DEPARTMENT OF COMPUTER SCIENCE

Modelling Internet Network Intrusion Detection in Smart City Ecosystems

By

WANDISA MFENGUZA

A dissertation submitted in fulfilment of the requirements for the Master of Computer Science Degree

Student Number: 201308763

Supervisor: Prof. K. Sibanda

Submission Date: May 2021

Declaration

I, **WANDISA MFENGUZA**, student number **201308763,** declare that this research is my own work. It has not been submitted before for any other degree, part of a degree or examination at this or any other university.

Signature: 

Date: 10 May 2021

Declaration on Plagiarism

I, **WANDISA MFENGUZA**, student number **201308763**, hereby declare that I am fully aware of the University of Fort Hare's policy on plagiarism and I have taken every precaution to comply with the regulations.

Signature: 

Date:10 May 2021

Declaration on Ethical Clearance

I, **WANDISA MFENGUZA**, student number **201308763**, hereby declare that I am fully aware of the University of Fort Hare's policy on research ethics and I have taken every precaution to comply with the regulations. I confirm that my research constitutes an exemption to Rule G17.6.10.5 and an ethical certificate with a reference number is not required.

Signature: 

Date: 21 May 2021

Dedication

This work is dedicated to my late mother and my amazing grandmother who encouraged me to pursue a master's degree. Thank you, Thahla, for being my pillar of strength when I felt too weak to stand on my own. You are amazing!

Abstract

Smart city systems are intended to enhance the lives of citizens through the design of systems that promote resource efficiency and the real-time provisioning of resources in cities. The benefits offered by smart cities include the use of internet of things (IoT) sensors to gather useful data such as power demand to inhibit blackouts and the average speed of vehicles to alleviate traffic congestion. Nonetheless, earlier studies have indicated a substantial increase in cyber-security issues due to the increase in the deployment of smart city ecosystems. Consequently, IoT cyber-security is recognised as an area that requires crucial scrutiny. This study begins by investigating the current state of intrusion detection in smart city ecosystems. Current intrusion detection frameworks lack the capability to operate under extremely limiting settings such as conditions of low processing power and fast response times. Moreover, the study also identifies that, despite intrusion detection being a highly researched thematic area, a plethora of previous studies tend to propose intrusion detection frameworks that are more suitable for traditional computer networks rather than wireless sensor networks (WSNs) which consist of heterogeneous settings with diverse devices and communication protocols. Subsequently, this study developed two candidate deep learning models, namely a convolutional neural network (CNN) and a long short-term memory (LSTM) network and presents evidence on their robustness and predictive power. Results have indicated that, unlike the CNN model, the LSTM model can quickly converge and offer high predictive power without the vigorous application of regularisation techniques. The proposed LSTM classification model obtained a remarkable 100% in detection rates and further reported 0% in false alarm and false negative rates. This study gives a broad overview of the current state of intrusion detection mechanisms for smart city ecosystems to guide future studies. The study also demonstrates that existing intrusion detection systems (IDSs) can be enhanced through the development of more robust and lightweight models that offer high detection rates and minimal false alarm rates to prevent security risks in smart city ecosystems to ensure sustainable and safe smart cities.

Acknowledgements

First, I would like to thank God for the abundant strength He has given me throughout this master's degree journey. I wish to express my earnest gratitude to my supervisor Professor Sibanda for his support, motivation, and guidance. A special thanks to the Department of Science and Innovation as well as the Council for Scientific and Industrial Research for their financial support. I am also indebted to my family for their support and encouragement. Finally, I would like to thank my friends who were part of this journey for all their support.

Table of Contents

Declaration....i
Declaration on Plagiarism....ii
Declaration on Ethical Clearance....iii
Dedication....iv
Abstract....v
Acknowledgements....vi
Table of Contents....vii
List of Figures....x
List of Tables....xiii
Chapter 1: Introduction....1
1.1 Introduction....1
1.2 Background....1
1.3 Problem identification....2
1.4 Research aim....2
1.5 Research questions....2
1.6 Research objectives....3
1.7 Study contribution....3
1.8 Dissertation outline....3
1.9 Summary....4
Chapter 2: Literature Review....5
2.1 Introduction....5
2.2 Smart city ecosystems....5
2.2.1 Dimensions of a smart city....5
2.2.2 Digitisation in a smart city....7
2.3 Security issues in smart city ecosystems....7
2.4 The taxonomy of denial-of-service attacks....10
2.5 Intrusion detection....12
2.6 Deep learning....16
2.6.1 Artificial neural networks....17
2.7 Related work....19
2.8 Summary....22
Chapter 3: Research Design and Methodology....23

3.1 Introduction....23
3.2 Research methodology....23
3.2.1 Qualitative approach....23
3.2.2 Quantitative approach....23
3.2.3 Simulation and modelling methodology....24
3.3 Research design....24
3.3.1 Data collection....24
3.4 Data pre-processing....27
3.5 Feature engineering and selection....28
3.6 Model training and performance evaluation....28
3.7 Summary....30
Chapter 4: Implementation....31
4.1 Introduction....31
4.2 Data processing and feature selection....31
4.3 Model training approach and performance evaluation....34
4.4 Summary....38
Chapter 5: Study Results and Discussion....39
5.1 Introduction....39
5.2 Results....39
5.2.1 Convolutional neural network performance analysis....39
5.2.2 Long short-term memory performance analysis....53
5.3 Discussion and model evaluation....69
5.4 Summary....71
Chapter 6: Conclusion....72
6.1 Introduction....72
6.2 Summary of the dissertation....72
6.2.1 Empirical findings....72
6.2.2 Research objectives....73
6.3 Implications of the study....74
6.4 Future studies....74
References....75
Appendix A: Data Collection....86
Appendix B: Training a Catboost Model....93
Appendix C: Training a Convolutional Neural Network Model....94

Appendix D: Training a Long Short-Term Memory Model.....95

List of Figures

Figure 2.1: TCP SYN flooding attack....11
Figure 2.2: Feed-forward neural network structure....17
Figure 3.1: SUMO and NS3 integration....26
Figure 4.1: Dropping duplicated features....31
Figure 4.2: Converting data types and encoding of categorical features....32
Figure 4.3: Generating a pandas-profile report....32
Figure 4.4: Catboost feature importance....33
Figure 4.5: Reshaping the data to a three-dimensional matrix....34
Figure 4.6: Plotting accuracy and loss curves....36
Figure 4.7: Plotting confusion matrix....36
Figure 4.8: Saving a model....37
Figure 4.9: Converting a regular model to a lightweight model....37
Figure 4.10: Loading and assessing the lightweight model....37
Figure 4.11: The implementation strategy....38
Figure 5.1: Model accuracy of an unregularised CNN....39
Figure 5.2:Training vs validation loss of the unregularised CNN....40
Figure 5.3: Validation set confusion matrix – unregularised CNN....41
Figure 5.4: Test set confusion matrix – unregularised CNN....41
Figure 5.5: Model accuracy of a CNN with 0.1 dropout in first layer only....42
Figure 5.6:Training vs validation loss of the CNN with 0.1 dropout in first layer only....42
Figure 5.7: Validation set confusion matrix – CNN with 0.1 dropout in first layer only....43
Figure 5.8:Test set confusion matrix – CNN with 0.1 dropout in first layer only....44
Figure 5.9: Model accuracy of a CNN with 0.2 dropout in first layer only....44
Figure 5.10: Training vs validation loss of the CNN with 0.2 dropout in first layer only....45
Figure 5.11: Validation set confusion matrix – CNN with 0.2 dropout in first layer only....45
Figure 5.12: Test set confusion matrix – CNN with 0.2 dropout in first layer only....46
Figure 5.13: Model accuracy of a CNN with 0.5 dropout in first layer only....46
Figure 5.14: Training vs validation loss of the CNN with 0.5 dropout in first layer only....47
Figure 5.15: Validation set confusion matrix – CNN with 0.5 dropout in first layer only....47
Figure 5.16: Test set confusion matrix – CNN with 0.5 dropout in first layer only....48
Figure 5.17: Model accuracy of a CNN with 0.5 dropout in first and second layers....49

Figure 5.18: Training vs validation loss of the CNN with 0.5 dropout in first and second layers.....49
Figure 5.19: Validation set confusion matrix – CNN with 0.5 dropout in first and second layers.....50
Figure 5.20: Test set confusion matrix – CNN with 0.5 dropout in first and second layers....50
Figure 5.21: Model accuracy of a CNN with 0.5 dropout in all three layers.....51
Figure 5.22: Training vs validation loss of the CNN with 0.5 dropout in all three layers.....51
Figure 5.23: Validation set confusion matrix – CNN with 0.5 dropout in all three layers.....52
Figure 5.24: Test set confusion matrix – CNN with 0.5 dropout in all three layers.....52
Figure 5.25: CNN lightweight version confusion matrix.....53
Figure 5.26: Model accuracy of an unregularised LSTM.....54
Figure 5.27: Training vs validation loss of the unregularised LSTM.....54
Figure 5.28: Validation set confusion matrix – unregularised LSTM.....55
Figure 5.29: Test set confusion matrix – unregularised LSTM.....56
Figure 5.30: Model accuracy of a LSTM with 0.1 dropout in first layer only.....56
Figure 5.31: Training vs validation loss of the LSTM with 0.1 dropout in first layer only.....57
Figure 5.32: Validation set confusion matrix – LSTM with 0.1 dropout in first layer only....58
Figure 5.33: Test set confusion matrix – LSTM with 0.1 dropout in first layer only.....58
Figure 5.34: Model accuracy of an LSTM with 0.2 dropout in first layer only.....59
Figure 5.35: Training vs validation loss of the LSTM with 0.2 dropout in first layer only.....60
Figure 5.36: Validation set confusion matrix – LSTM with 0.2 dropout in first layer only....60
Figure 5.37: Test set confusion matrix – LSTM with 0.2 dropout in first layer only.....61
Figure 5.38: Model accuracy of an LSTM with 0.5 dropout in first layer only.....61
Figure 5.39: Training vs validation loss of the LSTM with 0.5 dropout in first layer only.....62
Figure 5.40: Validation set confusion matrix – LSTM with 0.5 dropout in first layer only....63
Figure 5.41: Test set confusion matrix – LSTM with 0.5 dropout in first layer only.....63
Figure 5.42: Model accuracy of an LSTM with 0.5 dropout in first and second and layers. . .64
Figure 5.43: Training vs validation loss of the LSTM with 0.5 dropout in first and second layers.....65
Figure 5.44: Validation set confusion matrix – LSTM with 0.5 dropout in first and second layers.....65
Figure 5.45: Test set confusion matrix – LSTM with 0.5 dropout in first and second layers. 66
Figure 5.46: Model accuracy of an LSTM with 0.5 dropout in all three layers.....66
Figure 5.47: Training vs validation loss of the LSTM with 0.5 dropout in all three layers.....67

Figure 5.48: Validation set confusion matrix – LSTM with 0.5 dropout in all three layers....68
Figure 5.49: Test set confusion matrix – LSTM with 0.5 dropout in all three layers.....68
Figure 5.50: LSTM lightweight version confusion matrix.....69
Figure A1: Downloading a city from OpenStreetMaps.....93
Figure A2: Exporting the desired area.....94
Figure A3: SUMO configuration file.....95
Figure A4: SUMO simulation.....96
Figure A5: Saving packet data as CSV.....99
Figure A6: Concatenating multiple CSV files into a single file.....99

List of Tables

Table 2.1: A simple analysis of the performance metrics applicable in the evaluation of IDS suitable for resource-constrained networks.....15
Table 3.1: Simulation setting.....26
Table 3.2: Table illustrating a confusion matrix.....29
Table 4.1: Pandas-profiling data description snapshot.....32
Table 4.2: Pandas-profiling warnings snapshot.....32
Table 4.3: Training dataset sample.....34
Table 4.4: Model training parameters.....35
Table A.1: NS3 folder structure.....97
Table A.2: Wireshark packet details.....98

Keywords

Machine Learning

Deep Learning

Cybersecurity

Intrusions Detection

Internet of Things

Wireless Senor Networks

Smart Cities

Neural Networks

List of Acronyms

DoS	Denial of Service
TCP	Transmission Control Protocol
ICMP	Internet Control Message Protocol
HTTP	Hypertext Transfer Protocol
UDP	User Datagram Protocol
WSN	Wireless Sensor Network
TCP SYN	Transmission Control Protocol Synchronize
ANN	Artificial Neural Network
IDS	Intrusion Detection System
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
LSTM	Long Short-term Memory
TPR	True Positive Rate
FPR	False Positive Rate
IoT	Internet of Things
SUMO	Simulation of Urban Mobility
NS3	Network Simulator 3
NS2	Network Simulator 2

Chapter 1: Introduction

1.1 Introduction

This chapter introduces the study, provides an overview of the study background, and states the problem at hand. The chapter further outlines the questions that the study intends to answer and outlines the aims as well as the objectives of the study. Finally, the contributions of the study are presented.

1.2 Background

The emergence of the Fourth Industrial Revolution presents sophisticated futuristic technologies. These advances in technology have seen a tremendous surge in the adoption of Internet of Things (IoT) -based smart city systems, such as smart grids, traffic management systems, smart health, and smart homes, which are intended to augment the lives of citizens through their system design that fosters resource efficiency and the real-time provisioning of resources in cities. The benefits offered by smart cities include the use of IoT sensors to gather useful data such as power demand to inhibit blackouts and the average speed of vehicles to alleviate traffic congestion (AlDairi & Tawalbeh, 2017).

Previous studies have reported substantial increases in cyber-security issues due to the increase in the deployment of these smart city ecosystems. Consequently, cyber-security in IoT is recognised as an area that requires crucial scrutiny. Security breaches in smart cities are accredited to how smart city systems are designed. Several studies have revealed that smart city systems are designed without security considerations in mind and, consequently, offer limited authentication and integrity capabilities which make these ecosystems susceptible to cyber-attacks (Tuptuk & Hailes, 2018).

This view is supported by Mohamad Noor and Hassan (2019) who critically investigated trends in IoT security and identified poor authentication mechanisms, such as the use of weak and identical passwords and lack of mechanisms that offer automated intrusion detection, as the main factors that expose cyber-physical systems to cyber threats. In a similar study, Pacheco and Hariri (2016) argue that cyber-physical systems support poor authentication and do not enforce the use of strong passwords, hence it is easy for adversaries to guess passwords or use brute force to gain unauthorised access to specific systems. In the same way, Zhou, Zhang, and Liu (2019) explored security trends in cyber-physical systems such as smart homes, smart grids, and smart cities. They established that, when developing IoT inspired products, vendors tended to position their focus on implementing functionalities that

are positioned towards user satisfaction while neglecting security considerations in the process. Furthermore, their study maintains that such vendors do not only design resource-constrained devices but also devices that use default passwords and often have unpatched bugs which impede the deployment of efficient security mechanisms.

Due to the interconnectivity and sensitivity of the data shared among devices in smart city networks, any compromise in these ecosystems may lead to fatalities and life-threatening situations. Consequently, the security of smart city ecosystems has become an enormously researched area (Banerjee, Lee, & Choo, 2018).

1.3 Problem identification

In an attempt to realise the sustainable deployment of smart cities, Elrawy, Awad and Hamed (2018) proposed network intrusion detection as a necessary mechanism to protect against intrusions that compromise the confidentiality, integrity and availability of smart city systems. Banerjee, Lee and Choo (2018), Chen, Hasan and Mohan (2018), Mohamad Noor and Hassan (2019) and Pereira, Barreto and Amaral (2017) concede the necessity for the development of intrusion detection schemes that are capable of detecting malicious attacks. An intrusion detection scheme is a mechanism implemented at the network layer of an IoT-based system to analyse the data packets of the system of interest and generate responses in real time (Elrawy et al., 2018). Although intrusion detection has been an exhaustively explored research area as highlighted by Roux, Alata, Auriol, Nicomette and Kaâniche (2017), the majority of previous studies focused on wired networks rather than WSNs.

Smart city systems are largely based on wireless connections linking numerous devices. This makes it imperative to continuously design and optimise intrusion detection systems (IDSs) which will improve the security of smart city systems. Hence, this study seeks to propose a deep learning-based intrusion detection mechanism bespoke for IoT-based smart city systems.

1.4 Research aim

This study aims to develop an IDS that can perform well in an IoT environment.

1.5 Research questions

This study seeks to address the following questions:

- What are the cyber-security challenges currently facing the deployment of smart city ecosystems?

- What are the current intrusion detection methods in smart city ecosystems?
- Can a model that would detect intrusion in a smart city ecosystem with better accuracy be designed?
- How efficient is the proposed model?

1.6 Research objectives

The objectives of this study are as follows:

- To analyse cyber-security challenges in smart city ecosystems
- To identify state-of-the-art intrusion detection methods used in smart city ecosystems
- To propose a deep learning model as an intrusion detection mechanism for smart city ecosystems
- To evaluate the efficiency of the proposed deep learning model.

1.7 Study contribution

The literature survey of this study contributes to existing knowledge by exposing gaps that exist in intrusion detection in smart city ecosystems. The literature survey reveals that earlier studies were not mindful of the resource constraints that exist in IoT-based networks. Accordingly, this study proposes a lightweight deep learning model that will take into consideration the low processing power, heterogeneity, and self-organising nature of smart city ecosystems.

1.8 Dissertation outline

Chapter 1 introduces the study. The chapter encompasses the problem statement, background, research questions and research objectives. This chapter also presents the expected outcomes and impact of the study.

Chapter 2 provides a comprehensive survey of the literature related to the study.

Chapter 3 is a presentation of the methodological approaches adopted in the study. The chapter further discusses the philosophy followed in carrying out the research and the tools utilised to carry out the experiments. Furthermore, this chapter provides an overview of the methodologies that were undertaken for data analyses.

Chapter 4 gives a descriptive overview of how the feature selection, as well as the data used to train and evaluate the candidate models, was processed. The chapter also discusses the implementation approaches of the candidate models.

Chapter 5 presents a performance evaluation and comparative analysis of two candidate models. The two models are evaluated against each other based on convergence speed, detection rates and false alarm rates under varied dropout rates.
Chapter 6 concludes the entire study and gives an overview of potential future studies.

1.9 Summary

This chapter introduced the study. The chapter started by giving a brief presentation of the background of the study and further outlined the research questions and objectives. The chapter also presented the significance of the study and concluded with the outline thereof.

Chapter 2: Literature Review

2.1 Introduction

This chapter presents the literature relevant to the study. The chapter briefly provides an introductory overview of smart cities and the dimensions thereof. The chapter further provides a review of security issues in smart city ecosystems and state-of-the-art detection mechanisms as well as a perusal of deep learning techniques for intrusion detection.

2.2 Smart city ecosystems

The prevailing technological evolution presents innovations predetermined to integrate exhilarating technologies encompassing 5G networks, IoT and Artificial Intelligence (AI). These advances in technology have triggered the proliferation of the development of smart cities. AlDairi and Tawalbeh (2017) define smart cities as cities that integrate infrastructure and technology to enhance the lives of citizens.

Smart cities exploit AI, cloud computing, embedded computing, biometric systems as well as IoT technologies for efficient city automation and infrastructure governance and monitoring. Smart cities comprise subsystems incorporating smart energy meters, smart health and smart homes that are designed to enhance the quality of life in cities (Elmaghraby & Losavio, 2014). One of the conveniences offered by smart cities is smart urban transportation – a system of interconnected vehicles that exchange crucial information such as GPS coordinates as well as traffic and weather updates.

Supporting AlDairi and Tawalbeh's (2017) narrative on smart cities, Popescul and Radu (2016) assert that smart cities incorporate sophisticated and embedded systems such as smartphones, medical devices, wearable sensors, supervisory control and data acquisition (SCADA) systems as well as other sensor technologies which are utilised to monitor air quality and pollution, energy and water consumption, waste management and automotive traffic. The next section briefly outlines the distinguishing attributes of smart cities.

2.2.1 Dimensions of a smart city

Smart cities are designed to promote e-governance, sustainability, liveability as well as efficiency in urban areas using IoT and AI (Joshi, Saxena, Goodbole, & Shreya, 2016). Jucevičius, Patašienė and Patašius (2014) state that there are certain underpinnings that ultramodern cities should uphold for them to be conceded as smart cities. These underpinnings are outlined below:

2.2.1.1 Smart transportation

Smart transportation is an indispensable component of a smart city and uses sensors and other intelligent technologies to collect data to promote information sharing between vehicles as a means to mitigate traffic congestion as well as to monitor road infrastructure, alleviate road traffic accidents and improve the quality of life in cities (Olaverri-Monreal, 2016).

2.2.1.2 Smart energy

Smart energy involves the use of sensors, smart meters and renewable energy sources for the automation and monitoring of energy consumption. Smart grids are recognised as the most notable innovation towards the concept of smart energy management (UNCTAD, 2016). According to Otuoze, Mustafa and Larik (2018), smart grids are ecosystems that consist of advanced metering systems, communication networks and data management systems that are interconnected to exchange information and to collect data that can be utilised for real-time monitoring of energy consumption to aid dynamic pricing.

2.2.1.3 Smart health care

Smart health care is a smart city system designed to leverage wearable sensors, long-range communication networks, cloud technologies and machine learning techniques for the remote monitoring of non-critical patients. The sensors gather essential information such as pulse, respiratory rate, body temperature and blood pressure. The collected data can be sent to the cloud and further used to notify emergency services whenever the readings from the sensors vary. Machine learning is then used to recommend treatment based on the data that are collected from the wearable sensors (Baker, Xiang, & Atkinson, 2017).

2.2.1.4 Smart education

Smart education is a concept of fabricated and programmable educational spaces that leverage state-of-the-art technologies to provide efficient communication, administration, leadership and student management and performance monitoring in learning institutions (Williamson, 2015).

2.2.1.5 Smart environment

In a study that proposes storage architectures for the efficient monitoring of the smart environment, Fazio, Celesti, Puliafito and Villari, (2015) define a smart environment as an

interconnected ecosystem of cloud storage technologies, heterogeneous devices, actuators and sensors that exchange data to promote real-time monitoring of environmental variables such as waste management, air quality, pollution, health care and weather.

2.2.1.6 Smart security

This is a smart city system that uses IoT and ultramodern technologies, designed for intruder detection and the tracking of people and objects (Saifuzzaman, Khan, Moon, & Nur, 2017).

2.2.1.7 Smart industry

Smart industry is an ecosystem where oil, gas, manufacturing, mining, and aviation industries exploit sensor integrated applications to design intelligent machines. Intelligent machines are used for the real-time exchange of data to promote timeous communication between suppliers and retailers and to track goods (Butt & Afzaal, 2019).

2.2.2 Digitisation in a smart city

Regarding the smart city characteristics highlighted, it is clear that connectivity and automation are fundamental to the development of smart cities (Bruneo et al., 2019). This digitisation is achievable using mobile applications, Wi-Fi, AI and cloud-based architectures to facilitate resource monitoring in cities. Thus, smart city components are connected through wireless networks and generate data that may be used intelligently for city infrastructure management and monitoring (UNCTAD, 2016).

Although smart cities offer an intelligent, spatial, and economic competitive advantage, they are mired with security issues. Previous studies such as that of (AlDairi & Tawalbeh, 2017) established that due to high interconnectivity and the data shared among smart city systems, these systems are prone to cyber-attacks.

Motivated by the views of earlier scholars on security issues in smart city ecosystems, this study surveyed cutting-edge literature related to security hurdles in the concerned ecosystems. The next section is devoted to a survey of the literature relating to security challenges in smart cities.

2.3 Security issues in smart city ecosystems

The pinnacle of the development of smart cities has instigated a significant rise in cyber-security issues in these ecosystems. Therefore, the security of smart cities is an area that needs utmost attention from the research community.

Popescul and Radu (2016) identified limitations related to hardware, computational processing power, low energy consumption and memory as the main factors that hinder the implementation of security mechanisms in smart city systems. In the same study, they recognise scalability, heterogeneity, the use of diverse communication and network protocols and dynamic network topologies as other aspects that impede the deployment of security techniques in these systems. According to their study, the aforementioned circumstances have led to designers gauging the performance of their products based on the ability of the products to consume low energy and processing power while overlooking security considerations and viewing design and implementation of security mechanisms as just add-ons.

After evaluating the state-of-the-art protection methods used for smart cities, Cui, Xie, Qu, Gao and Yang (2018) established that numerous protection mechanisms employed in conventional systems, such as encryption, biometrics and anonymity, are redundant in smart city systems due to the limited computational processing power of sensors and other devices that are used in the construction of smart city systems. Moreover, their study is in accord with previous studies which maintain that heterogeneity, scalability, and the dynamic nature of IoT systems make deployment of security measures in these systems largely a cumbersome task to accomplish, subjecting smart cities to high-security risks.

Additionally, their study postulates that attackers are becoming smarter and have started developing tools that can bypass currently deployed security mechanisms by using AI to weaken the trained algorithms and make them less reliable. Furthermore, their study has pointed out denial-of-service (DoS) attacks as the most notable attacks that threaten the integrity and availability of smart city ecosystems.

An earlier study by Ferraz, André and Ferraz, (2014) investigated the impact of cyber-security on smart cities. Their study argued that modern communication media such as smartphones, laptops, tablets and easy Wi-Fi access enable cyber-criminals to perform unethical hacking activities effortlessly. Additionally, their study affirms that software patches and software products that are released without undergoing enough tests induce security loopholes in smart city systems.

Furthermore, to understand the applications of IoT in the development of smart cities, Khajenasiri, Estebsari, Verhelst and Gielen (2017) investigated the architecture of IoT. Their study agrees with several researchers that believe that hurdles in the implementation of adequate security mechanisms are due to the heterogeneous nature of IoT architecture and networks. The authors criticised the lagging in the enforcement of security standards,

claiming that this lagging resulted in leakages of sensitive information as the information was not protected through security strategies such as encryption.

Otuoze et al. (2018) draw attention to the sources of threats in smart cities, focusing mainly on smart grid systems. Their study maintains that the introduction of transmission control protocol (TCP) and the presence of communication nodes in smart grids have induced enormous security issues in the systems. According to their study, smart grids are currently facing security threats that are likely to result in power blackouts. These power blackouts may further lead to cascade failures, damage to consumer devices, chaos in the energy market, theft of sensitive data and jeopardising human lives. Furthermore, their study argues that these security loopholes in smart grids expose the systems to organised crimes such as hacking, rioting, terrorism, energy theft, information leakage, false data injection, sabotage, and the disruption of services.

After investigating security vulnerabilities in in-vehicle networks, Rizvi, Willet, Perino, Marasco and Condo (2017) highlighted that modern vehicles have shifted from being just 'metal boxes' to sophisticated machines that possess sufficient intelligence to make decisions. Their study adds that, due to this shift, vehicles are as susceptible to cyber-attacks as computers. Their study confirms the views of Cui, Xie, Qu, Gao and Yang (2018) on the notion that security mechanisms used in traditional computer networks fail to protect vehicle networks due to the nature and architecture of their WSNs. Earlier studies established that technologies used in connecting smart cities are a major issue in the security of systems. One such technology used in smart cities is the radio frequency identification (RFID) tag which is a core technology for interconnecting cyber-physical systems. This technology provides real-time information sharing. However, Popescul and Radu (2016) argue that RFID is extremely susceptible to attacks making it a security risk in smart cities.

One of the biggest issues in RFID is the sabotage of a reader where an adversary gains control of the reader and starts emitting electromagnetic waves to destroy all the data in the RFID tag. These devices can also be tracked without a user's consent. Additionally, RFID tags are highly prone to DoS attacks (Popescul & Radu, 2016). The surging deployment of smart cities is significantly growing data and network traffic. These cities are rendered inoperable without scalable city governance systems and techniques.

Additionally, smart city infrastructure, such as smart transportation systems which are active institutions in cities, use devices such as radars, Bluetooth detectors and licence plate cameras to collect data on the speed, flow, and travel times of vehicles. This information can be easily associated with drivers' identities and may reveal sensitive information (safety-critical

events, speed, destination, home and workplace addresses and time spent in a particular location) about the drivers and put them in jeopardy (Popescul & Radu, 2016).

Furthermore, earlier scholars established that cyber-physical systems exploit open-source platforms such as Android, which is renowned for attracting cyber-criminals into launching particularly DoS attacks which are the most renowned in WSNs (Elleithy, 2006; Zekri, El Kafhali, Aboutabit, & Saadi, 2018). The next section provides a brief overview of the characterisation of four popular DoS attack scenarios.

2.4 The taxonomy of denial-of-service attacks

A DoS attack is an attack that disrupts the operations of a server by incapacitating the server from providing services to its clients. Adversaries launch this attack by flooding a network server with invalid packets or spoofed IP addresses to slow down the compromised network (Elleithy, 2006. It is believed that DoS attacks are cumbersome to avert because the packets that are sent when a DoS attack is launched are disguised as legitimate packets.

Conti (2018) established that DoS attacks have gained popularity because they have become easier to launch due to technological advances that have availed high-speed networks and enabled ease of access to bots that can be exploited to implement volumetric DoS attacks such as TCP, ICMP, HTTP and UDP flooding.

Alguliyev, Aliguliyev and Abdullayeva (2019) argue that DoS attacks are designed to disrupt and halt services of WSN by saturating the already limited resources of WSNs. They add that when DoS attacks are successfully launched, they retard the performance of network resources and paralyse the normal operations of the network by causing failure to the affected sensors rendering the sensors inaccessible for communication.

Azahari Mohd Yusof, Hani Mohd Ali and Yusof Darus (2018) categorise DoS into different classes as follows:

- UDP flood: An attack that uses the UDP protocol and attacks the victim by sending a large number of packets to slow down and crash the compromised device.
- TCP SYN flood: An attacker sends recurrent SYN packets using fake IP addresses. Consequently, the victim will be unable to close connections, will thus receive a large number of packets while unable to send an acknowledgement, causing the server to crash. The orchestration of a TCP SYN flooding attack is as demonstrated in **Figure 2.1**.

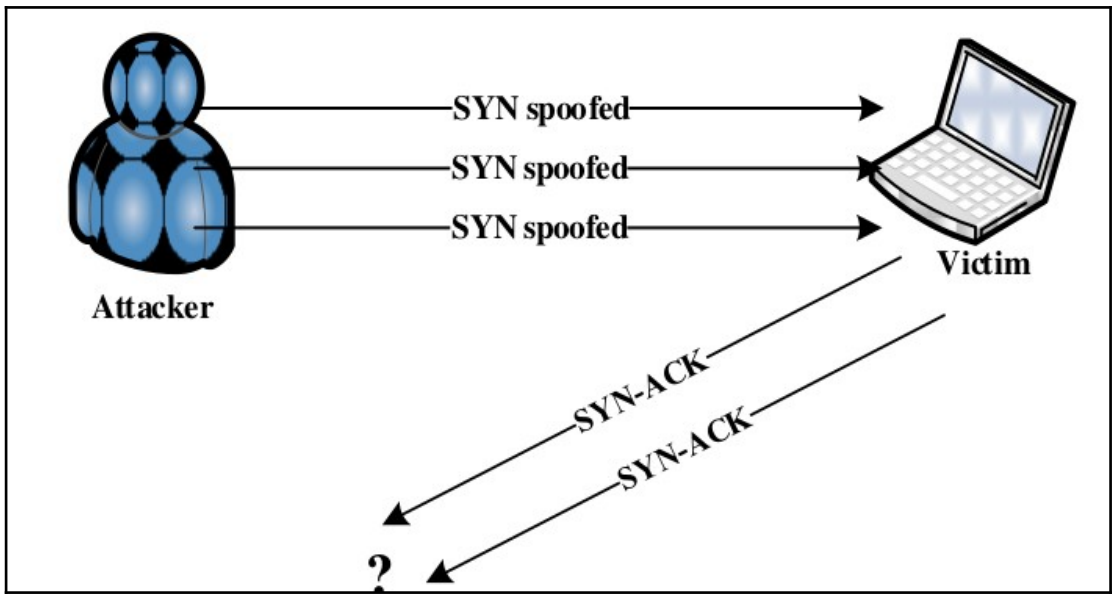

Figure 2.1: TCP SYN flooding attack

. Adapted from Azahari Mohd Yusof, Hani Mohd Ali and Yusof Darus (2018)

- Ping of death: In a ping of death, the adversary sends oversized and malformed packets to the victim. This leads to memory overflows that crash the compromised device. This will keep the compromised network resource so busy that it will not be able to service its legitimate clients.
- Smurf attack: This is an attack launched by sending extremely large ICMP packets to the victim thereby flooding the victim's device with spoofed ping messages to cripple the network resource.

Regarding the above characteristics of DoS attacks, it is indisputable that the repercussions of such attacks can be devastating. Thus, for any technology to thrive, the privacy and security of user data should be fundamental. According to Aris and Oktug (2017), IoT networks remain exposed to cyber-attacks despite numerous cryptography-based security mechanisms that have been continuously proposed and implemented. Accordingly, as an attempt to overcome security drawbacks in smart city technologies, Krimmling and Peter (2014) proposed intrusion detection as the most potent security mechanism that can be used to alleviate cyber-attacks on resource-constrained smart city networks.

Intrusion detection is a cyber-security approach that has the proficiency to monitor and analyse packet traffic, filter normal network behaviour from abnormal network behaviour and discern intruders and attacks in communication systems. Intrusion detection gained popularity due to the astounding defence capacity that the technique demonstrated in

traditional computer networks (Zarpelão, Miani, Kawakani, & de Alvarenga, 2017). Section 2.5 of this study is devoted to reviewing currently deployed intrusion detection methods.

2.5 Intrusion detection

The influx of smart city systems, such as intelligent transportation systems (ITS), compels the development of methods that can monitor and detect security threats in ecosystems (Aloqaily, Otoum, Al Ridhawi, & Jararweh, 2019). As mentioned in the preceding section, previous studies recommend intrusion detection as the best approach to mitigate cyber-security attacks in smart city systems. Garcia-Font, Garrigues and Rifà-Pous (2016) state that computations and energy constraints in WSNs make it impractical to deploy security mechanisms that are used in traditional network architectures. Hence, there is a need to design intrusion detection mechanisms that are tailored for smart city ecosystems. Inspired by the above notion, Sherasiya, Upadhyay and Patel (2016) reviewed a state-of-the-art IDS and discovered that currently deployed IDSs are inadequate for sensor networks due to massive computational overhead setbacks that IDSs exhibit.

Similarly, Zarpelão, Miani, Kawakani and de Alvarenga (2017) examined IDSs for IoT systems to understand how earlier scholars addressed the hurdles that impede the implementation of IDSs in IoT systems. Their study investigated 18 IDSs proposed between 2009 and 2016 and classified the IDSs under review based on detection method, placement strategy and the validation strategy used. They suggested that IoT IDS is still in its infancy and a lot more still needs to be done. Additionally, they established that detection methods used in the majority of current IDSs do not clearly demonstrate adequacy and validation strategies are not proven to be effective. They further suggested that there is an earnest need for researchers to focus on investigating flaws in currently deployed IDSs to improve the development of IDSs for IoT environments.

In the same vein, Santos, Rabadao and Gonçalves (2018) studied previously proposed IDSs to make recommendations for future research on the development of IDSs for IoT environments. Their methodology involved an exhaustive evaluation of 20 research papers published between 2009 and 2017. Their study suggested that current IDSs have low detection rates. Furthermore, they claim that the previously proposed IDSs are not adequate for IoT environments due to poorly devised implementation strategies. Furthermore, Jabbar and Aluvalu (2018) examined state-of-the-art IDSs. They established that, regardless of the breakthroughs made in the development of IDSs, current systems exhibit enormous

limitations that introduce complexities in IoT networks leading to poor detection rates in the IDSs making the systems less adequate for deployment in IoT environments.

In their work, Hodo et al. (2016) developed an IDS for sensor networks. For the development of the proposed model, their study simulated a network with five sensor nodes and collected data after launching a DoS attack on the simulated network. A neural network classifier was then trained and evaluated using the data. The trained network demonstrated a high level of accuracy in detecting DoS attacks with a true positive rate of 99.4% and a false positive rate of 0.6%. This is an impressive result in terms of detection and false positive rates, however, there is no evidence provided in terms of reduced computational overheads leaving its suitability for constrained networks uncertain.

Thanigaivelan, Nigussie, Virtanen and Isoaho (2018) proposed a hybrid IDS. Their proposed framework integrates the routing protocol for low-power and lossy networks (RPL) with the distress propagation object to monitor and analyse the packets received at the device level, enabling nodes to monitor and analyse the packets received and react against malicious traffic. The experimental results of their study indicated that the proposed IDS can efficiently detect anomalies with minimal false alarm rates and lower overheads making it efficient for IoT systems.

Doshi, Apthorpe and Feamster (2018) proposed a machine learning (ML) classification model for IoT distributed DoS (DDoS) attack detection testing five different ML classifiers, namely linear support vector machines (LSVM), K-nearest neighbour (KNN), random forests (RF), neural networks (NN) and decision trees (DT). They acquired the training and test datasets by simulating an IoT network to obtain normal and abnormal traffic data. Zhang and Xiao (2019) proposed a negative selection algorithm-based intrusion detection model to detect abnormal behaviour in WSNs. Their model is trained using the Iris dataset – whose relevance is not clear – classifying versicolor and virginica samples as abnormal traffic. The authors argue that their experimental results have clearly demonstrated that the proposed model can efficiently detect anomalies while saving sensor resources.

Aldaej (2019) proposed an IDS aimed at enhancing cyber-security in modern IoT. The study applies the flexible mobile adhoc networks intrusion detection system (FMIDS) to detect DDoS attacks in IoT WSNs. His algorithm is implemented using simulated data. However, the results of his study do not clearly indicate whether the proposed technique can reduce false positives and computational overhead. Thamilarasu and Chawla (2019) proposed an IDS that exploits deep learning (DL) algorithms incorporating them with network virtualisation techniques to detect abnormal network behaviour in smart city systems. Their

methodology involved simulating a smart home network where the connected devices communicated through CoAp, ZigBee, Bluetooth BLE and Wi-Fi protocols. They assessed the performance of their proposed model based on its ability to detect five different attacks including blackhole, opportunistic service, DDoS, sinkhole and wormhole attacks. The authors claim that their experimental results demonstrated high detection rates with minimal false positive rates. A deep belief network-based IDS tailored for connected vehicles in smart cities is proposed in the study of Aloqaily, Otoum, Ridhawi and Jararweh (2019). Their study used the KDD CUP 99 dataset to represent malicious network traffic and benign network traffic was simulated using NS3 where 40 vehicles in a smart city setup were used to collect training data for the model; both the KDD CUP 99 dataset and the data from the simulated network were used. The model was then evaluated on the KDD CUP 99 dataset where a detection rate of 98.5% and a 1.5 % false positive rate were obtained.

Hasan, Islam, Zarif and Hashem (2019) proposed an intrusion detection framework for IoT-based networks. Their framework was trained using an open-source synthetic dataset that contained the network traffic of an IoT smart environment. The data contained normal and malicious data with eight attack vectors. The authors applied several ML algorithms, namely, support vector machines, random forest, logistic regression, DT and NN and compared the robustness of the algorithms. All the trained algorithms demonstrated good detection rates with RF being the most robust with an accuracy of 99.4%.

Anthi, Williams, Slowińska, Theodorakopoulos and Burnap (2019) developed a three-layer IDS by applying a supervised ML approach to train a model that is not only able to classify if a packet is normal or malicious but is also able to identify the type of attack. The model was trained with data from an IoT testbed. The proposed model achieved an accuracy of 97% on the training set but dropped significantly to 90% on the test dataset; this demonstrates that the model is overfitted and unable to generalise.

Venkatraman and Surendiran (2019) proposed an automata controller-based intrusion detection model. Their approach is designed to detect known and unknown attacks. The data for training the proposed model was obtained by simulating an IoT-based smart home network resulting in 18000 network transactions of which 15000 were normal network traffic while 3000 were malicious network traffic. Their novel approach obtained an overall accuracy of 99%. This is a good detection rate even though there is no clear indication of whether the approach introduces computational overheads or not. The survey of cutting-edge intrusion detection frameworks undertaken by this study has identified that it is indisputably evident that several of the studies reviewed were not mindful of the dynamic characterisation

and resource limitations of IoT infrastructures such as smart city ecosystems. Some notable shortcomings of the proposed frameworks include the use of inappropriate datasets that contain antiquated attack vectors for training the proposed detection models and insubstantial evidence of the elimination of computational overheads.

A probable precursor of the identified limitations is the unavailability of appropriate datasets and the lack of frameworks that are tailored for the development of lightweight ML models. These findings agree with the findings of an earlier study by Arshad et al. (2018) who scrutinised state-of-the-art intrusion adetection frameworks for IoT architectures taking into account the limiting attributes of IoT networks. They pointed out that an overwhelming majority of currently deployed IDS models were trained using KDD 99 data, which is a dataset comprising network traffic for traditional computer networks with no reflection of current threats encountered in IoT environments. Moreover, they concluded that the current IDSs do not consider the limitations of IoT environments that encompass computational power constraints as well as the heterogeneity of devices and communication protocols.

Table 2.1 depicts a summary of the results of the conducted survey; a very brief outline of the limitations identified from each study is provided. A (-) symbol indicates that the result of a metric was not confirmed in the study.

Table 2.1: A simple analysis of the performance metrics applicable in the evaluation of IDS suitable for resource-constrained networks

Reference	**Dataset**	**TPR %**	**FPR %**	**Computational Overheads**	**Limitations**
Hodo et al. (2016)	Simulated	99.4	0.6	-	Evident overfitting
Thanigaivelan, Nigussie, Virtanen, & Isoaho (2018)	Simulated	-	-	864 B	-
Doshi, Apthorpe, & Feamster (2018)	Experimental IoT network traffic	99.0	-	-	-
Zhang & Xiao (2019)	Iris dataset	-	-	-	Irrelevant dataset; no evidence of suitability for

Reference	Dataset	TPR %	FPR %	Computational Overheads	Limitations
					constrained and heterogeneous networks
Aldaej (2019)	Simulated	-	-	-	No evidence of the performance and suitability of the model
Thamilarasu & Chawla (2019)	Simulated smart home	97.0	-	-	-
Venkatraman & Surendiran (2019)	Simulated smart home	99.0	-	-	-
Aloqaily, Otoum, Ridhawi, & Jararweh, (2019)	KDD CUP 99 dataset	99.0	1.5	-	Use of a dataset with outdated attack vectors
Hasan, Islam, Zarif, & Hashem (2019)	Simulated smart environment	99.4	-	-	-
Anthi, Williams, Slowińska, Theodorakopoulos, & Burnap (2019)	Experimental WSN	90.0	-	-	Overfitting

2.6 Deep learning

DL is a field of ML inspired by biological neural systems that emerged to advance traditional ML algorithms. DL offers the capability to eliminate the human effort that was previously needed by contemporary ML for tasks that include feature engineering and model training (Erickson et al., 2018). According to Wu, El-Maghraby and Pathak (2015), DL claimed its prominence from its outstanding performance in applications encompassing object detection, speech recognition, text processing, language translation and self-driving cars. The ML algorithms at the core of DL are artificial neural networks (ANNs). A general overview of ANNs is provided in the next section.

2.6.1 Artificial neural networks

ANNs are computational models that simulate the functionalities of the human brain. According to Wu et al. (2015), ANNs exhibit the capability to adeptly recognise complex

patterns in high-dimensional data points. A remarkable trade-off between NN and traditional ML techniques is the ability of ANNs to self-learn and adjust to changes in patterns of the data that are being modelled (Wu & Rahman, 2017). In a paper that investigates the applications of ANNs in HIV/AIDS studies, Sibanda and Pretorius (2012) classified ANNs into two basic architectures. They suggest that the architecture of any ANN is determined by the structure and the learning processes of the network. The two architectures are feed-forward NN and recurrent NN.

2.6.1.1 Feed-forward and recurrent artificial neural networks

A feed-forward NN is an ANN with an acyclic topology where information flows only in one direction, that is, from input to output without any feedback loops (Krenker, Bešter, & Kos, 2011). Feed-forward NNs are widely used in pattern recognition. The feed-forward architecture is diagrammatically demonstrated in **Figure 2.2**.

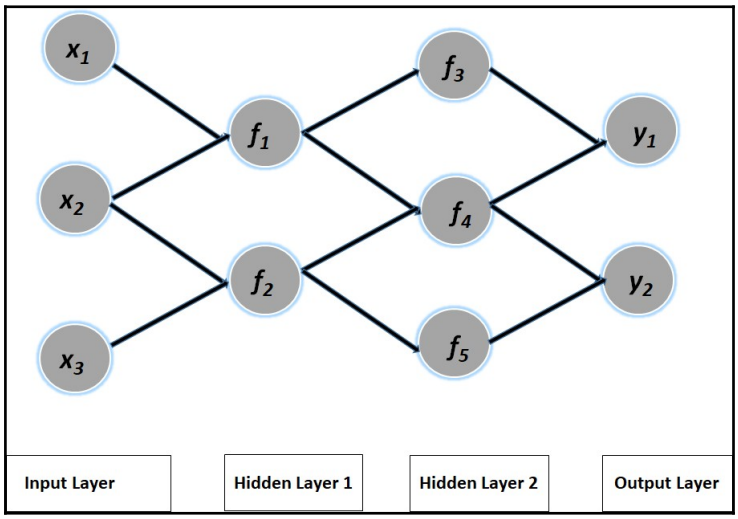

Figure 2.2: Feed-forward neural network structure

Recurrent neural networks (RNNs), on the other hand, are cyclic ANNs with a feedback loop attribute that enables every node in a layer to take input both from the previous layer and adjacent nodes (Azpiazu & Pera, 2019). This architecture allows RNNs to perform well in tasks that involve sequential data modelling. Furthermore, according to Rosindell and Wong (2018), this architecture also allows RNN models to store information about the past, thus leveraging them the capability to observe correlations between events that are far away from each other in the data.

Azpiazu and Pera, (2019) added that despite the robustness of RNNs, they suffer from the problem of a vanishing and exploding gradient which makes them cumbersome to train. As a means to address the problem of vanishing gradient, Hochreiter and Schmidhuber (1997) introduced LSTM. An LSTM NN is an RNN derivative that uses linear memory cells enclosed by multiplicative gate units to read, write, revitalise, and store information (Liang, 2017). An LSTM prevents the vanishing gradient problem by integrating nonlinear and data-dependent units into the RNN cell (Sherstinsky, 2018) .

2.6.1.2 The topology of artificial neural networks

ANNs are built from basic units called neurons whose basic functionalities are derived from biological neurons. The neurons consist of inputs, weights, and bias. A basic NN (bundle of neurons) comprises an input layer (that accepts the training features), at least one hidden layer (at which all computations and transmutations are performed) and the output layer (that returns the learned information).

Faghri, Martinelli and Demetsky (1997) decompose the architecture of NN into four facets, namely inputs, outputs, weights and activation functions. Given an input, the neuron computes the weighted sum of the given inputs and adds bias; thereafter, the activation function decides if it should send a new signal or not. The said scenario can be mathematically mapped as:

$$y = f(x,w) \qquad (2.1)$$

where

y = outputs,

x = inputs,

w = weights, and

f = activation function.

The activation function (f) can be sigmoid, hyperbolic tangent (tanh), SoftMax or rectified linear unit (ReLU).

2.6.1.2.1 Activation functions

An activation function is the aspect of an NN that is responsible for the transformation of inputs into outputs by weighting how robust an output should be from the neuron based on the sum of inputs. There are a variety of activation functions serving different purposes (Bach, 2018).

A. Sigmoid

The sigmoid activation function is a nonlinear 'squashing' function that maps inputs (-∞, ∞) to (0,1). The sigmoid is used immensely in feed-forward NN. The function is renowned for its remarkable performance in binary classification tasks (Nwankpa, Ijomah, Gachagan, & Marshall, 2018). The function can be mathematically defined as in (2.2).

$$f(x)=\frac{1}{e^{-x}} \tag{2.2}$$

In an NN that uses the sigmoid function, if the weights and the inputs are small as x increases, then f(x) converges to 1; if x becomes large and negative, then f(x) converges to 0 (Renals, 2015).

B. Rectified linear units

ReLU is the most popular activation function with robust mathematical and biological foundations. ReLu outputs 0 when $x < 0$, and, conversely, outputs a linear function when $x \geq 0$ (Agarap, 2018).

2.7 Related work

Hu and He (2001) presented a back-propagation NN based on self-organising map (SOM) and multi-layer perceptron (MLP) algorithms. Their solution entails accumulating data from different sensors and analysing the data by using detectors that use a SOM to identify intrusive behaviour in the data. Their approach uses a three-layer perceptron consisting of four input neurons, two hidden layers and two outputs. The authors declare that the proposed solution obtained a good detection rate of 96% and a false positive rate below 3%.

Correspondingly, Sammany, Sharawi, El-Beltagy and Saroit (2007) introduced a three-class intrusion detection framework based on back-propagation NN. Their model is trained using the Defence Advanced Research Projects Agency dataset containing 450 000 connection records. Their two-layer MLP is trained with an 80/20 split and assessed on 2600 unseen examples; an average detection rate of 93.43% was obtained.

Naoum, Abid and Al-Sultani, (2012) applied a back-propagation NN for intrusion detection using the KDD CUP 99 dataset at a varying number of hidden layers using the tanh activation function. They assert that their proposed network was able to distinguish between normal and

abnormal network behaviour with an accuracy of 94% and a false positive rate of approximately 16%.

A two-layer MLP intrusion detection framework was suggested by Moradi and Zulkernine (2004). A dataset consisting of 450 000 examples from the Defence Advanced Research Projects Agency was utilised for the training and validation of the proposed model. Their approach detects two attack vectors, namely, SYN Flood and Satan. The trained NN attained an overall accuracy of 99%; however, the model is strongly indicative of overfitting as the classification accuracy significantly declined when the trained model was applied on previously unseen data.

Yin, Zhu, Fei and He (2017) proposed an RNN-based IDS. Their approach involved investigating the impact of learning rates, the number of neurons and varying classification methods on the performance of the model. The model was trained using the NSL-KDD dataset with 41 input features. Their experiments entailed evaluating the robustness of the proposed model on binary classification and multi-class classification tasks. Furthermore, the performance of the RNN model was compared to the performance of models trained with other algorithms encompassing SVM, RF, MLP and naive Bayesian (NB). Their experimental results are satisfactory even though there is an evident indication of overfitting as their model performs impressively well on the training set, but the performance drops significantly on the test set.

Shenfield, Day and Ayesh, (2018) applied ANN to develop an anomaly-based IDS that detects malicious shell-code patterns in network traffic. Their solution was evaluated based on a 10-fold cross-validation technique to test the ability of the trained network to generalise. Their model obtained an average accuracy of 98% and a false positive rate of 2% which is a magnificent outcome especially in terms of false positive rates.

Furthermore, Saljoughi, Mehrvarz and Mirvaziri (2017) proposed NN for intrusion detection in cloud computing combining it with a particle swarm optimisation (PSO) algorithm for performance optimisation. Their approach uses a Kolmogorov-Smirnov correlation-based filter for feature selection. The proposed model was trained and evaluated using the KDD CUP 99 dataset. Their study deduced that the NN demonstrated a more remarkable performance when combined with the PSO algorithm than when it was implemented as a simple NN.

Althubiti, Nick, Mason, Yuan and Esterline (2018) investigated the applicability of LSTM RNNs in modelling network intrusion detection. Their framework was trained using the KDD CUP 99 dataset based on four attack scenarios. Their approach involved an exhaustive feature

engineering and selection where a J4.8 DT algorithm was used for selecting the most relevant features. The model was trained with a fixed number of epochs and varying learning rates to find the optimal parameters for the learning rate, network type and LSTM features. Their study concluded that LSTM offers excellent detection rates, especially when applied to the detection of high-frequency attacks such as DoS and network probe attacks. Mirsky, Doitshman, Elovici and Shabtai (2018) used autoencoders to develop an online and unsupervised network intrusion detection. The autoencoder NN was trained using real data obtained from an IP surveillance video camera. They concluded that autoencoders are efficient for the development of IDS.

Shone, Ngoc, Phai and Shi (2018) proposed a network intrusion detection system (NIDS) that stacks DL and shallow learning techniques based on the KDD 99 and NSL-KDD datasets. They implemented their proposed solution using a GPU-enabled TensorFlow using a dataset containing 41 features with five attack vectors. They trained their model using autoencoder NN and compared its performance to the performance of a deep belief NN. The proposed autoencoder model obtained an accuracy of 97.85%. Khan, Xiaosong, Alazab and Kumar (2018) proposed a CNN based IDS using the KDD CUP 99 dataset. The network comprised three hidden layers with each layer containing a convolutional and pooling layer. They compared the performance of their proposed technique to the performance of an SVM classifier and the CNN model outperformed the SVM with an accuracy of 98.50%.

Papamartzivanos, Gómez Mármol and Kambourakis (2019) proposed an autonomous and self-adaptive IDS based on a self-taught learning (STL) NN combining it with a MAPE-K control loop model. Their novel approach involved training a model using KDD CUP 99 and NSL-KDD datasets. They split the datasets into smaller subsets with randomly selected attack scenarios and evaluated the adaptability of the trained network on various network environments by changing the test datasets. According to the study, the STL NN attained an accuracy of 73.37%.

2.8 Summary

This chapter introduced the concept of smart cities outlining the underlying attributes that classify a city as ‘smart’. The chapter further gave a review of security issues that threaten the sustainability and safety of smart city ecosystems. Additionally, the taxonomy of DoS attacks, which generally have been denoted as the most prominent attacks in IoT systems, was outlined. Furthermore, the chapter presented a survey of the state of IDS and concluded

with a review of DL and the applications thereof in the development of intrusion detection frameworks.

Chapter 3: Research Design and Methodology

3.1 Introduction

This chapter provides an overview of the methods adopted by this study to achieve the results and answer the research questions. Section 3.2 highlights the methodologies adopted by the study and in Section 3.3 an overview of how the data was collected is provided. Data transformation and processing procedures are presented in Section 3.4. The feature engineering process is outlined in Section 3.5. Lastly, Section 3.6 describes the training and performance evaluation approach.

3.2 Research methodology

This study followed three (3) research approaches, namely, the qualitative, quantitative and simulation and modelling research methodologies.

3.2.1 Qualitative approach

A qualitative methodology is a research approach concerned with the qualitative elements of data. According to Daniel (2016) , the robustness of a qualitative research method lies in its flexibility which offers the ability to reconstruct the design of a study. The study used document analysis as a primary data source for the qualitative approach to conduct a systematic review of state-of-the-art intrusion detection frameworks. Document analysis is a systematic review method that permits analysis of documentary evidence.

In the systematic review, ten studies to interrogate previously proposed intrusion detection mechanisms were scrutinised. The investigation focused on studies published between 2014 and 2019. To meet the requirements of the inclusion criteria, the studies had to be grounded in intrusion detection in IoT architectures incorporating smart cities, cyber-physical systems and WSNs regardless of whether the proposed models were developed based on ML techniques or not. The selected studies were from Science Direct, Google Scholar, Academia.edu, Semantic Scholar and IEEE Explore databases. To retrieve the articles the phrase ‘intrusion detection in smart cities’ was used to search for relevant studies. Papers based on intrusion detection were surveyed and analysed.

3.2.2 Quantitative approach

The quantitative research method for feature engineering, feature selection and results analysis and interpretation were also utilised. Feature engineering and selection is clearly

explained in Section 3.5. Acaps (2012) describes quantitative methodology as a comprehensive and evidence-based research approach that can uncover correlational and causal relations between phenomena. The quantitative methodology involves the use of statistical and descriptive analysis methods to garner hidden insights into numerical data. In his study, Apuke (2017) stipulated that the quantitative approach is based on quantitatively analysing data to deduce results that give answers to research questions. The main trade-off of using the quantitative approach is its questionable ability to produce generalisable, reliable and replicable results (Acaps, 2012).

3.2.3 Simulation and modelling methodology

In addition to the qualitative and quantitative research approaches, the study followed the simulation and modelling approach. This is an approach that involves the imitation of a real-world system of interest to conduct experiments. For data collection for the experiments, the simulation and modelling approach was used to simulate a smart urban mobility ecosystem. The most fundamental benefit of using the simulation and modelling approach is that it offers the capability to model a complex phenomenon in a simplified form safely, cost-effectively and efficiently (Jaleesha & Ezhil, 2019).

For this study, the simulation and modelling methodology was employed to simulate a communication scenario between vehicles in a smart city setup to collect data. Thus, the simulation was the primary data source. The gathered data which comprised normal and malicious network packet traces data were used for model training and testing.

3.3 Research design

The tools and methods exploited to achieve the aim of this study are presented in this section.

3.3.1 Data collection

This section presents a technical description of the steps that were undertaken to acquire the data used for model training, validation, and testing. Since data sets that are suitable for training intrusion detection models appropriate for smart city networks are scarce, for data gathering for this study, a smart urban mobility infrastructure was simulated. The simulation was achieved using SUMO and NS3. The simulation stages are presented in **Appendix A**.

3.3.1.1 Smart urban mobility simulation

Due to the scarcity of datasets that demonstrate cyber-attack vectors in smart city ecosystems, a smart urban mobility system was simulated using Simulation of Urban Mobility (SUMO), an open-source road traffic simulator that offers the capability to simulate road networks and traffic demand. SUMO is used jointly with OpenStreetMaps. A digital map of an area of interest (city) from OpenStreetMaps is integrated into the SUMO simulation and then converted to a SUMO road network. SUMO was used to simulate a network of communicating vehicles popularly known as a vehicular ad hoc network (VANET). The VANET was then converted to a communication scenario using Network Simulator 3 and packet traces were captured from the simulation. Thereafter, the captured packets were decoded to human-readable data using Wireshark as the network analyser. In a SUMO simulation, each vehicle in the network is uniquely identified with a defined identifier. The vehicles' departure times and routes are defined; the routes are the connected sets of edges between a vehicle's departure point and destination. Appreciating the fact that the routes of vehicles cannot be controlled in the real world, the vehicle routes were assigned randomly to bring the simulation closer to reality.

The vehicles and city infrastructure were connected through wireless access in vehicular environment (WAVE), an IEEE 802.11p wireless communication standard that provides interoperable and wireless communication between vehicles and city infrastructure. WAVE is the most suitable for devices and network architectures with rapidly changing properties and topologies, respectively. By default, routing within the simulated network follows the shortest path first algorithm which means that, when searching for a communication line, a node (vehicle) in the network will always search for its nearest neighbours and send packets to the closest node.

3.3.1.2 Mobility network conversion

For seamless packet tracing and capturing, the SUMO-generated urban mobility network was integrated with Network Simulator 3 (NS3). NS3 is an open-source event-oriented network simulator. NS3 offers the ability to import a SUMO network and define the vehicles as simple nodes that communicate with each other. With NS3, various physical and application layer protocols and routing protocols can be defined. This study implemented TCP and UDP as transport layer protocols together with ad hoc on-demand distance-vector (AODV), destination-sequenced distance-vector routing (DSDV), dynamic source routing (DSR), and optimised link state routing (OLSR) as ad hoc routing protocols.

Integrating SUMO with NS3 required that the mobility network be converted into a network of communicating nodes. This was achieved by first converting the SUMO trace file to a Network Simulator 2 (NS2) mobility trace file. The NS2 mobility trace was then imported into the NS3 simulation and a communication scenario was configured. The process is depicted in **Figure 3.1**.

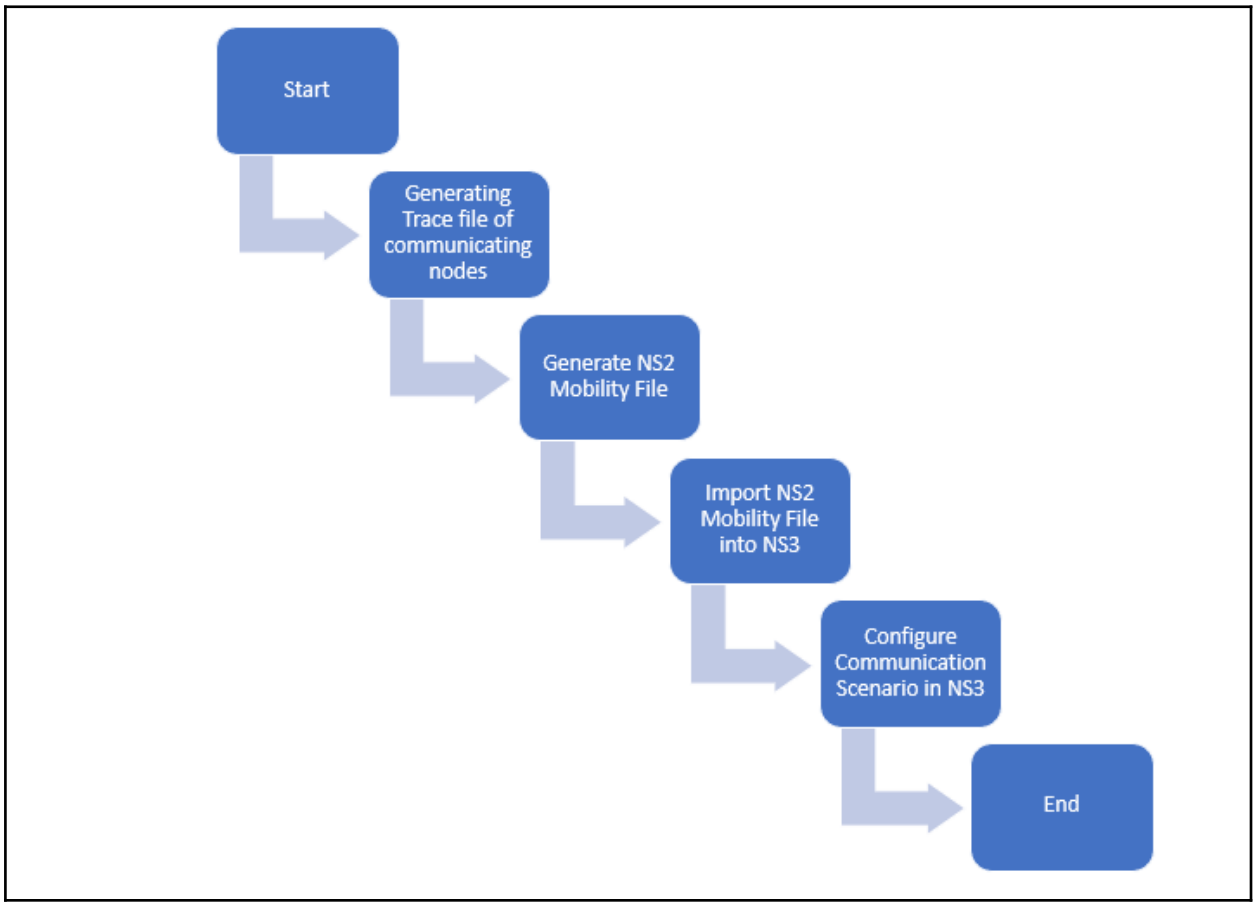

Figure 3.1: Simulation of Urban Mobility and Network Simulator 3 integration

Upon successfully configuring the network, the communication scenario within the network could then be defined. The configuration involved defining the routing protocols, communication standards, packet sizes and assignment of IP addresses. Two DoS attack scenarios (UDP and TCP flooding) were then launched on the simulated network and packet traces for both normal and malicious network traffic were captured. A summary of the overall simulation parameters is presented in **Table 3.1**.

Table 3.1: Simulation setting

Parameters	Values
Simulator	SUMO, NS3
Channel	Wireless
Wireless technologies	WAVE, Wi-Fi
Standards	IEEE 802.11p, IEEE 802.11b
Simulation time	SUMO: 2000 seconds NS3: 300 seconds
Number of vehicles	500
Speed of vehicles	40m/s
Transmission protocol	TCP and UDP
Traffic application	HTTP
Scenario	Urban
Topology	Dynamic
Communication range	50–500 meters
Routing protocols	AODV, DSDV, DSR, OLSR, TCP and UDP

3.3.1.3 Packet tracing and decoding

The packets captured within the NS3 simulated network were then exported as capture files and decoded with Wireshark for further data collection. Wireshark is a real-time network analyser software that decodes network packets in a human-readable format. The decoded data was then collected as comma separated value (CSV) files for further processing as explained in Section 3.4.

3.4 Data pre-processing

The normal and malicious traffic traces were extracted and saved as separate CSV files. The quality of data has a significant impact on the predictive power of an ML model. The garbage

in, garbage out (GIGO) concept is especially relevant in the training of ML models. Accordingly, to attain the best predictive power from the candidate ML models, a thorough data cleansing process was orchestrated to reduce the noise in the data. To achieve this, Pandas – an open-source data manipulation library – was used for the data cleaning and transformation process. The process further involved labelling the packet traces as either normal or malicious accordingly to create the target feature (packet status).

3.5 Feature engineering and selection

According to Chandrashekar and Sahin (2014), feature engineering and selection is a technique applied in the development of ML models to select a subset of variables. The technique, which selects only variables that effectively describe the input data in the interest of training ML models that offer high predictive power, is necessary to reduce dimensionality in the data.

In this study, feature engineering was performed through pandas-profiling. Pandas-profiling is a Python module popularly known for its robustness in providing quick exploratory data analysis insights of a given dataset. The process helps with insight that assists in deciding on features that should be removed from the data either due to high correlation with other features or a high number of missing values that may make the feature unusable. The insights gathered from the profile report also help with handling missing data, duplicate rows, and outliers. The feature engineering process resulted to a selection of six (6) features, namely:

- arrival time – the time at which a packet arrives to the host
- protocol – routing protocol
- length – packet size
- cumulative bytes – total number of transmitted bytes
- time delta from the previously captured frame – time difference between current two packets
- mac overhead – a ratio of the packet length and the throughput
- packet status – defines whether the transmitted packet is normal or malicious.

3.6 Model training and performance evaluation

Our model training followed the classification approach. Classification is a supervised learning technique that predicts a class for a given input. Five hundred thousand data points containing normal and malicious network traffic were selected. The data was then split into four datasets: train data, evaluation data (for performance evaluation during the training

process) and two validation sets which were used to perform 2-fold cross-validation. All datasets consisted of both normal and malicious traffic.

To propose and build a robust ML model, two different NN – CNN and LSTM – were trained using TensorFlow. TensorFlow is an end-to-end ML framework developed by Google. The framework offers robust, scalable,,, and flexible functionalities to easily build powerful ML models. The trained model that demonstrated the highest level of robustness was then converted to a lightweight version with TensorFlow-lite.

In evaluating the performance of the models, the learning rate was set to 0.0001 and kept constant throughout the experiment. Moreover, the activation function was set to ReLu at the input layers and sigmoid at the output layer. The models were consistently trained, validated, and tested using identical parameters and datasets to promote an unbiased performance evaluation of the models at all the training iterations. During the training and validation process, the loss and accuracy of the models were monitored. The loss was monitored to track how well the model fitted the data without overfitting, while the accuracy monitored the model's generalisation ability as well as its predictive power.

The testing approach was based on 2-fold cross-validation, where the models' performance was evaluated on two different datasets, which throughout this study are referred to as the validation dataset and the test dataset. The models were first evaluated on a validation set that had 100 000 data points (99 790 normal and 210 malicious network packets). The models were further tested on another 100 000 data points (99 775 normal and 225 malicious network packets). Both the validation and test datasets were completely unseen data.

In the interest of examining the robustness of the models on imbalanced data, which is what would be found in real-world scenarios, all datasets were left imbalanced, that is, no class balancing technique was applied. As far as loss is concerned, an ideal situation is when training and validation loss decrease as the model fits and learns the data. The performance of the lightweight model was further evaluated to investigate whether the lightweight version preserved the predictive power of the original (regular DL) model. Since this model is a binary classifier model, the results were presented using a confusion matrix. A confusion matrix is a table that visualises the performance of a classification model. A confusion matrix layout is shown in **Table 3.2**.

Table 3.2: Table illustrating a confusion matrix

Actual Class	Predicted Class		
	Negative		Positive
	Negative	True negative	False positive
	Positive	False negative	True positive

Where:

- True negative is a class that is negative and is predicted as negative
- False negative is a positive class that is predicted as negative
- True positive is a class that is predicted as positive and is actually positive
- False positive is a negative class that is predicted as positive.

The performance is also presented in terms of accuracy, which is the proportion of the total number of correct predictions as per (3.1):

$$Accuracy = \frac{\text{true positive} + \text{true negative}}{\text{true positive} + \text{true negative} + \text{false positive} + \text{false negative}} \quad (3.1)$$

3.7 Summary

This chapter presented an overview of methodologies and tools that were used to aid this study to answer the research questions. The chapter opened with a run-down of the methodology and design employed in the study and further outlined the tools used to achieve the objectives of the study. The chapter concluded by discussing how the datasets were analysed and how the results were validated. The next chapter will discuss in detail how the tools were utilised to gather the data and train the models.

Chapter 4: Implementation

4.1 Introduction

This chapter presents the implementation stages undertaken to train the two candidate intrusion detection models during this study. Section 4.2 describes the steps taken to simulate a communication scenario to collect data for training the models. The data processing and feature selection processes are explained in Section 4.3 and the implementation of the candidate intrusion detection models is discussed in Section 4.4. Section 4.5 concludes the chapter.

4.2 Data processing and feature selection

The predictive power of an ML model is highly dependent on the quality of the training dataset. Hence, intensive data processing and feature engineering is a crucial step that cannot be overlooked. In this study, the quality of the data was enhanced by removing duplicated features from the dataset. **Figure 4.1** shows how the duplicated features were removed from the datasets.

```
#Drop duplicated features if any
duplicated_features = X_train.duplicated()
features_to_keep = [not index for index in duplicated_features]
```

Figure 4.1: Dropping duplicated features

Additionally, the dataset was enriched by adding a MAC overhead feature computed as the ratio of the packet length and the throughput; throughput is the sum of packets received by all nodes in the network. The processing further involved converting the data types of the features to make the data suitable to feed into an ML model. Moreover, features that required label encodings, such as the protocol type and the target feature (packet status), were encoded appropriately. Label encoding refers to converting text or string labels into numerical values. **Figure 4.2** demonstrates how the data types conversion and the encoding of categorical features were performed.

```
# Converting data types
data['length'] = data.length.astype(int)
data['cumulative_bytes'] = data.cumulative_bytes.astype(float)
data['mac_overhead'] = data.mac_overhead.astype(float)
data['time_delta_from_previous_captured_frame'] = data.time_delta_from_previous_captured_frame.astype(float)

# Encode categorical features
le = LabelEncoder()
data['protocol'] = le.fit_transform(data['protocol'])
data['packet_status']= data.packet_status.map( {'normal':0 , 'malicious':1})
```

Figure 4.2: Converting data types and encoding of categorical features

Upon completion of the data processing and transformation, a descriptive data analysis was performed through pandas-profiling to generate an intensive exploratory report of the dataset. To generate the pandas-profiling report the pandas-profiling library is imported and the profile report function, as shown in **Figure 4.3,** is invoked.

```
import pandas_profiling
data.profile_report()
```

Figure 4.3: Generating a pandas-profile report

Table 4.1 and **Table 4.2** are sneak peeks of the profile report gathered from the dataset before any feature selection was orchestrated.

Table 4.1: Pandas-profiling data description snapshot

Dataset statistics	
Number of variables	8
Number of observations	800000
Missing cells	0
Missing cells (%)	0.0%

Variable types	
CAT	4
NUM	4

Table 4.2: Pandas-profiling warnings snapshot

Dataset has 688 (0.1%) duplicate rows	Duplicates
`arrival_time` has a high cardinality: 622604 distinct values	High cardinality
`source_ip` has a high cardinality: 788 distinct values	High cardinality
`mac_overhead` is highly correlated with `length`	High Correlation
`length` is highly correlated with `mac_overhead`	High Correlation
`time_delta_from_previous_captured_frame` is highly skewed (γ1 = 57.52150881)	Skewed

This study further leveraged a beneficial feature importance functionality offered by most tree-based algorithms to help decide on the most appropriate features for developing the ML model. Feature importance is a statistical computation that measures a score for each feature in a dataset. The higher the score, the more relevant the feature towards estimating the output variable. The feature importance is built-in for various tree-based ML algorithms that include RF, XGBoost, Catboost and LightGBM.

In this study, a Catboost model was trained with the aim of extracting the features whose importance was considered significant in the Catboost training process. The process followed to train the Catboost model and generate feature importance is illustrated in **Appendix B**. **Figure 4.4** shows the generated feature importance plot.

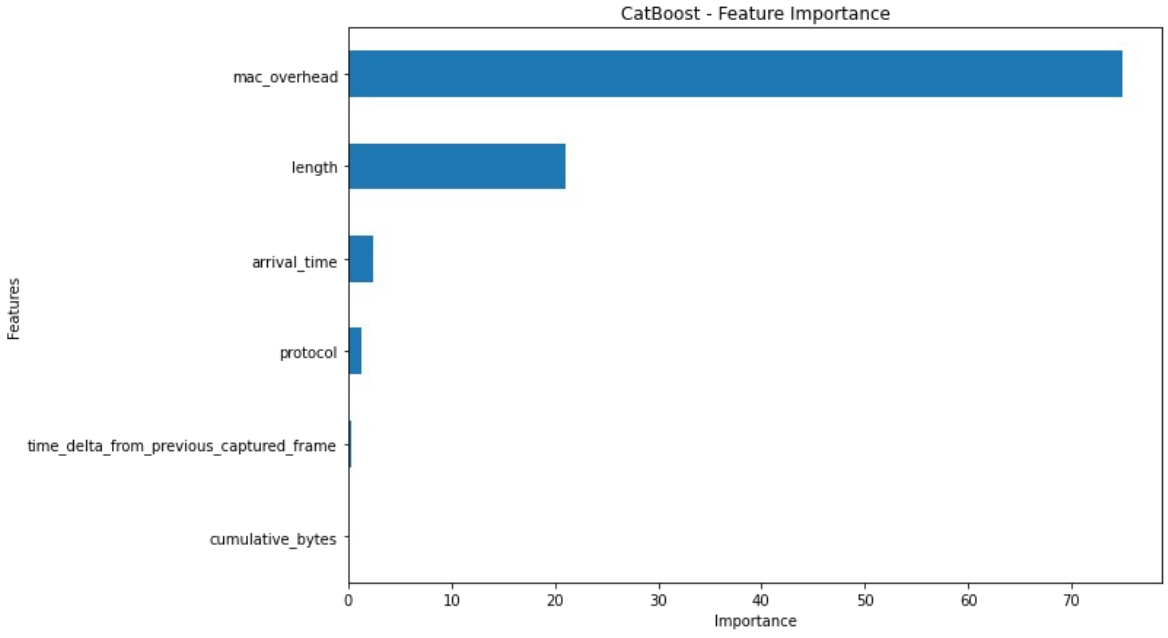

Figure 4.4: Catboost feature importance

The final training features are presented in **Table 4.3**.

Table 4.3: Training dataset sample

arrival_time	protocol	length	cumulative_bytes	time_delta_from_previous_captured_frame	mac_overhead
95	6	716	3417932.0	0.007063	8.631448e-06
230	1	84	18271290.0	0.000244	6.402024e-07
160	3	88	7406936.0	0.000532	1.276946e-06
43	1	84	2630640.0	0.007083	6.402024e-07
261	1	84	27512966.0	0.000232	6.402024e-07

By default, convolutional NNs and LSTMs, take three-dimensional matrix input data; subsequently, the last data transformation step before training the model was to reshape the data into a three-dimensional matrix. This step is illustrated in **Figure 4.5**.

```
# Reshaping data
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_metric = X_metric.reshape((X_metric.shape[0], X_metric.shape[1], 1))
X_val = X_val.reshape((X_val.shape[0], X_val.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Figure 4.5: Reshaping the data to a three-dimensional matrix

4.3 Model training approach and performance evaluation

The modelling approach followed four (4) steps:

1. Training a regular DL model
2. Evaluating the performance of the regular DL model
3. Converting the regular DL model to a lightweight version of the model
4. Assessing the lightweight model's performance to determine whether the lightweight model is as effective and reliable as the regular model.

Six (6) features – arrival time, protocol, length, cumulative bytes, time delta from the previously captured frame and mac overhead to predict the packet status (whether the transmitted packet is normal or malicious) – were used in training the models.

The target feature comprises two classes – the normal class and the malicious class. A class is considered normal if the transmitted packet does not contain malware and is considered malicious if it contains malware. In building a binary classification model, two NN-based candidate models, viz, CNN and LSTM, were trained.

The aim of training two models was to compare the classification efficiency of the two models and to select the model that offers the best predictive power. Each of the two models consists of three (3) fully connected layers. The ReLu activation function was applied at the input layers and the sigmoid function at the output layer. In the first training step, two (2) regular candidate models were trained iteratively, tuning the different hyperparameters and adding dropout as a regularisation technique. The regularisation technique is applied to prevent overfitting, thereby improving the model's robustness and generalisation ability.

In the training approach, the dropout strategy is such that a varying number of nodes are randomly dropped into the network. For example, a dropout of 0.5 implies that 50% of the nodes will be dropped randomly in the layer in which the dropout is applied.

The models were set to train for 500 epochs. When training ML models, too many epochs may lead to the overfitting of the learning model, while too few epochs may lead to underfitting. To mitigate the likelihood of the model overfitting, an early stopping was defined to monitor the validation loss and automatically halt the training once the model started overfitting the data. **Table 4.4** presents a summary of the training parameters. The processes to train the candidate models are presented in **Appendix C** and **Appendix D**.

Table 4.4: Model training parameters

Hyperparameter	**Value**
Optimiser	Adam
Learning rate	0.0001
Evaluation metric	Accuracy
Loss	Binary cross-entropy
Number of epochs	500
Regularisation technique	Dropout (varies from 0.1 to 0.5)

The performance evaluation metric applied was accuracy and a binary cross-entropy loss function was defined to monitor the loss at each training iteration. The models were evaluated based on three (3) primary evaluation metrics, namely, the model accuracy, training and validation loss curves and the confusion matrix. A code snippet that demonstrates how the

model accuracy and loss curves were plotted is shown in **Figure 4.6** and a snippet that shows the plotting of a confusion matrix is depicted in **Figure 4.7**.

```
#plot training and validation accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train' , 'Test'],loc = 'upper left')
plt.show()

#Plot training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train' , 'Test'],loc = 'upper left')
plt.show()
```

Figure 4.6: Plotting accuracy and loss curves

```
#Plot confusion matrix
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(y_test,test_pred)
ax= plt.subplot()
sns.heatmap(mat, annot=True, ax = ax,fmt='g',cmap='Blues');
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix: Test Set');
ax.xaxis.set_ticklabels(['normal', 'malicious']); ax.yaxis.set_ticklabels(['normal', 'malicious']);
```

Figure 4.7: Plotting confusion matrix

A model that demonstrated robustness during the validation step was saved and converted to a lightweight version. **Figure 4.8** and **Figure 4.9** depict the conversion of a regular NN model to a lightweight version.

```
#Save model
keras_file = 'model.h5'
keras.models.save_model(model,keras_file)
```

Figure 4.8: Saving a model

```
#Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open('model.tflite', "wb").write(tflite_model)
```

Figure 4.9: Converting a regular model to a lightweight model

The last step of the evaluation was loading the lightweight model and assessing its performance on the test set to evaluate whether the model preserves the performance of the regular model. The loading and evaluation of the lightweight model are demonstrated in **Figure 4.10**.

```
#Load TFLite model and allocate tensors
result = []
X_test = X_test.astype(np.float32)
interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()

#Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

#Test model on test data.
for i in range(len(X_test)):
    input_shape = input_details[0]['shape']
    interpreter.set_tensor(input_details[0]['index'], X_test[i].reshape(1,6,1))
    interpreter.invoke()
    output_data = interpreter.get_tensor(output_details[0]['index'])
    result.extend(np.array(interpreter.get_tensor(output_details[0]['index']).astype(int)))

#Plot confusion matrix
mat = confusion_matrix(y_test,result)
ax= plt.subplot()
sns.heatmap(mat, annot=True, ax = ax,fmt='g',cmap='Blues')

ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels')
ax.set_title('TfLite Model Confusion Matrix')
ax.xaxis.set_ticklabels(['malicious', 'normal'])
ax.yaxis.set_ticklabels(['malicious', 'normal'])
```

Figure 4.10: Loading and assessing the lightweight model

The entire study's implementation strategy is presented in **Figure 4.11**.

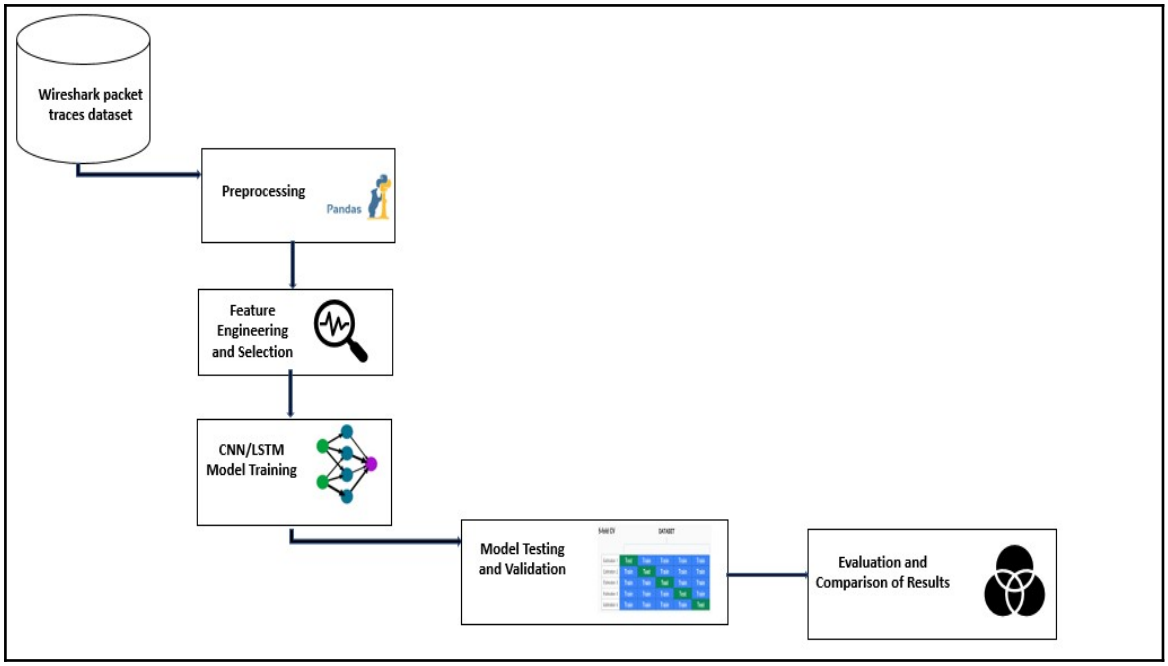

Figure 4.11: The implementation strategy

4.4 Summary

This chapter gave an overview of the implementation techniques adopted by this study and outlined how each technique was applied to achieve the aim of the study. The methods followed to gather the data used for the training and testing of the candidate models was described. Data processing and featuring engineering approaches exploited in this study were demonstrated. Finally, the implementation of the training strategy of the models and the methods adopted for the evaluation of the trained models were presented.

Chapter 5: Study Results and Discussion

5.1 Introduction

In this study, a CNN with three (3) fully connected layers was used as a baseline model and an LSTM network with three fully connected layers was proposed. This chapter focuses on comparing the performance of the baseline model against the performance of the proposed model. The performance evaluation of the models is primarily focused on three pivotal metrics, namely, detection rate (true positives), false alarm rate (false positives) and false negative rates. Furthermore, the evaluation outcomes of both models are concisely analysed and discussed.

5.2 Results

5.2.1 Convolutional neural network performance analysis

The approach involved training two candidate models. This section presents the performance analysis of the CNN classifier. The first attempts to train the models were without the application of any regularisation technique. **Figure 5.1** presents the model accuracy of a CNN without any regularisation applied, that is, no dropout has been added in any of the network layers. As can be seen in **Figure 5.1**, the accuracy of the model on the training set is 100% and drops slightly to 99% on the validation set. This indicates that the model is overfitting and is not likely to generalise well when applied to unseen data.

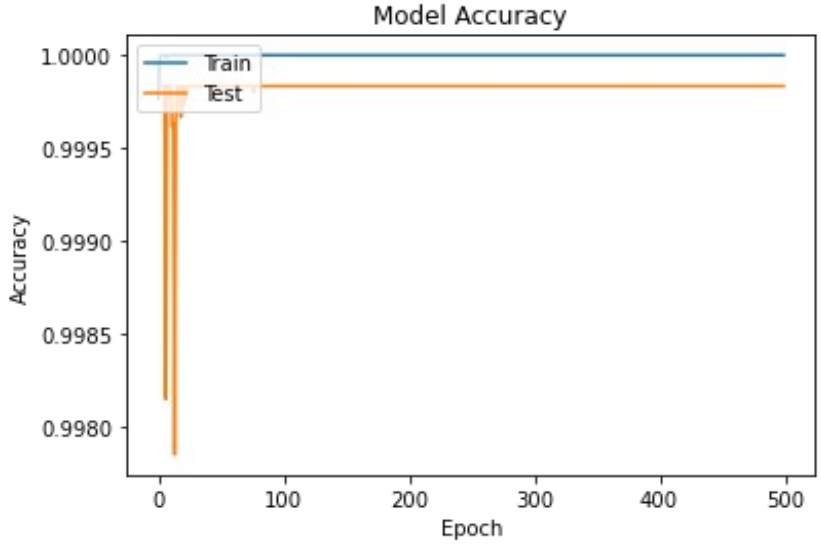

Figure 5.1: Model accuracy of an unregularised CNN

Figure 5.2 shows the loss during the training process of the unregularised CNN. Loss is defined as a penalty for bad prediction. Ideally, the loss should decrease as the model learns the data. However, in **Figure 5.2** it can be observed that the validation loss fluctuated instead of decreasing. This is a clear indicator that the model does not converge and is likely to overfit the data.

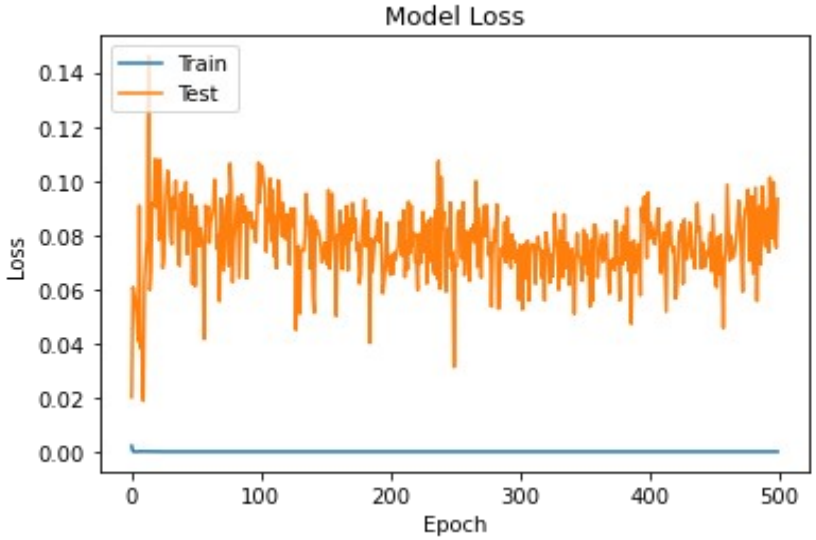

Figure 5.2:Training vs validation loss of the unregularised CNN

The confusion matrix shown in **Figure 5.3** presents the classification accuracy of the validation set. It can be seen in **Figure 5.3** that, out of the 210 instances, 190 were correctly classified as malicious while 20 malicious instances were misclassified as normal, yielding a false negative rate of 9.5%. This is an overall 86% detection rate and the false alarm rate is 0%. This implies that although the model exhibits a high detection accuracy, it would fail to raise an alarm 9.5% of the time.

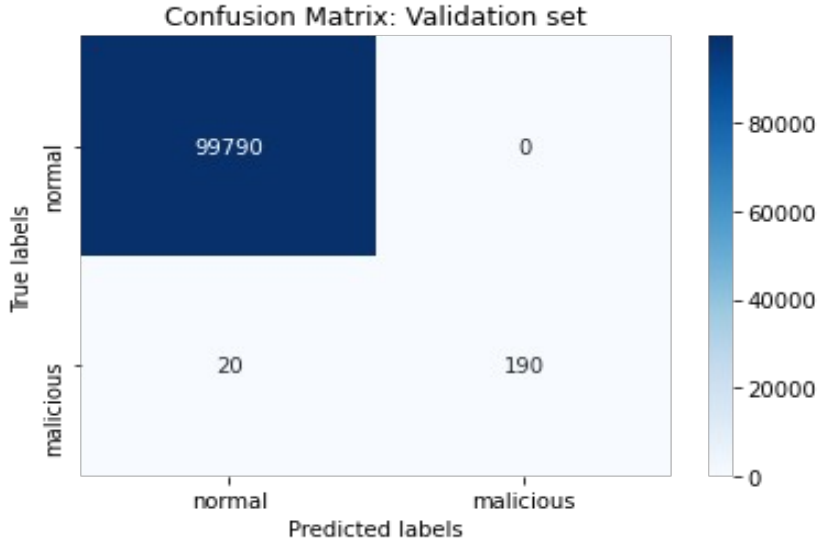

Figure 5.3: Validation set confusion matrix – unregularised CNN

Figure 5.4 presents the confusion matrix for the test set classification accuracy. From the figure, it can be seen that the classification accuracy of the model gradually declined. Out of 225 instances, 181 were correctly classified as malicious while 44 malicious instances were incorrectly classified as normal. This yields a detection rate of 80% and a 19.5% false negative rate. From this result, it can be observed that the detection and false negative rates gradually drop by 6% and 10%, respectively, while the false alarm rate remains at 0%. This is a clear indicator that the model's generalisation ability is poor as we can see that it performs poorly on the test set compared to the validation set.

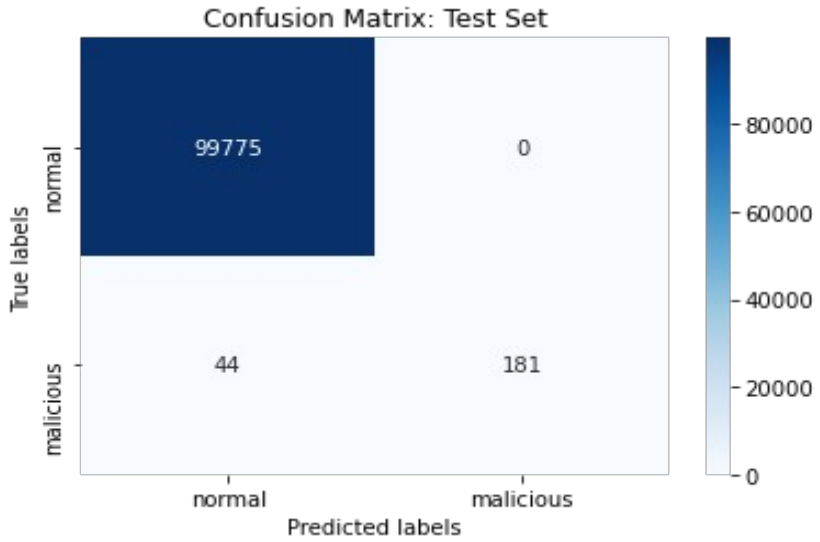

Figure 5.4: Test set confusion matrix – unregularised CNN

In **Figure 5.5**, the accuracy of the CNN model with a dropout of 0.1 applied in the first layer as a regularisation technique is shown. Again, the validation accuracy drops. This is an indication that the model is still overfitting the data.

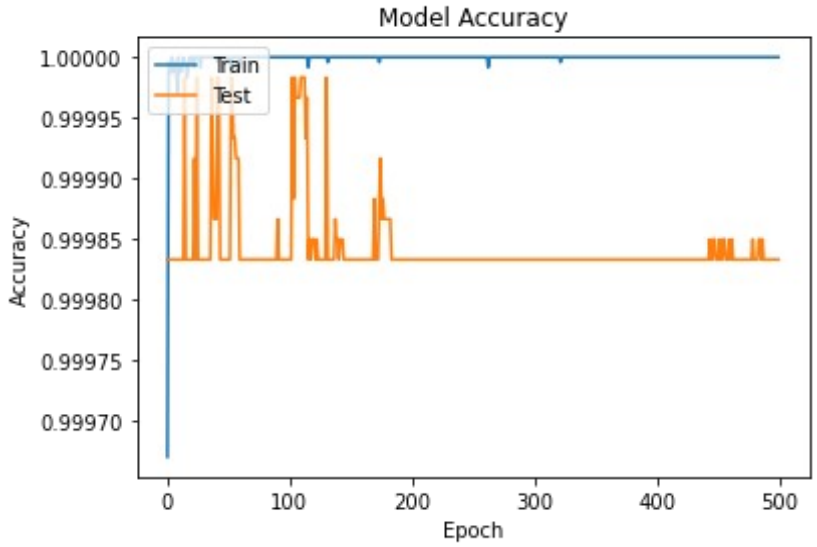

Figure 5.5: Model accuracy of a CNN with 0.1 dropout in first layer only

Training and validation loss of the CNN with 0.1 dropout in the first layer is depicted in **Figure 5.6**. From the figure, it can be noted that although the validation loss increases and decreases unstably, it is slightly better than the validation loss presented in **Figure 5.2**, where the model is not regularised. This implies a slightly better generalisation ability of the regularised model compared to its unregularised counterpart.

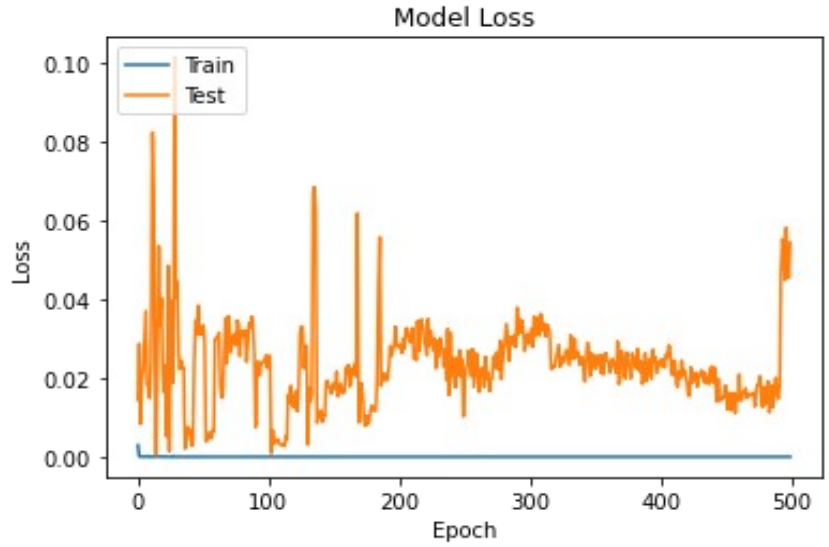

Figure 5.6:Training vs validation loss of the CNN with 0.1 dropout in first layer only

The confusion matrix shown in **Figure 5.7** demonstrates a slight improvement in the detection and false negative rates. From **Figure 5.7**, it can be noted that the detection rate improved from 86% to 90% while the false negative rate improved from 9.5% to 0.9% when

the model was assessed on the validation set. This is an improvement of 4% and 5% in detection and false negative rates, respectively. This minor improvement in generalisation ability demonstrated by the validation loss of the model as depicted in **Figure 5.6** is visible in the model's classification accuracy as shown in **Figure 5.8**. It is worth noting that adding a dropout of 0.1 in the model's first layer did not remarkably improve the predictive power of the model – the false alarm rates remained at 0%.

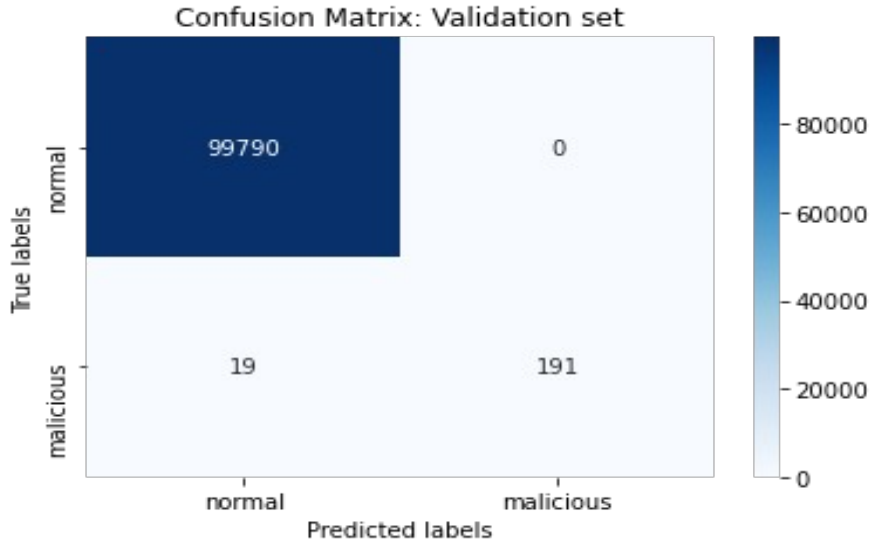

Figure 5.7: Validation set confusion matrix – CNN with 0.1 dropout in first layer only

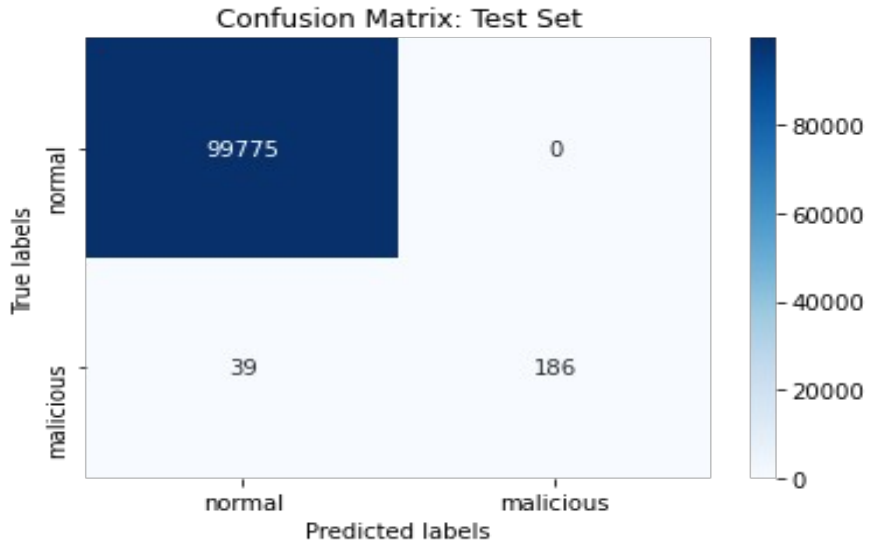

Figure 5.8:Test set confusion matrix – CNN with 0.1 dropout in first layer only

Realising that regularising the CNN model by adding a dropout of 0.1 in the first layer did not demonstrate much significance in terms of detection rates, false negative rates and

generalisation ability, the dropout rate in the first layer was increased to 0.2. Regarding **Figure 5.9**, it can be deduced that increasing the dropout rate improved the model accuracy to a certain degree, however, the change in accuracy was not compelling. **Figure 5.10** shows the loss for the CNN when the dropout rate applied is 0.2. From the figure, it is notable that the validation loss is still fluctuating and far above zero at some epochs. This is a strong indicator that the model's penalty for bad predictions is still higher.

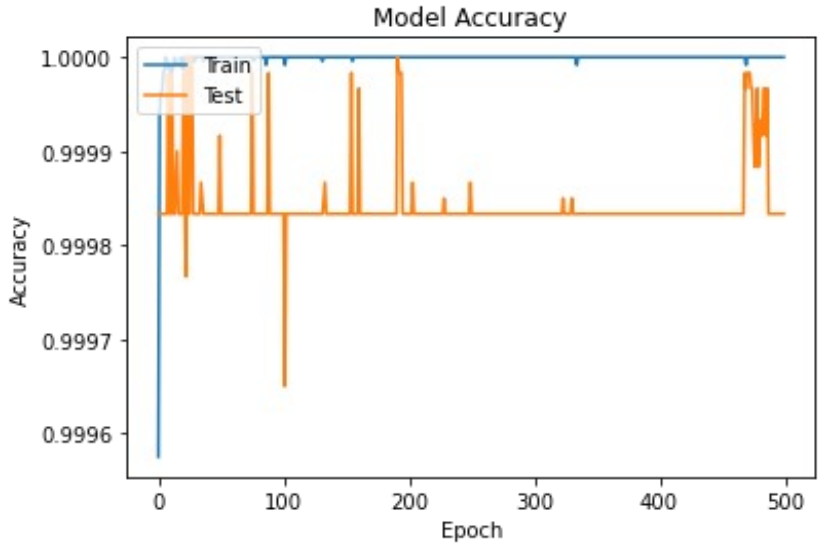

Figure 5.9: Model accuracy of a CNN with 0.2 dropout in first layer only

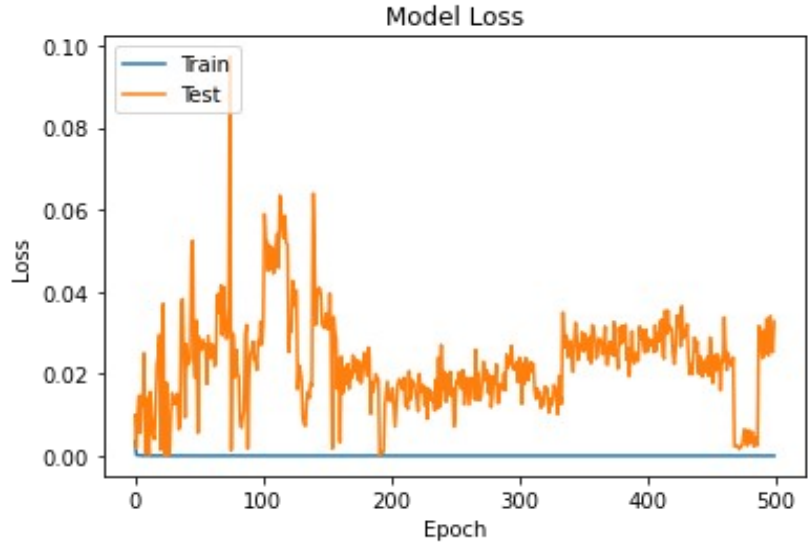

Figure 5.10: Training vs validation loss of the CNN with 0.2 dropout in first layer only

The confusion matrix presented in **Figure 5.11** demonstrates that when a dropout of 0.2 is applied, the detection rates, false negative rates and false alarm rates on the validation set do

not improve or deteriorate compared to when the dropout rate was 0.1. **Figure 5.12** presents the confusion matrix that depicts how the CNN model performs on the test set when the dropout rate in the first layer is increased to 0.2. From the confusion matrix, a slight improvement in detection and false negative rates compared to when the dropout rate was 0.2 can be observed. From this observation, it can be concluded that increasing the dropout rate improved the model's predictive power even though the validation loss of the model was relatively higher, which implies that the generalisation ability thereof is still poorer.

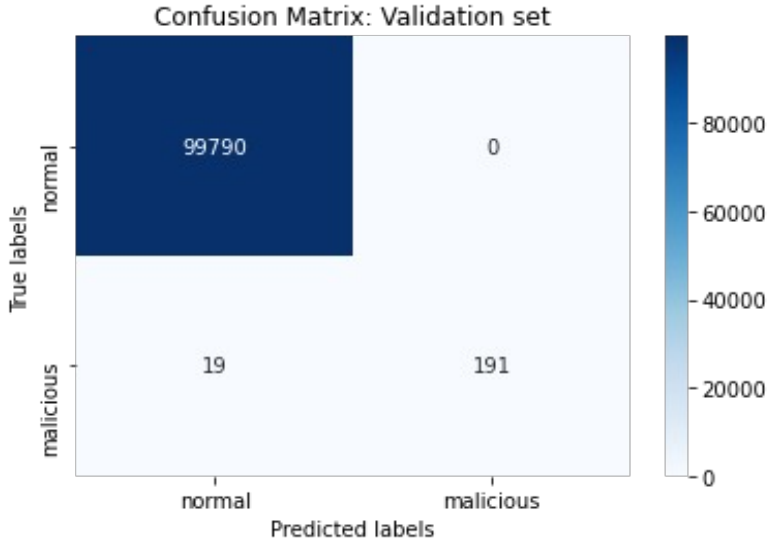

Figure 5.11: Validation set confusion matrix – CNN with 0.2 dropout in first layer only

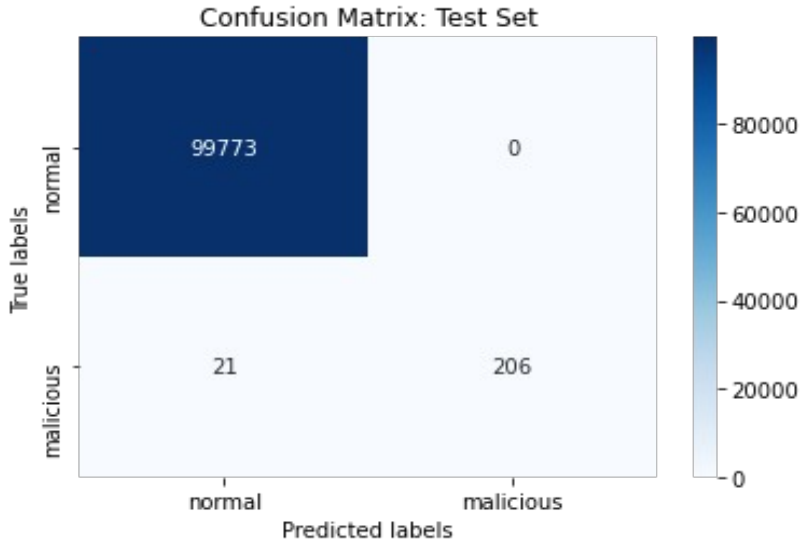

Figure 5.12: Test set confusion matrix – CNN with 0.2 dropout in first layer only

Presented in **Figure 5.13** is a model accuracy of a CNN with a dropout rate of 0.5 in the first layer. A slight improvement in accuracy is visible when the dropout rate is increased to 0.5. **Figure 5.14** illustrates the training and validation loss of a CNN with a dropout rate of 0.5 applied as a regularisation technique only in the first layer of the network. Again, there is no significant change compared to when the dropout rates were 0.1 and 0.2.

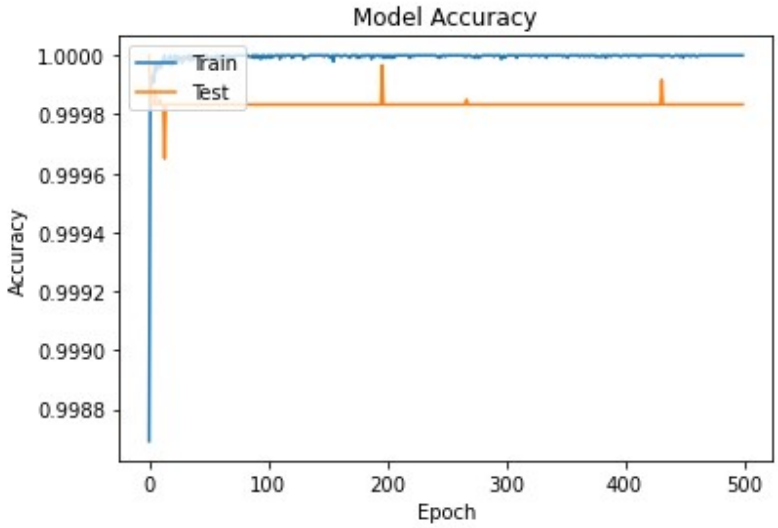

Figure 5.13: Model accuracy of a CNN with 0.5 dropout in first layer only

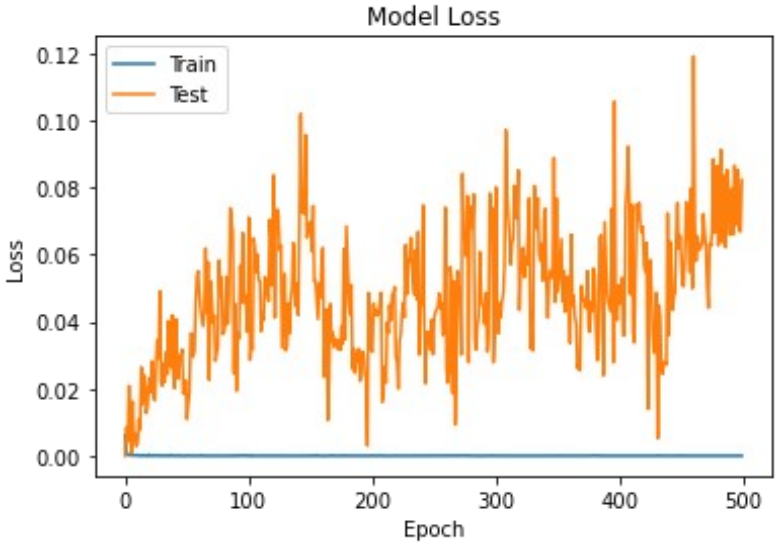

Figure 5.14: Training vs validation loss of the CNN with 0.5 dropout in first layer only

In **Figure 5.15**, a validation set confusion matrix that illustrates the performance of the CNN classifier when a dropout rate of 0.5 is applied in the first layer of the model is presented. The results indicate that dropping 50% of the nodes in the first layer of the network deteriorated the predictive power of the model. This is observable in the slight increase in false negative

rates compared to when 20% of the network nodes were dropped. The results of adding a 0.5 dropout rate in the first layer are similar to the results obtained when the dropout rate was 0.1.

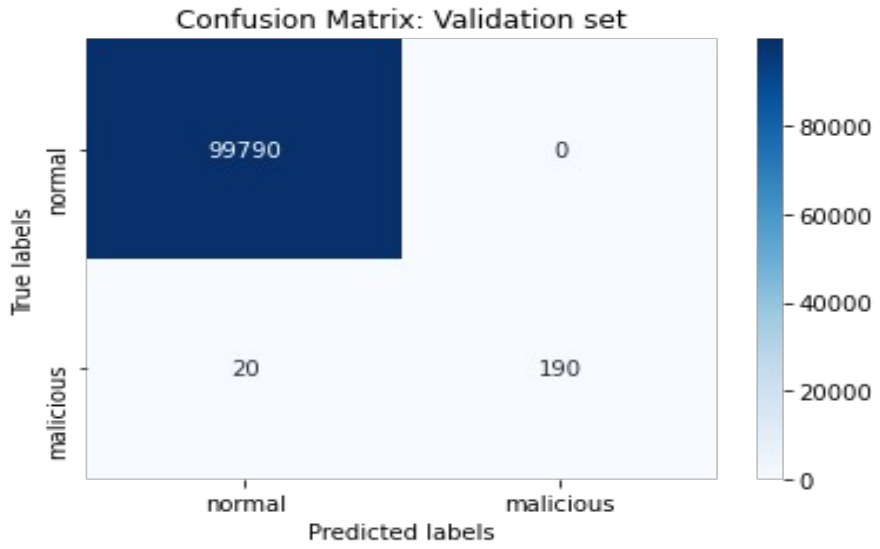

Figure 5.15: Validation set confusion matrix – CNN with 0.5 dropout in first layer only

Figure 5.16 presents the performance of the CNN model on the test set when the dropout rate is 0.5. From the figure, a 10% detection rate can be observed which is indicative of a model that is unable to generalise. The observations demonstrated in **Figure 5.15** and **Figure 5.16** clearly suggest that accuracy alone is not a reliable metric to measure the predictive power and generalisation ability of a classification model.

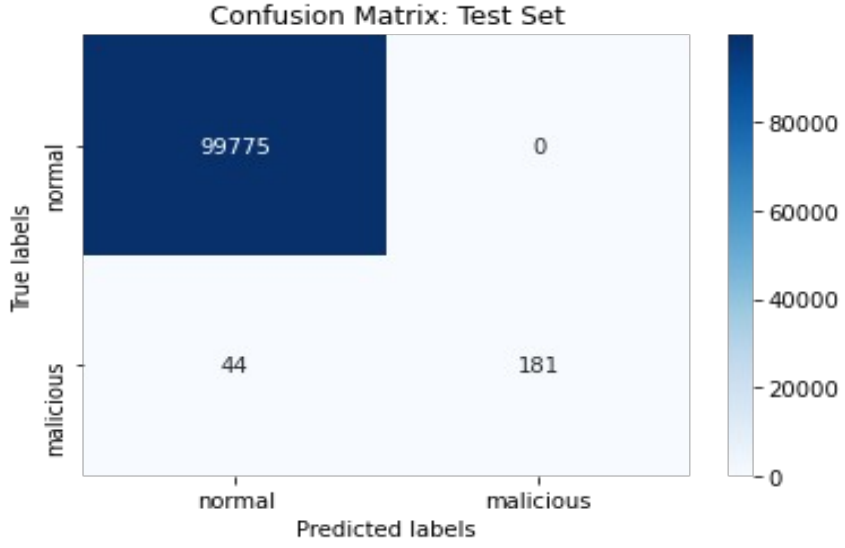

Figure 5.16: Test set confusion matrix – CNN with 0.5 dropout in first layer only

The second last experiment on the CNN was adding a dropout rate of 0.5 in the first two layers of the CNN. **Figure 5.17** demonstrates the performance of the classifier in terms of accuracy. From the figure, it can be observed that the best approach would have been to train the classifier with fewer epochs when the dropout is 0.5 in the first two layers. However, training the model with fewer epochs does not guarantee a generalising model as shorter training time may result in underfitting the model. Regarding the accuracy demonstrated in the figure, the model performed well during the first 100 iterations/epochs and dropped slightly afterwards.

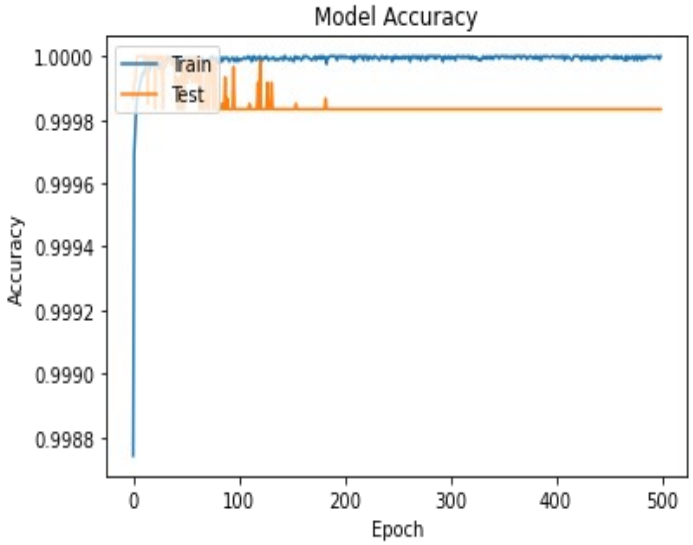

Figure 5.17: Model accuracy of a CNN with 0.5 dropout in first and second layers

Figure 5.18 is a clear demonstration of the model's train versus validation loss. From the figure, it is quite clear that the validation loss was lower before the first 100 epochs of the training. This implies that the model started overfitting the data just before reaching the first 100 epochs. Additionally, what this implies is that adding a 0.5 dropout rate in the first two layers of the CNN is more optimal with fewer training epochs. **Figure 5.19** is a validation confusion matrix that demonstrates the performance of the CNN classifier when a dropout of 0.5 in the first two layers is defined. The results obtained when applying this dropout rate are not indifferent to the results when the dropout rate was 0.5 in the first layer only.

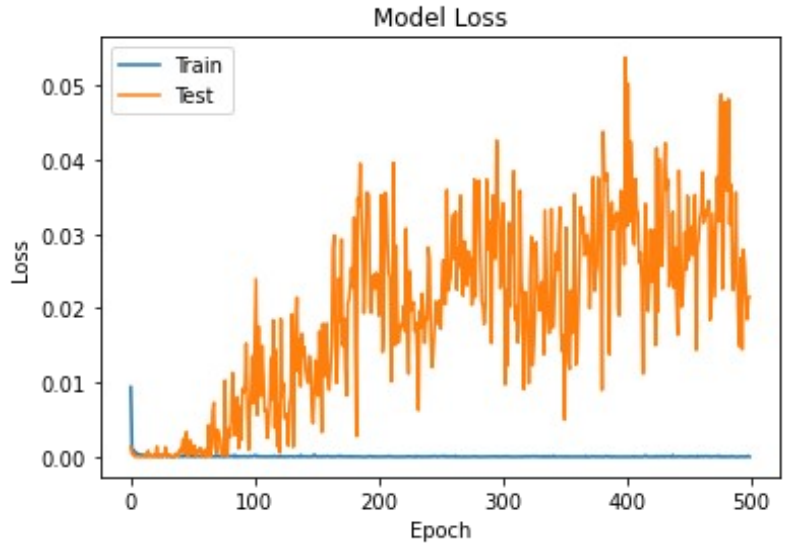

Figure 5.18: Training vs validation loss of the CNN with 0.5 dropout in first and second layers

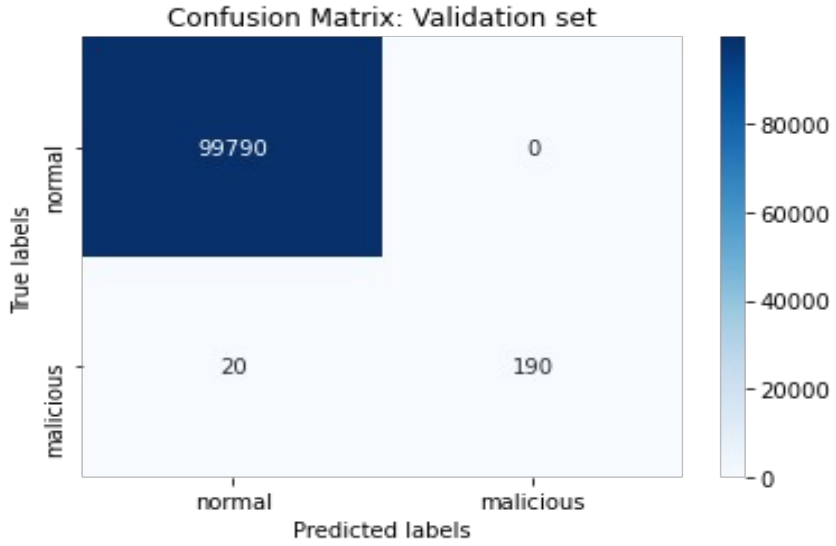

Figure 5.19: Validation set confusion matrix – CNN with 0.5 dropout in first and second layers

The validation set confusion matrix presented in **Figure 5.20** shows a slight improvement in generalisation ability when a dropout of 0.5 in the first two layers is defined compared to when the dropout of 0.5 is applied only in the first layer of the CNN classifier. However, the generalisation ability of the model is still poor as it can be observed that the detection rate in the test set dropped when compared to the detection rate in the validation set. **Figure 5.21** presents the CNN classifier accuracy when the regularisation technique is such that a dropout

rate of 0.5 is applied in all the layers of the network. The demonstrated accuracy suggests that the said regularisation technique improved the accuracy of the model remarkably.

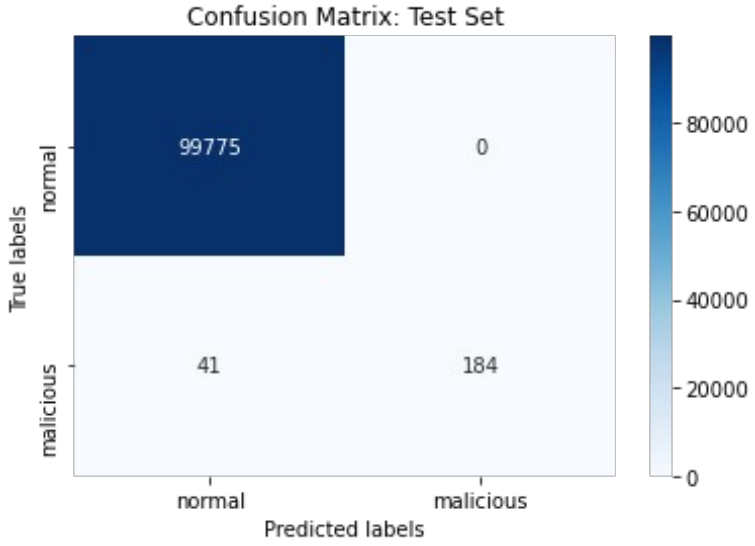

Figure 5.20: Test set confusion matrix – CNN with 0.5 dropout in first and second layers

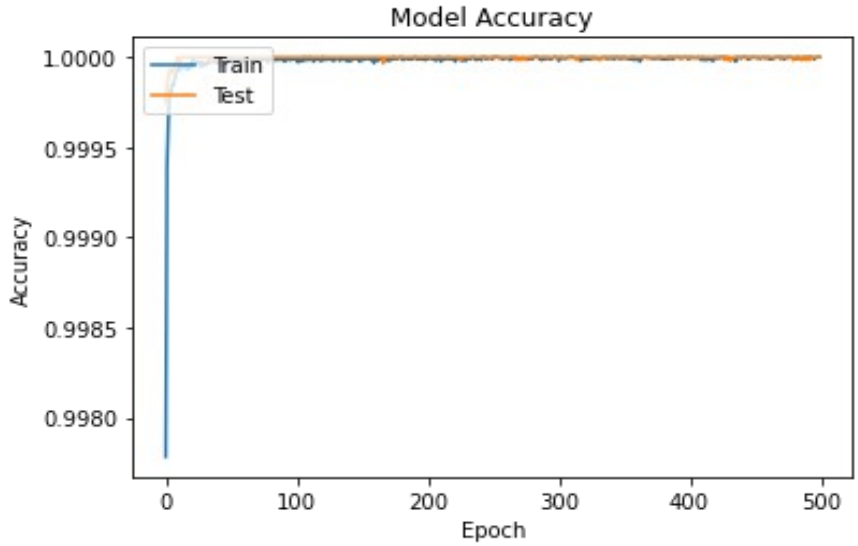

Figure 5.21: Model accuracy of a CNN with 0.5 dropout in all three layers

In **Figure 5.22,** the training and validation loss curves when a dropout rate of 0.5 is applied in all the layers of the CNN classifier are shown. From the results presented in the figure, it can be observed that defining a dropout rate of 0.5 in all three layers reduces the penalty poor prediction power. This strongly suggests that the model exhibits a relatively high predictive power and can generalise unseen data well.

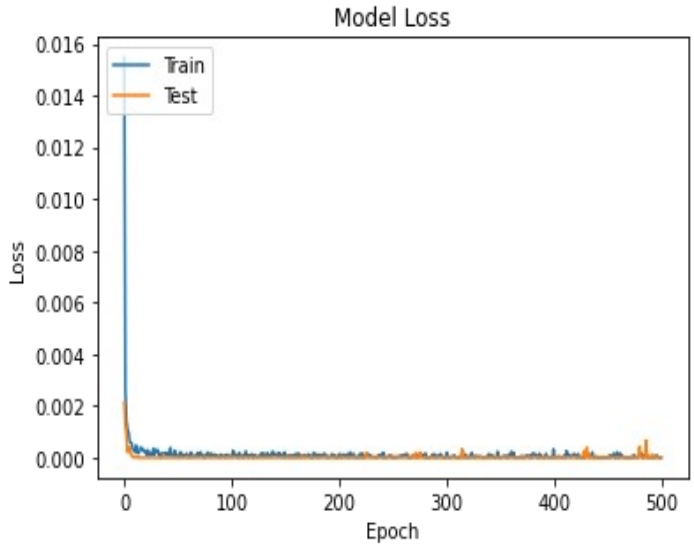

Figure 5.22: Training vs validation loss of the CNN with 0.5 dropout in all three layers

Figure 5.23 is a visual demonstration of the performance of the CNN classifier when a dropout rate of 0.5 is applied in all three defined layers of the CNN model. The results demonstrated that defining a higher dropout in all the defined layers optimised the detection rates yielded by the model. From the validation set confusion matrix, it can be observed that the model obtained 100% in detection, false negative and false alarm rates.

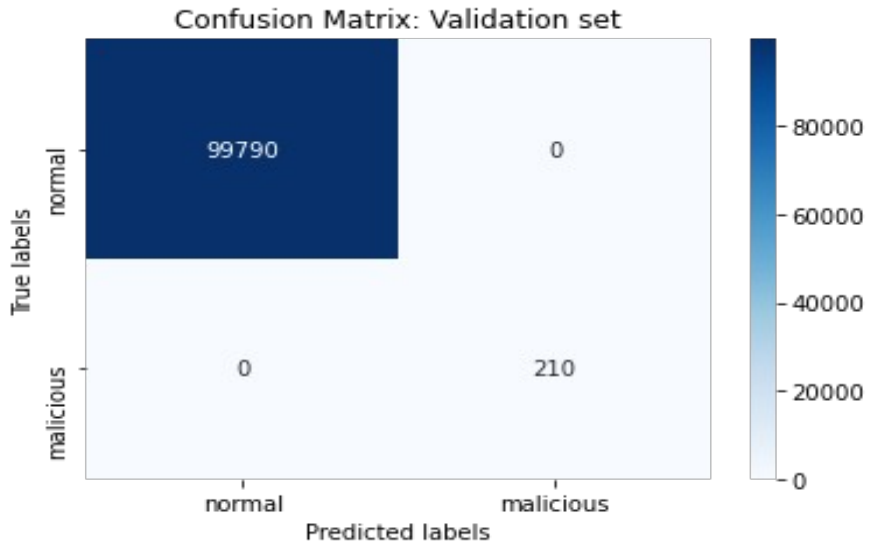

Figure 5.23: Validation set confusion matrix – CNN with 0.5 dropout in all three layers

Figure 5.24 is a confusion matrix that shows the performance of the CNN classifier when the classifier is assessed on the test set. The model maintained a 100% score in detection, false alarm and false negative rates when compared to how the model performed in the validation

set. This indicated that the model is robust enough to predict the packet status – whether the transmitted packet is normal or malicious – accurately.

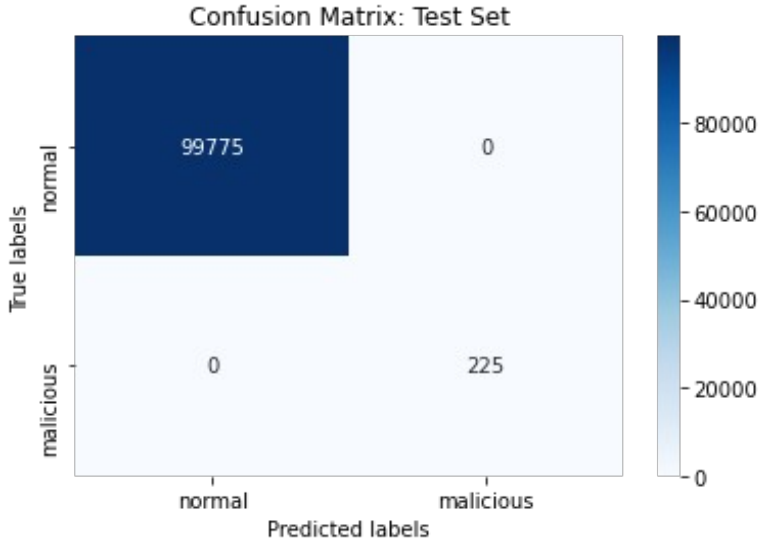

Figure 5.24: Test set confusion matrix – CNN with 0.5 dropout in all three layers

The robust model whose performance has been demonstrated in **Figures 5.21** to **5.24** was converted to a lightweight version. **Figure 5.25** shows how the lightweight version performed when cross-validated on the validation set. Regarding the results displayed in the figure, it can be observed that the lightweight version preserved the performance of the regular CNN model as it maintained 100% detection, false alarm and false negative rates.

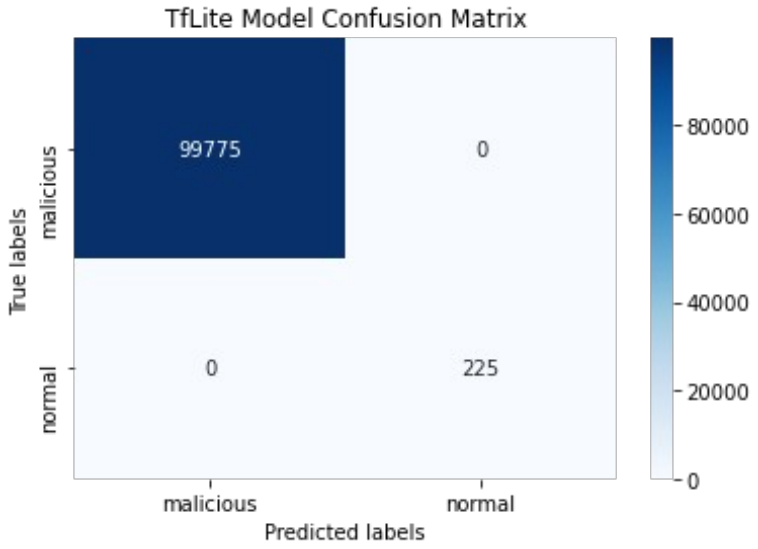

Figure 5.25: CNN lightweight version confusion matrix

5.2.2 Long short-term memory performance analysis

This section presents the analysis of the performance of the LSTM classification model. **Figure 5.26** shows the classification accuracy of the LSTM classifier when there is no regularisation technique applied. The results demonstrated in the figure show that the classifier obtained a training accuracy of 0%. This implies that the model underperforms extremely when it is not regularised. Moreover, this could imply that the number of training iterations was not sufficient to allow the model to converge. However, results demonstrated in **Figure 5.16** indicated that accuracy alone is not the best metric to measure the performance of a classifier.

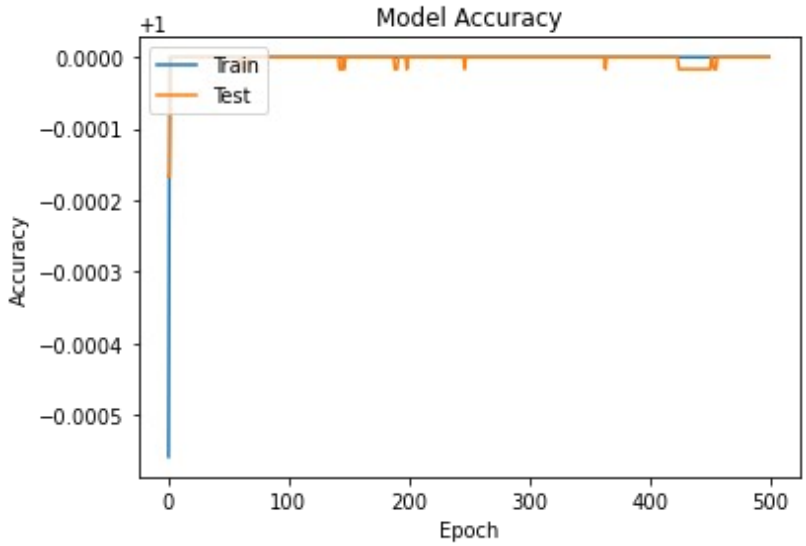

Figure 5.26: Model accuracy of an unregularised LSTM

In **Figure 5.27** the training and validation loss curves of the unregularised LSTM classifier are presented. From the figure, it can be observed that the training is 0 throughout the training process while an unsteady loss is evident on the validation loss curve. Looking at the validation loss curve, the loss is 0 during approximately the first 100 epochs and starts increasing around 150 epochs. The model then starts converging again as it approaches 200 iterations of the training and the loss increases again as the training approaches 500 iterations of the training process. With this loss, the model is expected to generalise well when applied to unseen data.

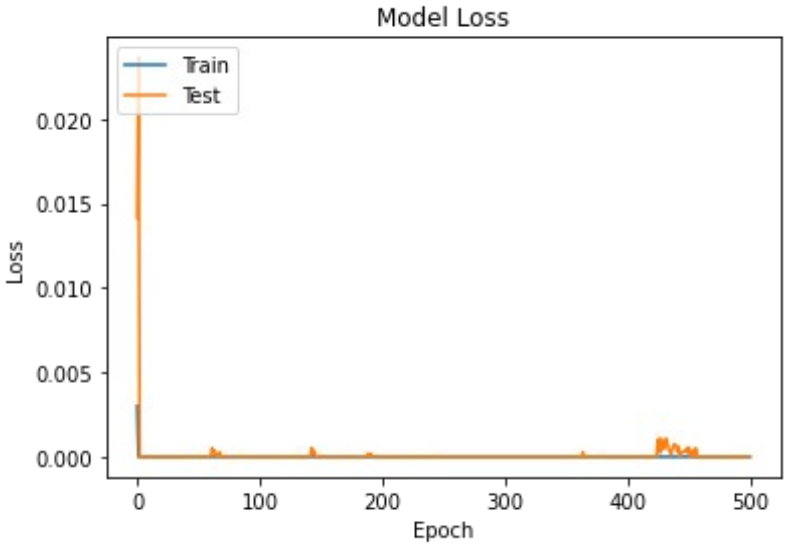

Figure 5.27: Training vs validation loss of the unregularised LSTM

Figure 5.28 presents a confusion matrix that shows the classification performance of the unregularised LSTM on the validation set. The validation set consisted of 100 000 packet traces of which 99 790 were normal while 210 were malicious traces. As it can be observed in **Figure 5.28**, all the 100 000 packets were correctly classified as either normal or malicious. From these results, it can be established that the model has proffered impressive performance on the validation set as it attained a 100% detection, false alarm, and false negative rate.

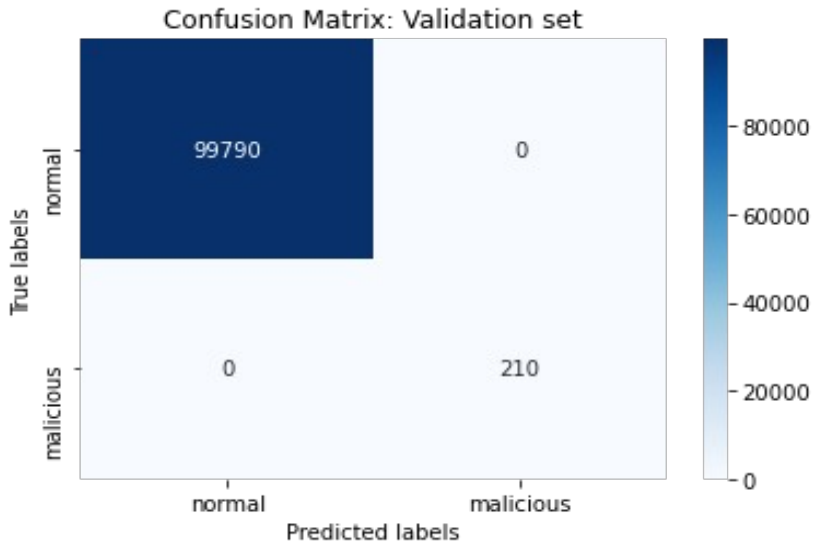

Figure 5.28: Validation set confusion matrix – unregularised LSTM

Similarly, the model was cross-validated on a test set also comprising 100 000 packet traces where 99 775 of the packet traces were normal while the remaining 225 were malicious traces. A confusion matrix that clearly displays the performance of the unregularised LSTM classifier on the test set is presented in **Figure 5.29.** Of the 100 000 packets, only three packets were misclassified. The three packets were malicious but incorrectly predicted as normal. This implies that the model failed to detect three intrusions in the test set. As can be observed, there is a marginal drop in the classification performance of the model when the model is validated on the test set. The drop in performance resulted in a 0.01 rise in false negative rates. Furthermore, the drop in performance in the second fold of the 2-fold cross-validation is evidence of overfitting.

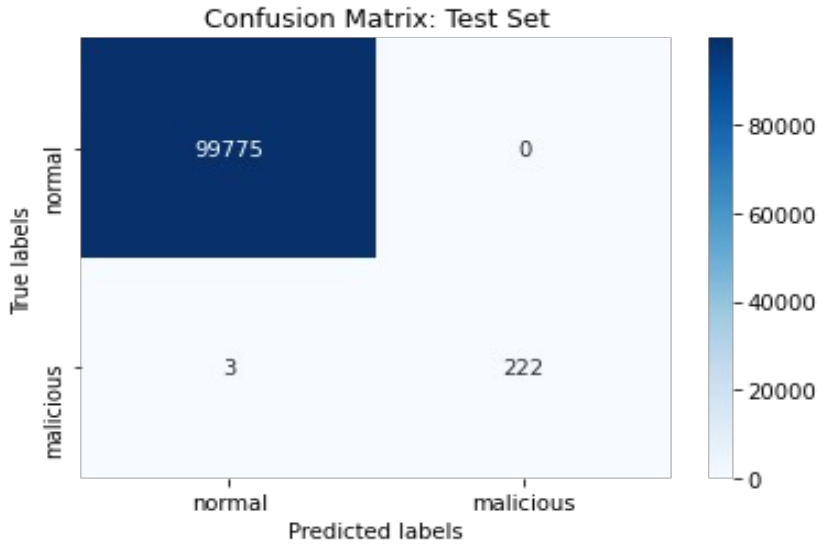

Figure 5.29: Test set confusion matrix – unregularised LSTM

Figure 5.30 shows the training and validation accuracy of the LSTM classifier when a dropout of 0.1 is added in the first layer as a regularisation technique. The classifier's learning behaviour demonstrated in the figure is similar to the behaviour demonstrated by the unregularised LSTM classifier as far as accuracy is concerned.

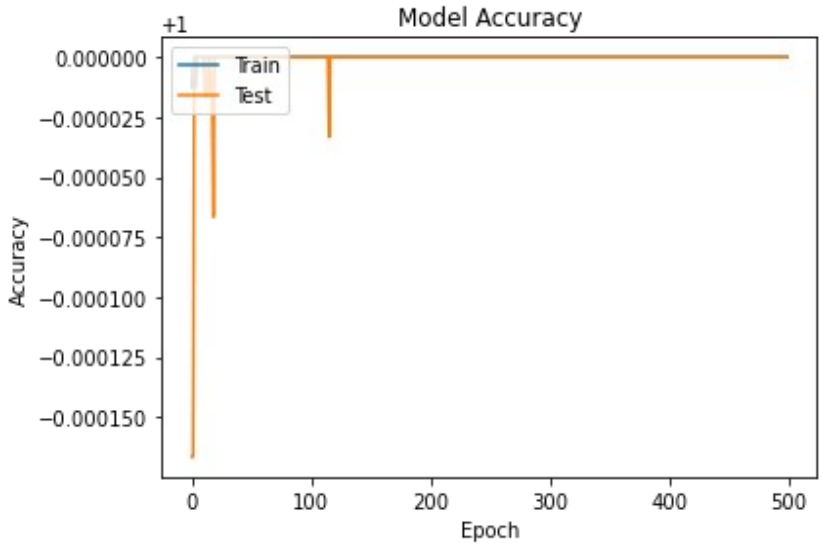

Figure 5.30: Model accuracy of a LSTM with 0.1 dropout in first layer only

Figure 5.31 presents the train and validation loss curves of the LSTM classifier when a dropout of 0.1 is applied only in the first layer of the classifier. From the curves displayed in **Figure 5.31,** the model completely converges after the first 100 epochs as the loss decreases and remains constant. Before reaching the first 100 epochs, a bit of variability in the validation loss is observed but the loss stabilises after reaching 100 epochs. After 100 epochs, both the train and validation loss remain constant at 0. This is indicative of a robust model that can generalise well even on unseen data.

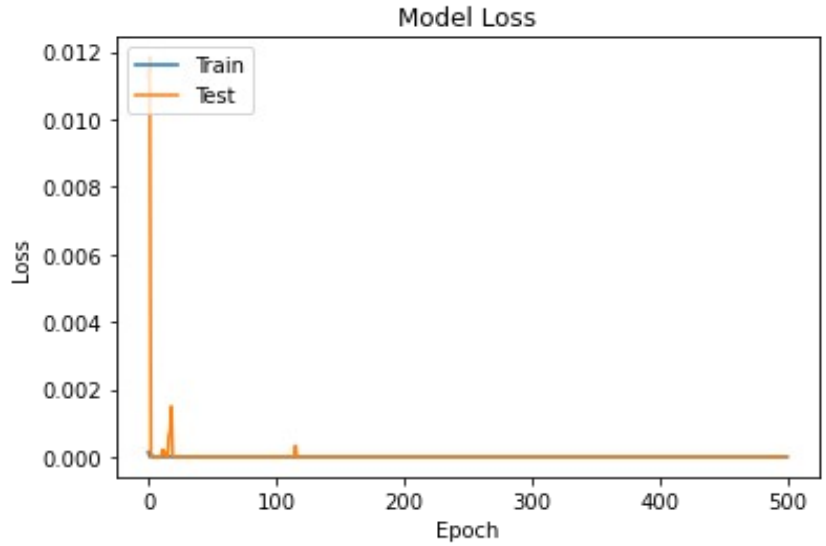

Figure 5.31: Training vs validation loss of the LSTM with 0.1 dropout in first layer only

Figure 5.32 shows a confusion matrix that demonstrates the results of the performance evaluation of the LSTM classifier on the validation set when a dropout of 0.1 is added only in the first layer. From the confusion matrix, it is observed that the model correctly classified all the packets in the validation set. Furthermore, the results demonstrate that the model can detect the normality or enmity of a transmitted packet 100% of the time. Moreover, it can be concluded that the performance of the classifier was exceptional as it obtained a 100% in detection, false alarm, and false negative rates.

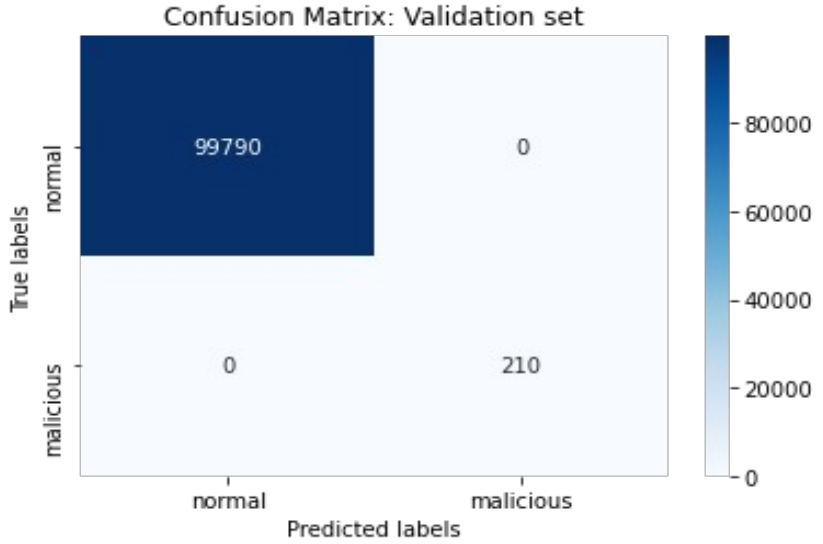

Figure 5.32: Validation set confusion matrix – LSTM with 0.1 dropout in first layer only

In **Figure 5.33**, the second fold of the cross-validation is presented. The model classified all 100 000 packets in the test dataset accurately. The outcomes of this cross-validation step suggest that the model can generalise well as it performed exceptionally well both on the first fold and the second fold of the cross-validation. From the presented results, it can be deduced that regularising an LSTM classifier plays a crucial role in the convergence speed and performance of the model and its ability to generalise.

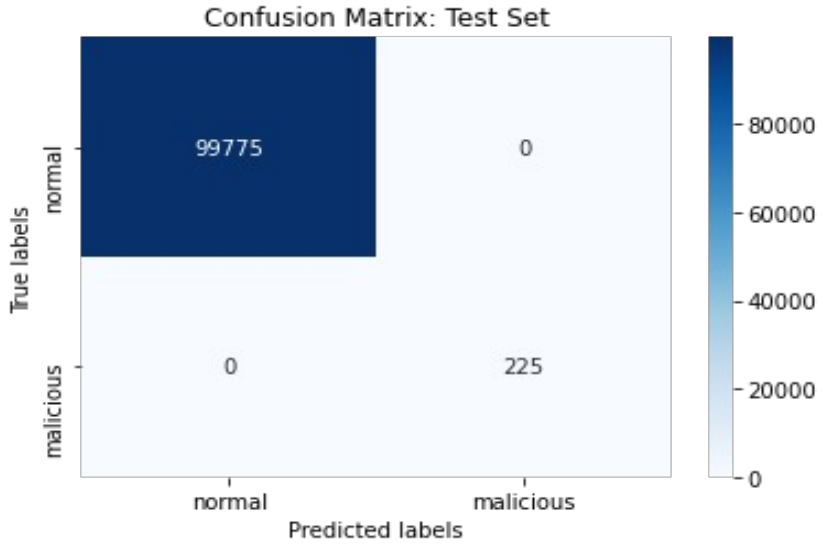

Figure 5.33: Test set confusion matrix – LSTM with 0.1 dropout in first layer only

Results presented from **Figures 5.30** to **5.33** have clearly demonstrated that adding a dropout of 0.1 in the first layer eminently improved the classification accuracy and the generalisation ability of the LSTM. To understand whether increasing the dropout rate would impact the classifier's performance, the dropout rate in the first layer was increased to 0.2 and the model was evaluated again.

Figure 5.34 is a demonstration of the model accuracy when the dropout rate is increased to 0.2 in the first layer. From the figure, the model proffers a pleasing performance as both the learning and validation accuracy are 100%.

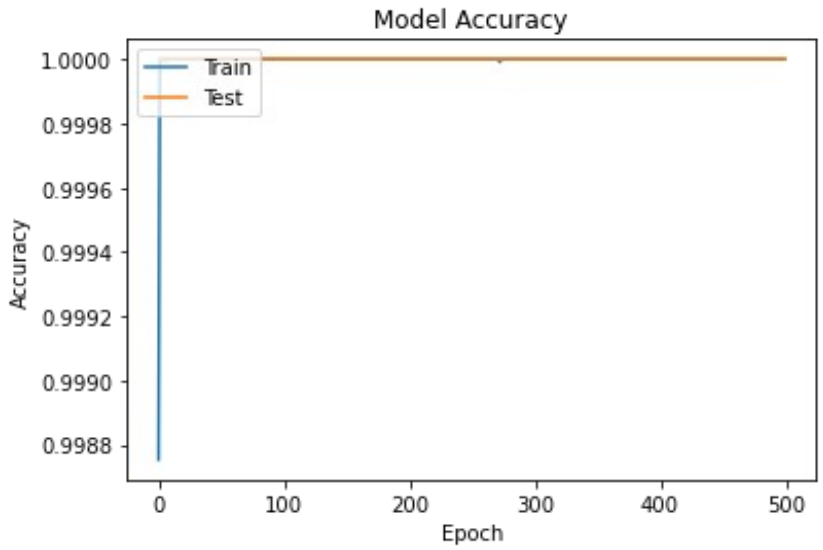

Figure 5.34: Model accuracy of an LSTM with 0.2 dropout in first layer only

Figure 5.35 shows the loss of the LSTM classification model when a 0.2 dropout is applied in the first layer. Regarding the train and validation loss curves presented in the figure, the model instantaneously converges when the dropout rate is increased to 0.2. As can be observed in the figure, both the train and validation loss remain constant at 0 throughout the training process. From this result, it can be perceived that increasing the dropout optimised the convergence speed of the model immensely. This implies that increasing the dropout rate to 0.2 in the first layer reduces the model's training time to a great extent.

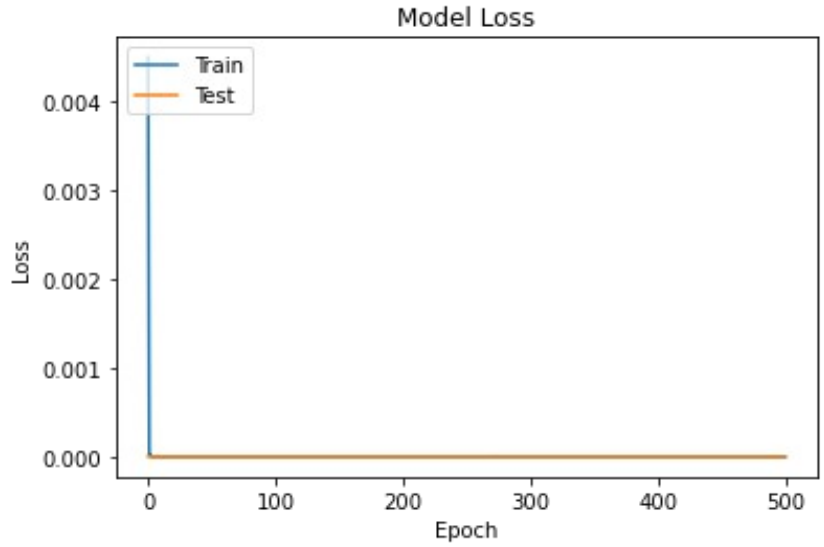

Figure 5.35: Training vs validation loss of the LSTM with 0.2 dropout in first layer only

In **Figure 5.36**, a confusion matrix shows the performance of the model on the validation set when the dropout rate is increased to 0.2. Once more, it can be observed that the model classified all 210 packets in the validation set accurately.

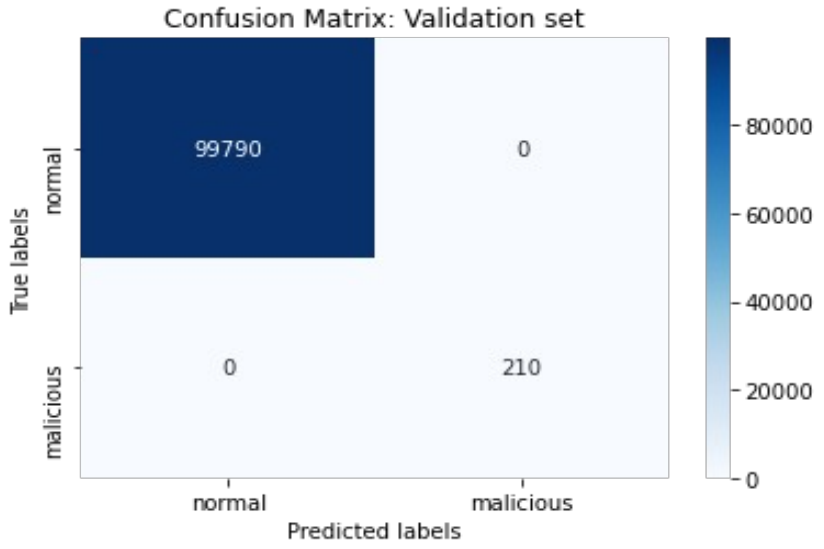

Figure 5.36: Validation set confusion matrix – LSTM with 0.2 dropout in first layer only

Figure 5.37 presents the performance of the model on the test set when the dropout rate is increased to 0.2. It can be observed that increasing the dropout in this instance introduced a negligible drop in the detection rate obtained by the classifier as the model was unable to read three of the packets accurately. However, the model was able to read 222 out of 225

malicious packets correctly while predicting all the 99 790 normal packets correctly which gives the model an impressive accuracy score.

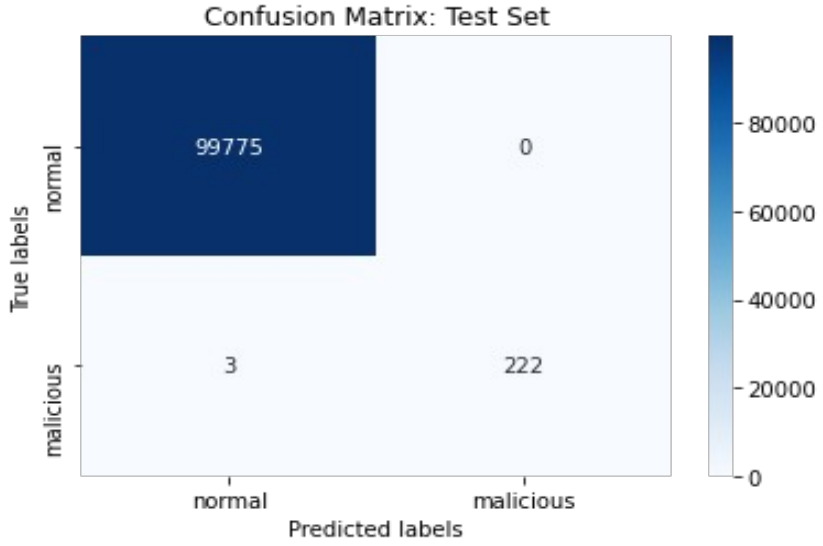

Figure 5.37: Test set confusion matrix – LSTM with 0.2 dropout in first layer only

In a similar manner as in the previous experiment, the dropout in the first layer was raised to 0.5. The results of the experiment are presented in **Figures 5.38** to **5.41**. **Figure 5.38** shows the classification accuracy of the model. From the results displayed, it can be observed that the accuracy was a bit unstable during the first 200 epochs of the training and stabilised to a 100% accuracy score subsequently.

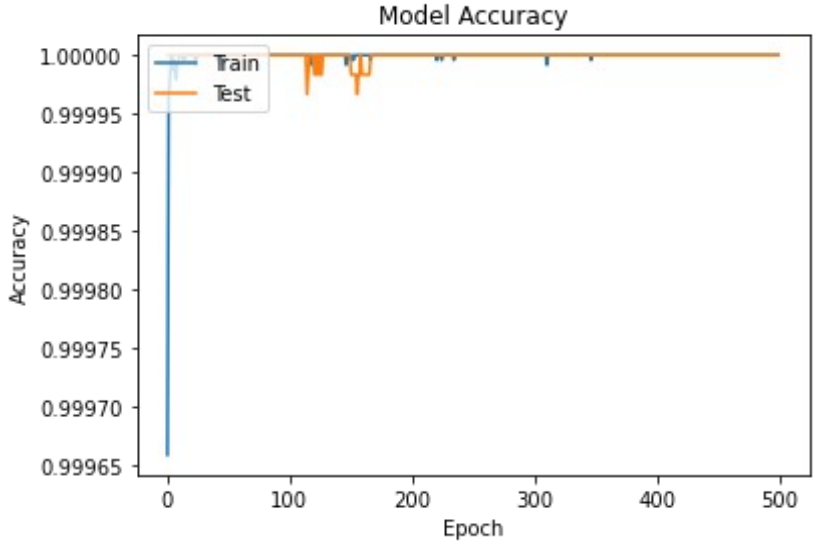

Figure 5.38: Model accuracy of an LSTM with 0.5 dropout in first layer only

Presented in **Figure 5.39** are the training and validation loss curves of the LSTM classifier when the dropout rate was increased to 0.5 in the first layer. From the graph, it can be seen that there is a bit of unsteadiness in the train loss during the first 300 epochs. The validation loss on the other hand demonstrates this unsteadiness only after 100 epochs and stability in the validation loss is gained after 200 epochs. Moreover, it can be observed that the model completely converges after roughly 300 training epochs. From this observation, it can be concluded that 300 epochs are sufficient to make the model converge. However, when compared to applying a dropout of 0.2 as demonstrated in **Figure 5.35**, increasing the dropout to 0.5 dropped the model's convergence speed.

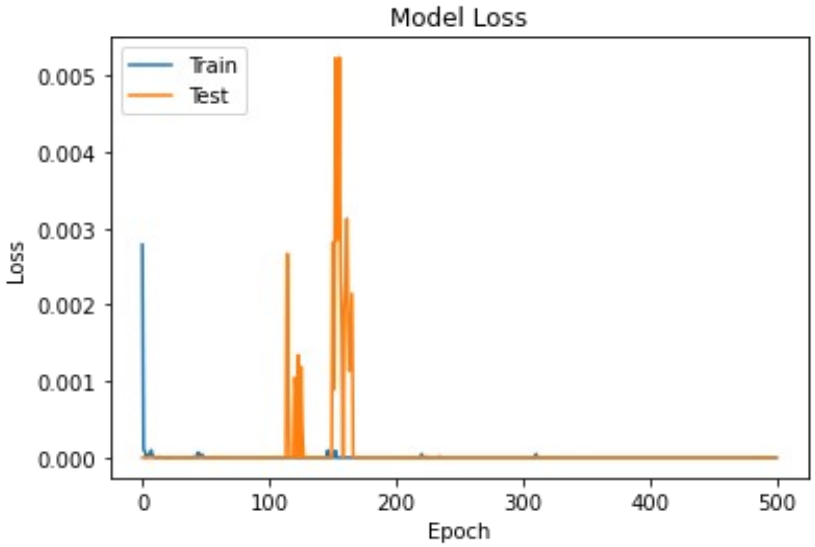

Figure 5.39: Training vs validation loss of the LSTM with 0.5 dropout in first layer only

In **Figure 5.40** a confusion matrix that demonstrates how the model performs on the validation set is presented. From the figure, it can be observed that the model correctly predicted all 100 000 packets in the validation set leading to the model obtaining a 100% detection rate in the validation set.

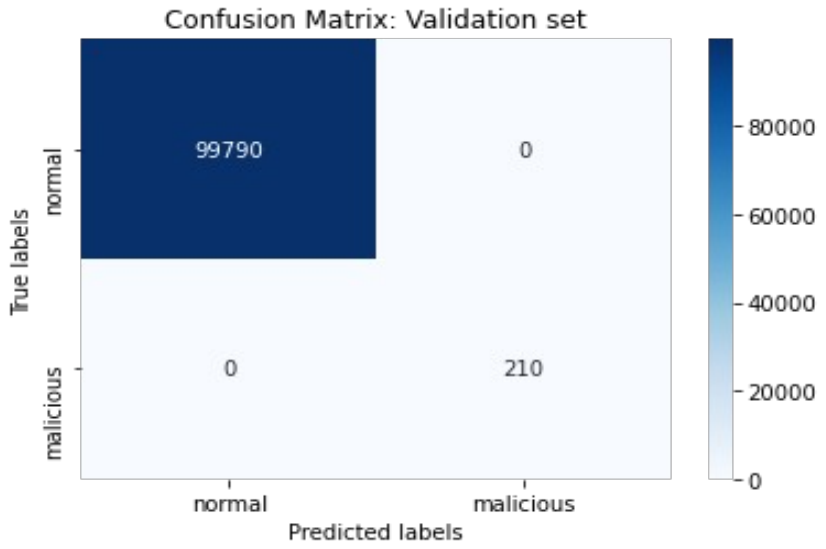

Figure 5.40: Validation set confusion matrix – LSTM with 0.5 dropout in first layer only

Figure 5.41 visually displays the performance results of the LSTM classifier on the test set when the dropout rate is raised to 0.5 in the first layer. Again, the model correctly predicted all 100 000 packets in the test set. From these results, it can be established that the model is robust enough to detect malicious packets with great accuracy.

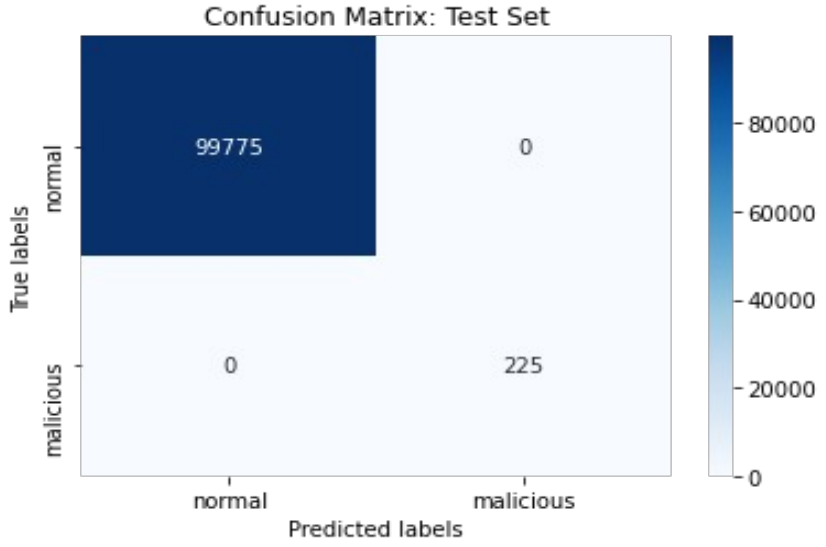

Figure 5.41: Test set confusion matrix – LSTM with 0.5 dropout in first layer only

To understand how the dropout implementation strategy impacts the model performance, varying dropout rates were applied in the different defined layers and the performance of the model was evaluated. The results of the experiments are demonstrated in **Figures 5.42** to **5.49**.

Figure 5.42 depicts the accuracy of the LSTM classifier when a 0.5 dropout rate is applied in the first and second layers of the LSTM. It can be observed that the classification accuracy is consistently 100% during the first 300 training iterations, plunges slightly to 99% at epoch 300 but stabilises eventually. This could indicate that the model started overfitting at the 300th iteration of the training process.

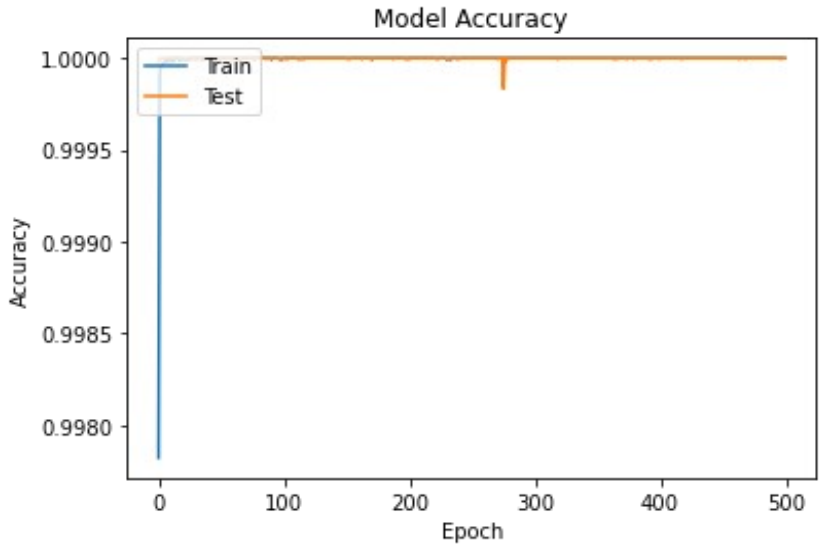

Figure 5.42: Model accuracy of an LSTM with 0.5 dropout in first and second and layers

In **Figure 5.43**, the training and validation loss of the LSTM with a 0.5 dropout rate in the first two layers is presented. Looking at the figure, a negligible instability in the train loss is observed until after roughly 490 epochs. On the validation, the loss is 0 and an increase of 0.04 is observed at the 300th epoch. The validation loss steadies after 300 epochs while the train loss remains until after 490 epochs. This observation confirms the overfitting and drop in accuracy observed in **Figure 5.42**. It can be seen that the model indeed started overfitting the data as the loss started increasing but the model converged again thereafter.

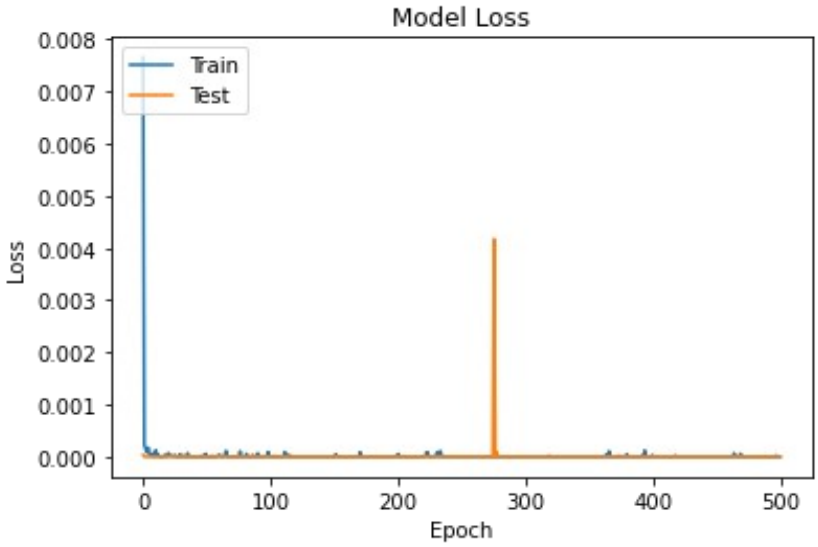

Figure 5.43: Training vs validation loss of the LSTM with 0.5 dropout in first and second layers

Figure 5.44 presents a confusion matrix that displays the performance of the LSTM with a 0.5 dropout rate in the first two layers of the LSTM. The model demonstrated the capability to predict the status of the packets as it classified all 100 000 packets in the validation set accurately. This implies that the LSTM classifier was able to classify the validation set correctly with 100% accuracy.

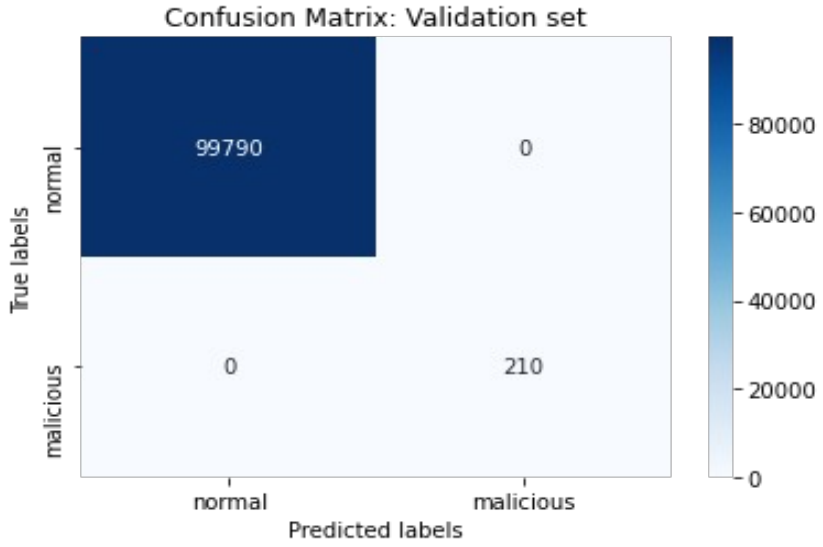

Figure 5.44: Validation set confusion matrix – LSTM with 0.5 dropout in first and second layers

Presented in **Figure 5.45** is a confusion matrix that shows the performance of the model on the second fold of the cross-validation. From the figure, it can be seen that the model

misclassified two malicious packets as normal while 99 998 packets were classified correctly. This implies a 0.008% drop in detection rates when compared to how the model performed on the validation set during the first fold of the cross-validation. This implies that the overfitting observed in **Figure 5.42** and **Figure 5.43** did have an impact on the performance of the classifier to a small degree.

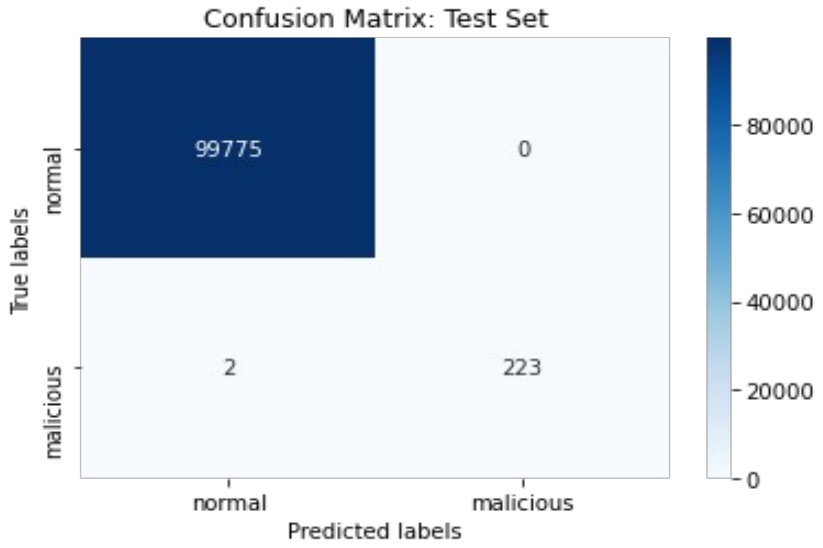

Figure 5.45: Test set confusion matrix – LSTM with 0.5 dropout in first and second layers

The last experiment was applying a 0.5 dropout rate in all the three defined layers of the LSTM. **Figures 5.46** to **5.49** depict the evaluation results of the LSTM. In **Figure 5.46** the accuracy of the model with the contemporary dropout strategy is depicted. The results indicate that the model maintained 100% accuracy throughout the training process.

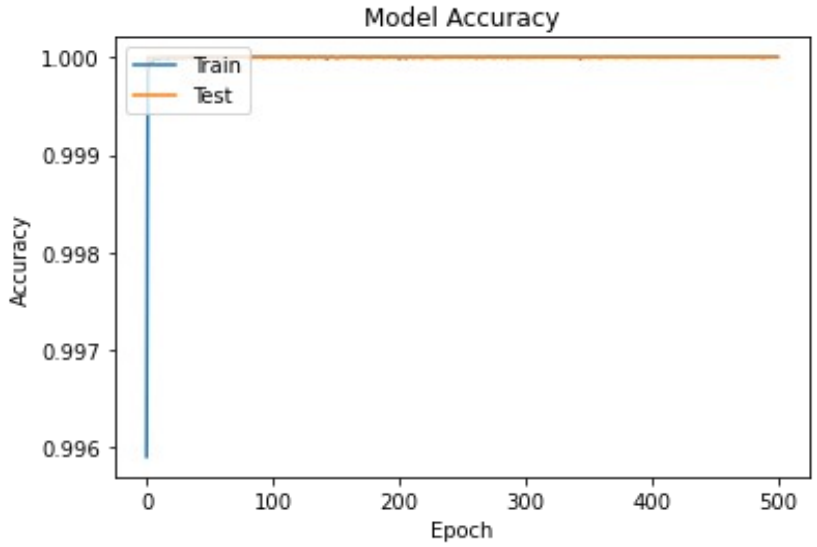

Figure 5.46: Model accuracy of an LSTM with 0.5 dropout in all three layers

In **Figure 5.47**, the training and validation loss curves of the LSTM with a dropout rate of 0.5 in all the network layers are presented. Even though not so significant, some variability can be observed in the train loss curve. On the loss curve, however, the loss decreases instantly after the model starts training. The low loss in both the train and validation loss curves is indicative of a model that can generalise even though a negligent delay in convergence is clear in the train loss curve.

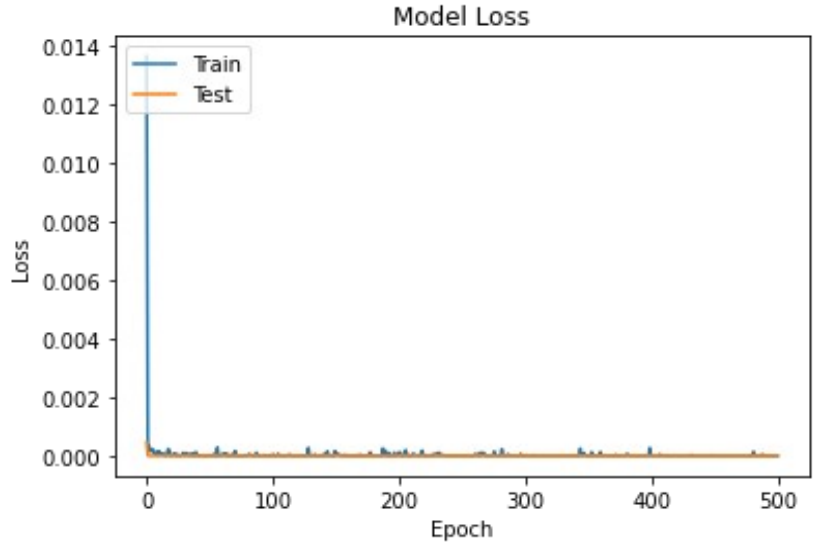

Figure 5.47: Training vs validation loss of the LSTM with 0.5 dropout in all three layers

Figure 5.48 presents a validation set confusion matrix that depicts the classification accuracy when the dropout strategy is such that a dropout rate of 0.5 is applied in all three layers of the

LSTM. With this strategy and as demonstrated in the confusion matrix, the model was able to predict all 100 000 packet traces accurately. As can be seen in the figure, the results indicate that the model was able to classify the status of the transmitted packets with 100% accuracy.

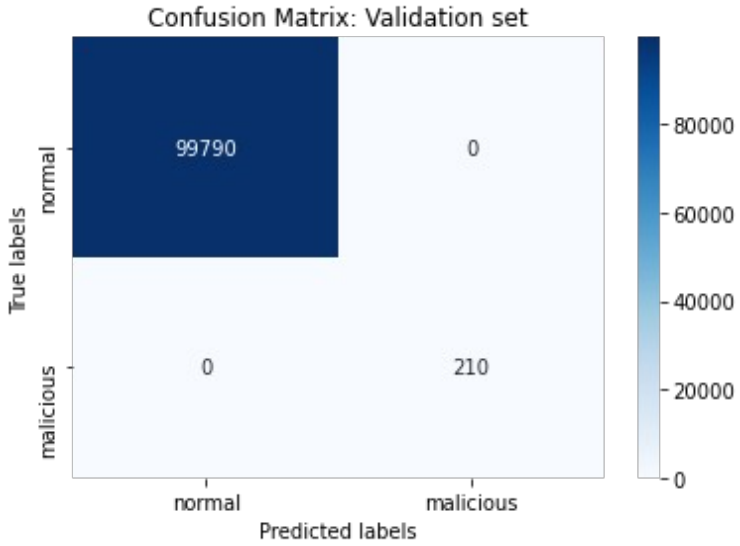

Figure 5.48: Validation set confusion matrix – LSTM with 0.5 dropout in all three layers

In **Figure 5.49,** the test set confusion matrix of the LSTM with the 0.5 dropout rate in all the layers strategy is illustrated. Once again, the model predicted all the 100 000 packets in the test set correctly. Regarding the results demonstrated in the confusion matrix, there is a clear indication that applying a dropout of 0.5 in all three layers defined in the LSTM greatly elevated the performance and the generalisation ability of the model.

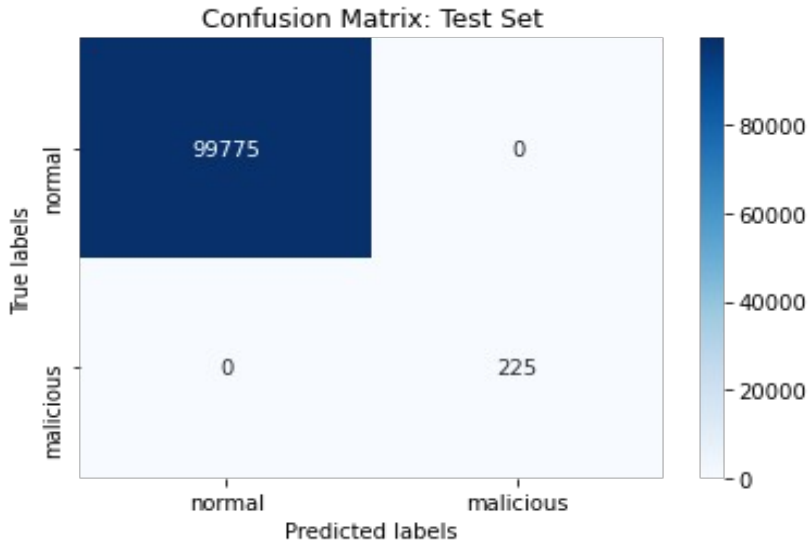

Figure 5.49: Test set confusion matrix – LSTM with 0.5 dropout in all three layers

The best performing LSTM model was converted to a lightweight version of the trained model and the performance of the lightweight model was again evaluated on unseen data. As demonstrated in **Figure 5.50**, the results strongly suggest that the lightweight version preserved the performance of the regular LSTM model as it maintained the 100% detection rate obtained by the regular LSTM model.

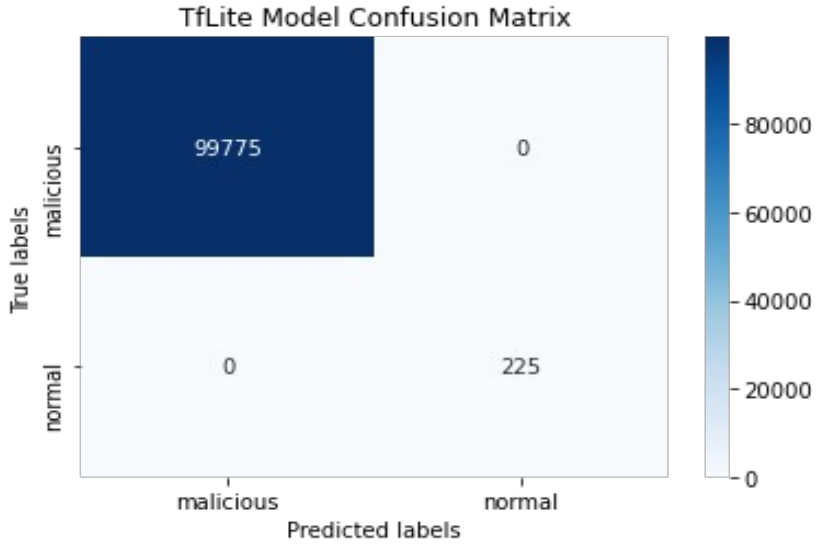

Figure 5.50: LSTM lightweight version confusion matrix

5.3 Discussion and model evaluation

This study commenced by surveying cutting-edge intrusion detection frameworks. The suitability of the surveyed IDS four (4) metrics – true positive rates (TPR), false positive rates (FPR), computational overheads that may be induced by the IDS and the dataset used to train the model – were evaluated. Unlike in conventional networks where TPR and FPR are sufficient to evaluate the suitability of an IDS, in resource-constrained networks such as smart city ecosystems, metrics that evaluate energy consumption and computational overheads introduced by the detection frameworks are essential to determine whether the proposed IDS technique is suitable for deployment in resource-constrained architectures.

Additionally, since smart city networks are heterogeneous and extremely dynamic, the dataset used for training the model must reflect heterogeneity and dynamicity. Hence, the suitability of each of the proposed IDSs was also evaluated on the dataset used for the development and evaluation of the proposed framework. Accordingly, the analysis was focused on the relevance of the dataset used for training the proposed model and the ability of

the model to exhibit high detection rates with minimal FPR. Since the proposed models were designed for deployment in WSNs, it was also essential to evaluate the ability of the model to perform well without introducing computational overheads.

The results of the survey revealed that most studies reviewed were not mindful of the dynamic characterisation and resource limitations of IoT infrastructures such as smart city ecosystems. Some notable shortcomings of the proposed frameworks include the use of amiss datasets that contain antiquated attack vectors for training the proposed detection models and insubstantial evidence of the elimination of computational overheads. To address the identified limitations of the surveyed frameworks, this study trained and evaluated two candidate models to propose the model that demonstrated the highest predictive power.

In intrusion detection, a model is considered effective if it demonstrates the capability to detect the status/nature of network packets with high detection but low false alarm rates. The experimental results have demonstrated that regularisation plays a vital role in augmenting the detection and generalisation ability of both the CNN and LSTM models.

With CNN, it was observed that applying a regularisation – adding dropout in only one input layer – was not sufficient to optimise the generalisation ability of the model. Thus, adding a dropout of at least 0.5 in all input layers significantly improved the model's ability to generalise. This observation is demonstrated in **Figure 5.23** and **Figure 5.24**. Moreover, a comparative analysis between the performance of the CNN and the performance of the LSTM models was orchestrated. When comparing the performance of the two models, it could be observed that a dropout of just 0.1 in the first input layer was sufficient to make the LSTM model converge more quickly and produce remarkable classification results – high detection rates and low false alarm rates. The indicated efficiency of the LSTM at each regularisation iteration is demonstrated in **Figures 5.26** to **5.49**. Regarding **Figure 5.47**, it is evident that the LSTM model converges much more quickly when regularisation is applied to all the input layers.

However, in **Figures 5.2**, **5.6**, **5.10**, **5.14** and **5.18**, the loss fluctuates which is indicative of overfitting. This overfitting is clear in the classification accuracy as well as the proportion of false alarm rates. This observation is presented in **Figures 5.3**, **5.4**, **5.7**, **5.8**, **5.11**, **5.12**, **5.15**, **5.16**, **5.19**, **5.20**, **5.29**, **5.37** and **5.45**. In the said figures, it can be observed that the classification accuracy drops when the model is applied to unseen data. The most common scenario is that the false alarm rates are notably higher when the model classifies the test dataset.

Regarding the evaluation results, it can be concluded that the implementation strategy of the regularisation technique is a remarkable factor in the behaviour of the model and its robustness.

Furthermore, one of the study's objectives was to propose a DL model as an intrusion detection mechanism for smart city ecosystems. Subsequently, as an attempt to reduce computational overheads that may be introduced by the deployment of trained intrusion detection models, the models that attained high predictive power were saved and converted to lightweight versions using TensorFlow-lite.

The sizes of both models shrank remarkably after the conversion. The memory size of the CNN model reduced from 859 kilobytes to 269 kilobytes and the LSTM reduced from 2.0 megabytes to 683 kilobytes. This means that TensorFlow-lite more than halves the memory size of a model. This should then result in reduced computational overheads. The lightweight versions were then tested using the test dataset that was used to evaluate the regular models. The experiments have exposed that the lightweight versions preserve the performance of the original models. This observation is demonstrated in **Figures 5.25** and **5.50.**

5.4 Summary

This chapter gave an overview and discussed the findings of the study. Additionally, a rigorous comparative analysis of the baseline and the proposed models was performed. The comparison was mainly focused on the convergence speed of the models, generalisation ability, detection and false alarm rates.

Chapter 6: Conclusion

6.1 Introduction

This chapter concludes the study. The chapter presents the empirical findings of the study and outlines how the research objectives were achieved. The chapter further presents the implications of the study and concludes by highlighting the potential for future studies. The study aims to review the current state of intrusion detection in smart city ecosystems and proposes a DL model tailored for smart city ecosystems.

6.2 Summary of the dissertation

This section provides a detailed summary of the study. The section presents the empirical findings of this study, research objectives and the future work is highlighted.

6.2.1 Empirical findings

Connectivity and automation are indispensable aspects of the realisation of sustainable smart cities. This high connectivity and automation to facilitate resource monitoring in cities are achievable with mobile applications, Wi-Fi, AI and cloud-based architectures. Thus, smart city components are connected through wireless networks and generate data that may be used intelligently for city infrastructure management and monitoring. Even though digitisation places smart cities in an excellent position to offer an intelligent, spatial and economic competitive advantage, it renders the cities susceptible to security issues. Several studies have proved that despite numerous cryptography-based security mechanisms that have been continuously proposed and implemented, IoT networks remain exposed to cyber-attacks. Considering this, the security of smart cities is an area that undeniably needs the utmost attention.

Numerous studies proposed intrusion detection that can be used to alleviate cyber-attacks as the most potent security mechanism for resource-constrained smart city networks. This study reviewed IDSs that have been previously proposed for resource-constrained environments.

The review investigated the features of the proposed intrusion detection frameworks and their suitability for deployment in resource-constrained and dynamic environments. This study identified that many studies are not mindful of resource constraints and the dynamic nature of IoT-based networks such as smart city networks. Furthermore, more investigations on the design of proper IDSs for smart cities still need to be orchestrated.

This study proposed a lightweight binary classification NN. The proposed model is lightweight and proffers high detection and low false alarm rates. Additionally, the results of

the study's experiments strongly suggest that using dropout as a regularisation technique boosts the performance of a DL model tremendously. The results further demonstrated that achieving the best results depends highly on the strategy by which the dropout is implemented. Moreover, the study discovered that DL models can be converted into lighter versions without detracting from the performance standard of non-lightweight DL models.

6.2.2 Research objectives

- *To analyse cyber-security challenges in smart city ecosystems*

The study presented cyber-security issues prevalent in smart city ecosystems. Advances in technology present countless conveniences that, among others, include the automation of cities to effectuate smart cities. Smart cities offer plenty of appealing benefits such as the use of IoT sensors to gather useful data on power demand to prevent power blackouts as well as the average speed of vehicles to mitigate traffic congestion on roads. The reviewed work revealed a wide variety of security challenges that threaten the safety and sustainability of smart cities with DoS attacks discerned as the most notorious cyber-attacks that threaten this sustainability.

- *To identify state-of-the-art intrusion detection methods used in smart city ecosystems*

State-of-the-art intrusion detection frameworks were surveyed. The results of the survey conveyed that most frameworks were developed with less consideration for the dynamic characterisation and resource limitations of IoT infrastructures such as smart city ecosystems.

- *To propose a deep learning model as an intrusion detection mechanism for smart city ecosystems*

A CNN binary classifier and an LSTM binary classifier were implemented as candidate models. The most robust model, which offers a relatively higher convergence speed and high detection rates, was converted to a lightweight version. The conversion was performed to make the model lighter and thus more suitable for deployment in resource-constrained environments. The results of the empirical study demonstrated that the LSTM binary classifier proffers the best performance. Therefore, this classifier is proposed as the most suitable model for intrusion detection in smart city ecosystems.

The robustness demonstrated by the LSTM classifier is in accordance with the results of an earlier study by Althubiti et al. (2018) that investigated the applicability of LSTM RNNs in modelling network intrusion detection. However, their model was trained using the KDD 99 cup dataset which, in the view of the researcher, is an inappropriate dataset for training models tailored for smart city environments. Additionally, their study did not present

sufficient evidence of whether the produced model was lightweight enough to operate in resource-constrained environments.

- *To evaluate the efficiency of the proposed deep learning model*

The efficiency of the proposed model was evaluated. The proposed model obtained a 100% detection rate and occupied only 683 kilobytes of memory. The model demonstrated high detection rates with low false alarm and false negative rates. This implies that the model is robust enough to detect cyber intrusions and the memory size of the model indicates that the model is lightweight enough to be deployed in resource-constrained environments.

6.3 Implications of the study

Security is crucial for the sustainability, availability, confidentiality, and integrity of smart city ecosystems. To give direction for future research, this study provides an overview of the current state of intrusion detection mechanisms available for smart city ecosystems. The study further demonstrates that current intrusion detection mechanisms can be improved. This can be achieved by developing more robust and lightweight models that offer high detection rates and minimal false alarm rates to prevent security risks in smart city ecosystems and to ensure sustainable and safe smart cities.

6.4 Future studies

Although the proposed model demonstrated remarkable robustness, it can currently only detect two types of DoS attacks – UDP and TCP flooding attacks. Future studies should delve into introducing more attack vectors to assess the adaptability of the proposed model. Additionally, the effectiveness of the proposed model should be scrutinised by deploying the model on a resource-constrained device (such as Raspberry-pi) and monitoring the computational overheads introduced when the model is in production.

References

Acaps. (2012). *Qualitative and quantitative research techniques for humanitarian needs assessment*. United Nations Office for the Coordination of Humanitarian Affairs (OCHA), New York, USA. Retrieved from http://reliefweb.int/report/world/qualitative-and-quantitative-research-techniques-humanitarian-needs-assessment

Agarap, A. F. (2018). Deep learning using rectified linear units (ReLU), *1*, 2-8. Retrieved from http://arxiv.org/abs/1803.08375

Aldaej, A. (2019). Enhancing cyber security in modern internet of things (IoT) using intrusion prevention algorithm for IoT (IPAI). *IEEE Access*, 1-1. https://doi.org/10.1109/access.2019.2893445

AlDairi, A., & Tawalbeh, L. (2017). Cyber security attacks on smart cities and associated mobile technologies. *Procedia Computer Science*, *109*(2017), 1086-1091. https://doi.org/10.1016/j.procs.2017.05.391

Alguliyev, R. M., Aliguliyev, R. M., & Abdullayeva, F. J. (2019). The improved LSTM and CNN models for DDoS attacks prediction in social media. *International Journal of Cyber Warfare and Terrorism*, *9*(1), 1-18. https://doi.org/10.4018/IJCWT.2019010101

Aloqaily, M., Otoum, S., Al Ridhawi, I., & Jararweh, Y. (2019). An intrusion detection system for connected vehicles in smart cities. *Ad Hoc Networks*, *90*. https://doi.org/10.1016/j.adhoc.2019.02.001

Althubiti, S., Nick, W., Mason, J., Yuan, X., & Esterline, A. (2018, April 19-22). *Applying long short-term memory recurrent neural network for intrusion detection*. Paper presented at the IEEE SoutheastCon in St. Petersburg, Florida, USA. Retrieved from https://doi.org/10.1109/SECON.2018.8478898

Anthi, E., Williams, L., Slowińska, M., Theodorakopoulos, G., & Burnap, P. (2019). A supervised intrusion detection system for smart home IoT devices. *IEEE Internet of Things Journal*, *6*(5), 9042-9053. https://doi.org/10.1109/JIOT.2019.2926365

Apuke, O. D. (2017). Quantitative research methods a synopsis approach. *Arabian Journal of Business and Management Review (Kuwait Chapter)*, *6*(10). https://doi.org/ 10.12816/0040336

Aris, A., & Oktug, S. F. (2017, February). *Poster: State of the art IDS design for IoT*. Paper presented at the 2017 International Conference on Embedded Wireless Systems and Networks (EWSN), Uppsala, Sweden (pp. 196-197).

Arshad, J., Azad, M. A., Salah, K., Jie, W., Iqbal, R., & Alazab, M. (2018). *A review of performance, energy and privacy of intrusion detection systems for IoT*. Retrieved from http://arxiv.org/abs/1812.09160

Atieno, O. P. (2009). An analysis of the strengths and limitation of qualitative and quantitative research paradigms. *Problems of Education in the 21st Century*, *13*. Retrieved from http://www.scientiasocialis.lt/pec/node/files/pdf/Atieno_Vol.13.pdf

Azahari Mohd Yusof, M., Hani Mohd Ali, F., & Yusof Darus, M. (2018). Detection and defense algorithms of different types of DDoS attacks. *International Journal of Engineering and Technology*, *9*(5), 410-444. https://doi.org/10.7763/ijet.2017.v9.1008

Azpiazu, I. M., & Pera, M. S. (2019). Multiattentive recurrent neural network architecture for multilingual readability assessment. *Transactions of the Association for Computational Linguistics*, *7*, 421-436. https://doi.org/10.1162/tacl_a_00278

Bach, A. A. (2018). *WORD2VEC embeddings for playlist recommendation*. Facultat de Matem`atiques i Inform`atica, Universitat de Barcelona. Retrieved from http://diposit.ub.edu/dspace/bitstream/2445/130481/3/memoria.pdf

Baker, S. B., Xiang, W., & Atkinson, I. (2017). Internet of things for smart healthcare: Technologies, challenges, and opportunities. *IEEE Access*, *5*, 26521-26544. https://doi.org/10.1109/ACCESS.2017.2775180

Banerjee, M., Lee, J., & Choo, K. R. (2018). A blockchain future for internet of things security: A position paper. *Digital Communications and Networks*, *4*(3), 149-160. https://doi.org/10.1016/j.dcan.2017.10.006

Bruneo, D., Distefano, S., Giacobbe, M., Minnolo, A., Longo, F., Merlino, G., … Tapas, N. (2019). An IoT service ecosystem for smart cities: The #SmartME project. *Internet of Things*, *5*, 12-33. https://doi.org/10.1016/j.iot.2018.11.004

Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16-28. https://doi.org/10.1016/ j.compeleceng.2013.11.024

Chen, C., Hasan, M., & Mohan, S. (2018). Securing real-time internet-of-things. *Sensors*, *18*(12), 4356. https://doi.org/10.3390/s18124356

Conti, M. (Ed.) (2018). *Versatile cybersecurity*. New York, NY: Springer. https://doi.org/ 10.1007/978-3-319-97643-3

Cui, L., Xie, G., Qu, Y., Gao, L., & Yang, Y. (2018). Security and Privacy in Smart Cities: Challenges and Opportunities. IEEE Access, 6, 46134-46145. doi: 10.1109/access.2018.2853985

Daniel, E. (2016). The usefulness of qualitative and quantitative approaches and methods in researching problem-solving ability in science education curriculum. *Journal of Education and Practice*, *7*(15), 91-100. https://eric.ed.gov/?id=EJ1103224

Doshi, R., Apthorpe, N., & Feamster, N. (2018, May 24). *Machine learning DDoS detection for consumer internet of things devices*. A paper presented at the 2018 IEEE

Symposium on Security and Privacy Workshops, San Francisco, California, USA (pp. 29-35). https://doi.org/10.1109/SPW.2018.00013

Elleithy, K., & Blagovic, D. (2006). Denial of service attack techniques: Analysis, implementation and comparison. *Systemics, Cybernetics and Informatics*, *3*(1), 66-71. Retrieved from http://www.iiisci.org/Journal/CV$/sci/pdfs/P129065.pdf

Elmaghraby, A. S., & Losavio, M. M. (2014). Cyber security challenges in smart cities: Safety, security and privacy. *Journal of Advanced Research*, *5*(4), 491-497. https://doi.org/10.1016/j.jare.2014.02.006

Elrawy, M. F., Awad, A. I., & Hamed, H. F. A. (2018). Intrusion detection systems for IoT-based smart environments: A survey. *Journal of Cloud Computing*, *7*(21). https:// doi.org/10.1186/s13677-018-0123-6

Erickson, B. J., Korfiatis, P., Kline, T. L., Akkus, Z., Philbrick, K., & Weston, A. D. (2018). Deep learning in radiology: Does one size fit all? *Journal of the American College of Radiology*, *15*(3), 521-526. https://doi.org/10.1016/j.jacr.2017.12.027

Faghri, A., Martinelli, D., & Demetsky, M. J. (1997). Neural network applications in transportation engineering. *Artificial Neural Networks for Civil Engineers: Fundamentals and Applications*, *1*, 137-159.

Fazio, M., Celesti, A., Puliafito, A., & Villari, M. (2015). Big data storage in the cloud for smart environment monitoring. *Procedia Computer Science*, *52*, 500-506. https://doi.org/10.1016/j.procs.2015.05.023

Ferraz, F. S., & Ferraz, C. A. G. (2014). Smart City Security Issues: Depicting Information Security Issues in the Role of an Urban Environment. Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014. https://doi.org/10.1109/UCC.2014.137

Garcia-Font, V., Garrigues, C., & Rifà-Pous, H. (2016). A comparative study of anomaly detection techniques for smart city wireless sensor networks. *Sensors*, *16*(6), 868. https://doi.org/10.3390/s16060868

Hasan, M., Islam, M. M., Zarif, M. I. I., & Hashem, M. M. A. (2019). Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. *Internet of Things*, *7*, 100059. https://doi.org/10.1016/j.iot.2019.100059

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hodo, E., Bellekens, X., Hamilton, A., Dubouilh, P. L., Iorkyase, E., Tachtatzis, C., & Atkinson, R. (2016). *Threat analysis of IoT networks using artificial neural network intrusion detection system*. Paper presented at the 2016 International Symposium on Networks, Computers and Communications, Hammamet, Tunisia (pp. 4-8). https://arxiv.org/ftp/arxiv/papers/1704/1704.02286.pdf

Hu, L., & He, Z. (2001). Neural network-based intrusion detection systems. Proceedings of the Sixth International Conference for You Computer Scientist: In Computer Science and Technology in New Century, 106(18), 296–298. https://doi.org/10.5120/18705-9636

Jabbar, M. A., & Aluvalu, R. (2018, April 22-23). *Intrusion detection system for the internet of things: A review*. A paper presented at the Smart Cities Symposium, Bahrain, Bahrain.

Jaleesha, B. K., & Ezhil, S. S. (2019). A performance on simulation with methodologies. *International Journal of Innovative Technology and Exploring Engineering*, *8*(7), 1166-1170.

Joshi, S., Saxena, S., Goodbole, T., & Shreya. (2016). Developing smart cities: An integrated framework. *Procedia Computer Science*, *93*, 902-909. https://doi.org/10.1016/j.procs.2016.07.258

Jucevičius, R., Patašienė, I., & Patašius, M. (2014). Digital dimension of smart city: Critical analysis. *Procedia – Social and Behavioral Sciences, 156*(April), 146-150. https://doi.org/10.1016/j.sbspro.2014.11.137

Khan, R. U., Xiaosong, Z., Alazab, & Kumar, R. (2018). An improved convolutional neural network model for intrusion detection in networks. Retrieved from https://easychair.org/publications/preprint/bj3s.

Khajenasiri, I., Estebsari, A., Verhelst, M., & Gielen, G. (2017). A Review on Internet of Things Solutions for Intelligent Energy Control in Buildings for Smart City Applications. Energy Procedia, 111(September 2016), 770–779. https://doi.org/10.1016/j.egypro.2017.03.239

Krenker, A., Bešter, J., & Kos, A. (2011). Introduction to the artificial neural networks. In K. Suzuki (Ed), *Artificial neural networks: Methodological Advances and biomedical applications* (pp. 1-18). Retrieved from https://doi.org/10.5772/15751

Krimmling, J., & Peter, S. (2014). *Integration and evaluation of intrusion detection for CoAP in smart city applications*. A paper presented at the 2014 IEEE Conference on Communications and Network Security, San Francisco, California, USA (pp. 73-78). https://doi.org/10.1109/CNS.2014.6997468

Liang, X. (2017). *20: Convolutional and recurrent neural networks*. Retrieved from https://www.cs.cmu.edu/~epxing/Class/10708-17/notes-17/10708-scribe-lecture20.pdf.

Mirsky, Y., Doitshman, T., Elovici, Y., & Shabtai, A. (2018). *Kitsune: An ensemble of autoencoders for online network intrusion detection*. Ben-Gurion University of the Negev, Beersheba, Israel. Retrieved from https://doi.org/10.14722/ndss.2018.23204

Mohamad Noor, M., & Hassan, W. H. (2019). Current research on internet of things (IoT) security: A survey. *Computer Networks*, *148*, 283-294. https://doi.org/10.1016/j.comnet.2018.11.025

Moradi, M., & Zulkernine, M. (2004, November 15-18). *A neural network based system for intrusion detection and classification of attacks*. A paper presented at the 2004 IEEE International Conference on Advances in Intelligent Systems – Theory and Applications, Luxembourg. Retrieved from https://www.researchgate.net/publication/236030027_A_Neural_Network_Based_System_for_Intrusion_Detection_and_Classification_of_Attacks

Naoum, R. S., Abid, N. A., & Al-Sultani, Z. N. (2012). An enhanced resilient backpropagation artificial neural network for intrusion detection system. *International Journal of Computer Science and Network Security*, *12*(3), 11-16.

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. Retrieved from http://arxiv.org/abs/1811.03378

Olaverri-Monreal, C. (2016). Intelligent technologies for mobility in smart cities. *Hiradastechnika Journal*, *71*, 29-34.

Otuoze, A. O., Mustafa, M. W., & Larik, R. M. (2018). Smart grids security challenges: Classification by sources of threats. *Journal of Electrical Systems and Information Technology*, *5*(3), 468-483. https://doi.org/10.1016/j.jesit.2018.01.001

Pacheco, J., & Hariri, S. (2016). IoT Security Framework for Smart Cyber Infrastructures. *2016 IEEE 1St International Workshops On Foundations And Applications Of Self* Systems (FAS*W)*. doi: 10.1109/fas-w.2016.58

Papamartzivanos, D., Gómez Mármol, F., & Kambourakis, G. (2019). Introducing deep learning self-adaptive misuse network intrusion detection systems. *IEEE Access*, *7*, 13546-13560. https://doi.org/10.1109/ACCESS.2019.2893871

Pereira, T., Barreto, L., & Amaral, A. (2017). Network and information security challenges within Industry 4.0 paradigm. *Procedia Manufacturing*, *13*, 1253-1260. https://doi.org/10.1016/j.promfg.2017.09.047

Popescul, D., & Radu, L. D. (2016). Data security in smart cities: Challenges and solutions. *Informatica Economică*, *20*(1). https://doi.org/10.12948/issn14531305/20.1.2016.03

Renals, S. (2015). *Multi-layer neural networks*. Retrieved from https://www.inf.ed.ac.uk/teaching/courses/asr/2014-15/asr07-nnDetails.pdf

Rizvi, S., Willet, J., Perino, D., Marasco, S., & Condo, C. (2017). A threat to vehicular cyber security and the urgency for correction. *Procedia Computer Science*, *114*, 100-105. https://doi.org/10.1016/j.procs.2017.09.021

Rosindell, J., & Wong, Y. (2018). Biodiversity, the tree of life, and science communication. In R. Scherson & D. Faith (Eds.), *Phylogenetic diversity: Applications and challenges in biodiversity science* (pp. 41-71). Cham, Switzerland: Springer. https://doi.org/10.1007/978-3-319-93145-6_3

Roux, J., Alata, E., Auriol, G., Nicomette, V., & Kaâniche, M. (2017, September). *Toward an intrusion detection approach for IoT based on radio communications profiling*. A paper presented at the 13th European Dependable Computing Conference, Geneva, Switzerland.

Saifuzzaman, M., Khan, A. H., Moon, N. N., & Nur, F. N. (2017). Smart security for an organization based on IoT. *International Journal of Computer Applications*, *165*(10), 33-38. https://doi.org/10.5120/ijca2017913982

Sammany, M., Sharawi, M., El-Beltagy, M., & Saroit, I. (2007). *Artificial neural networks architecture for intrusion detection systems and classification of attacks.* A paper presented at the 5th International Conference INFO2007. Retrieved from http://infos2007.fci.cu.edu.eg/Computational Intelligence e/07177.pdf

Santos, L., Rabadao, C., & Gonçalves, R. (2018, June). *Intrusion detection systems in internet of things: A literature review.* A paper presented at the 5th Iberian Conference on Information Systems and Technologies, Caceres, Spain (pp. 1-7). https://doi.org/10.23919/CISTI.2018.8399291

Shenfield, A., Day, D., & Ayesh, A. (2018). Intelligent intrusion detection systems using artificial neural networks. *ICT Express*, *4*(2), 95-99. https://doi.org/10.1016/j.icte.2018.04.003

Sherasiya, T., Upadhyay, H., & Patel, H. B. (2016). A survey: Intrusion detection system for internet of things. *International Journal of Computer Science and Engineering*, *5*(2), 91-98. Retrieved from http://oaji.net/articles/2016/1870-1456139720.pdf

Sherstinsky, A. (2018). *Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network*. Retrieved from http://arxiv.org/abs/1808.03314.

Shokuh Saljoughi, A., Mehrvarz, M., & Mirvaziri, H. (2018). Attacks and intrusion detection in cloud computing using neural networks and particle swarm optimization algorithms. *Emerging Science Journal*, *1*(4), 179-191. https://doi.org/10.28991/ijse-01120

Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*(1), 41-50. https://doi.org/10.1109/tetci.2017.2772792

Sibanda, W., & Pretorius, P. (2012). Artificial neural networks – a review of applications of neural networks in the modeling of HIV Epidemic. *International Journal of Computer Applications*, *44*(16), 1-5.

Thamilarasu, G., & Chawla, S. (2019). Towards deep-learning-driven intrusion detection for the internet of things. *Sensors*, *19*(9). https://doi.org/10.3390/s19091977

Thanigaivelan, N. K., Nigussie, E., Virtanen, S., & Isoaho, J. (2018). Hybrid internal anomaly detection system for IoT: Reactive nodes with cross-layer operation. *Security and Communication Networks*, *2018*. https://doi.org/10.1155/2018/3672698

Tuptuk, N., & Hailes, S. (2018). Security of smart manufacturing systems. *Journal of Manufacturing Systems*, *47*, 93-106. https://doi.org/10.1016/j.jmsy.2018.04.007

UNCTAD. (2016). *Smart cities and infrastructure.* Report of the Secretary-General of the Economic and Social Council. Retrieved from https://doi.org/10.1017/S0020818300006640

Venkatraman, S., & Surendiran, B. (2019). Adaptive hybrid intrusion detection system for crowd sourced multimedia internet of things systems. *Multimedia Tools and Applications*, *79*, 3993-4010. https://doi.org/10.1007/s11042-019-7495-6

Williamson, B. (2015). Educating the smart city: Schooling smart citizens through computational urbanism. *Big Data & Society*. https://doi.org/10.1177/2053951715617783

Wu, Z. Y., El-Maghraby, M., & Pathak, S. (2015). Applications of deep learning for smart water networks. *Procedia Engineering*, *119*, 479-485. https://doi.org/10.1016/j.proeng.2015.08.870

Wu, Z. Y., & Rahman, A. (2017). Optimized deep learning framework for water distribution data-driven modeling. *Procedia Engineering*, *186*, 261-268. https://doi.org/10.1016/j.proeng.2017.03.240

Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, *5*, 21954-21961. https://doi.org/10.1109/ACCESS.2017.2762418

Zarpelão, B. B., Miani, R. S., Kawakani, C. T., & de Alvarenga, S. C. (2017). A survey of intrusion detection in internet of things. *Journal of Network and Computer Applications*, *84*, 25-37. https://doi.org/10.1016/j.jnca.2017.02.009

Zekri, M., El Kafhali, S., Aboutabit, N., & Saadi, Y. (2018, October 24-26). *DDoS attack detection using machine learning techniques in cloud computing environments*. A paper presented at the 2017 3rd International Conference of Cloud Computing Technologies and Applications, (CloudTech), Rabat, Morocco (pp. 1-7). Retrieved from https://doi.org/10.1109/CloudTech.2017.8284731

Zhang, R., & Xiao, X. (2019). Intrusion detection in wireless sensor networks with an improved NSA based on space division. *Journal of Sensors*, *2019*(5451263). https://doi.org/10.1155/2019/5451263

Zhou, W., Zhang, Y., & Liu, P. (2019). The effect of IoT new features on security and privacy: New threats, existing solutions, and challenges yet to be solved. *IEEE Internet of Things Journal*, *6*(2), 1606-1616. https://arxiv.org/abs/1802.03110

Appendix A: Data Collection

This section presents the steps followed by this study for data gathering. The steps are explained below.

a) Downloading and installing the SUMO simulator

The SUMO simulator was downloaded from the official SUMO site: https://sumo.dlr.de/docs/Installing/index.html. Additionally, as a prerequisite to run the simulation, Python had to be installed. Python was downloaded from https://www.python.org/downloads/.

b) Downloading an OpenStreetMap of the city of interest

A map of a small area in residential Cape Town South Africa was downloaded from https://www.openstreetmap.org//. The process followed is demonstrated below:

i) Search for the city on the OpenStreetMap search bar (**Figure A1**).

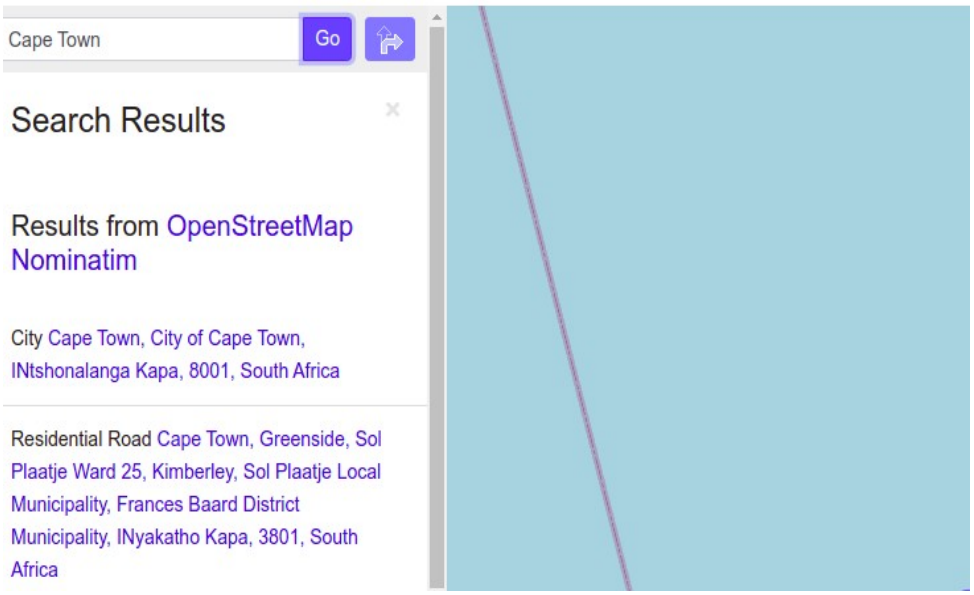

Figure A1: Downloading a city from OpenStreetMaps

ii) Select an area of interest from the search results and click export (**Figure A2**).

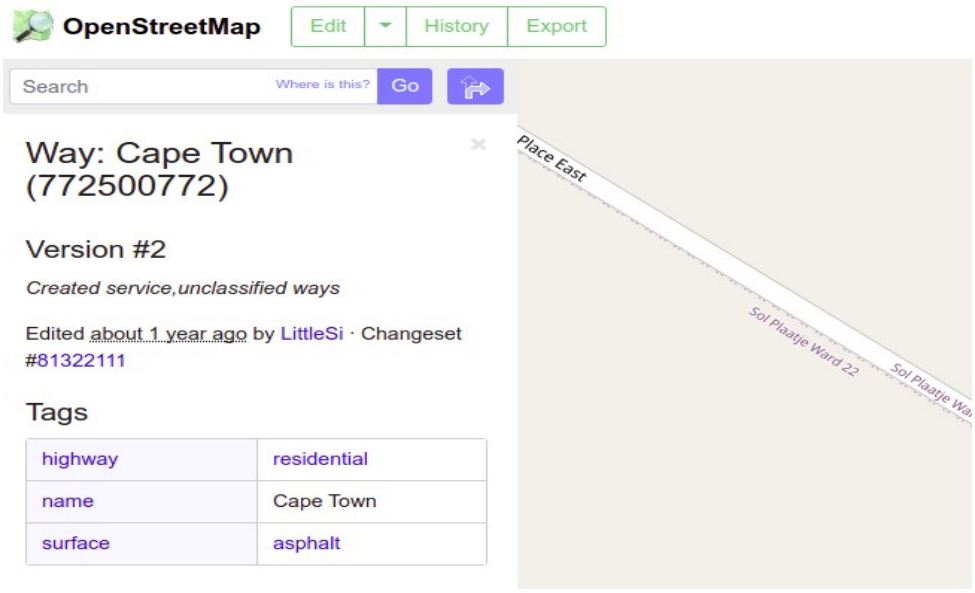

Figure A2: Exporting the desired area

c) Converting the map into a SUMO network

The downloaded file is stored in a .osm file extension. It must be noted that in this study all SUMO simulation files were named such that they contained 'capetown' before the file extension. To convert the downloaded map into a SUMO network, navigation to the folder containing the map in a command line terminal is undertaken and the command below is run:

netconvert capetown.osm -o capetown.net.xml

Running the command generates a network file. The generated network file contains node identifiers, node types and the priority of the nodes.

d) Adding trips and routes to the network

By default, SUMO comes with a randomTrips.py python file which is used to add trips and routes to the network. The output of this step is a route file with extension .rou. The route file comprises identifiers of the vehicles and their routes. This is achieved by running the following command in the command line terminal:

python randomTrips.py -n capetown.net.xml -r capetown.rou.xml -e 50 -l

e) Generating the SUMO configuration file

The configuration file takes the network file and the route file as input. The file contains the simulation start time and end time. For this study, the simulation time was set to 2000 seconds. The file has a.sumocfg extension. The extensible markup language (xml) script that creates the SUMO configuration file is presented in **Figure A3.**

```
<configuration>
<input>
<net-file value="capetown.net.xml"/>
<route-files value="capetown.rou.xml"/>
  <additional-files value="capetown.poly.xml"/>
</input>
<time>
<begin value="0"/>
<end value="2000"/>
<step-length value="1.0"/>
</time>
</configuration>
```

Figure A3: SUMO configuration file

The generated configuration file is used to run the simulation, either by double-clicking the configuration file or by using the following command in the command line terminal:

sumo-gui -c capetown.sumocfg

The simulation is shown in **Figure A4**.

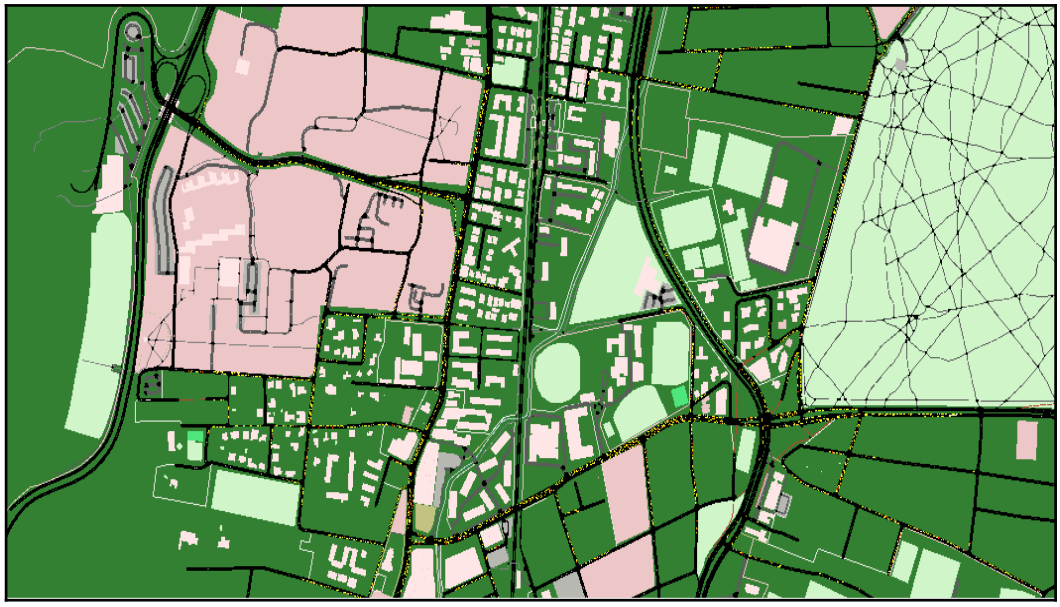

Figure A4: SUMO simulation

f) Creating a trace file of communicating nodes

The first step towards creating the trace file was to convert the entire simulation into a trace file of communicating nodes. The conversion produces the trace.mxl file (in this case, capetown.xml file). This is achieved by running the next command:

sumo -c capetown.sumocfg --fcd-output capetwon.xml

g) Generation of an NS2 mobility file

This step produces an NS2 mobility file with a .tcl file extension and is achieved by running a traceExporter.py file provided by SUMO with the following command:

python traceExporter.py capetown.xml --ns2mobility-output=ns2mobility.tcl

h) Importing an NS2 mobility file into NS3

As a prerequisite to run this step, NS3 was downloaded from https://www.nsnam.org/releases/ns-3-29/download/.

i) Configuration of a communication scenario in NS3

Upon the complete installation of NS3, a folder with a name that follows the pattern '*ns3-allinone-version*' (for example, ns3-allinone-3.29) is automatically created. The folder structure is illustrated in **Table A1**.

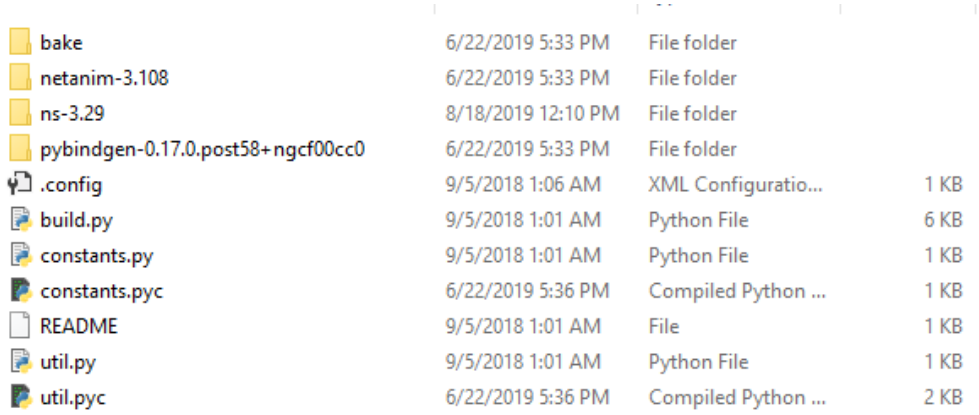

Figure A5: NS3 folder structure

To completely convert the SUMO simulation to a network of communicating nodes within NS3, a vanet-routing-compare.cc code that comes with NS3 is used. The vanet-routing-compare.cc script as shown below was used for navigation:

ns3-allinone-3.29 > src > wave > example > vanet-routing-compare.cc

For a normal network traffic simulation, within the vanet-routing-compare.cc code, the simulation parameters were defined as follows:

// Realistic vehicular trace in Cape Town
// "low density, 478 total vehicles"
m_traceFile=".../mobility.tcl";
m_logFile = "mobcpt.log";
m_mobility = 1;
m_nNodes = 478;
m_TotalSimTime = 962.01;

m_nodeSpeed = 20;

m_nodePause = 0;

m_CSVfileName = "mobcpt.csv";

m_CSVfileName = "mobcpt2.csv";

j) Launching a DoS attack on the simulated network

To launch a DoS attack, new parameters inside the vanet-routing-compare.cc code were set. The defined parameters are shown below:

// Network traffic parameters to launch UDP and TCP flooding attacks

#define TCP_SINK_PORT 9000

#define UDP_SINK_PORT 9001

#define BULK_SEND_MAX_BYTES 2097152

#define MAX_SIMULATION_TIME 50.0

#define ATTACKER_START 0.0

#define ATTACKER_RATE (std::string)"12000kb/s"

#define ON_TIME (std::string)"0.25"

#define BURST_PERIOD 1

#define OFF_TIME (std::string)"0.75"

//std::to_string(BURST_PERIOD – stof(ON_TIME))

#define SENDER_START 0.75

k) Running NS3 simulation

The NS3 simulation is run using the following command:

./waf --run vanet-routing-compare.cc

l) Packet tracing and decoding

The output of steps i and j are capture files (pcap files) that contain network packets captured from the simulated network. The capture files were decoded using Wireshark and data was saved as comma-separated-values files for further processing. A snapshot of data generated from the decoded packets is demonstrated in **Table A2**. **Figure A5** demonstrates the process of exporting the decoded packets to a CSV file.

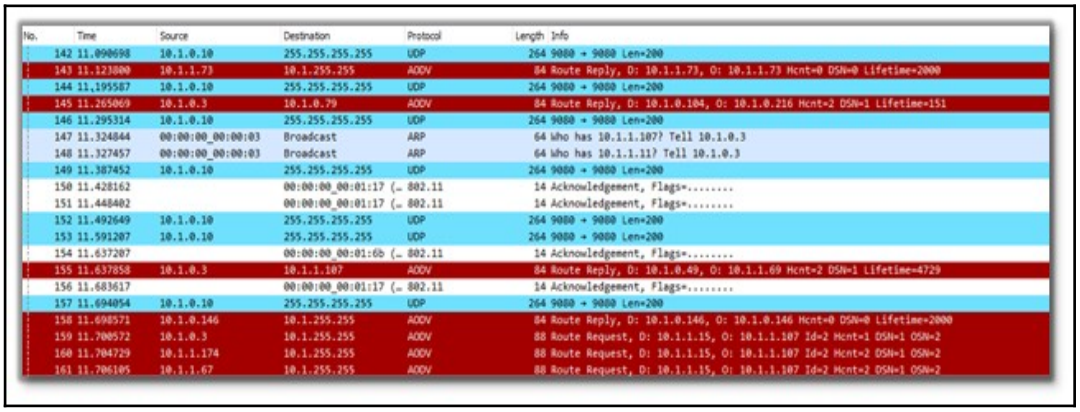

Figure A6: Wireshark packet details

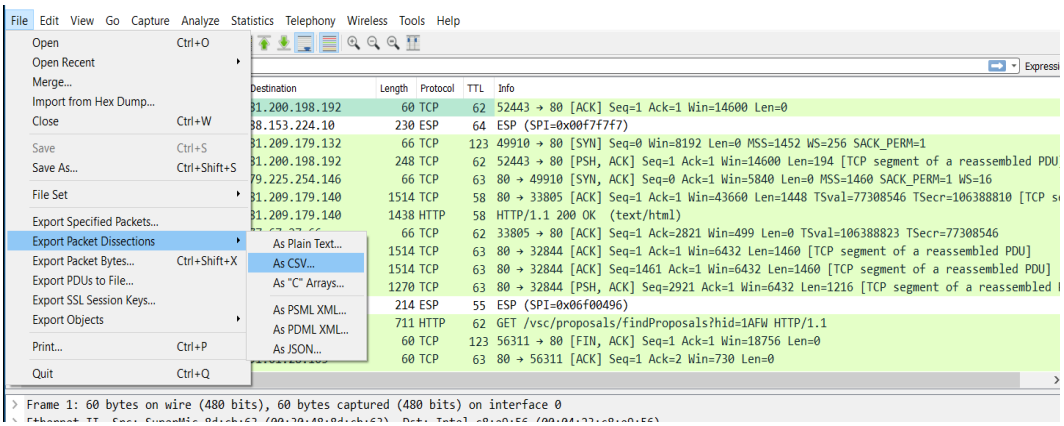

Figure A7: Saving packet data as CSV

m) **Concatenating multiple CSVs to a single CSV file**

The CSV files exported from Wireshark were then loaded into Pandas Dataframes and concatenated to make a single CSV. The procedure is demonstrated in **Figure A6**.

```
import pandas as pd

df1 = pd.read_csv('/node1.csv',engine='python')
df2 = pd.read_csv('/node2.csv',engine='python')
df3 = pd.read_csv('/node3.csv',engine='python')
df4 = pd.read_csv('/node4.csv',engine='python')
df5 = pd.read_csv('/node5.csv',engine='python')
full_data  = pd.concat([df1,df2,df3,df4,df5])
full_data.to_csv('/full_data.csv')
```

Figure A8: Concatenating multiple CSV files into a single file

Appendix B: Training a Catboost Model

```
#Import required modules
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split

# Split dataset into train and test datasets
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_state = 0
                                                 ,stratify = y)
categorical_features_indices = np.where(X.dtypes != np.float)[0]
# Define the model
catboost_model=CatBoostClassifier(iterations=500, depth=3, learning_rate=0.01
                                  ,class_weights=[0.1, 0.9],loss_function= 'Logloss'
                                  ,use_best_model=True,random_seed=100
                                  ,early_stopping_rounds = 100,verbose = 100)
#Fit the model
catboost_model.fit(X_train, y_train,cat_features=categorical_features_indices
                   ,eval_set=(X_test, y_test),plot=True)

#Genetate feature importance scores
fea_imp = pd.DataFrame({'imp': catboost_model.feature_importances_, 'col': X.columns})
fea_imp = fea_imp.sort_values(['imp', 'col'], ascending=[True, False]).iloc[-30:]

#Plot feature importance
fea_imp.plot(kind='barh', x='col', y='imp', figsize=(10, 7), legend=None)
plt.title('CatBoost - Feature Importance')
plt.ylabel('Features')
plt.xlabel('Importance');
```

Appendix C: Training a Convolutional Neural Network Model

```
# set global seed to ensure reproducibility of the results
tf.random.set_seed(seed = 100)

#Define model and add layers
model = Sequential()
model.add(Conv1D(filters=32,kernel_size=3,activation='relu',input_shape=(X_train.shape[1], 1),padding='same'))
model.add(BatchNormalization())
model.add(MaxPool1D(2,padding='same'))
model.add(Dropout(0.5))

model.add(Conv1D(filters=64,kernel_size=3,activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPool1D(2,padding='same'))
model.add(Dropout(0.5))

model.add(Conv1D(filters=128,kernel_size=3,activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPool1D(2,padding='same'))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer=Adam(lr = 0.0001),loss = 'binary_crossentropy',metrics=['accuracy'])
history = model.fit(X_train,y_train,epochs = 500,shuffle=True,validation_data=(X_metric,y_metric),verbose=100)
keras.callbacks.EarlyStopping(monitor='val_loss',min_delta=0, patience=0,verbose=0, mode='auto')
```

Appendix D: Training a Long Short-Term Memory Model

```
#Define LSTM model and add layers
model = Sequential()
model.add(LSTM(units=32,activation='relu', return_sequences = True,input_shape=(X_train.shape[1], 1)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(LSTM(units=64,activation='relu', return_sequences = True))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(LSTM(units=128,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer=Adam(lr = 0.0001),loss = 'binary_crossentropy',metrics=['accuracy'])
history = model.fit(X_train,y_train,epochs = 500,shuffle=True,validation_data=(X_metric,y_metric),verbose=100)
keras.callbacks.EarlyStopping(monitor='val loss',min delta=0, patience=0,verbose=0, mode='auto')
```