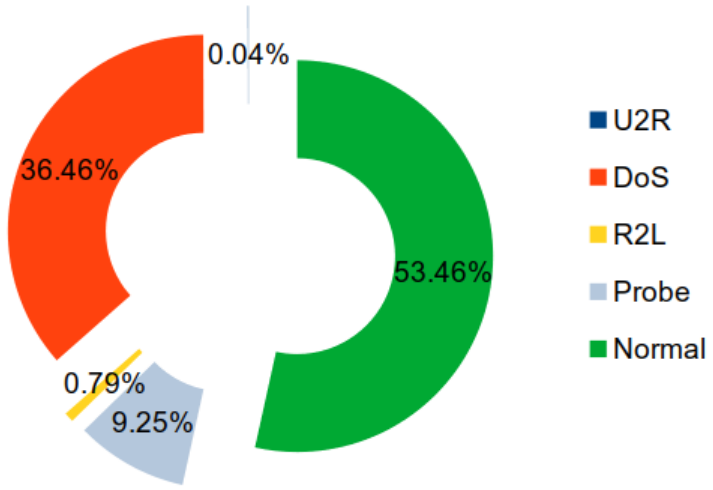


Figure 3.2: Network traffic distribution of NSL-KDD Train+ dataset

3.5 Proposed Methodology

In this Section, the proposed methodology and the architecture of the implemented CNN model are presented.

Categorical feature transformation: One hot encoding has been used to restrain the ML algorithm from assuming certain ranks when the values of the categorical features are converted to numerical values.

Data Balancing: Figure 3.2 demonstrates how highly imbalanced the NSL-KDD dataset is. To prevent the ML algorithms from having a bias toward highly represented attacks, Algorithm 2 has been applied; however, unlike the model presented in the previous chapter, the proposed NID system in this chapter, combines the balancing algorithm with two-dimensional CNN model. Algorithm 2 is a hybrid approach designed to consider both under-sampling and over-sampling. However, for generalization, rather than fixing the balancing strategy, two hyper-parameters have been incorporated into the algorithm that can be tuned for different datasets and problems. The two hyper-parameters are the targeted proportionality ($prop$) and the reference traffic to use for balancing (ref). Tuning and having control over which traffic to use for balancing can be beneficial to alleviate unnecessary high data volume used for training the model leading to a more scalable and more sustainable model.

The algorithm uses the reference traffic to determine the ideal number of records other traffic types should relate to (Line 3). Then, it iterates through the set of traffic types (T) present in the input dataset. For each traffic type, the algorithm calculates the ratio of the total number of samples of the traffic associated with the iteration and the total number of samples of the traffic associated with the reference traffic ($scale$ in Line 6). The ratio will determine whether the algorithm needs to over-sample or under-sample the traffic associated with the iteration.

Due to the effectiveness and simplicity of RUS, it is the under-sampling technique used in Algorithm 2. Since Algorithm 2 has been designed for balancing highly imbalanced datasets where the majority classes would have a large number of samples compared to the minority classes, selecting random samples to be removed may not affect the distribution within these classes very much, especially if the majority class has a uniform distribution. However, when over-sampling, rather than creating synthesized samples of minority classes that include some noise or using randomly selected samples, a replication process was employed. Notwithstanding the effectiveness of ROS, replicating the existing minority samples almost equally may result in a much more balanced dataset, especially when the selected balancing strategy requires duplicating the minority classes twice or more.

Algorithm 2 Data balancing algorithm

INPUT: Imbalanced Training Dataset (D_{train}), Reference traffic to use for balancing (ref), Proportion to add ($prop$)
OUTPUT: More Balanced Training Dataset (D_b)

procedure BALANCE_IOT_DATASET($D_{train}, prop, ref$)
  $D_b \leftarrow \emptyset$
  $no\_rec \leftarrow size(D_{train}[type == ref])$
  for each $t \in \mathcal{T}$ **do** ▷ $\mathcal{T}$ is a set of the traffic types present in D_{train}
    $D_t \leftarrow D_{train}[type == t]$
    $scale \leftarrow no\_rec/size(D_t)$
    if $scale <= 1.0$ **then**
      $D_{scaled} \leftarrow sample\_records(D_t, scale)$
      $D_b \leftarrow D_b \cup D_{scaled}$
    else
      $rc \leftarrow int(scale * prop)$ ▷ rc is a replication counter
      if $rc = 0$ **then** $rc \leftarrow 1$
      end if
      while $rc \neq 0$ **do**
        $D_b \leftarrow D_b \cup D_t$
        $rc \leftarrow rc - 1$
      end while
    end if
  end for
  $D_b \leftarrow shuffle(D_b)$
  return D_b
end procedure

Applying a method that adds noise to the original samples on a highly imbalanced dataset may lead to a high noise level that can affect the performance of ML algorithms. A performance comparison between Algorithm 2 and other data balancing techniques, such as, ROS, SMOTE, and ADASYN is conducted in Section 2.8 of this chapter. After tuning ref and $prop$, the Service Scan traffic type has been used as a reference traffic to balance the BoT-IoT dataset with $prop$ set to 1.0 and DoS has been used as a reference traffic to balance the NSL-KDD training dataset with $prop$ set to 1.0 (see Fig. 3.3).

Data Normalization: Without normalizing input features, the trained ML model may have a bias toward features with higher values which can lead to other features being neglected. Furthermore, normalization expedites the training process [71]. Equation (3.1) demonstrates how the min-max normalization is applied.

$$x^{'} = a + \frac{x - min(X)}{max(X) - min(X)} * (b - a) \tag{3.1}$$

where X is the input dataset, $x^{'}$ is the normalized value of $x \in X$, and $[a, b]$ is the interval in which the transformed input values should fall within.

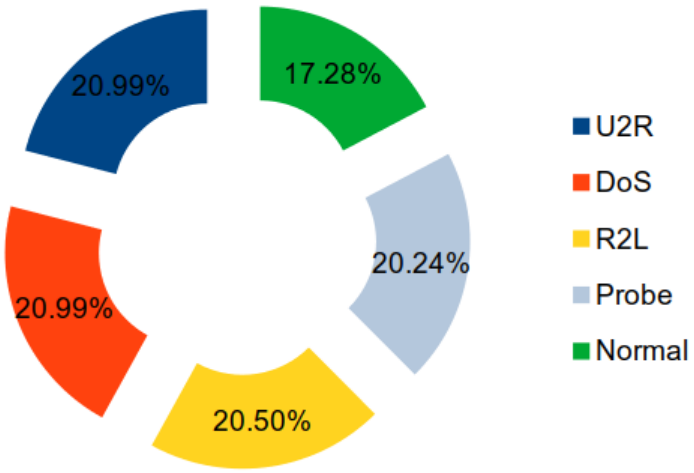


Figure 3.3: More balanced training NSL-KDD dataset

3.5.1 Proposed DL model

CNN is a deep learning algorithm that has the ability to extract spatial features [31, 35]. Since its introduction, CNN has been used as a base for most of the state-of-the-art image classification models. Nevertheless, CNN has also been applied to other fields, such as, speech recognition and natural language processing [71, 35]. Researchers used different types of CNN models in the field of NID. For instance, Fatani et al. [22] have used one-dimensional CNN with 64 filters of 1x3 size to extract spatial features. Jiang et al. [35] reshaped the flat input features into a two-dimensional format to apply two convolutional layers with a different number of filters and different kernel sizes. Some of the CNN based models reported in the literature, such as, IGAN-IDS [22] and TSODE [31], were successful in classifying different traffic types present in the NSL-KDD dataset. However, others had poor performance, such as, the ReseNet applied by Li et al. [44]: it achieved an accuracy of 81.57%; however, its FAR was around 99% high on the NSL-KDD Test-21 dataset. The variation in performance of the reported CNN models may be related to the architecture and the hyper-parameters used by different researchers. Furthermore, CNN models that were combined with balancing techniques and other DL methods had more success.

The proposed CNN model in this research, illustrated in Fig. 3.4, uses a two-dimensional convolutional layer of 64 filters with the size of 3x3. Rectified Linear Unit (RELU) activation is used for this layer. Then batch normalization and drop out of 50% are applied to the output of the convolutional layer to avoid over-fitting. The extracted spatial features

go into a feed-forward neural network of two layers.

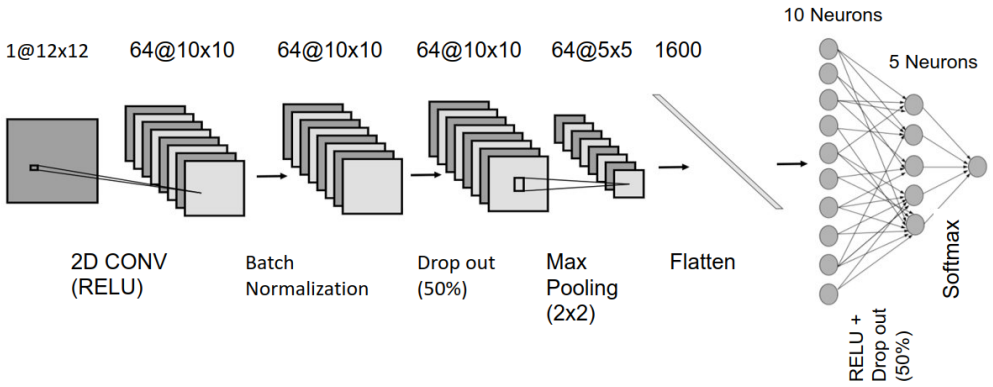


Figure 3.4: The proposed CNN model

3.6 Results and discussions

3.6.1 Multi-class classification results

The proposed CNN model combined with Algorithm 2 achieves high performance: weighted F1-Score of 85.96%, macro F1-Score of 72.15%, and weighted recall of 85.49% on the NSL-KDD Test+ dataset. Algorithm 2 enhances the performance of the CNN model on the minority classes remarkably while maintaining high performance on the majority classes. With data balancing, the recall of the R2L class rises drastically from 8% to 72%, and, similarly, the recall of the U2R class improves significantly from 0% to 70%. However, this comes at a small cost in the recall performance of the Probe class. The recall of the Probe class decreases from 74% to 66%. In spite of this, the F1-Score of the Probe class increases from 71% to 76%. Considering the slight performance decrease in the Probe class and the immense improvement in the minority classes (R2L and U2R), the proposed CNN model performance, when combined with the data balancing technique, is far better than the model performance without data balancing.

To further validate the performance of this research's approach, it has been tested against the other difficult NSL-KDD test dataset (Test-21). Then, a comparison between its performance and the performance of other models that used other data balancing techniques was conducted. Since some of the compared models used DNN, another model

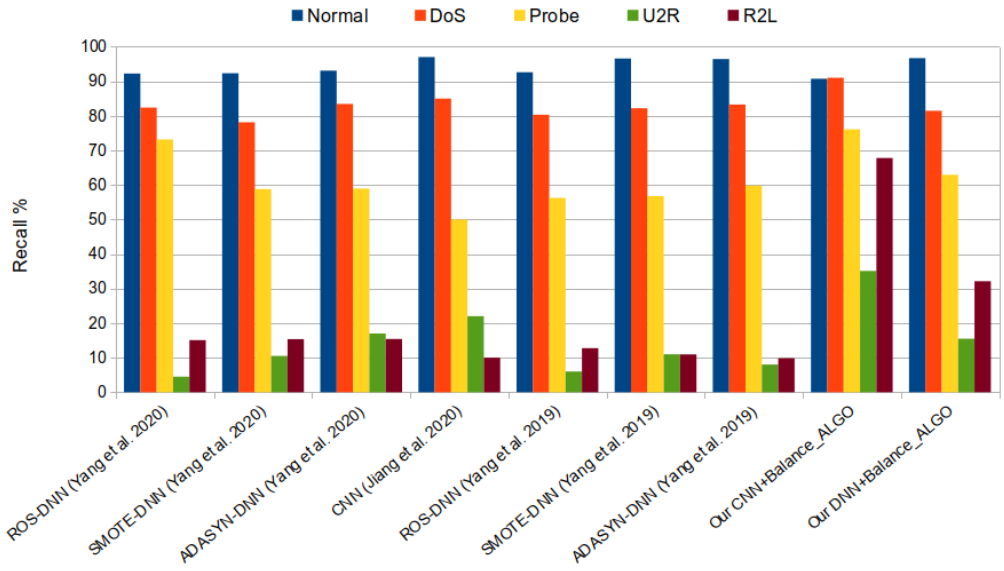


Figure 3.5: Comparison of different balancing techniques using the Test+ dataset

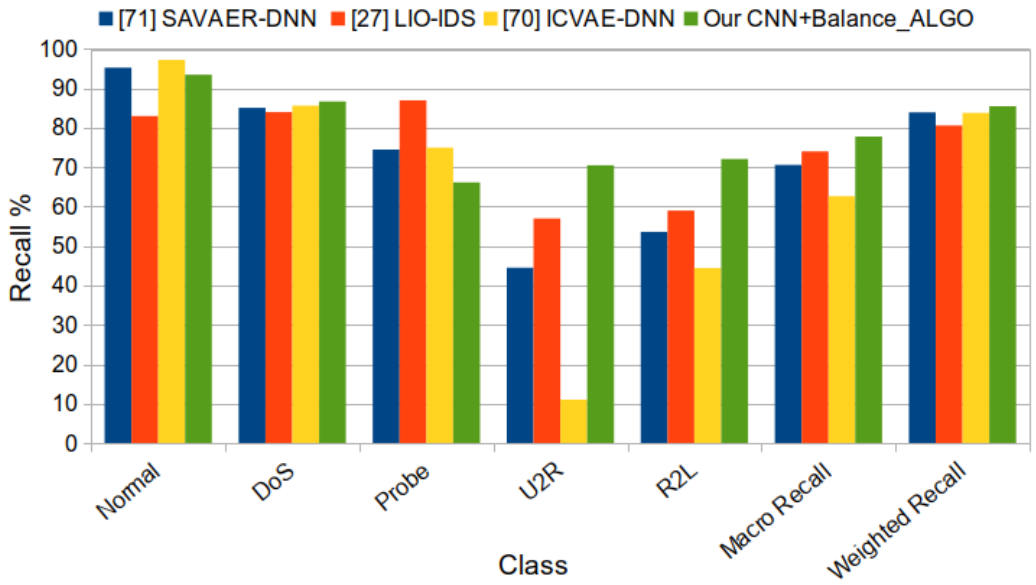


Figure 3.6: Performance comparison between the best models using the Test+ dataset

that uses DNN instead of CNN was implemented and combined with the Algorithm 2 to further evaluate the performance of Algorithm 2. Figure 3.5 shows that both the CNN and the DNN models, when combined with Algorithm 2, achieve better results than other DL models reported in the literature that were combined with other balancing techniques, such as, ROS, SMOTE, and ADASYN. As depicted in Figure 3.5, all methods have similar performance in the two majority classes: the normal class and the DoS class. However, they differ in the probe class and the two minority classes: the R2L class and the U2R class. The proposed CNN model combined with Algorithm 2 has a superior performance in these two minority classes. Furthermore, the implemented DNN model combined with Algorithm 2 outperforms the other DNN models that were combined with ROS, SMOTE, and ADASYN in the recall of the R2L class. It achieves a recall of 32.14% on the NSL-KDD Test+ dataset. As for the other classes, it achieves similar performance to the other methods. SAVAER-DNN [71] achieves better results than the implemented DNN model in this research. However, the CNN model proposed in this chapter of the research, achieves much higher recall scores. Moreover, the proposed system's ability to classify the different attacks in the BoT-IoT dataset has been validated. It achieves 100% F1-Score on the keylogging class compared to 86% achieved in by the DNN model proposed in the previous chapter.

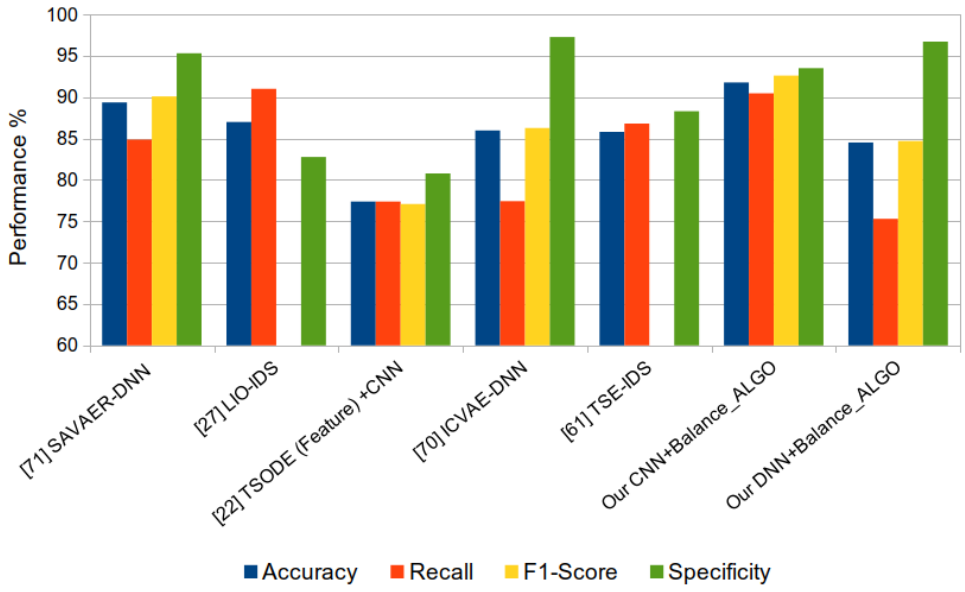


Figure 3.7: Comparison of binary classification results of the best models

Table 3.1: Multiple metrics comparison in both test datasets in the following format (Result on Test+, Result on Test-21), where **B**: the proposed model has a better result, **W**: the proposed model has a worse result, and "**-**": the result was not reported.

Metric	Class	[35] CNN-BiLSTM	[22] TSODE+CNN	[27] LIO-IDS	[71] SAVAER-DNN	[70] ICVAE-DNN	[61] TSE-IDS
Recall	Normal	(-,-)	(-,-)	**(B,-)**	(W,W)	(W,W)	(-,-)
	DoS	(-,-)	(-,-)	**(B,-)**	**(B,B)**	(B,W)	(-,-)
	Probe	(-,-)	(-,-)	(W,-)	(W,W)	(W,W)	(-,-)
	U2R	(-,-)	(-,-)	**(B,-)**	**(B,B)**	**(B,B)**	(-,-)
	R2L	(-,-)	(-,-)	**(B,-)**	**(B,B)**	**(B,B)**	(-,-)
	Macro	(-,-)	(-,-)	**(B,-)**	**(B,B)**	**(B,B)**	(-,-)
	Weighted	(-,-)	(-,-)	**(B,-)**	(B,W)	(B,W)	(-,-)
F1-Score	Normal	(W,-)	(W,-)	**(B,-)**	(-,-)	(-,-)	(-,-)
	DoS	**(B,-)**	**(B,-)**	**(B,-)**	(-,-)	(-,-)	(-,-)
	Probe	**(B,-)**	**(B,-)**	(W,-)	(-,-)	(-,-)	(-,-)
	U2R	**(B,-)**	**(B,-)**	(W,-)	(-,-)	(-,-)	(-,-)
	R2L	**(B,-)**	**(B,-)**	**(B,-)**	(-,-)	(-,-)	(-,-)
	Macro	**(B,-)**	**(B,-)**	**(B,-)**	(-,-)	(-,-)	(-,-)
	Weighted	**(B,-)**	**(B,-)**	**(B,-)**	(-,-)	(-,-)	(-,-)
Accuracy		(-,-)	**(B,-)**	**(B,-)**	**(B,B)**	**(B,B)**	**(B,B)**
Precision		(-,-)	**(B,-)**	(-,-)	(W,B)	(W,B)	**(B,B)**
Recall		(-,-)	**(B,-)**	(W,-)	**(B,B)**	**(B,B)**	**(B,B)**
F1-Score		(-,-)	**(B,-)**	(-,-)	**(B,B)**	**(B,B)**	(-,-)
FAR		(-,-)	**(B,-)**	**(B,-)**	(W,W)	(W,W)	(B,W)

3.6.2 Binary classification results and comparison with state-of-the art models

Improper use of data balancing may lead to a good classification performance in some classes, while it may result in a poor performance on the binary classification task or the overall multi-class classification performance. This section reports and compares the proposed NID system's performance on the NSL-KDD test datasets to other models reported in the literature. The proposed system performs consistently on both datasets, which is a good sign for its generalizability. Furthermore, it achieves a low FAR of 6.50% (which is the second best among the compared models) and it achieves the highest F1-Score (92.60%) on the Test+ dataset. As depicted in Fig. 3.6, the proposed system competes with other state-of-the-art models. It scores the best macro recall on both test datasets and the best weighted recall on the Test+ dataset.

Figure 3.7 demonstrates the competitiveness of the proposed system. SAVAER-DNN [71] and the proposed system in this thesis achieve high but balanced performance when considering multiple metrics. On the other hand, ICVAE-DNN [70] scores a low recall and a high specificity of 77.43% and 97.26%, respectively, while the proposed NID system scores a recall and specificity of 90.46% and of 93.5%, respectively. LIO-IDS [27] scores a competitive recall score of 91.00%, whereas it has a very high FAR of 17.24% which

evaluates in a specificity of 82.76%. When using the Test-21 dataset as a benchmark, the proposed NID system shows a consistently high performance similar to its performance on the Test+ dataset. Despite having a slightly lower specificity of 76.57% on the Test-21 dataset compared to a specificity range of 81-87% for the other compared models, the proposed model has the highest accuracy, recall, and F1-Score (above 92%). The other compared models: Two-Stage Classifier Ensemble for Intelligent Anomaly-Based Intrusion Detection System (TSE-IDS) [61], ICVAE-DNN [70], and SAVAER-DNN [71] had recall scores of 72.50%, 72.86%, and 79.96%, respectively. Table 3.1 summarizes the performance of the proposed NID system in comparison with other proposed systems in the literature. It may be noted that the proposed system wins in most of the F1-Score comparisons and all accuracy comparisons.

3.7 Summary

To study the effect of data balancing and feature extraction on the classification performance of ML algorithms on highly imbalanced datasets, in this chapter, a simple yet effective NID system was presented. This chapter showed that through proper data balancing and through a good feature extraction technique i.e., via CNN, significant improvements in the performance of DL models on the minority classes are achievable.

Chapter 4

Conclusion

To measure the performance of the ML/DL algorithms when used as a base for NID systems, in this thesis, two important factors were addressed: data balancing and HP optimization. This thesis demonstrates the importance of HP optimization to improve the performance of the applied ML/DL models. A thorough comparison between three different optimization algorithms has been conducted. Moreover, a data balancing technique to handle the imbalance in some IoT datasets was presented. The proposed algorithm can help build NID systems that can distinguish different attack types more accurately regardless of their representation proportionality within the original dataset.

References

[1] Two modifications of cnn. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11):769–772, 1976.

[2] Mohamed Ahzam Amanullah, Riyaz Ahamed Ariyaluran Habeeb, Fariza Hanum Nasaruddin, Abdullah Gani, Ejaz Ahmed, Abdul Salam Mohamed Nainar, Nazihah Md Akim, and Muhammad Imran. Deep learning and big data technologies for iot security. *Computer Communications*, 151:495–517, 2020.

[3] Giuseppina Andresini, Annalisa Appice, Luca De Rose, and Donato Malerba. Gan augmentation to deal with imbalance in imaging-based intrusion detection. *Future Generation Computer Systems*, 123:108–127, 2021.

[4] Fatima Zohra Belgrana, Nacéra Benamrane, Mohamed Amine Hamaida, Abdellah Mohamed Chaabani, and Abdelmalik Taleb-Ahmed. Network intrusion detection system using neural network and condensed nearest neighbors with selection of nsl-kdd influencing features. In *2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, pages 23–29, 2021.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[6] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[7] Salah Bouktif, Ali Fiaz, Ali Ouni, and Mohamed Adel Serhani. Multi-sequence lstm-rnn deep learning and metaheuristics for electric load forecasting. *Energies*, 13(2), 2020.

[8] Leo Breiman. Random forests. *Machine Learning*, 45(2), 2001.

[9] J. Brownlee. Difference between a batch and an epoch in a neural network, machine learning mastery, 2019.

[10] Radhika Chapaneri and Seema Shah. Enhanced detection of imbalanced malicious network traffic with regularized generative adversarial networks. *Journal of Network and Computer Applications*, 202:103368, 2022.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, jun 2002.

[12] Zhewei Chen, Linyue Zhou, and Wenwen Yu. ADASYN-random forest based intrusion detection model. In *2021 4th International Conference on Signal Processing and Machine Learning*. ACM, aug 2021.

[13] Zina Chkirbene, Habib Ben Abdallah, et al. Data augmentation for intrusion detection and classification in cloud networks. In *Int. Wireless Communications and Mobile Computing (IWCMC)*, pages 831–836, 2021.

[14] Hyunghun Cho, Yongjin Kim, Eunjung Lee, Daeyoung Choi, Yongjae Lee, and Wonjong Rhee. Basic enhancement strategies when using bayesian optimization for hyperparameter tuning of deep neural networks. *IEEE Access*, 8:52588–52608, 2020.

[15] Marc Claesen, Jaak Simm, Dusan Popovic, Yves Moreau, and Bart De Moor. Easy hyperparameter search using optunity. *CoRR*, abs/1412.1114, 2014.

[16] Hongwei Ding et al. Imbalanced data classification: A knn and generative adversarial networks-based hybrid approach for intrusion detection. *Future Generation Computer Systems*, 131:240–254, 2022.

[17] Abhishek Divekar et al. Benchmarking datasets for anomaly-based network intrusion detection: Kdd cup 99 alternatives. In *IEEE 3rd Int. Conf. on Computing, Communication and Security (ICCCS)*, pages 1–8, 2018.

[18] Omar Elghalhoud, Kshirasagar Naik, et al. Data balancing and hyper-parameter optimization for machine learning algorithms for secure iot networks. *Proceedings of the 18th ACM International Symposium on QoS and Security for Wireless and Mobile Networks*, 2022.

[19] Omar Elghalhoud, Kshirasagar Naik, Marzia Zaman, and Nishith Goel. Data balancing and hyper-parameter optimization for machine learning algorithms for secure iot networks. In *Proceedings of the 18th ACM International Symposium on QoS and Security for Wireless and Mobile Networks*, Q2SWinet '22, page 71–78, New York, NY, USA, 2022. Association for Computing Machinery.

[20] Omar Elghalhoud, Kshirasagar Naik, Marzia Zaman, and Ricardo Manzano S. Data balancing and cnn based network intrusion detection system. In *2023 IEEE Wireless Communications and Networking Conference (WCNC)*, 2023.

[21] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446. PMLR, 10–15 Jul 2018.

[22] Abdulaziz Fatani, Mohamed Abd Elaziz, et al. Iot intrusion detection system using deep learning and enhanced transient search optimization. *IEEE Access*, 9:123448–123464, 2021.

[23] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[25] Martin Grill, Tomáš Pevný, and Martin Rehak. Reducing false positives of network anomaly detection by local adaptive multivariate smoothing. *Journal of Computer and System Sciences*, 83(1):43–57, 2017.

[26] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5769–5779, Red Hook, NY, USA, 2017. Curran Associates Inc.

[27] Neha Gupta, Vinita Jindal, and Punam Bedi. Lio-ids: Handling class imbalance using lstm and improved one-vs-one technique in intrusion detection system. *Computer Networks*, 192:108076, 2021.

[28] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

[29] Mahmudul Hasan, Md. Milon Islam, Md Ishrak Islam Zarif, and M.M.A. Hashem. Attack and anomaly detection in iot sensors in iot sites using machine learning approaches. *Internet of Things*, 7:100059, 2019.

[30] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *IN: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE), IJCNN 2008*, pages 1322–1328, 2008.

[31] Shuokang Huang and Kai Lei. Igan-ids: An imbalanced generative adversarial network towards intrusion detection system in ad-hoc networks. *Ad Hoc Networks*, 105:102177, 2020.

[32] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 754–762, Bejing, China, 22–24 Jun 2014. PMLR.

[33] Ayesha Jamal, Muhammad Faisal Hayat, and Muhammad Nasir. Malware detection and classification in iot network using ann. *Mehran University Research Journal of Eng. and Tech.*, 41(1):80–91, 2022.

[34] James Bergstra, Dan Yamins, and David D. Cox. Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. In Stéfan van der Walt, Jarrod Millman, and Katy Huff, editors, *Proceedings of the 12th Python in Science Conference*, pages 13 – 19, 2013.

[35] Kaiyuan Jiang, Wenya Wang, Aili Wang, and Haibin Wu. Network intrusion detection combined hybrid sampling with deep hierarchical network. *IEEE Access*, 8:32464–32476, 2020.

[36] Taejoon Kim, Sang C. Suh, Hyunjoo Kim, Jonghyun Kim, and Jinoh Kim. An encoding technique for cnn-based network anomaly detection. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2960–2965, 2018.

[37] Mikołaj Komisarek, Marek Pawlicki, Rafał Kozik, Witold Hołubowicz, and Michał Choraś. How to effectively collect and process network data for intrusion detection? *Internet of Things*, 7:100059, 2021.

[38] Nickolaos Koroniotis, Nour Moustafa, et al. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *CoRR*, abs/1811.00701, 2018.

[39] Nickolaos Koroniotis, Nour Moustafa, and Elena Sitnikova. A new network forensic framework based on deep learning for internet of things networks: A particle deep framework. *Future Generation Computer Systems*, 110:91–106, 2020.

[40] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *In Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, 1997.

[41] Prabhat Kumar, Govind P. Gupta, and Rakesh Tripathi. An ensemble learning and fog-cloud architecture-driven cyber-attack detection framework for iomt networks. *Computer Communications*, 166:110–124, 2021.

[42] Joffrey L. Leevy, John Hancock, Taghi M. Khoshgoftaar, and Jared Peterson. Detecting information theft attacks in the bot-iot dataset. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 807–812, 2021.

[43] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18(1):6765–6816, jan 2017.

[44] Zhipeng Li, Qin, et al. Intrusion detection using convolutional neural networks for representation learning. In *Neural Information Processing*, pages 858–866, Cham, 2017. Springer International Publishing.

[45] Wei-Chao Lin, Shih-Wen Ke, and Chih-Fong Tsai. Cann: An intrusion detection system based on combining cluster centers and nearest neighbors. *Knowledge-Based Systems*, 78:13–21, 2015.

[46] Jingmei Liu, Yuanbo Gao, and Fengjie Hu. A fast network intrusion detection system using adaptive synthetic oversampling and lightgbm. *Computers & Security*, 106:102289, 2021.

[47] S. M. Mahedy Hasan, Md. Fazle Rabbi, Arifa Islam Champa, and Md. Asif Zaman. A machine learning-based model for early stage detection of diabetes. In *2020 23rd International Conference on Computer and Information Technology (ICCIT)*, pages 1–6, 2020.

[48] Mariama Mbow, Hiroshi Koide, and Kouichi Sakurai. An intrusion detection system for imbalanced dataset based on deep learning. In *2021 Ninth International Symposium on Computing and Networking (CANDAR)*, pages 38–47, 2021.

[49] Giovanna Menardi and Nicola Torelli. Training and assessing classification rules with imbalanced data. *Data Mining and Knowledge Discovery*, 28:92–122, 2012.

[50] Nour Moustafa. A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets. *Sustainable Cities and Society*, 72:102994, 2021.

[51] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015.

[52] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651. PMLR, 06–11 Aug 2017.

[53] Mohammad Reza Parsaei, Samaneh Miri Rostami, and Reza Javidan. A hybrid data mining approach for intrusion detection on imbalanced nsl-kdd dataset. *International Journal of Advanced Computer Science and Applications*, 7(6), 2016.

[54] Marek Pawlicki, Michał Choraś, Rafał Kozik, and Witold Hołubowicz. On the impact of network data balancing in cybersecurity applications. In Valeria V. Krzhizhanovskaya, Gábor Závodszky, Michael H. Lees, Jack J. Dongarra, Peter M. A. Sloot, Sérgio Brissos, and João Teixeira, editors, *Computational Science – ICCS 2020*, pages 196–210, Cham, 2020. Springer International Publishing.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[56] Eréndira Rendón, Roberto Alejo, Carlos Castorena, Frank J. Isidro-Ortega, and Everardo E. Granda-Gutiérrez. Data sampling methods to deal with the big data multi-class imbalance problem. *Applied Sciences*, 10(4), 2020.

[57] Mohanad Sarhan, Siamak Layeghy, Nour Moustafa, and Marius Portmann. Towards a standard feature set of NIDS datasets. *CoRR*, abs/2101.11315, 2021.

[58] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Feature analysis for ml-based iiot intrusion detection. *CoRR*, abs/2108.12732, 2021.

[59] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In Paolo Mori, Steven Furnell, and Olivier Camp, editors, *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018*, pages 108–116. SciTePress, 2018.

[60] Bruno Silva, Ricardo Silveira, Manuel Silva Neto, Paulo Cortez, and Danielo Gomes. A comparative analysis of undersampling techniques for network intrusion detection systems design. *Journal of Communication and Information Systems*, 36(1):31–43, Feb. 2021.

[61] Bayu Adhi Tama, Marco Comuzzi, and Kyung-Hyune Rhee. Tse-ids: A two-stage classifier ensemble for intelligent anomaly-based intrusion detection system. *IEEE Access*, 7:94497–94507, 2019.

[62] Bayu Adhi Tama and Kyung Hyune Rhee. An in-depth experimental study of anomaly detection using gradient boosted machine. *Neural Computing and Applications*, 31:955–965, 2017.

[63] Mahbod Tavallaee et al. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, 2009.

[64] Ikram Sumaiya Thaseen, Vanitha Mohanraj, Sakthivel Ramachandran, Kishore Sanapala, and Sang-Soo Yeo. A hadoop based framework integrating machine learning classifiers for anomaly detection in the internet of things. *Electronics*, 10(16), 2021.

[65] Jan N. van Rijn and Frank Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2367–2376, New York, NY, USA, 2018. Association for Computing Machinery.

[66] Pablo Velarde-Alvarado, Hugo Gonzalez, Rafael Martínez-Peláez, Luis J. Mena, Alberto Ochoa-Brust, Efraín Moreno-García, Vanessa G. Félix, and Rodolfo Ostos. A novel framework for generating personalized network datasets for nids based on traffic aggregation. *Sensors*, 22(5), 2022.

[67] Anzhou Wang and Yaojun Ding. Network malicious traffic identification method based on wgan category balancing. In *2021 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 1–6, 2021.

[68] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[69] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[70] Yanqing Yang, Kangfeng Zheng, et al. Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network. *Sensors*, 19(11), 2019.

[71] Yanqing Yang, Kangfeng Zheng, Bin Wu, Yixian Yang, and Xiujuan Wang. Network intrusion detection based on supervised adversarial variational auto-encoder with regularization. *IEEE Access*, 8:42169–42184, 2020.

[72] Show-Jane Yen and Yue-Shi Lee. Cluster-based under-sampling approaches for imbalanced data distributions. *Expert Systems with Applications*, 36(3, Part 1):5718–5727, 2009.

[73] Lian Yu and Nengfeng Zhou. Survey of imbalanced data methodologies, 2021.

[74] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *CoRR*, abs/2003.05689, 2020.

[75] J. Zhang and I. Mani. In *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets*.

[76] Runchi Zhang and Zhiyi Qiu. Optimizing hyper-parameters of neural networks with swarm intelligence: A novel framework for credit scoring. *PLoS ONE*, 15(6), 2020.

[77] Xiaoxuan Zhang, Jing Ran, and Jize Mi. An intrusion detection system based on convolutional neural network for imbalanced network traffic. In *IEEE 7th International Conference on Computer Science and Network Tech. (ICCSNT)*, pages 456–460, 2019.

[78] Yan Zhang, Hongmei Zhang, Xiangli Zhang, and Dongsheng Qi. Deep learning intrusion detection model based on optimized imbalanced network data. In *2018 IEEE*

18th International Conference on Communication Technology (ICCT), pages 1128–1132, 2018.