University of Pardubice
Faculty of Economics and Administration

Detection of IoT Cyberattacks in Smart Cities using Deep Neural Networks

By

Zeru Kifle

Advisor

prof. Ing. Petr Hajek, Ph.D.

A Thesis Presented to Faculty of Economics and Administration of University of Pardubice
in partial fulfillment of the Requirements for the Degree of Master of Science in Informatics and System Engineering

Pardubice, Czech Republic

April 30, 2023

Author's Declaration

I declare:

The thesis entitled Detection of IoT Cyberattacks in Smart Cities using Deep Neural Net is my own work. All literary sources and information that I used in the thesis are referenced in the bibliography.

I have been acquainted with the fact that my work is subject to the rights and obligations arising from Act No. 121/2000 Sb., On Copyright, on Rights Related to Copyright and on Amendments to Certain Acts (Copyright Act), as amended, especially with the fact that the University of Pardubice has the right to conclude a license agreement for the use of this thesis as a school work under Section 60, Subsection 1 of the Copyright Act, and that if this thesis is used by me or a license to use it is granted to another entity, the University of Pardubice is entitled to request a reasonable fee from me to cover the costs incurred for the creation of the work, depending on the circumstances up to their actual amount.

I acknowledge that in accordance with Section 47b of Act No. 111/1998 Sb., On Higher Education Institutions and on Amendments to Other Acts (Higher Education Act), as amended, and the Directive of the University of Pardubice No. 7/2019 Rules for Submission, Publication and Layout of Theses, as amended, the thesis will be published through the Digital Library of the University of Pardubice.

In Pardubice on

Kebede Zeru Kifle, b.o.h.
(Roll No.E21821)

April 30, 2023

Acceptance Certificate

Faculty of Economics and Administration

University of Pardubice

The thesis report entitled **Detection of IoT Cyberattacks in Smart Cities using Deep Neural Networks** submitted by Mr. Zeru Kifle is carried under my supervision and guidance and fulfilling the nature and standard required for the partial fulfillment of requirements of the master of science degree in Informatics and System Engineering. The work encapsulated in this thesis has not been submitted somewhere for the degree.

prof. Ing. Petr Hajek, Ph.D.
Faculty of Economics and Administration
University of Pardubice

Signature

April 30, 2023

Abstract

Nowadays, IoT and smart cities are increasingly becoming popular topics among both researchers and practitioners. IoT applications are the main backbone for building a smart city. Many governments use IoT applications to provide better services for their citizens, and other non-governmental organizations also use them to provide better services and products for their customers. Moreover, the day-to-day activities of society and device interactions in a smart city are carried out over IoT applications. Conversely, new and intelligent attacks are greatly increasing due to the behavior of these applications. As a result, security becomes one of the most crucial concerns that need to be addressed. To date, several intrusion detection models have been proposed by several researchers for ensuring the security of IoT devices in the smart city. In this thesis, I proposed deep neural network-based models MLP, LSTM, and GRU for detecting binary and muti-class IoT cyber attacks, using an imbalanced big data set. The most recent datasets, UNSW-NB15 and CICIDS 2017, were used for model training and evaluation, which are enhanced by a variety of recently added cyber attacks. The experimental results for the UNSW-NB15 dataset show that the MLP model outperformed other models in terms of recall, precision, F1-score, and FPR (false positive rate) with values of 99.17%, 99.17%, 99.17%, and 0.0037, respectively. Furthermore, the LSTM model achieved a higher accuracy of 99.26%. In the case of conventional and ensemble models, Random Forest outclasses other models with respect to all metrics when trained and evaluated with the UNSW-NB15 dataset. Further, when the dataset CICIDS2017 was used for training and evaluating the Random Forest model, it outperformed other conventional and ensemble methods. Among the deep neural network models, the MLP model classified attacks with the accuracy of 98.10%, precision of 98.20%, F1-score of 98.12%, and FPR of 0.0202, which makes it the best-performing deep learning model.

Key Words: - *IoT; Cyber Attack; Smart City; Attack Detection; Deep Learning; Big Data.*

Acknowledgment

In the first, place I want to forward my sincere thanks to God and Saint Mary. Next, I would like to give my grateful and sincere thanks to Professor Petr Hajek for his guidance, patience, and support during my graduate studies.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Chapter 1 Introduction

1.1 Introduction

Internet of Things (IoT) means the process of sharing real-time data or information between physical devices through the Internet without the intervention of human beings. These physical devices are equipped with different sensors, which are embedded in different objects or systems to share information and communicate with each other to control or monitor the system autonomously (Singh et al., 2021).

IoT devices are utilized for developing smart homes, offices, and cities. Some of the systems developed with the help of IoT are smart door access control systems, lighting for homes and offices, automated gate and garage systems, thermostats and humidity controllers, traffic management, lighting on streets, pollution monitoring and reporting, parking solutions, water management, waste management, wearable devices such as smartwatches, healthcare, autonomous driving systems, agriculture and smart farming, industrial IoT for manufacturing, disaster management, logistics, and fleet management, smart grids and energy management, and big data analytics (Singh et al., 2021; Page, 2023).

IoT is the main component for the development of a smart city. IoT devices in smart cities are interconnected with each other through the Internet and perform their tasks autonomously. One of the drawbacks of IoT devices in a smart city is their vulnerability to cyber attacks. This vulnerability occurred due to the limited resources (Sha et al., 2020) of the IoT device, such as its open communication medium, its lack of processing capability, and its lack of storage space for implementing classical cyber attack detection and prevention algorithms on the device as another network device.

Generally, the IoT is vulnerable to cyber attacks because of a lack of sufficient authentication and authorization, unreliable user interfaces, insecure networks, privacy issues, inadequate transport encryption, the inadequacy of the security configuration, and poor physical security

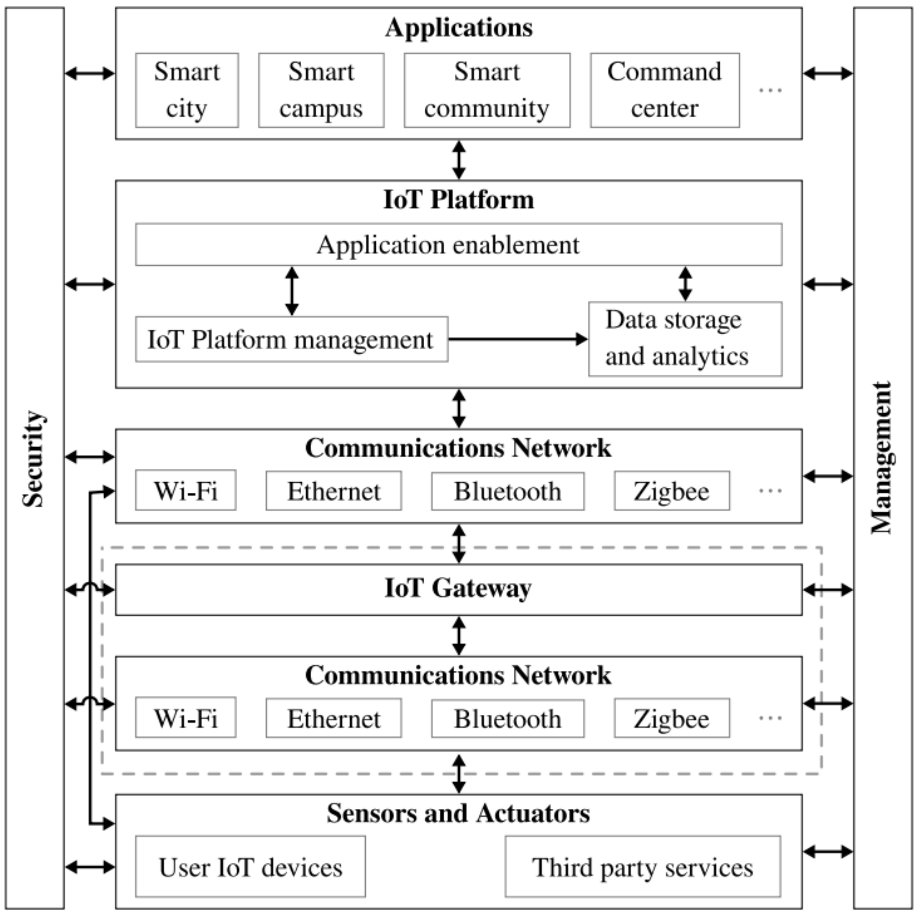


Figure 1.1: Basic architecture of IoT(Domínguez-Bolaño et al., 2022)

(Vishwakarma & Jain, 2020).

The main purpose of the attacker on IoT devices is to divert the normal operation of the device in its operating environment by controlling all functionalities of the device, blocking data transmission between IoT devices, altering the data transmission from source to destination and vice versa, drooping some part of the data transmission, etc. Botnets, denial of service attacks (DoS), distributed DoS (Vishwakarma & Jain, 2020), man-in-the middle, identity theft, and data theft, ransomware, remote recording, and advanced persistent threats (Chen et al., 2021) are some of the cyber attacks for IoT device (Singh et al., 2021).

Based on classical Machine Learning (ML) algorithms, various IoT cyber attack detection models for smart cities have been developed, trained, and evaluated with small, outdated, balanced datasets. However, limited research has been done for an IoT cyber attack detection model with deep learning techniques and model training and evaluation on the most recently available datasets, which consist of variant attack types for IoT devices in a smart city, so there is a gap for deep learning models to detect and classify intrusions in the context of smart cities, which needs to be addresssed by experimenting on big and imbalanced datasets.

In this thesis, I propose a deep neural network-based IoT cyber attack detection model for smart cities. The deep neural network algorithms used for modeling are Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU). These detection and classification models were evaluated using accuracy, F1-score, precision, recall, and false positive rate (FPR) metrics. In addition to this, I also compare the performance of these models with other popular algorithms, such as Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), Artificial Neural Network (ANN), and K-Nearest Neighbors (KNN), which are the most common and powerful algorithms for classification tasks. I used two of the most recent datasets for training and evaluating the model. These datasets are publicly available for cyber attack researchers, and they are UNSW-NB15 (Moustafa & Slay, 2016; Janarthanan & Zargari, 2017) and CICIDS 2017 (of New Brunswick, 2017; Sharafaldin et al., 2018). These models were simulated in the Python programming language with the Jupiter notebook editor.

1.2 Statement of The Problem

Several research works haved evaluated different intrusion detection models using several datasets to detect and classify IoT cyber attacks in smart cities to reduce IoT cyber attacks in smart cities. These research works include both classical ML and deep learning models for prediction and classification. Still, they evaluated their models with outdated datasets with a small diversity of attacks, and on the other hand, some researchers used the small size of the most recent datasets for their model training and evaluation. Classical machine learning models such as LR, SVM, DT, RF, ANN, KNN by (Alrashdi et al., 2019; Rashid, Kamruzzaman, Hassan, et al., 2020; Rashid, Kamruzzaman, Imam, et al., 2020) research has been done and evaluated using the small size of the most recent datasets (UNSW-NB15 and CICIDS2017). A deep learning model based on Convolutional Neural Network (CNN) and recurrent ANN by (Al-Taleb & Saqib, 2022) evaluated by recent datasets, UNSW-NB15 and CICIDS2017, and outdated datasets, BoT_IoT and ToN-IoT, KDDcup99, NSL-KDD, Koyot, and WSN-DS (Vinayakumar et al., 2019). The performance of the cyber attack detection model is affected by the types of algorithms and datasets used for modeling. Today, modeling efficient attack detection model with deep learning techniques on big datasets is the hot research topic area.

Due to the diversity of attacks and dynamics of attack behavior, previous work is insufficient to handle the current IoT cyber attack in smart cities. So, several researchers recommend developing a new IoT cyber attack detection framework with new approaches, such as using advanced modeling algorithms and a recent dataset enriched with new attack types.

By taking into consideration of the above issues, in this thesis, Deep Neural Networks (DNNs) such as MLP, LSTM, and GRU is used to model IoT cyber attack detection for smart cities. The proposed model is evaluated using the most recent USNW-NB15 and CICIDS2017 datasets, which were enhanced with varieties of IoT cyber attacks. The diversity of attacks in each dataset is described in detail in chapter 4 section 4.2, Table 4.1. The proposed model's performance is evaluated using well-known model evaluation metrics: accuracy, precision, F1-score, recall, and false positive rate (FPR).

1.3 Objective

1.3.1 General Objective

Characterize IoT architecture for smart cities, summarize existing approaches to detecting IoT cyber attacks, propose a DNN-based model for detecting IoT cyber attacks, validate the model using datasets relevant to smart cities, and discuss implications of the results for smart cities.

1.3.2 Specific Objectives

- Select recent publicly available datasets enhanced with IoT cyber attack variants of smart city.
- To perform data prepossessing for proposed model input.
- Modeling IoT cyber attack detection and prediction model using DNN methods.
- Evaluate the DNN-based attack detection model with selected datasets and compare the result with classical and ensemble classification algorithms.

1.4 Scope of The Thesis

The scope of this thesis focuses on modeling DNN-based models (LSTM, GRU, and MLP) for IoT cyber attacks in smart cities and evaluating the model with large, imbalanced datasets. The classical and ensemble machine learning classifiers were trained and evaluated with the same datasets without modification of the hyperparameter tuning parameters of their default configurations for comparison.

1.5 Contribution

Various IoT cyber attack detection models have been developed by several researchers in the past few years, and they are still being worked on. Each scholar uses several classification algorithms for modeling and several datasets for evaluating their models. The research done by (Alrashdi et al., 2019; Rashid, Kamruzzaman, Hassan, et al., 2020) for intrusion detection models by RF, LR, SVM, DT, RF, ANN, and KNN algorithms, and their model trained and evaluated with a subset of USNW-NB15 and CICIDS2017 datasets, on the other hand, (Shafiq et al., 2020) used the ANN machine learning algorithm to model an intrusion detection system and evaluated their model using outdated datasets, such as BoT-IoT_IoT. In this thesis, I model an IoT attack detection model using DNNs (LSTM, GRU, and MLP) and evaluate it with the most recent big datasets (USNW-NB15 and CICIDS2007) that are enhanced with varieties of attacks. The main contribution of this thesis is listed as follows:

- Existing work does not consider much for deep learning algorithms on both datasets. In this thesis, I demonstrate deep learning algorithms with all records of the datasets, UNSW-NB15 and CICIDS2017.
- I explore the performance of deep learning algorithms with big datasets for attack detection and classification.
- To provide the attack detection and classification performance of a classical and ensemble machine learning classifiers and a DNN model on big-size binary and multi class datasets.
- To provide the output of DNN-based model ability to detect and classify IoT cyber attacks in a smart city environment for the researchers.

1.6 Research Methodology

1.6.1 Literature Review

Internet of Things (IoT) cyber attack prediction is a hot research topic in the smart city environment. Numerous studies have been done to protect the IoT from cyber attacks. The literature review conducted in this thesis mainly focuses on the existing cyber attack detection models related to smart cities, the algorithms used for modeling the detection model, the datasets used for model evaluation, the performance achieved, feature recommendations, and gaps that I have figured out for further improvement relating to IoT cyber attacks.

1.6.2 Data Collection

A variety of datasets are available online for conducting cyber attack prevention research. In this research, I collected the latest datasets containing new varieties of attack types that are relevant for smart cities to evaluate the proposed IoT attack detection model. USNW-NB15 (Moustafa & Slay, 2015; Janarthanan & Zargari, 2017) and CICIDS2017 (of New Brunswick, 2017; Sharafaldin et al., 2018) datasets are the most recent datasets enriched with recent attack types that are relevant for a smart city. UNSW-NB15 created by the dataset includes generic, fuzzers, analysis, backdoor, exploit, reconnaissance, shellcode, and worm attack categories, while CICIDS2017 created by the Canadian Institute of Cybersecurity also includes brute force attacks, heartbleed attack, botnet, DoS attack, distributed DoS (DDoS) attack, web attack, and infiltration attacks for further explanation, see in Chapter 4 at section 4.2.1.

1.6.3 IoT Attack Detection System Design and Implementation

In the proposed IoT cyber attack detection model designed and implemented phase, the overall architecture of the model is designed, DNNs, classical machine learning, and ensemble methods used for modeling are explained; relevant datasets for model evaluation are clearly described, preprocessed, and prepared for training and evaluating the model. The algorithms for the model and the dataset for model training and evaluation are selected based on the gaps identified during the literature review and other researchers' recommendations. Finally, the model is simulated using the Python programming language, and I used Jupiter Notebook as an editor; for further information, see in Chapter 4.

1.6.4 Result Evaluation and Discussion

To evaluate the performance of the IoT cyber attack detection model, I used commonly used model performance metrics, accuracy, precision, F1-score, recall, time to train, and time to predict.

1.7 Thesis Outline

The remaining part of this document is organized as follows. In Chapter 1, the introduction to IoT, cyber attack challenges in IoT application smart city, research objective, statement of the problem, the contribution of the thesis, and scope of the work are presented. The research methodology is also presented in this chapter. In Chapter 2, a detailed description of IoT applications in smart cities, smart city architecture, and the cyber attack in IoT technologies challenges are discussed. In Chapter 3, a literature review focused on deep learning methods and classical machine learning algorithms used for IoT cyber attack detection and classification models which are more related to the current work are presented. In Chapter 4, the proposed IoT cyber attack prediction model in smart city, algorithms used for the model description and datasets used for evaluating the model with its description, and necessary steps of the prepossessing phase in detail are presented. Finally, in Chapter 5 and Chapter 6, the discussion, result evaluation, conclusion, and future recommendation of this thesis are presented.

Chapter 2 Theoretical Background

2.1 IoT Cyberattacks in Smart Cities

The infrastructures in smart cities are equipped with IoT devices that are used to provide information access or services to the citizens (see Figure 2.1). The IoT application in the smart city is used to simplify the life of people by improving the quality of service, reducing the amount of time for getting information, and also the service providers easily provide their service to their customers through customer IoT appliance (Chen et al., 2021).

Smart cities are characterized by heterogeneity, resource constraints, mobility, connectivity, scalability, and user involvement. Heterogeneity is expressed by the existence of the variety of IoT devices, platforms, communication protocols, technologies, mobility means, diverse hardware performances, and so on (Cui et al., 2018) in smart city applications (Zhang et al., 2017) as shown in Figure 2.2, while the characteristic of mobility indicates wireless communication and real-time data flow monitoring. IoT devices implemented in smart cities are resource-limited, such as in memory, battery capacity, and processing capabilities. This limitation occurred due to the small size of the IoT device. Smart cities express connectivity and scalability by involving various types of IoT devices for various applications on the smart city platform, and scalability is expressed by the expansion of smart cities and the increment in the number of IoT devices in the communication network (Cui et al., 2018). User requirements and participation in demanding technology are one of the basic needs for the expansion of smart cities.

2.2 IoT-based Smart City Architecture

There is no consistent architecture for an IoT-based smart city. According to Cui et al. (2018), IoT-based smart city architecture is organized into four layers. These are the perception layer (sensors, devices, etc.), network layer (wired and wireless communication), support layer (cloud computing, fog computing, etc.), and application layer (smart health care, smart

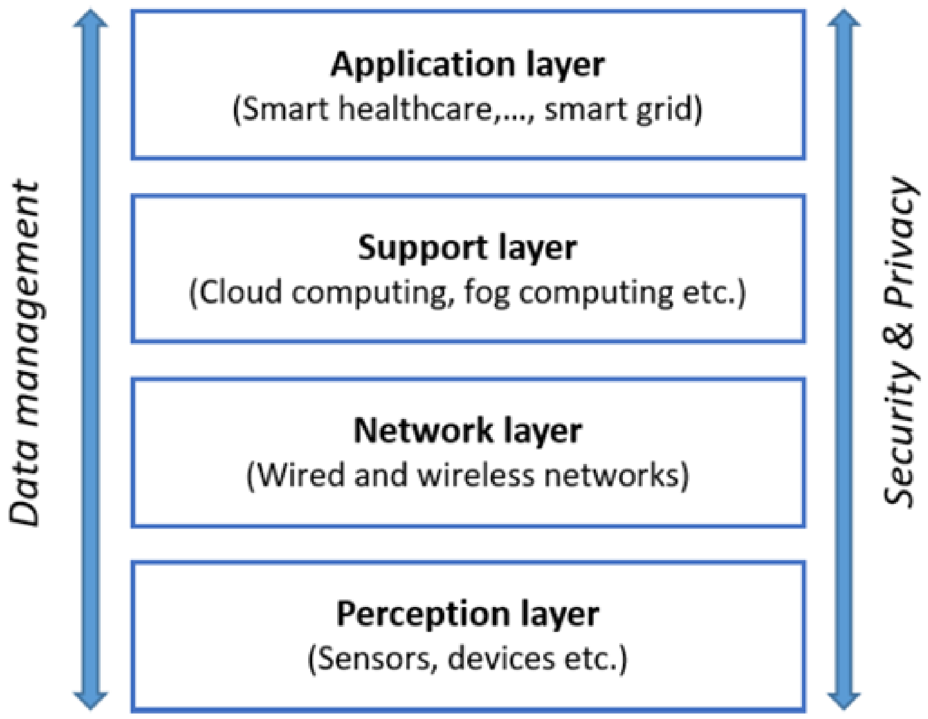


Figure 2.1: Smart city architecture (Cui et al., 2018)

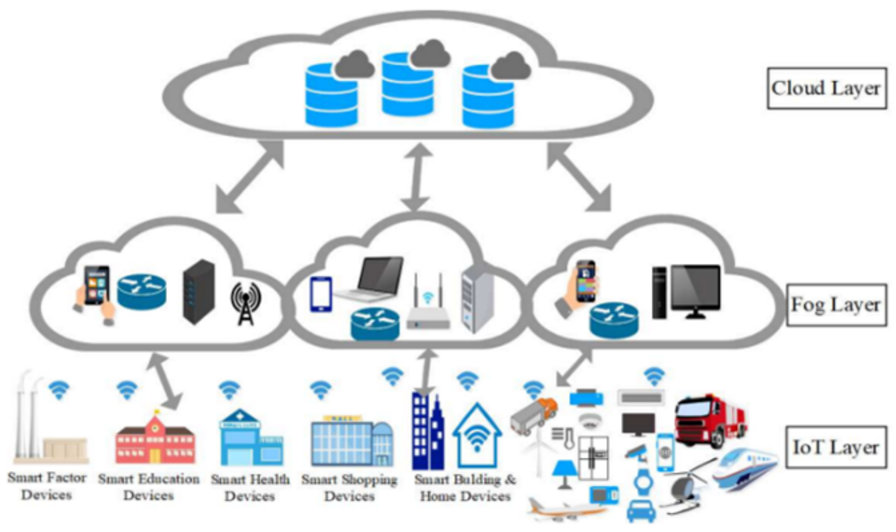


Figure 2.2: Smart city architecture with three layers (Alrashdi et al., 2019)

agriculture, smart home, etc.).

2.3 IoT applications in Smart Cities

The IoT technologies are widely applied for smart grids, smart transportation, smart environment, smart living, smart health, and smart energy (Al-Turjman et al., 2022), see Figure 2.3. In 2020, the global market value of smart city expected to reach US$ 1200 billion (Zhang et al., 2017).

Smart environment

Environmental pollution and global warming are the main issues in this era. Implementing an IoT-based smart environmental system in an urban area is one of the mechanisms to reduce global warming and environmental pollution. The presence of IoT technologies and sensors is one of the mechanisms for creating a smart environmental monitoring system in a smart city. Some of the environmental problems that disrupt normal human living conditions are air pollution (Alshamsi et al., 2017), water pollution, radiation pollution (Ullo & Sinha, 2020), environmental change, weather forecasting, and so on. So that currently, smart cities integrate IoT devices and sensors to monitor and control those disasters in their early stages.

Smart Transportation

According to the Allied Market Research report, the cost of IoT for the transportation system in 2016, is $135 billion USD, and in 2023, it is expected to reach $328 billion USD (MEDIA, 2020). Integration of IoT technology in transportation systems has benefits for enhancing customer experience by providing real-time and up-to-date data to them, improving safety, operational performance, environmental impact, and energy usage improvement (MEDIA, 2020). In smart transportation systems, customers can use their smart transport application on their phone to access the required real-time information about transportation without going to the physical transportation office (Cui et al., 2018).

The applications of IoT technologies in the transportation system are traffic management (smart parking, traffic lights, smart accident assistance), toll and ticketing, connected cars, vehicle tracking systems (trip schedules, fleet tracking, driving times, driver rest break scheduling, alerts for speeding, harsh cornering, acceleration, or braking, monitoring of vehicle load,

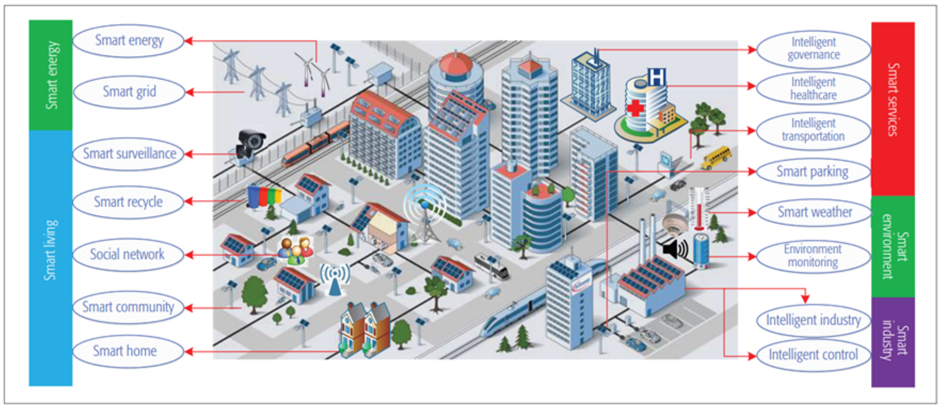


Figure 2.3: Smart city building component (Zhang et al., 2017), page 123

distance traveled, and fuel consumption), and public transport management, which includes real-time vehicle tracking, data analysis and real-time management, and personalized travel information (MEDIA, 2020).

2.4 Cyberattack in IoT

Nowadays, to simplify the living standard of citizens in urban areas, the infrastructure and public services of the city are interconnected with IoT technologies, which build a smart city. In a smart city, with the help of IoT devices, a huge amount of data is exchanged from different devices in different directions. Due to the nature of IoT devices and applications, the smarty city is easily vulnerable to attacks such as DoS, collusion attacks, Sybil attacks, eavesdropping attacks, outside forgery attacks, spam attacks, outside forgery, likability attacks, inside curious attacks, identity attacks (Cui et al., 2018), DDoS attacks (Vishwakarma & Jain, 2020), and so on.

In 2015, around 230 thousand of Ukrainian are out of electric service due to the DoS attack on the smart grids (Cui et al., 2018).

2.4.1 Cyber Attack Issues

The data or information can be collected to analyze the business situation, to support the decision-making process, to take advantage of the competitors, or intentionally harm the individual or the society. The data or information of the society living in a smart city can be accessed by the smart device manufacturer, or service provider, through appliances used in their home without harm (Cui et al., 2018). On the other hand, their data or information is intentionally accessed by hackers. Protecting smart city from IoT cyber attack by using cryptography, blockchain, biometrics, and other techniques, which are commonly used for other resource reach application is insufficient for IoT device because the resource of IoT device in a smart city is limited to implement such techniques (Cui et al., 2018). In this thesis, I focus on machine learning-based and deep learning-based cyber attack detection and identification of attacks in IoT-based smart cities. Almarshdi et al. (2023) categorizes IoT attacks into the physical layer, network layer, support layer, and application layer as shown in Figure 2.4, which is taken on page 300.

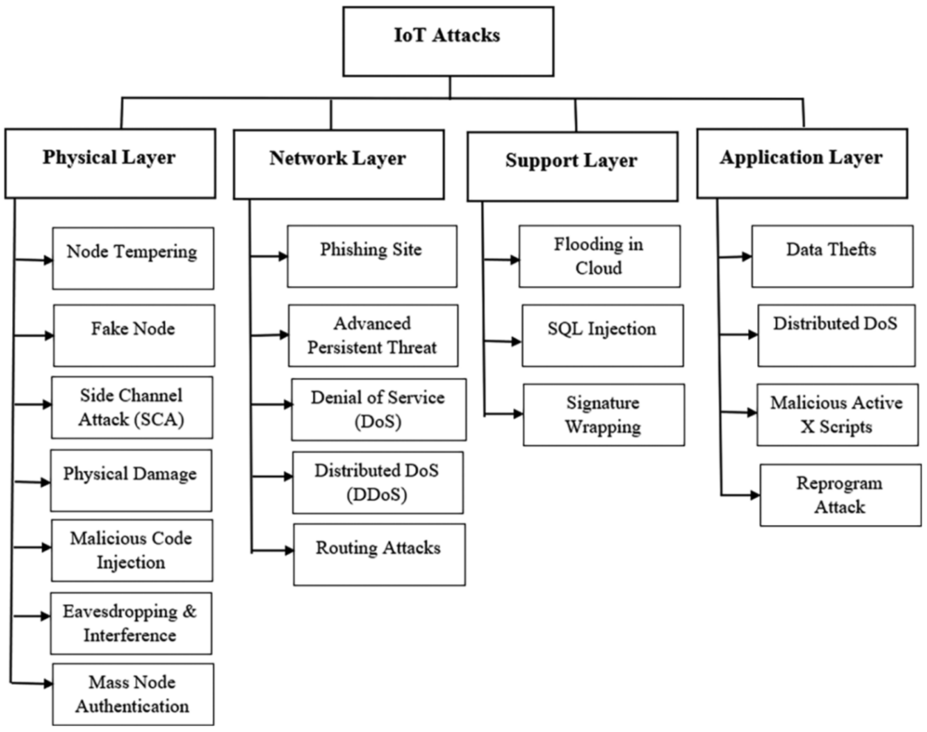


Figure 2.4: IoT attacks families in smart city architecture layers

Chapter 3 Review of Literature

Nowadays, many IoT attack defense models have been developed. Most of them are modeled based on classical machine learning algorithms, and a few are modeled based on DNN-based methods. In this section, I present research done by various scholars related to IoT cyber attack detection and mitigation using several machine learning algorithms and a few DNNs, as well as the datasets used to evaluate the models.

3.1 Standard Machine Learning-Based IoT Device Attack Detection in Smart Cities

Alrashdi et al. (2019) proposed a machine learning-based anomaly detection model for IoT cyber attacks in smart cities. The model (AD-IoT) proposed by (Alrashdi et al., 2019) detects the IoT devices that are compromised by a malicious attack in smart cities by using RF machine learning. This model was evaluated with the UNSW-NB15 dataset, and its accuracy score was 99.34% for the binary classification problem (benign or malicious). This work was done on a single-machine algorithm for attack detection, and they also recommend to use CNNs to improve their work.

Rashid, Kamruzzaman, Hassan, et al. (2020) proposed machine learning techniques that are used to detect and mitigate cyber attacks in IoT-based smart city applications. The authors' motivation was to reduce IoT device failures in smart cities and improve the efficiency of single attack classifiers that have been done previously (Alrashdi et al., 2019). They used DT, RF, LR, SVM, KNN, and ANN machine learning algorithms to build a model for IoT cyber attack detection and mitigation in smart cities. In addition to this, both single and ensemble classifier methods of machine learning algorithms with advanced feature selection and performance evaluation were done for attack detection in smart city IoT applications. The ensemble (stacking) detection model scored better results than the single classifier approach with the performance metrics of accuracy, precision, recall, and F1-Score using the UNSW-NB15 and CICIDS2017 datasets. However, for further improvement, they recommend using deep learning.

To select an efficient machine learning algorithm for intrusion detection or cyberattacks in IoT-based smart city applications, (Shafiq et al., 2020) proposed a machine learning selection framework that applies a bijective soft set approach and its algorithm. This framework used the Bot-IoT dataset for the proposed framework's evaluation. NB, BayesNet, C4.5, RF, and Random Tree were the algorithms used for it. From these algorithms, the NB machine learning algorithm was selected for anomaly and intrusion detection of IoT device attacks in smart cities. This algorithm performed better in terms of accuracy and the time taken to build the model of performance metrics.

3.2 Deep Learning Model-Based IoT Device Attack Detection in Smart Cities

Chen et al. (2021) conducted a paper review relating to IoT application cyber attacks in smart city detection and attack classification using deep learning algorithms. The authors (Chen et al., 2021) discussed deep belief networks, Boltzmann machines, restricted Boltzmann machines, CNNs, recurrent ANNs, and generative adversarial networks for attack detection and classification for smart cities. In addition to this, the authors presented a few deep learning-based cyber attack detection models for IoT applications in a smart city.

Al-Taleb & Saqib (2022) proposed an intelligent cyber threat identification model using a hybrid machine learning algorithm for a smart city environment. A CNN and quasi-recurrent network (CNN-QRNN)-based hybrid deep learning model designed by Al-Taleb & Saqib (2022) was used to classify and analyze cyber threats in a real-time environment of smart cities. The proposed model was evaluated with the BoT-IoT and ToN-IoT datasets. Its results show an improvement in accuracy and a reduction in FPRs.

Rashid, Kamruzzaman, Imam, et al. (2020) proposed ANN-based cyber attack mitigation techniques for smart city applications. They used the UNSW-NB15[1] dataset for evaluating the proposed model. The proposed model is an ANN model. Its performance was evaluated using

[1]https://research.unsw.edu.au/projects/unsw-nb15-dataset

the most commonly used performance metrics, such as accuracy, precision, recall, and F1-score, and its scores were 85.1%, 84%, 85%, and 84%, respectively. However, to improve the performance of the detection model, they recommend a deep learning model for future work.

Vinayakumar et al. (2019) explores DNNs that are used to develop an adaptable and efficient intrusion detection model to detect and categorize unplanned and unpredictable cyber attacks in the network using various freely available cyber community malware datasets. The behavior of malware attacks is dynamic, and this study aims to identify the best and most effective algorithms for detecting cyber attacks. A DNN model for cyber attack detection experiments was performed on the NSL-KDD[2], UNSW-NB15, Kyoto, WSN-DS, CICIDS 2017[3], and KDDCup 99 datasets[4], but it outperformed on the KDDCup 99 dataset. Vinayakumar et al. (2019) model Scaled-hybrid_IDS with hybrid DNNs for detecting malware in the network. This model can be used to monitor host-level and network traffic cyber attacks in real-time environments.

Meidan et al. (2018) proposed a network-based cyberattack anomaly detection method using deep autoencoders. This model focused on Mirai and BASHLITE IoT-based botnet attacks that were generated from nine infected IoT devices. The model's false alarm rate performance metric is lower than that of other anomaly detection algorithms. But the number of attacks incorporated into the data set is small. Table 3.1 illustrates the summary of this review of the literature.

To sum up, from the related works, it is clear that the IoT devices in smart cities need improvement in security aspects to ensure seamless operation without any problems that are discussed in the above literature. Thus, this is the motivation for this investigation, raises from the security concerns reported by several authors, of a deep learning-based attack classification model which is capable of detecting several IoT networks related cyber-attacks with the imbalanced datasets for learning. As a result, the present study will present a characterization of IoT architecture for smart cities, summarises and compares existing methods developed for detecting IoT cyber-attacks, propose a DNN-based model for detecting IoT cyber attacks, validate

[2]https://www.unb.ca/cic/datasets/nsl.html
[3]https://www.unb.ca/cic/datasets/ids-2017.html
[4]https://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data

the model using datasets relevant to smart cities, and discuss implications of the results with regards of the IoT application in the smart cities context.

Authors	Proposed work	Algorithm	Dataset	Train and Test	Result	Gap and recommendation
Alrashdi et al. (2019)	AD-IoT System designed for IoT cyber attack detection in the smart city	RF	UNSW-NB15	Train: 65546, Test: 634388	Accuracy: 99.34%, it has the ability to detect unknown attack unlike previously signature-based intrusion detection system	Experemnt done only with the RF algorithm and small dataset size and recommend CNN algorithm for improvement.
Rashid, Kamruzzaman, Hassan, et al. (2020)	Detection and mitigation of cyber attacks in IoT-based smart city applications	LR, SVM, DT, RF, ANN and KNN; feature selection technique (information gain ratio) used to select relevant features	UNSW-NB15 & CICIDS2017	UNSW-NB15 Train and Test: 119241 & 56000, CICIDS2017 Train and Test: 41997 & 148777	Stacking ensemble model performance: accuracy scores 83.0% and 99.9%) on UNSW-NB15 & CICIDS2017, respectively.	The authors recommend improving attack detection using deep learning techniques in the future.
Chen et al. (2021)	The author presented a review of a deep learning-based cyber attack detection model for securing a smart city IoT application	Boltzmann machines, restricted Boltzmann machines, deep belief networks, recurrent neural networks, CNNs, and generative adversarial networks are presented.	—	—	Introduced some deep learning technologies and recommended different types of cyber attacks to be detected by them in smart city applications	Recommended LSTM for attack detection
Al-Taleb & Saqib (2022)	Proposed CNN and quasi-recurrent neural network-based hybrid deep learning model for cyber threat intelligence	CNN and quasi-recurrent neural network used for modeling	BoT-IoT and TON_IoT	BoT_IoT: 3668045, TON-IoT:461043, *Note:* train and test not specified	Achieved better accuracy on BoT-IoT (99.9%) and TON_IoT (99.9%), and FPR on BoT-IoT (0.0003) and TON_IoT (0.001)	Recommendation: implementation of the model in distributed environment
Rashid, Kamruzzaman, Imam, et al. (2020)	An ANN-based model for mitigating cyber attacks in smart city applications	ANN	UNSW NB15	Train: 65866, Test: 16466	Accuracy, precision, recall, and F1-score of 85.16%, 84%, 85%,84%, for training and 85.10%,84%,85%,84% for testing dataset, respectively	Recommended ensemble and deep learning techniques
Vinayakumar et al. (2019)	The authors evaluated the performance of DNN and classical machine learning algorithm model for intelligent intrusion detection and they proposed scalable and hybrid DNNs framework for real-time cyber attack detection	DNNs with different numbers of layers	KDDCup 99, NSL-KDD, UNSW-NB15, Kyoto, WSN-DS &CICIDS 2017	Train: 494021, Test: 311029; Train: 125973, Test: 22544; Train; 935000 Test: 28481; Train: 3054682, Test: 1563923; Train: 262260 Test: 112400; & Train: 93500, Test: 28481, respectively.	The DNN model performs better than the classical machine learning model for the identification and classification of unforeseen and unpredictable cyber attacks	Experiments done on small samples of each dataset and the number of no-nattack instances small compared to attack instances, the proposed scalable and hybrid DNNs framework was not tested with existing intrusion detection dataset
Meidan et al. (2018)	Proposed a deep autoencoder-based network-anomaly detection model for attacks generated by comprised IoT device	Deep Autoencoder	N-BaIoT	—	Applied autoencoders for anomaly detection purposes rather than dimensionality reduction	Number of attacks was small and anomaly detector trained only with benign dataset

Table 3.1: Review of literature summary

Chapter 4 Architecture of the Proposed IoT Cyber Attack Detection System

4.1 Model for Detection of IoT Cyber Attacks in Smart Cities

As shown in Figure 4.1, the first three models, MLP, LSTM, and GRU, are a group of DNNs, and the remaining DT, AdaBoost, RF, LR, and Gaussian NB form a group of conventional machine learning algorithms used to model IoT cyber attack detection system for smart cities.

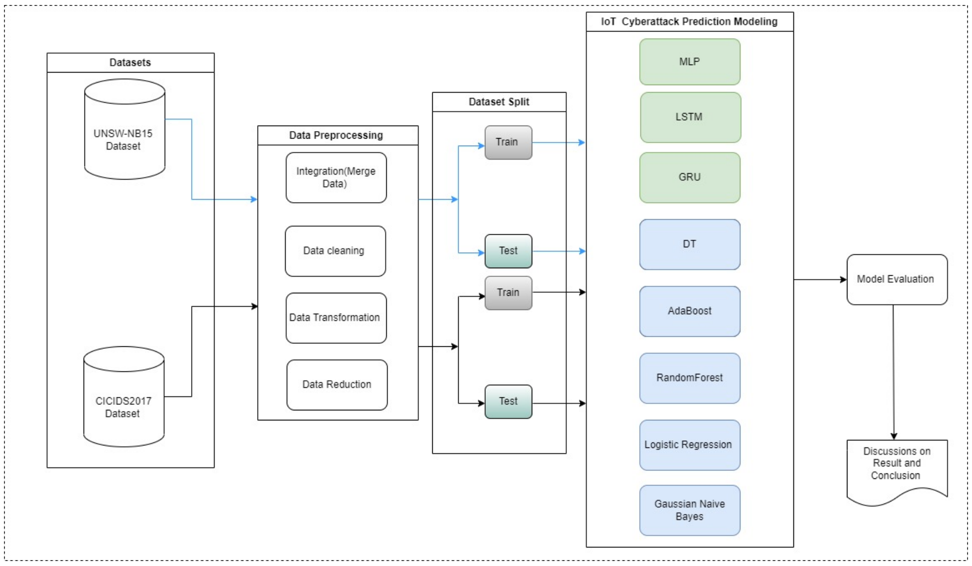


Figure 4.1: Architecture for detecting IoT cyber attack in smart cities

4.1.1 Deep Learning Methods

Because the deep learning-based intrusion detection model exhibits an impressive performance in many security applications, many researchers now favor it for modeling intrusion detection systems (Huang, 2021). The deep learning approach is appropriate for analyzing high-dimensional features on large datasets. Deep learning-based models for intrusion detection systems include both traditional MLP by (Mohammed et al., 2020), and recurrent models, such as GRU by (Kasongo, 2023), and LSTM by (Huang, 2021; Kasongo, 2023). In this thesis, I simulate IoT cyber attack detection and classification in smart cities using deep learning techniques.

4.1.1.1 Long Short-Term Memory

The LSTM neural network is a family of ANNs that is an enhanced version of recurrent neural networks used in deep learning and artificial intelligence. LSTM was originally proposed to handle the problem of gradient and vanishing gradient problems in recurrent neural networks (Dey & Salem, 2017; Huang, 2021). The LSTM architecture is composed of three subunits, which are called an input gate, an output gate, and a forget gate, as shown in Figure 4.2. The cell is a memory of the LSTM that stores important information for a random period. The hidden state is considered short-term memory and the cell is considered long-term memory in the LSTM architecture.

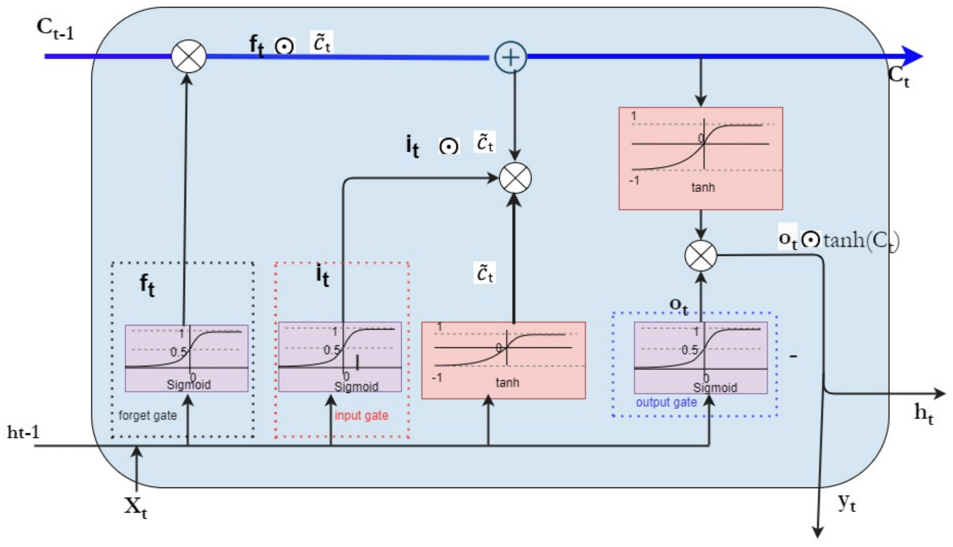


Figure 4.2: LSTM structure adopted from Kandpal (2018); Dey & Salem (2017)

Working Principles of LSTM

To determine the current memory state c_t value, we can compute it using the equation (4.1). Initially, the cell takes the previous memory state (C_{t-1}) and multiplies it element-wise with the forget output value to come up with the decision point to pass or not to the memory (cell) state. The decision of whether to pass or not pass the previous memory state mainly depends on the output of the forget gate. The output value from forget (f) is 1 or 0. If the forget output value is equal to 1, the previous memory state passes to the cell; otherwise, it does not. The new memory (cell) state, C_t, in equation (4.2), is computed by the addition of the current memory state and the element-wise multiplication of the input vector to it and the candidate memory cell state in equation (4.3). Finally, the output is calculated by using the tanh(C_t) function or $g(C_t)$, where g is the activation function; it can be a "hyperbolic tangent (tanh) function“ or "rectified linear unit (ReLU)“ (Dey & Salem, 2017; Dobilas, 2022; YADAV, 2019).

$$c_t = C_{t-1} \odot f_{t-1} \tag{4.1}$$

$$C_t = c_t + i_t \odot \hat{c}_t \tag{4.2}$$

$$\hat{c}_t = g \times (W_c Xt + U_c h_{t-1} + b_c) \tag{4.3}$$

$$h_t = o_t \odot g(c_t) \tag{4.4}$$

In our case, g represents the activation function, here g is tanh, h_t represents the current hidden state, W and U, represent the weight parameters for input vector X_t and previous hidden state (h_{t-1}), b represents for bias, and $\hat{c}_t$ represents the cell state or candidate memory. The LSTM architecture uses the "propagation through time" method to update the weight parameters during model training (Huang, 2021).

The input, forget, and output gate or control signals of LSTM are expressed in equation (4.5), (4.6), (4.7), respectively. The output gate is derived from the combination of the input and forget gate:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{4.5}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{4.6}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{4.7}$$

4.1.1.2 Gated Recurrent Units

An update gate and a reset gate, which are represented by equations (4.8), (4.9), make up a GRU. GRU has fewer gates than LSTM, but its computing performance is nearly identical to LSTM; in some circumstances, it even outperforms LSTM (Dey & Salem, 2017). Figure 4.3 illustrates how the current input and past memory signals flowing into the GRU network regulate the activation function and recent state updates.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{4.8}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{4.9}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \tag{4.10}$$

$$h_t = g(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{4.11}$$

Working Principles of GRU

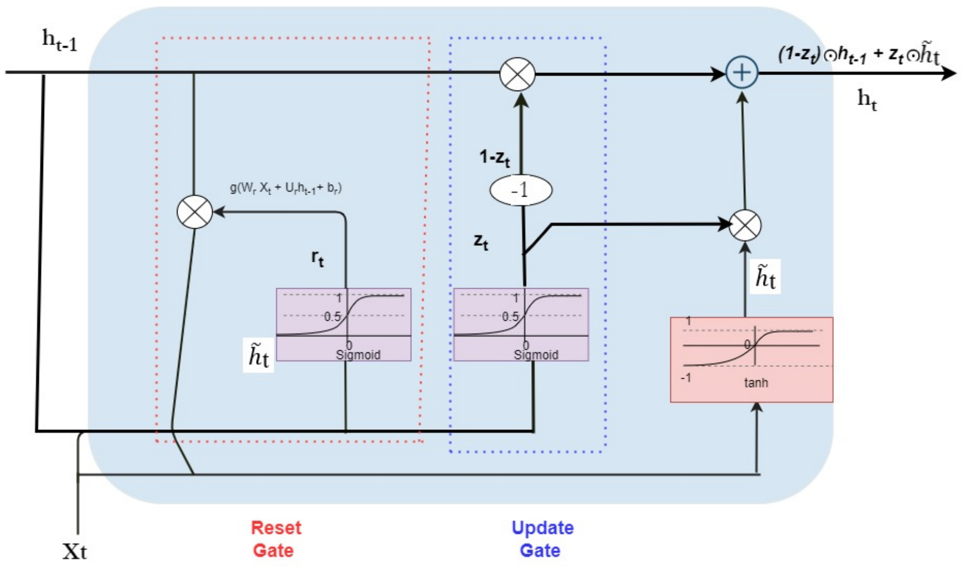


Figure 4.3: GRU structure adopted from Dey & Salem (2017); Dobilas (2022)

The activation function at the reset gate stage of GRU takes the combination of the input vector X_t at time t and the previous hidden state h_{t-1} and then computes the activation function output. The output can be 0, 1 or between 0 and 1 because the output range for the sigmoid function is between 0 and 1. This output decides to the previous hidden (h_{t-1}) state is partially passed, totally passed, or discarded to pass the update state (Dobilas, 2022; YADAV, 2019).

The activation function output is equal to 1, then the previous hidden state (h_{t-1}) is multiplied by 1, and then totally passed to the update gate. On the other hand, the output value between 0 and 1 is then multiplied by the previously hidden state (h_{t-1}) and partially passed to the update gate. The previously hidden state ($h_t - 1$) is not transferred to the update gate, while the output from the activation function is 0.

The update gate z_t value is computed by passing W_z and U_z through the sigmoid activation function. Finally, the output and the new hidden state are computed using W_z and U_z and h_{t-1} as shown in Figure 4.3.

4.1.1.3 Multi-Layer Perceptron

An MLP is a neural network category and a family of feed-forward networks, which is built from three basic layers, as shown in Figure 4.4. These are the input layer, hidden layer, and output layer. As shown in Figure 4.4, input layers that contain the attributes of x_1, x_2, x_3, ..., x_{n-1}, x_n are given as input for a first hidden layer that contains four neurons, and then the output from this layer is also passed as input for the second hidden layer, which contains three neurons, and the output from this layer is given as input for the last output layer in the feed-forward network manager from input to output layers. MLP can be designed with one or more hidden layers with a different number of neurons. The input layer is equivalent to the number of features (attributes) in the dataset, whereas the number of neurons in the hidden layer and output layers varies based on the problem that we need to solve. Mostly, the number of output neurons is decided based on the number of outputs that we need to generate and the activation function applied to output neurons.

Working Principles of MLP

In an MLP neural network model, neurons take the input d)ata set features as input, multiply these dataset features with their respective randomly generated weights, sum up the multiplication result together, and feed this result to the activation function at the hidden layer. Each hidden layer's activation function output was sent forward as input for the subsequent hidden layer or output layer neurons. The error occurred in this model when the model output deviated from the predefined target value, which is the difference between the target value and the model

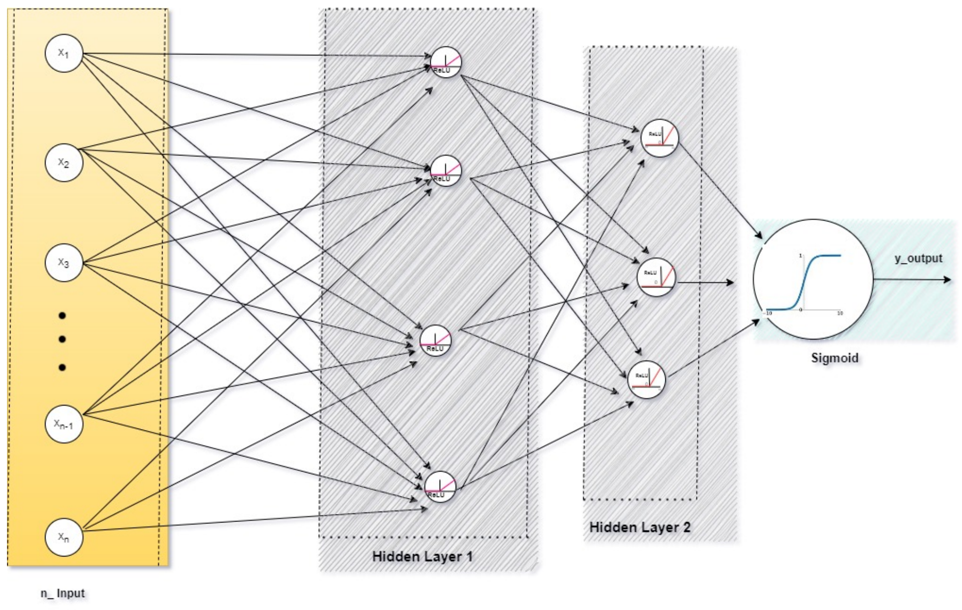


Figure 4.4: MLP architecture

prediction value. During model training, an MLP model employs a backpropagation learning technique to minimize this error (Taud & Mas, 2018). This method is used to tune the input feature weight parameters using the backpropagation algorithm until the model result reaches the expected output. At the final stage, the output layer neurons accept input from preceding layers, perform computation, and generate the final output based on the activation function deployed in them. In our case, the output layer classifies our input data into normal and attack classes.

4.1.2 Random Forest Classifier

A RF classifier is an ensemble machine learning model that combines different decision trees and is used to solve classification problems. A random subset of the dataset's features and a random subset of the dataset itself are utilized to build the decision tree that is used to build a RF. The forest is built using this decision tree construction method repeatedly. The RF classifier model takes the average of each decision tree prediction result to predict the final model output. Due to the ensemble learning method of the RF classifier, it is less prone to overfitting problems compared to a single decision tree (Breiman, 2001). This model is applicable for classification problems, such as fraud detection, image classification, and natural language processing.

4.1.3 Logistic Regression

LR is a statistical analysis method that is used in machine learning models for predicting or classifying classes of input variables in binary tasks. The output for the logistic regression model is either (yes or no) or (1 or 0). A mathematical logistic function is applied in logistic regression to map the probability of input variables between 0 and 1. Based on the mathematical logistic function mapping result, the logistic regression model decides that the output class of all input variable classes is (1 or 0) or (yes or no) (Agresti, 2015). In our case, this model is used to classify the network into attack class (1) or non-attack class (0). For modeling categorization issues, the Python library includes this statistical analysis technique.

4.1.4 Naive Bayes Algorithm

The NB classifier algorithm was created using the "Bayes' theorem. When the independent variables that are utilized to forecast the dependent variable (intended output) have no correlation with one another, this approach performs very well. Equation 4.12 contains the mathematical formula for the NB classifier algorithm (Thakar, 2020):

$$P(_{C|X}) = \frac{P(_{X|C}) \times P(_C)}{P(_X)}, \tag{4.12}$$

where C represents the target class such as *class* 0 and *class* 1 in our case as attack class and normal class, X represents the input or dependant variable with the features of x_1, x_2, x_3, ... x_n, $P(_{C|X})$ is a posterior probability, $P(_C)$ is called the prior, $P(_{X|C})$ is the likelihood which is the probability of predictor given class, and $P(_X)$ is predictor prior probability.

4.1.5 AdaBoost

Adaptive boosting (AdaBoost) is the most known ensemble machine learning method. This algorithm was created by combining multiple weak classification algorithm models to improve the performance of the classification model. The process of creating classifier models during training continues until the expected result is generated (Das et al., 2020).

Working Principles of AdaBoost

Initially, all original datasets were equally weighted to train the first model, and the prediction was done on the same dataset for the first iteration. The second iteration's instance weight update for the second model training and prediction dataset is based on the outcome of the first model's prediction. The dataset instance was misclassified, with the first model prediction weighted higher than another instance during the weight update. Then the weighted updated dataset was used as input for the second model's training and prediction. Similarly, the training dataset weight for the third model was updated by assigning a higher weight to the data instance misclassified by the previous model (Das et al., 2020; Navlani, 2018). This weighted update dataset instance for the new model process is continued until the predetermined model building termination criteria are achieved and the predicting error reaches its expected minimum value. Each successive model focuses on the weaknesses of the preceding models, except for the first model in the first iteration. The final output of the model is the aggregation value of all models built throughout each iteration (Das et al., 2020; Navlani, 2018).

4.1.6 Decision Tree

The DT algortihm is one of the most well-known and effective class of supervised machine learning techniques. It is more suitable for classification and prediction problems. The structure of the DT is hierarchical and constructed from the root node, branch, internal node, and leaf node from top to bottom, respectively. The root node is the starting point for decision tree construction. The root node is selected from the given dataset attribute using an attribution selection measure such as information gain, gain ratio, or Gini index (Priyanka & Kumar, 2020). Branch and internal nodes that correspond to the distinct values of the root node attribute are produced next to the root node selection. The internal node processes the data and sends the decision to the terminal (leaf node). The output stored in the leaf node is the final result of the DT. Between the first internal node and the leaf node of each decision tree, there are one or many branches and internal nodes.

The advantage of the DT is that it has an easily understandable structure, works with datasets that contain missing values, requires little computational effort and relatively high performance, and is applicable for feature selection, clustering, regression, and classification in the areas of business, medicine, industry, intelligent vehicles, remote sensing, and so on.

4.2 Data Collection and Preprocessing

4.2.1 Dataset Description

The dataset is the heart of the machine learning and deep learning algorithm-based model. The presence of the dataset is not only enough to model an efficient machine learning system, but we also, rather than its presence, have to focus on the quality of the dataset for the given scenario. For intrusion detection systems, there are numerous datasets. Some of the datasets for are as follows: UNSW-NB15, CICIDS2017, CSE-CIC-IDS2018, BoTNeTIoT-L01, KDD98, DARPA98, KDDCUP99, NSLKDD, and others (Moustafa & Slay, 2015), but all datasets do not include the most recent varieties of attacks.

In this thesis, I used the two most recent publicly available datasets. These datasets (CICIDS2007 and UNSW-NB15) include variant attacks compared to previous datasets. The variants in CICIDS2017 are DoS, DDoS, Brute Force, XSS, SQL Injection, Infiltration, Port scan, and Botnet and the variants in UNSW-NB15 are generic, fuzzers, analysis, backdoor, exploit, reconnaissance, shellcode, and worm. Thus, these datasets include the most recent attack variants for intrusion detection systems.

A. **UNSW-NB15 dataset**

The UNSW-NB15 dataset was created by (Moustafa & Slay, 2016; Janarthanan & Zargari, 2017) at the Australian Center for CyberSecurity by using the IXIA traffic generator. To generate the CSV files from raw network traffic Pcap files with respective feature names, they used Argus and Bro-IDS Tools, SQL Server 2008, and 12 algorithms developed with the C# programming language. The generated CSV file contains seven CSV files. The first four CSV files UNSW-NB15_1.csv, UNSWNB15_2.csv, UNSW-NB15_3.csv, and UNSW-NB15_4.csv, contain the complete UNSW-NB15 dataset records. The ground truth table is named UNSW-NB15_GT.CSV, which contains the labels of attack (Moustafa & Slay, 2015), the description of all features is named UNSW-NB15_features.csv, and the list of events file is called UNSW-NB15_LIST_EVENTS, which contains a list of events in all datasets. The first four CSV files above (Moustafa & Slay, 2016), include the complete UNSW-NB15 dataset. The first four CSV files contain 2,540,044 records with 49 features, including labels. According to the research (Moustafa & Slay, 2016) detailed description, these features are divided into five categories: flow features, basic features, time features, content features, and additional generated features. In addition to this, there are also UNSW-NB15_training-set.csv and UNSW-NB15_testing-set.csv files prepared from the UNSW-NB15 dataset. The total size of the raw data set is 100 GB. This data set includes the most recent attack varieties, categorized into nine main categories: generic, fuzzers, analysis, backdoor, exploit, reconnaissance, shellcode, and worm.

B. **Dataset CICISD 2017**

The Canadian Institute of Cybersecurity creates different types of datasets for cyber security researchers: IoT datasets, malware datasets, DNS datasets, dark web datasets, intrusion detection datasets, and ISCX datasets. The CICIDS2017 is one of the intrusion detection datasets provided by the Canadian Institute of Cybersecurity that is publicly available for intrusion detection system purposes (of New Brunswick, 2017; Sharafaldin et al., 2018). Many academics choose to use this dataset to test their intrusion detection models because it contains various recent cyber attack types. Brute force attacks, Heartbleed attacks, botnet attacks, DoS attacks, DDoS attacks, web attacks, and infiltration attacks are among the attacks in this data collection that are organized by name. The description of each attack group is in Table 4.1. Eight distinct CSV file names with normal and attack variants are generated within five days and are included in the CICIDS2017 dataset. In total, 2,830,743 records with 79 attributes, including target features, are present.

4.2.2 Data Preprocessing

The data preprocessing stage is one of the most challenging aspects of modeling and assessing deep learning and machine learning algorithms. Figure 4.5 illustrates the typical preprocessing stage that was used for both datasets in this thesis.

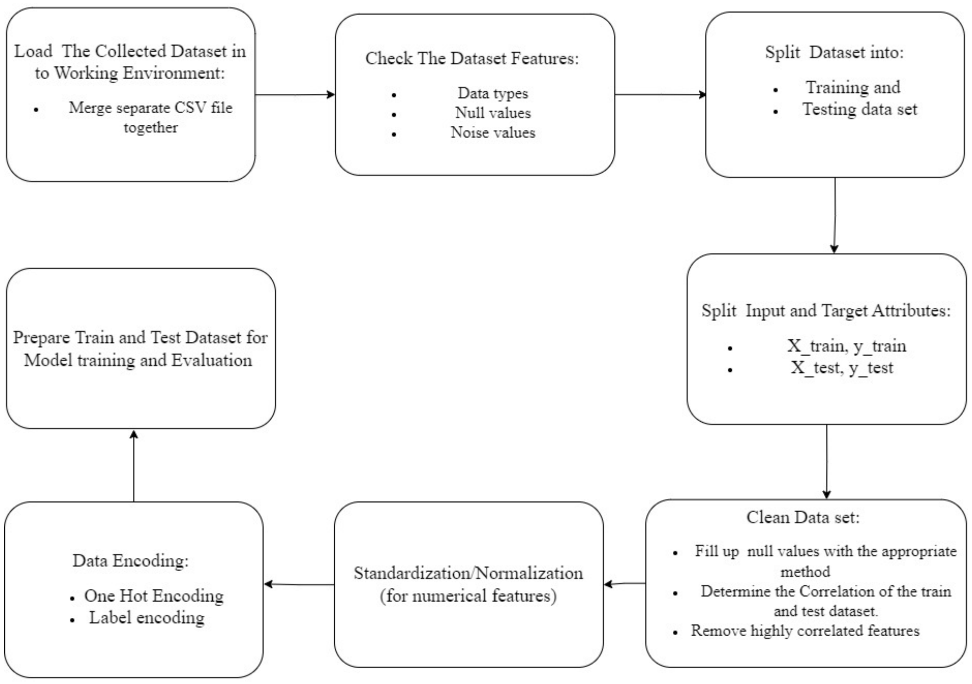

Load The Collected Dataset in to Working Environment:
Merge separate CSV file together
Check The Dataset Features:
Data types
Null values
Noise values
Split Dataset into:
Training and
Testing data set
Split Input and Target Attributes:
X_train, y_train
X_test, y_test
Clean Data set:
Fill up null values with the appropriate method
Determine the Correlation of the train and test dataset.
Remove highly correlated features
Standardization/Normalization (for numerical features)
Data Encoding:
One Hot Encoding
Label encoding
Prepare Train and Test Dataset for Model training and Evaluation

Figure 4.5: Data preprocessing phases

A. **Feature Scaling Technique**

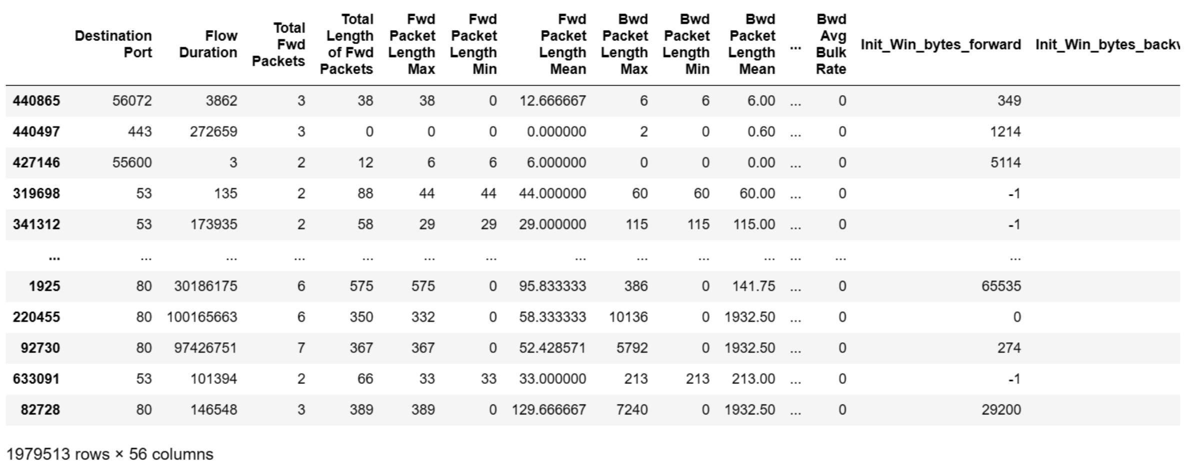
Normalization and standardization feature scaling techniques are used for data preprocessing in machine learning and statistics to transform numeric data values in the dataset to the required scales and ranges of data formats.

Normalization

Normalization is the process of converting the value of an attribute into a range between 0 and 1. It is called a min-max scaler. It is preferable for datasets that have a wider range between minimum and maximum values. It is computed as follows:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.13}$$

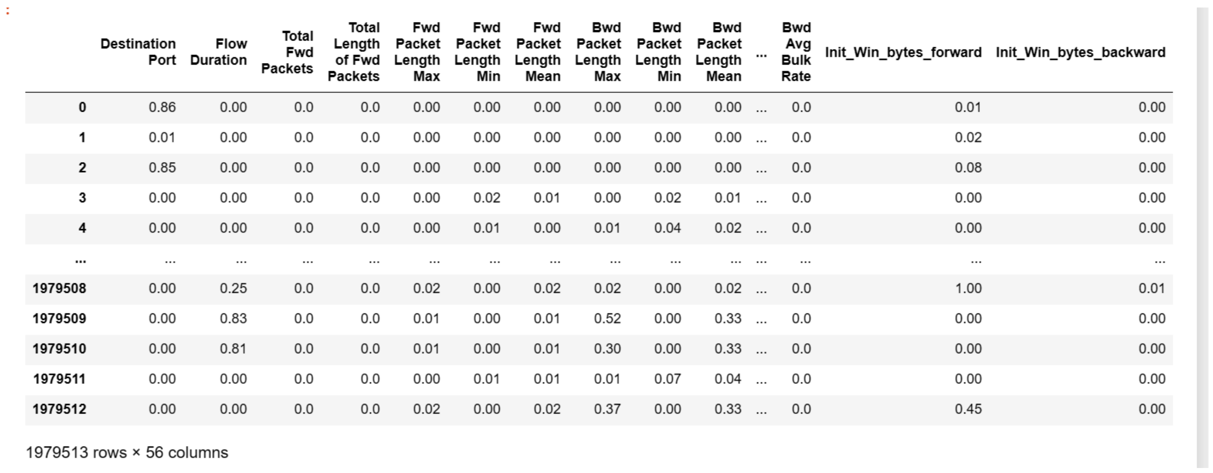
where X_{max} is the maximum value and X_{min} is the minimum value of the feature in the dataset, respectively. X is the feature value to be scaled and X_{norm} is the new equivalent value of X after normalization. Figure 4.6 depict how the datasets appeared before and after normalization, respectively.



	Destination Port	Flow Duration	Total Fwd Packets	Total Length of Fwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Bwd Packet Length Max	Bwd Packet Length Min	Bwd Packet Length Mean	...	Bwd Avg Bulk Rate	Init_Win_bytes_forward	Init_Win_bytes_back
440865	56072	3862	3	38	38	0	12.666667	6	6	6.00	...	0	349	
440497	443	272659	3	0	0	0	0.000000	2	0	0.60	...	0	1214	
427146	55600	3	2	12	6	6	6.000000	0	0	0.00	...	0	5114	
319698	53	135	2	88	44	44	44.000000	60	60	60.00	...	0	-1	
341312	53	173935	2	58	29	29	29.000000	115	115	115.00	...	0	-1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1925	80	30186175	6	575	575	0	95.833333	386	0	141.75	...	0	65535	
220455	80	100165663	6	350	332	0	58.333333	10136	0	1932.50	...	0	0	
92730	80	97426751	7	367	367	0	52.428571	5792	0	1932.50	...	0	274	
633091	53	101394	2	66	33	33	33.000000	213	213	213.00	...	0	-1	
82728	80	146548	3	389	389	0	129.666667	7240	0	1932.50	...	0	29200	

1979513 rows × 56 columns

(a) Dataset before normalization



	Destination Port	Flow Duration	Total Fwd Packets	Total Length of Fwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Bwd Packet Length Max	Bwd Packet Length Min	Bwd Packet Length Mean	...	Bwd Avg Bulk Rate	Init_Win_bytes_forward	Init_Win_bytes_backward
0	0.86	0.00	0.0	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.01	0.00
1	0.01	0.00	0.0	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.02	0.00
2	0.85	0.00	0.0	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.08	0.00
3	0.00	0.00	0.0	0.0	0.00	0.02	0.01	0.00	0.02	0.01	...	0.0	0.00	0.00
4	0.00	0.00	0.0	0.0	0.00	0.01	0.00	0.01	0.04	0.02	...	0.0	0.00	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1979508	0.00	0.25	0.0	0.0	0.02	0.00	0.02	0.02	0.00	0.02	...	0.0	1.00	0.01
1979509	0.00	0.83	0.0	0.0	0.01	0.00	0.01	0.52	0.00	0.33	...	0.0	0.00	0.00
1979510	0.00	0.81	0.0	0.0	0.01	0.00	0.01	0.30	0.00	0.33	...	0.0	0.00	0.00
1979511	0.00	0.00	0.0	0.0	0.00	0.01	0.01	0.01	0.07	0.04	...	0.0	0.00	0.00
1979512	0.00	0.00	0.0	0.0	0.02	0.00	0.02	0.37	0.00	0.33	...	0.0	0.45	0.00

1979513 rows × 56 columns

(b) Dataset after normalization

Figure 4.6: Dataset UNSW-NB15 before and after normalization

Standardization

Feature scaling is applied only to dataset features that have numerical values. This is done to make sure that each feature contributes equally to the final output. The process of transforming all datasets to have a mean of 0 and a standard deviation of 1 is known as feature standardization as shown in figure 4.7. The value of the attribute is not constrained within a range, unlike normalization. These techniques are mostly applicable for data that behaves in a Gaussian (bell curve) distribution, where the distribution concentrates around the mean of the data. It is called a standard scaler. It is computed as follows:

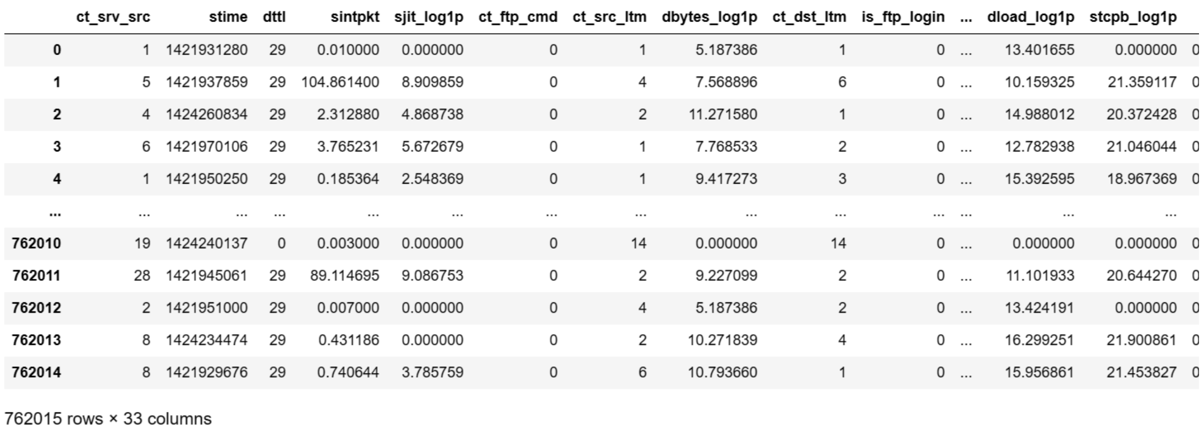
$$X_{std} = \frac{X - \mu}{\sigma} \tag{4.14}$$

where σ standard deviation of the feature value, μ is the mean of the feature value, X is the feature value to be scaled, and X_std is the new equivalent value of X after standardization. If the feature is not called for model training, the feature that has a higher value has a higher probability of getting a higher weight factor(Mohmand et al., 2022)

B. **Feature Selection - Correlation Formula and Explanation**

All attributes in the dataset do not have the same significance level to influence the model output. Most attributes have higher significance; some do not (Reddy et al., 2020). So that by removing or dropping these fewer significant attributes, we can improve the model's importance and reduce the training time of the model. In this work, we use the person correlation dimension reduction method to reduce less important attributes in our dataset.

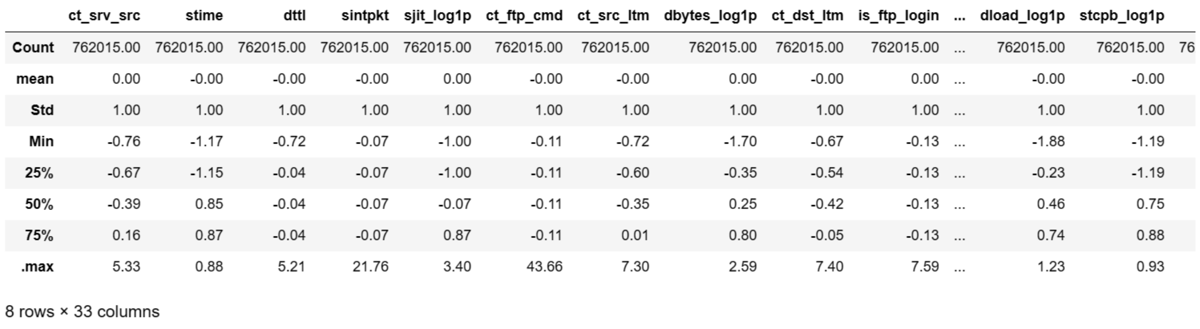
The Pearson correlation coefficient is used to identify the relationship between two variables or attributes in the dataset. These two variables are highly associated with one another if their correlation coefficients are close to 1 for positive linear relations and -1 for negative linear relations, respectively. Due to the correlation coefficient showing that each variable's relevance is similar and that one variable is sufficient for model training, we can delete one of the variables from the dataset. Before removing a variable, it is important to verify its relationship to the desired model result. If each variable is highly related to the target



	ct_srv_src	stime	dttl	sintpkt	sjit_log1p	ct_ftp_cmd	ct_src_ltm	dbytes_log1p	ct_dst_ltm	is_ftp_login	...	dload_log1p	stcpb_log1p	
0	1	1421931280	29	0.010000	0.000000	0	1	5.187386	1	0	...	13.401655	0.000000	0
1	5	1421937859	29	104.861400	8.909859	0	4	7.568896	6	0	...	10.159325	21.359117	0
2	4	1424260834	29	2.312880	4.868738	0	2	11.271580	1	0	...	14.988012	20.372428	0
3	6	1421970106	29	3.765231	5.672679	0	1	7.768533	2	0	...	12.782938	21.046044	0
4	1	1421950250	29	0.185364	2.548369	0	1	9.417273	3	0	...	15.392595	18.967369	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
762010	19	1424240137	0	0.003000	0.000000	0	14	0.000000	14	0	...	0.000000	0.000000	0
762011	28	1421945061	29	89.114695	9.086753	0	2	9.227099	2	0	...	11.101933	20.644270	0
762012	2	1421951000	29	0.007000	0.000000	0	4	5.187386	2	0	...	13.424191	0.000000	0
762013	8	1424234474	29	0.431186	0.000000	0	2	10.271839	4	0	...	16.299251	21.900861	0
762014	8	1421929676	29	0.740644	3.785759	0	6	10.793660	1	0	...	15.956861	21.453827	0

762015 rows × 33 columns

(a) Dataset before standardization



	ct_srv_src	stime	dttl	sintpkt	sjit_log1p	ct_ftp_cmd	ct_src_ltm	dbytes_log1p	ct_dst_ltm	is_ftp_login	...	dload_log1p	stcpb_log1p	
Count	762015.00	762015.00	762015.00	762015.00	762015.00	762015.00	762015.00	762015.00	762015.00	762015.00	...	762015.00	762015.00	76
mean	0.00	-0.00	-0.00	-0.00	0.00	-0.00	-0.00	0.00	-0.00	0.00	...	-0.00	-0.00	
Std	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	...	1.00	1.00	
Min	-0.76	-1.17	-0.72	-0.07	-1.00	-0.11	-0.72	-1.70	-0.67	-0.13	...	-1.88	-1.19	
25%	-0.67	-1.15	-0.04	-0.07	-1.00	-0.11	-0.60	-0.35	-0.54	-0.13	...	-0.23	-1.19	
50%	-0.39	0.85	-0.04	-0.07	-0.07	-0.11	-0.35	0.25	-0.42	-0.13	...	0.46	0.75	
75%	0.16	0.87	-0.04	-0.07	0.87	-0.11	0.01	0.80	-0.05	-0.13	...	0.74	0.88	
.max	5.33	0.88	5.21	21.76	3.40	43.66	7.30	2.59	7.40	7.59	...	1.23	0.93	

8 rows × 33 columns

(b) Dataset after standardization

Figure 4.7: Dataset CICIDS 2017 before and after standardization

value, we have to keep it as it is. The person correlation coefficient mathematical equation is represented as in equation 4.15.

$$r = \frac{\sum(X_i - \bar{X}) \times (Y_i - \bar{Y})}{\sqrt{\sum((X_i - \bar{X}))^2 \times \sum((Y_i - \bar{Y}))^2}}, \tag{4.15}$$

where r represents Pearson correlation coefficient, X_i represents x variable samples, $\bar{X}$ represents the mean of value in x variable, Y_i represents y variable samples, and $\bar{Y}$ represents the mean of value in y variable.

C. **Encoders**

Encodes in machine learning and deep learning models are used for converting multiple-column data types into numerical data types. Label encoders and One Hot encoder are the most popular categorical encoders used for it. In this thesis, I used the label encoder for CICIDS2017 dataset categorical features, which is the target ('Label') attribute value to decode into numerical data. And also, I used one hot encoder for the USNW-NB15 dataset to convert categorical features or attributes (proto, state service) into numerical data. The reason for using one-hot encoding for it is that the features are not expressed in an ordinary way, and the one-hot encoder is suitable for non-ordinary data types (Brownlee, 2020).

4.2.2.1 UNSW-NB15 Dataset Preprocessing

The preprocessing of this dataset started with merging four separate CSV files into a single file, which made it easy to preprocess the whole dataset. The first step after merging is to split the dataset into training and testing subsets. Then check the missing data set and ensure that each feature data type aligns with the information provided by the dataset creator, as shown in table 4.2 and adjust the data according to the given information.

Dataset split into train and test and handle missing and null values in dataset

The dataset was split into 70% train and 30% test. After splitting the training and testing datasets of the merged USNW-NB15 dataset, there are some missing values and incorrect data type representations. The training dataset features named "ct_flw_http_mthd“ contain 943,876 null values in the training dataset and 404,381 null values in the test dataset, "is_ftp_login“ contains 1,001,037 null values in the training dataset and 428,849 null values in the testing dataset, and "attack_cat“ contains 1,552,862 null values and 665,706 null values in test dataset. A number of flow methods such as Get and Post in the http service (ct_flw_http_mthd) 1,348,257 null values substituted by 0 or 1 Moustafa & Slay (2016). Therefore, I added 0 to the value that was missing.

On the other hand, the features in the training dataset are named with "is_ftp_login“ (if the ftp session is accessed by the user and password, then 1 otherwise, 0), which is a binary datatype. So that 1,429,886 null values within this feature name of a column were filled up with 0 Moustafa & Slay (2016, 2015). Under the attack category, 2,218,568 null records of (attack_cat) were filled up with the nominal data type "normal", which indicates the value of the nonattack records. Originally "is_sm_ips_ports" and "is_ftp_login" are binary data types but assigned as numerical, and also "ct_ftp_cmd", is categorized as a nominal data type but it is a numerical data type as presented in Table 4.2.

Source IP address, source port number, destination IP address, destination port number, and attack category named with srcip, sport, dstip, dsport, attack_cat respectively were not important features for network attack detection in this scenario. In addition to this, after analyzing the correlation between features such as sloss, dloss, dpkts, dwin, ltime, ct_srv_dst, ct_src_dport_ltm, and ct_dst_src_ltm were dropped from the training and testing dataset.

4.2.2.2 CICISD 2017 Dataset Preprocessing

From the total instance count of 2,830,743 in the CICIDS 2017 dataset, there were 1358 null instances. The number of instances in this instance is very small compared to the main dataset, so we dropped it. After dropping it, 2,827,876 instances left in the dataset. Then, the dataset was split into 70% train and 30%. Attacks that were small in number and had similar behaviors have been grouped into single groups by renaming the attack label (Panigrahi & Borah, 2018). So that I relabeled web-based attacks such as Web Attack - XSS, Web Attack - Sql Injection into Web Attack, FTP-Patator, SSH-Patator, Brute Force as Brute Force, DoS GoldenEye, DoS Hulk, DoS Slowhttptest, DoS slowloris into DoS and finally, we have the static records of the cleaned dataset presented with Table 4.5.

Table 4.1: Attack categories description in both datasets

Attack Type	**Attacking Mechanism**	**Dataset**
Fuzzers	Identifying security loop hoes and send random data through it to crash the system such as operating system, program or network	UNSW-NB15
Generic	A type of penetration attack through spam emails, port scan, web script	UNSW-NB15
Backdoor	Unauthorized access of systems using uncovered security gap of device	UNSW-NB15
Exploit	Attack performed by using fault of a computer program, system, and machine	UNSW-NB15
Reconnaissance	Collect information that is used to control computers	UNSW-NB15
Shellcode	penetrates a slight piece of code starting from a shell to control the compromised machine	UNSW-NB15
Worm	It replicated itself using a computer network to control the victim computer	UNSW-NB15
Brute Force attacks	Web application attack without using password	CICIDS2017
Heartbleed attack	Transport layer security issue exploited by malformed heartbeat	CICIDS2017
Botnet	Setal data, send spam, create the accessibility of device for attackers through interconnected device	CICIDS2017
DoS attacks	Disrupt the operation of machines and networks temporarily by making them unavailable the resource or making the system busy and disrupting authorized users	UNSW-NB15 & CICIDS2017
DDoS attack	Generating huge traffic and sending multiple requests to the victim system to make it out of control	CICIDS2017
Web attack	Which is performed in the form of SQL injection, brute force over http to discover the password, Cross-Site Scripting (XSS)	CICIDS2017
Infiltration attacks	Performed by finding vulnerabilities in the software installed on the computer and performing backdoor attacks such as IP sweep, Full port scan, etc.	CICIDS2017

Table 4.2: UNSW-NB15 Dataset statistics

Dataset Class Name	**Number of Instance in each Class**
Normal	2218764
Attack	
Generic	215481
Exploits	44525
Fuzzers	24246
DoS	16353
Reconnaissance	13987
Analysis	2677
Backdoor	2329
Shellcode	1511
Worms	174
Missing Data Feature Name	
ct_flw_http_mthd	1348145
is_ftp_login	1429879
attack_cat	2218764
Number of total features	49
Data type in dataset	
Categorical(Nominal)	srcip, sport, dstip, dsport, proto, state service, attack_cat
Binary	is_sm_ips_ports, is_ftp_login, label
Numerical	the rest of 37 features are numerical

Table 4.3: CICIDS2017 Dataset Statistics

File Names	**Dataset Class**	**No.of Records in Class**	**No.of Total Records**
Monday-WorkingHours.pcap_ISCX.csv	BENIGN	529918	529918
Tuesday-WorkingHours.pcap_ISCX.csv	BENIGN FTP-Patator SSH-Patator	432074 7938 5897	445909
Wednesday-workingHours.pcap_ISCX.csv	BENIGN DoS Hulk DoS GoldenEye DoS slowloris DoS Slowhttptest Heartbleed	440031 231073 10293 5796 5499 11	692703
Thursday-WorkingHours Morning-WebAttacks.pcapISCX_.csv	BENIGN Web Attack - Brute Force Web Attack - XSS Web Attack - Sql Injection	168186 1507 652 21	170366
Thursday-WorkingHours Afternoon-Infilteration.pcap.ISCX.csv	BENIGN Infiltration	288566 36	288602
Friday-WorkingHours Morning.pcap_ISCX.csv	BENIGN Bot	189067 1966	191033
Friday-WorkingHours-Afternoon PortScan.pcap_ISCX.csv	BENIGN PortScan	127537 158930	286467
Friday-WorkingHours-Afternoon DDos.pcap_ISCX.csv	BENIGN DDoS	97718 128027	225745
Total		**2,830,743**	**2,830,743**

Table 4.4: Train and test dataset (UNSW-NB15)

Normal and attack class category	Number of records	Train dataset	Test dataset records
Normal	2218764	155330	665706
Generic	215481	151011	64470
Exploits	44525	31182	13343
Fuzzers	24246	16876	7370
DoS	16353	11419	4934
Reconnaissance	13987	9779	4208
Analysis	2677	1857	820
Backdoor	2329	1671	658
Shellcode	1511	1054	457
Worms	174	125	49

Table 4.5: Train and test dataset (CICIDS2017)

Dataset class category	Number of records	Train dataset	Test data set
BENIGN	2271320	1590306	681014
DoS	251712	175867	75845
PortScan	158804	110968	47836
DDoS	128025	89718	38307
Brute Force	13832	9685	4147
Web Attack	2180	1546	634
Bot	1956	1392	564
Infiltration	36	26	10
Heartbleed	11	5	6

4.5(a) Train & test dataset split with random-state=42

Dataset class category	Number of records	Train dataset	Test data set
BENIGN	2271320	1589931	681389
DoS	251712	175854	75858
PortScan	158804	111396	47408
DDoS	128025	89761	38264
Brute Force	13832	9653	4179
Web Attack	2180	1513	667
Bot	1956	1374	582
Infiltration	36	26	10
Heartbleed	11	5	6

4.5(b) Train & test dataset split with random-state=1

Chapter 5 Evaluation and Discussion of Results

5.1 Expermental Parameters and Environmental Setup

5.1.1 Simulation Environment Setup

Software and hardware environments used for simulating the proposed DNN-based model for detecting IoT cyber attacks are presented as follows: The hardware used for it was a Lenovo IdeaPad 5 14ITL05 laptop, which consists of an 11th Gen Intel(R) Core(TM) i5-1135G7 CPU (2.40 GHz), 16 GB of RAM, and runs on 64-bit Microsoft Windows 11 Home. The software, we used for the simulation was Python 3.11 with Jupyter notebook as an editor.

5.1.2 Model Parameter Configuration

The model parameters configuration of all classical machine learning, ensemble methods, and deep learning models used in this thesis are presented in this section. For the experiments, I used a similar model configuration for the UNSW-NB15 and CICIDS2017 datasets.

The performance of a deep learning model is affected by model parameters, such as the number of hidden layers, number of neurons in each hidden layer, epochs, activation function, and loss function, used by the model during model training. During the experiments, I focused on the parameters of the number of epochs, batch size, number of hidden layers, and number of neurons in each hidden layer. I created four different setups with different parameters to select suitable parameters for our model and the dataset used for the experiment for DNN models. In the first setup, two hidden layers were used with 20 and 20 neurons for each layer and an epoch number of 200 by varying the batch size to 1024 and 2048. For the second setup, two hidden layers were employed, with the first hidden layer having 20 neurons and the second hidden layer having 10 neurons, the number of epochs equals 150, and we vary the batch size to 1024 and 2048. The third experiment setup was with two hidden layers, with each layer having 20 and 15 neurons, a batch size of 1024, and an epoch number equal to 150. The final experiment set up was one hidden layer with 10 and 10 neurons, with a batch size of 1024 and an epoch number equal to 150. Batch size and epoch number have a significant effect on model training time and performance (McCandlish et al., 2018; Aldin & Aldin, 2022). Based on the experiment's results, I selected the experiment parameters illustrated in Tables 5.1 and 5.2 for our work. For the MLP model with Scikit-Learn, we used the default number of epochs for all experiments.

A *Deep leaning model parameters*

The DNN model for attack classification used in this thesis is modeled with Scikit Learn and Keras libraries. The setup of the MLP, which was used with both libraries, is shown in Table 5.1. In addition to this configuration, I simulated this model using single hidden layers with 10 and 20 neurons. The remaining DNN algorithms, LSTM and GRU, were simulated using the Keras library with one hidden layer and two hidden layers, respectively. Table 5.2 shows how the model was set up with two hidden layers.

Table 5.1: MLP with Scikit learn and MLP with Keras libraries

Parameters	**MLP (Scikit learn)**	**MLP (Keras)**
hidden_layer_sizes	2 layers with (20,10) neurons	2 layers with (20,10) neurons
activation	ReLU	ReLU
solver	Adam	Adam
random_state	1	1
batch_size	1024	1024
epoch	-	150
max_iter	200	-

Table 5.2: LSTM and GRU with Keras library

Parameters	**LSTM**	**GRU**
hidden_layer_sizes	2 layers :(20,10) neurons	2 layers: (20,10) neurons
return_sequence	True	True
activation	sigmoid	sigmoid
optimizer	Adam	Adam
loss	binary_crossentropy	binary_crossentropy
batch_size	1024	1024
epoch	150	150
verbose	2	2

B *Classical machine learning algorithms and ensemble method model parameters*

I modeled conventional and ensemble machine learning models for attack classification using the Scikit learn library. The configuration parameters for modeling DT, AdaBoost, RF, LR, and NB models are illustrated in Tables 5.3 and 5.4.

Table 5.3: Decision tree and random forest parameters configuration

Parameters	**DT**	**RF**
criterion	gini	gini
max_depth	None	None
min_samples_leaf	1	1
min_samples_split	2	2
random_state	1	0
max_leaf_nodes	None	-
n_estimators	-	100
max_features	None	auto

Table 5.4: AdaBoost, logistic regression and Naive Bayes parameters configuration

AdaBoost		**Logistic regression**		**Naive Bayes**	
base_estimator	DT	max_iter	100	priors	None
max_depth	1	solver	lbfgs	var_smoothing	1e-09
learning_rate	1	Verbose	0		
random_state	96	n_jobs	none		
n_estimators	50	penalty	l2		
algorithm	SAMME.R				

5.2 Result Evaluation Metrics

The performance of machine learning models was evaluated with different metrics. Accuracy, recall, precision, F1-score, and FPR were used in this research (Vinayakumar et al., 2019). To calculate the metrics, the following values from confusion matrix are used:

True Positive (TP):- The number of non-attack instances classified as an exactly non-attack class.

True Negative (TN):- The number of attack instances classified as exact attack class.

False Positive (FP):- The number of attack instances classified as non-attack class i.e., assigned in incorrect class. In the original dataset, the instance or record is a group of attacks but the model is classified as a non-attack class.

False Negative (FN):- The number of non-attack instance classified as attack class.The model misclassified non-attack instances into attack classes.

Then, the evaluation criteria are defined as follows:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{5.1}$$

If the number of instances of false positives and false negatives is low, the accuracy is high.

$$Precision = \frac{TP}{TP+FP} \tag{5.2}$$

If a false positive is low the precision is high, this indicates the non. Attack instances are classified as non-attack classes.

$$Recall = \frac{TP}{TP+FN} \tag{5.3}$$

The recall is the ratio of the true positive class to the total number of real true classes in the original instance.

$$F-measure(F1-score) = 2 \times \frac{Recall \times Precison}{Recall + Precision}, \tag{5.4}$$

has the value between 0 and 1. Precision is low, either precision or recall is low.
False positive rate (FPR) of a classification model can be obtained as follows (Powers, 2020):

$$FPR = \frac{FP(\text{attack considered as normal})}{N(\text{All negative instance in dataset})} = \frac{FP}{FP+TN} \tag{5.5}$$

5.3 Results of Modeling

In this section, I present the attack classification performance of classical and ensemble machine learning and DNN models.

5.3.1 Classical and Ensemble ML Model Attack Classification

5.3.1.1 Classical and Ensemble ML Model Attack Classification Using CICIDS2017 Dataset

As shown in Table 5.5, classical and ensemble model results were generated by varying the random_state parameters of neural network parameters. The train and test split of the dataset with different values of the random_state parameters is presented in Table 4.5.

The random_state parameters in the machine learning algorithm and DNN model are used to create the same and consistent number of train and test datasets. This parameter is used to ensure the consistency of model results whenever the algorithm is executed with the specified value of the random_state parameter. However, if we change the value of it the similarity of each record in train and test data is not consistent. The newly created instances for the train and test datasets are not the absolutely same in type for different values of the random_state parameter, but the overall number of records that exist in train and test datasets are similar. The variation of the train and test dataset due to the random_state parameter value is shown in Table 4.5.

As we have seen in Table 5.5, the RF model outperforms than other models it scores 99.897% and 99.47% in terms of accuracy, recall, precision, and F1-score and it also scores 0.000913 and 0.0008 in FPR, 396.3 seconds, and 587.70 seconds to train models in both configuration random_state = 42 and random_state = 1 respectively. While the decision trees model is better next to the random forests model. It scores 99.864% in accuracy, recall, precision, and F1-score, 0.0008 in FPR, and 49.2 seconds to train the model at random_state=42 but random_state=1 AdaBoost model is the second. In terms of time to train the model decision tree is better than the random forest and AdaBoost model.

As shown in Table 5.5(a), NB models perform less than others in terms of all model performance metrics except model training time. The reason this model performs less than others is the nature of the dataset containing more correlated input attribute features, which is not suitable for the NB algorithm as discussed in 4 section 4.1.4.

The following Figures 5.1 and 5.2 confusion matrix show true positive (TP) true negative

Table 5.5: Conventional and ensemble ML algorithm model result using different data split configuration

Model	Accuracy	Recall	Precision	F1-Score	FPR	Time to Train Model in Second
DecisionTree	99.867%	99.867%	99.867%	99.867%	0.00084	49.2
AdaBoost	98.856%	98.108%	96.171%	97.130%	0.009599	230.6
Random Forest	99.897%	99.897%	99.897%	99.897%	0.000913	396.3
Logistic Regression	92.252%	92.252%	92.069%	92.124%	0.038068	18.8
Naive Bayes	59.730%	59.730%	85.671%	49.510%	0.632595	1.8

5.5(a) DT, AdaBoost, Random Forest, Logistic Regression and Naive Bayes at random_state=42

Model	Accuracy	Recall	Precision	F1-Score	FPR	Time to Train Model in Second
DecisionTree	96.08%	96.08%	96.12%	96.08%	0.0057	71.76
AdaBoost	97.19%	90.80%	94.71%	92.71%	0.0124	359.82
Random Forest	99.47%	99.47%	99.47%	99.47%	0.0008	587.70
Logistic Regression	92.24%	92.24%	92.06%	92.11%	0.0377	22.28
Naive Bayes	80.32%	80.32%	84.19%	71.55%	0.0000	2.51

5.5(b) DT, AdaBoost, Random Forest, Logistic Regression and Naive Bayes at random_state=1

(TN), false positive (FP), and false negative (FN) of the decision tree and random forest respectively. From the 848363 test data set 680439 predicted as TP, 166798 predicted as TN, 575 predicted as FP and 551 predicted as FN instances by a decision tree, and 680392 predicted as TP, 167097 predicted as TN, 622 predicted as FP and 252 predicted as FN by the RF model.

5.3.1.2 Classical and Ensemble ML Models Attack Classification Using UNSW-NB15 Dataset

As we have seen, DTs and RFs outperform other models when used to classify attacks using the CICIDS 2017 dataset. These models also performed better when used to classify attacks than other models used in this thesis on the UNSW-NB15 dataset. But other models also showed some improvement compared to the CICIDS2017 dataset attack classification. The RF and DT models scored 99.68% and 99.34% in accuracy, recall, precision, F1-score, 0.0019 and 0.0046 in FPR and 1281.1 seconds and 63.5 seconds in model training time, respectively. Since the RF algorithm model is more complex than the DT model, its training time is longer.

Table 5.6: Conventional and ensemble method model output using UNSW-NB15 dataset

Model	Accuracy	Recall	Precision	F1-Score	FPR	Time to Train Model in Second
DecisionTree	99.34%	99.34%	99.35%	99.34%	0.0046	63.5
AdaBoost	99.09%	96.49%	96.22%	96.35%	0.0055	252.3
Random Forest	99.68%	99.68%	99.68%	99.68%	0.0019	1281.4
Logistic Regression	98.93%	98.93%	98.95%	98.93%	0.0085	21.3
Naive Bayes	88.84%	88.84%	90.00%	84.79%	0.0002	8.1

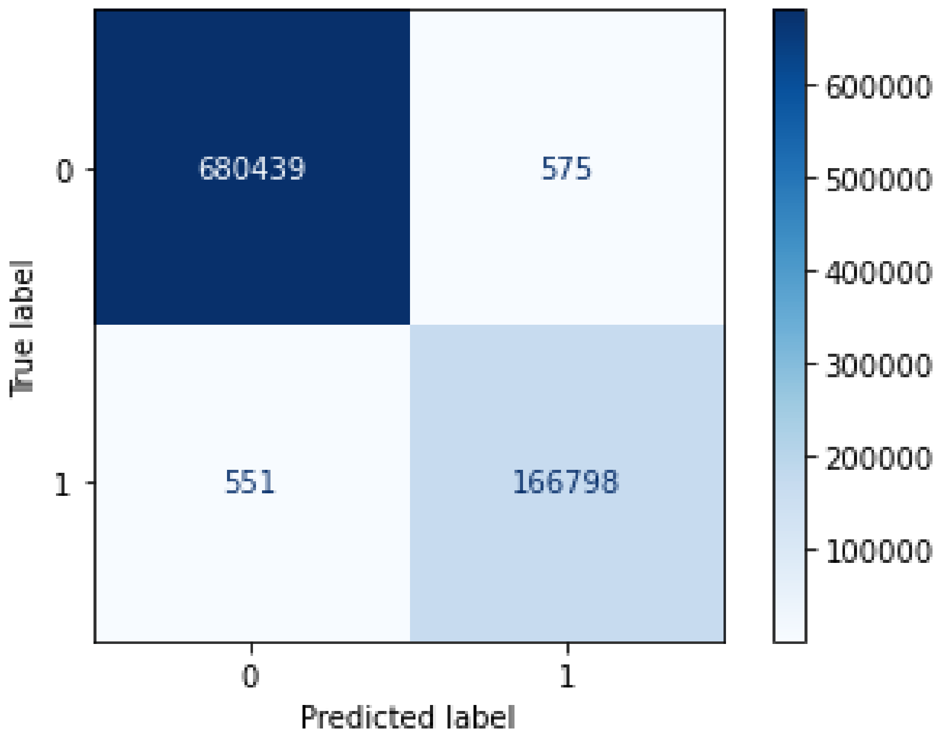


Figure 5.1: Confusion matrix for decision tree on CICIDS2017

The following Figure 5.3 shows the confusion matrix of the RF model using the UNSW-NB15 dataset. From 762015 instances, 2452 instances were misclassified which is 1266 instances is predicted as false positive and 1186 is predicted as false negative.

5.3.2 Deep Neural Network Model Attack Classification

5.3.2.1 Deep Neural Network Model Evaluation Using CICIDS2017 Dataset

In this section, I discuss the DNN models' attack classification performance using the CICIDS 2017 dataset. The performance of DNN models was mainly affected by the number of hidden layers, the number of neurons in each layer, the number of epochs, the batch size, and the learning rate. In this thesis, I present different results by a varying number of hidden layers and neuron parameters of the model.

MLP, LSTM, and GRU neural network models with one hidden layer with 10 and 20 neurons, two hidden layers 20 and 10 neurons attack classification performance was evaluated based on the parameters illustrated in Table 5.1 and 5.2.

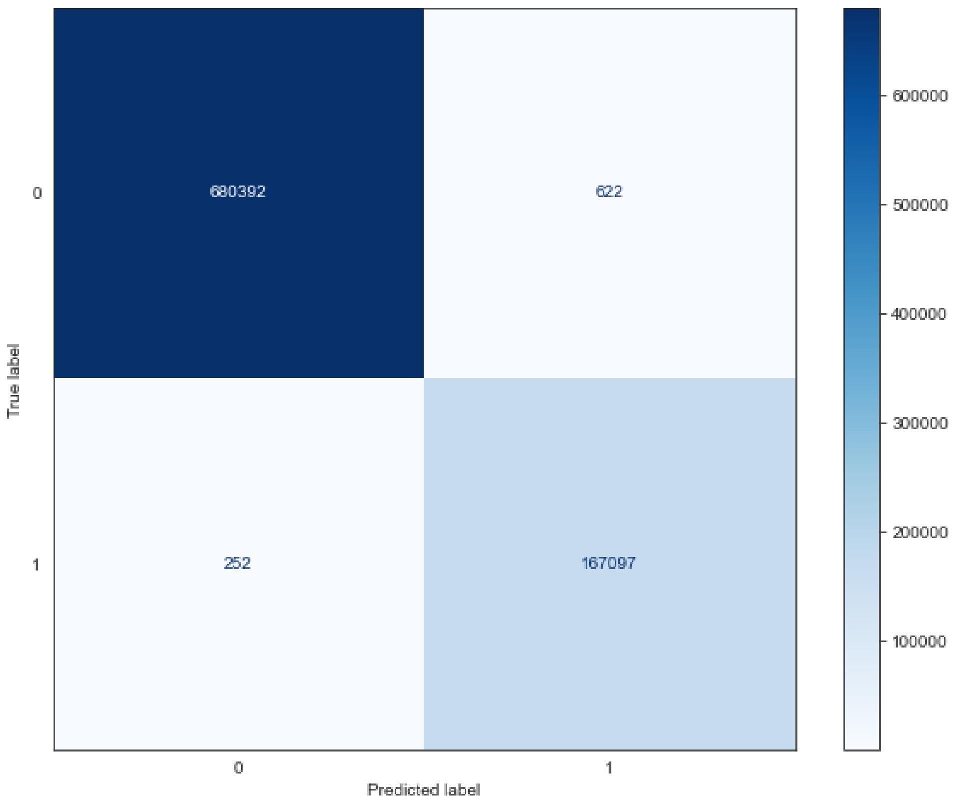


Figure 5.2: Confusion matrix for a random forest on CICIDS2017

A Scenario One: Train and Test data split scenario one (random-sate = 42)

Table 5.7: MLP, LSTM, and GRU with one hidden layer and 10 neurons on CICIDS2017 at random_state=42

Model	Accuracy	Recall	Precision	F1-Score	FPR	Time to Train Model in Second
MLP	97.441%	97.441%	97.576%	97.476%	0.02536	435.1
MLP (Keras)	97.969%	98.235%	91.960%	94.551%	0.02114	537.4
GRU	98.353%	98.187%	93.691%	95.515%	0.01624	804.4
LSTM	98.245%	97.913%	93.426%	95.223%	0.01693	969.7

In terms of detection accuracy, GRU with a single layer with 10 neurons and 20 neurons and two hidden layers with 20 neurons and 10 neurons outperforms other deep neural on the CICIDS2017 dataset using the train and test data split with random_state parameter value equal to 42. Whereas the overall performance metric MLP detection capability with the same parameters mentioned above is greater than others on this dataset as shown in table 5.9,5.8 and 5.7.

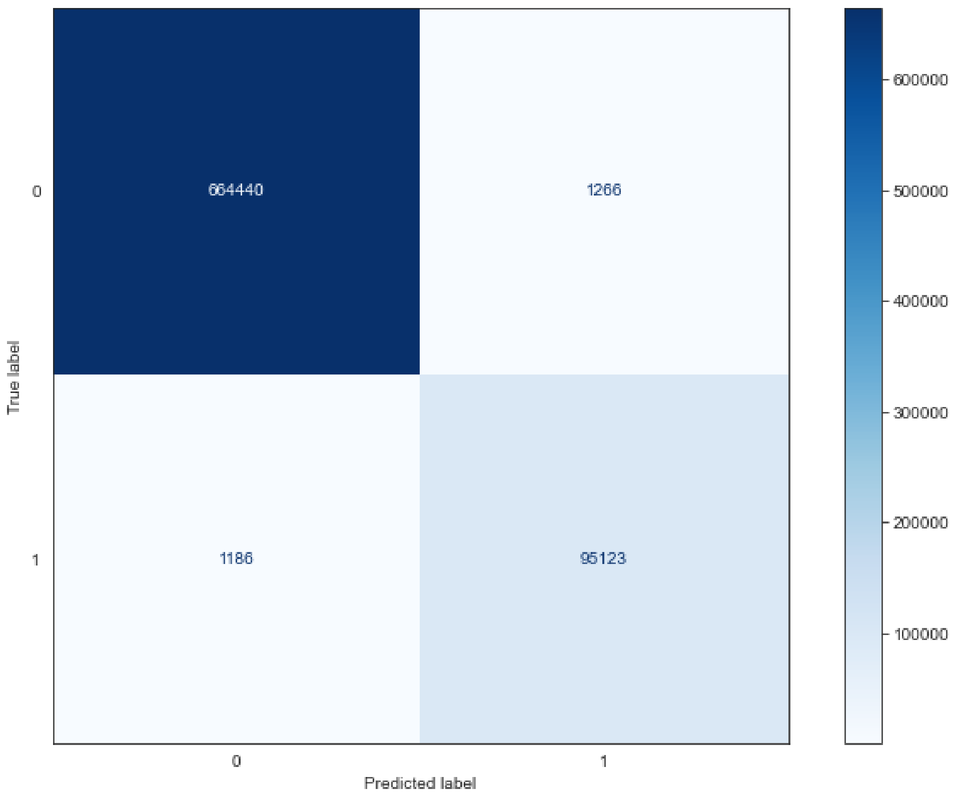


Figure 5.3: Confusion matrix for a random forest on UNSW-NB15

Table 5.8: MLP, LSTM, and GRU with one hidden layer and 20 neurons on CICIDS2017 at random_state=42

Model	Accuracy	Recall	Precision	F1-Score	FPR	Time to Train Model in Second
MLP	98.119%	98.119%	98.114%	98.116%	0.02183	425.8
MLP (Keras)	98.107%	98.639%	92.238%	94.916%	0.02042	1355.0
GRU	98.385%	98.482%	93.595%	95.613%	0.01144	4227.4
LSTM	98.192%	98.284%	92.891%	95.111%	0.01849	758.8

B Scenario Two: Train and Test data split scenario two (random-sate = 1) Tables 5.10 and 5.11 show the DNN model attack classification result using the CICIDS2017 dataset that was split into the train and test dataset by setting random_state parameters to 1.

For the second configuration of the CICIDS2017 dataset split into train and test with random'_state values equal to 1, in terms of accuracy and FPR, GRU outperforms with one hidden layer with 10 and 20 neurons whereas LSTM outperforms with two hidden layers with 20 and 10 neurons. But in the overall performance, MLP with Scikit Learn library performed well because the Sckit Learns library is more stable than the Keras library as shown in table 5.14,5.11 and 5.10.

Table 5.9: MLP, LSTM, and GRU with two hidden layers and 20, 10 neurons on CICIDS2017 at random_state=42

Model	**Accuracy**	**Recall**	**Precision**	**F1-Score**	**FPR**	**Time to Train Model in Second**
MLP	98.119%	98.119%	98.114%	98.116%	0.01062	213.9
MLP (Keras)	98.529%	96.730%	95.721%	95.875%	0.01047	463.9
GRU	98.726%	98.123%	95.429%	96.460%	0.01145	1243.3
LSTM	98.561%	97.524%	95.181%	96.001%	0.01203	2690.1

Table 5.10: MLP, LSTM, and GRU with one hidden layer and 10 neurons on CICIDS2017 at random_state=1

Model	**Accuracy**	**Recall**	**Precision**	**F1-Score**	**FPR**	**Time to Train Model in Second**
MLP	97.26%	97.26%	97.57%	97.32%	0.0332	298.34
MLP (Keras)	96.88%	99.41%	86.83%	91.97%	0.0369	404.67
GRU	97.40%	99.21%	88.75%	93.15%	0.0307	961.31
LSTM	97.27%	99.63%	87.98%	92.90%	0.0333	1005.94

Table 5.11: MLP, LSTM, and GRU with one hidden layer and 20 neurons on CICIDS2017 at random_state=1

Model	**Accuracy**	**Recall**	**Precision**	**F1-Score**	**FPR**	**Time to Train Model in Second**
MLP	97.70%	97.70%	97.91%	97.75%	0.0272	465.10
MLP (Keras)	96.97%	99.68%	86.79%	92.20%	0.0372	568.13
GRU	98.16%	99.39%	91.80%	95.03%	0.0217	1738.61
LSTM	97.24%	99.67%	87.82%	92.82%	0.0339	2005.71

Table 5.12: MLP, LSTM, and GRU with two hidden layers and 20, 10 neurons on CICIDS2017 at random_state=1

Model	**Accuracy**	**Recall**	**Precision**	**F1-Score**	**FPR**	**Time to Train Model in Second**
MLP	98.10%	98.10%	98.20%	98.12%	0.0202	867.63
MLP (Keras)	96.03%	94.38%	86.52%	89.47%	0.0360	768.31
GRU	97.17%	99.35%	87.81%	92.66%	0.0339	6249.95
LSTM	97.27%	99.58%	88.02%	92.90%	0.0332	3590.27

5.3.2.2 Deep Neural Network Model Evaluation Using UNSW-NB15 Dataset

Using the UNSW-NB15 dataset for model training and evaluation, the performance of DNN is presented on attack classification using several performance indicators. LSTM and GRU scores were 99.23% and 99.22% in accuracy with one hidden layer with ten neurons, and the training time for the model was 772.1 seconds and 1083.4 seconds, respectively. LSTM and GRU models with two hidden layers and 20 and 10 neurons scored 99.26% and 99.25% in accuracy, respectively. This is slightly better than this model with one hidden layer and ten neurons, but the model training is much longer because of the increment in the number of neurons and the complexity of the models.

The Scikit learn library was more stable, and MLP with Scikit learn performed better than MLP trained with the Keras library. In general, MLP with the Scikit learns library achieved scores of 99.11% to 99.17% in accuracy, recall, precision, and F1-score. All deep neural models and their performance on the UNSW-NB15 dataset with different configurations and parameters are presented in Tables 5.13,5.14 and Figures A.3,A.4,A.5,A.6,A.8.

Table 5.13: Deep neural network model with one hidden layer using UNSW-NB15 dataset

Model	Accuracy	Recall	Precision	F1-Score	FPR	Time to Train Model in Second
MLP	99.11%	99.11%	99.11%	99.11%	0.0059	151.6
MLP (Keras)	99.12%	96.30%	94.42%	94.86%	0.0066	513.7
GRU	99.22%	95.07%	96.17%	95.18%	0.0035	1083.4
LSTM	99.23%	95.47%	95.93%	95.26%	0.0040	772.1

5.13(a) Deep neural network with 10 neurons classification performance

Model	Accuracy	Recall	Precision	F1-Score	FPR	Time to Train Model in Seconds
MLP	99.17%	99.17%	99.17%	99.17%	0.0054	195.1
MLP (Keras)	99.19%	95.12%	95.96%	95.07%	0.0039	925.5
GRU	99.24%	96.07%	95.54%	95.38%	0.0047	3132.8
LSTM	99.25%	95.40%	96.07%	95.30%	0.0037	2072.0

5.13(b) Deep neural network with 20 neurons classification performance

Table 5.14: MLP, LSTM, and GRU with two hidden layers and 20, 10 neurons on UNSW-NB15 dataset

Model	Accuracy	Recall	Precision	F1-Score	FPR	Time to Train Model in Second
MLP	99.17%	99.17%	99.17%	99.17%	0.0037	269.7
MLP (Keras)	99.23%	95.26%	96.06%	95.21%	0.0038	982.5
GRU	99.25%	95.54%	96.00%	95.34%	0.0039	6284.9
LSTM	99.26%	95.54%	96.08%	95.39%	0.0038	10984.2

Figure 5.4 shows the TP, TN, FP, and FN using the UNSW-NB15 dataset using two hidden layers with 20 neurons and 10 neurons. The model predicted 3923 instances as normal but the

instance is attacked as well as 3211 instances predicted as attacks but the actual instance was non-attack.

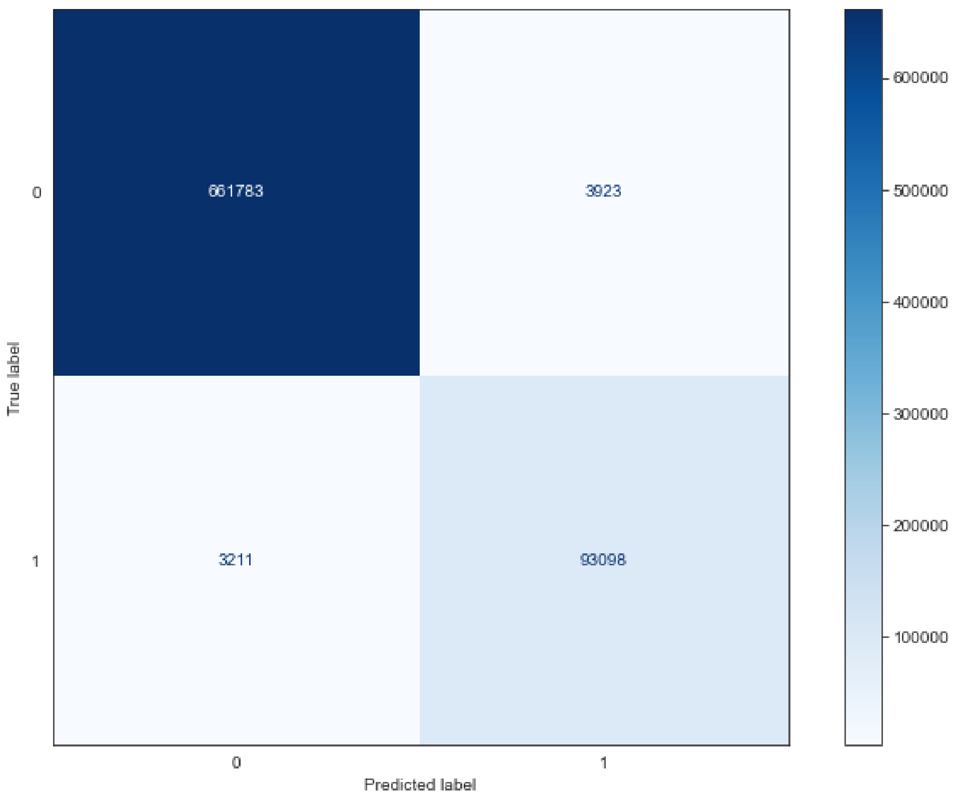


Figure 5.4: Confusion matrix for an MLP on UNSW-NB15

5.3.2.3 Performance Comparison with Earlier Research

A. CICIDS2017 Dataset

The MLP and RF models were used for the CICIDS 2017 by (Sharafaldin et al., 2018) to model an attack detection system. The accuracy of their MLP and RF model detection was 77% and 98%, respectively. Their model's precision and recall were 83%, 97%, 76%, and 97% for MLP and RF, respectively. My results illustrated in Table 5.15 show an improvement in all performance metrics.

B. UNSW-NB15 Dataset

Using RF and a small portion of the UNSW-NB15 dataset, the IoT attack detection model

Table 5.15: Ranks of models based on accuracy using CICIDS2017 dataset

Model	Rank	Accuracy	Recall	Precision	F1_Score	FPR
RF	1	99.90%	99.90%	99.90%	99.90%	0.000913
DecisionTree	2	99.87%	99.87%	99.87%	99.87%	0.00084
AdaBoost	3	98.86%	98.11%	96.17%	97.13%	0.009599
GRU	4	98.73%	99.58%	88.02%	92.90%	0.0332
LSTM	5	98.56%	99.35%	87.81%	92.66%	0.0339
MLP (Keras)	6	98.53%	94.38%	86.52%	89.47%	0.036
MLP	7	98.12%	98.10%	98.20%	98.12%	0.0202
Logistic Regression	8	92.25%	92.25%	92.07%	92.12%	0.038068
Naïve Bayes	9	59.73%	59.73%	85.67%	49.51%	0.632595

provided by (Alrashdi et al., 2019) obtained a 99.34% accuracy rate. Using the entire UNSW-NB15 dataset, I was able to achieve 99.68% accuracy with RF, 99.34% accuracy with a DT, 99.26% accuracy with LSTM models, and 99.25% accuracy with GRU models in my research. MLP, GRU, and LSTM models for attack detection proposed by (Disha & Waheed, 2022) performance evaluation using a selected subset of the UNSW-NB15 dataset with F1-score metrics archived 87.2%, 86.39%, and 90.31%, respectively. While my models using the entire instance of the UNSW-NB15 dataset improved the F1-score metric scores to 99.17%, 95.34%, and 95.35%, respectively.

Table 5.16: Ranks of models based on accuracy using UNSW-NB15 dataset

Model	Rank	Accuracy	Recall	Precision	F1-Score	FPR
Random Forest	1	99.68%	99.68%	99.68%	99.68%	0.0019
DecisionTree	2	99.34%	99.34%	99.35%	99.34%	0.0046
LSTM	3	99.26%	95.54%	96.08%	95.39%	0.0038
GRU	4	99.25%	95.54%	96.00%	95.34%	0.0039
MLP(Keras)	5	99.23%	95.26%	96.06%	95.21%	0.0038
MLP	6	99.17%	99.17%	99.17%	99.17%	0.0037
AdaBoost	7	99.09%	96.49%	96.22%	96.35%	0.0055
Logistic Regression	8	98.93%	98.93%	98.95%	98.93%	0.0085
Naïve Bayes	9	88.84%	88.84%	90.00%	84.79%	0.0002

According to the experimental results shown in table 5.16 the Random Forest model performs better in terms of accuracy on the dataset UNSW-NB15, from the conventional machine learning models the decision tree ranked second based on the same metrics. from the deep, learning algorithm LSTM outclasses the other deep learning models and generally, it is the third-ranked model in this experiment. based on the precision matrix the MLP model has a higher score than other deep learning models. whereas, the Naive base model performs the least among both the deep learning and conventional & ensemble models when evaluated in terms of all the matrices except false positive rate metrics.

The limitation of this research is the threshold value selection for correlated features in the dataset. To reduce the correlated features, I performed only two experiments with threshold values of 0.90 and 0.96. For the UNSW-NB15 dataset, 0.90 performed better, but for the CICIDS 2017 dataset, features are highly correlated, and a threshold value of 0.96 achieved better performance. In addition to this, models implemented with the Keras library show little variation in the results.

5.3.3 Individual Attack Classification

In this section, I have discussed the performance of the conventional machine learning model, ensemble machine learning model, and DNN model on each attack class using the UNSW-NB15 dataset and CICIDS2017 datasets.

5.3.3.1 Conventional and Deep Neural Network Model for Individual Attack Classification Using CICIDS2017

The attack classification performance of several models for individual attack classes in the CICIDS2017 dataset is displayed in Table 5.17. Based on the results, for attack type bots and infiltration, the DT model outperformed others, whereas the detection capabilities of other models were weak. Even if the overall performance of the IoT attack detection model scored a high detection accuracy rate for binary classification, that does not mean the detection model performs well for all categories of attacks. My experiment resulted in the overall detection performance of our model achieving higher detection accuracy, but the model's detection accuracy was very low on some attack types. As shown in Table 5.17, the model performance attack classification on individual attack types that exist in the CICIDS 2017 dataset was lower for bots and infiltration. In a smart city environment, several IoT devices are used to provide products or services, so when we implement an attack detection system we have to focus on which varieties of attacks affect our system and which type of detection system is more suitable for our business environment rather than selecting based considering the overall performance of the detection systems.

Table 5.17: Individual attack category classification using CICIDS2017 dataset

Models	BENIGN	Bot	Brute Force	DDoS	DoS	Heartbleed	Infiltration	PortScan	Web Attack
DecisionTree	99.92%	82.27%	99.98%	99.98%	99.88%	100.00%	100.00%	99.29%	99.05%
AdaBoost	99.04%	1.42%	97.32%	99.80%	98.26%	83.33%	0.00%	98.98%	5.21%
Random Forest	99.91%	68.62%	99.98%	99.96%	99.96%	83.33%	80.00%	99.98%	97.32%
Logistic Regression	96.19%	26.77%	28.33%	78.46%	71.53%	16.67%	0.00%	87.62%	0.16%
Naïve Bayes	50.49%	62.41%	51.07%	99.96%	98.31%	0.00%	60.00%	99.32%	6.78%
MLP	98.62%	36.70%	96.17%	98.56%	99.55%	33.33%	0.00%	95.23%	10.73%
MLP (keras)	98.70%	36.88%	97.71%	99.18%	99.77%	50.00%	10.00%	94.64%	11.36%
LSTM	98.73%	34.93%	99.45%	98.97%	99.84%	0.00%	10.00%	95.95%	34.23%
GRU	98.84%	37.06%	89.53%	98.51%	99.83%	83.33%	0.00%	96.50%	9.15%

5.3.3.2 Deep Neural Model Performance on Individual Attack Categories Using UNSW-NB15

The performance of several models for specific attack classes in the UNSW-NB15 dataset is displayed in Table 5.18. In comparison to other attack categories, the fuzzer attack category had inferior detection capabilities across all models. RF, DT, LR, LSTM, and GRU models each had a detection rate of 88.32%, 79.13%, 79.61%, 67.80%, and 66.23% for this attack type, respectively. The shellcode attack was more challenging to recognize next to the fuzzer attack category with those models.

Table 5.18: Individual attack category classification using UNSW-NB15 dataset

Models	Normal	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Reconnaissance	Shellcode	Worms
DecisionTree	99.54%	92.93%	99.54%	98.87%	98.59%	79.13%	99.95%	98.81%	89.06%	95.92%
AdaBoost	99.45%	90.24%	93.47%	97.08%	96.57%	66.08%	99.89%	99.38%	86.43%	97.96%
Random Forest	99.79%	93.17%	99.85%	99.80%	99.54%	88.32%	99.99%	99.86%	97.16%	100.00%
Logistic Regression	99.15%	90.98%	95.74%	96.62%	96.36%	79.61%	99.90%	96.06%	88.84%	93.88%
Naïve Bayes Prediction	99.98%	73.54%	76.60%	72.01%	36.38%	10.05%	0.85%	14.21%	0.00%	0.00%
MLP	99.63%	81.71%	97.11%	97.75%	96.90%	62.43%	99.91%	95.06%	85.12%	97.96%
MLP(Keras)	99.62%	83.29%	97.87%	98.16%	97.31%	64.18%	99.91%	99.33%	85.78%	100.00%
LSTM	99.61%	82.44%	98.02%	98.46%	97.17%	67.80%	99.91%	98.74%	90.81%	95.92%
GRU	99.62%	82.32%	97.42%	98.64%	97.72%	66.23%	99.93%	99.12%	91.68%	97.96%

Chapter 6 Conclusion and Future Recommendation

In this thesis, I propose DNN models for IoT attack detection and classification frameworks in smart cities. In addition to this, I also provide conventional machine learning and ensemble machine learning methods for attack detection and classification for comparison purposes. I have used two of the most recent datasets, the CICIDS2017 and UNSW-NB15 datasets, for training both kinds of models and evaluation. These datasets are chosen due to the fact that they are enhanced by a variety of recently added cyber attacks that exist in smart city environments.

The proposed DNN models, that is MLP, LSTM, and GRU, were evaluated in terms of accuracy, recall, precision, F1-score, and FPR. I also use those performance metrics for the evaluation of other models, such as DT, RF, AdaBoost, LR, and NB models. Using the UNSW-NB15 dataset, the MLP model outperforms other models in terms of recall, precision, F1-score, and FPR with values of 99.17%, 99.17%, 99.17%, and 0.0037, respectively. In contrast, the LSTM model achieves a higher accuracy of 99.26%. In the case of conventional and ensemble models, RF outclasses other models with respect to all metrics when trained and evaluated with the UNSW-NB15 dataset. Despite this, when the dataset CICIDS2017 was used for training and evaluating the RF model, it outperformed other conventional and ensemble methods. Among the DNN models, the MLP model classified attacks with a high accuracy (98.10%), precision (98.20%), F1-score (98.12%), and FPR (0.0202), which makes it the best DNN performer for this dataset.

Mostly, the results show that RF has better accuracy than MLP in attack detection in terms of all evaluation metrics. My findings imply that the DT and RF models performed best in terms of stability for different attack category classification. The deep learning models struggled for several attack types in the CICIDS2017 dataset (Bot, Heartbleed, and Infiltration), which suggests that more data are needed for these attack categories to further improve the performance of the DNN models. Though, smart cities intensively use IoT devices that are generating a huge volume of data. As a result, a big concern might be using conventional attack classification models as they have dimensionality problems. In this research, I have observed that the performance of conventional methods generally degraded as the dimensionality is increasing. Thus, I recommend the deep learning model for attack detection in smart cities where high-dimensional data are available. For the future work, I propose evaluating the models with real-time smart city traffic datasets.

References

Agresti, A. (2015). *Foundations of linear and generalized linear models*. John Wiley & Sons.

Aldin, N. B., & Aldin, S. S. A. B. (2022). Accuracy comparison of different batch size for a supervised machine learning task with image classification. In *2022 9th international conference on electrical and electronics engineering (iceee)* (pp. 316–319).

Almarshdi, R., Nassef, L., Fadel, E., & Alowidi, N. (2023). Hybrid deep learning based attack detection for imbalanced data classification. *Intelligent Automation & Soft Computing*, *35*(1), 297–320.

Alrashdi, I., Alqazzaz, A., Aloufi, E., Alharthi, R., Zohdy, M., & Ming, H. (2019). Ad-iot: Anomaly detection of iot cyberattacks in smart city using machine learning. In *2019 ieee 9th annual computing and communication workshop and conference (ccwc)* (pp. 0305–0310).

Alshamsi, A., Anwar, Y., Almulla, M., Aldohoori, M., Hamad, N., & Awad, M. (2017). Monitoring pollution: Applying iot to create a smart environment. In *2017 international conference on electrical and computing technologies and applications (icecta)* (pp. 1–4).

Al-Taleb, N., & Saqib, N. A. (2022). Towards a hybrid machine learning model for intelligent cyber threat identification in smart city environments. *Applied Sciences*, *12*(4), 1863.

Al-Turjman, F., Zahmatkesh, H., & Shahroze, R. (2022). An overview of security and privacy in smart cities' iot communications. *Transactions on Emerging Telecommunications Technologies*, *33*(3), e3677.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Brownlee, J. (2020). *Why one-hot encode data in machine learning?* Retrieved 04-26-2023, from https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

Chen, D., Wawrzynski, P., & Lv, Z. (2021). Cyber security in smart cities: A review of deep learning-based applications and case studies. *Sustainable Cities and Society*, *66*, 102655.

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.

Das, H., Naik, B., & Behera, H. (2020). An experimental analysis of machine learning classification algorithms on biomedical data. In *Proceedings of the 2nd international conference on communication, devices and computing: Iccdc 2019* (pp. 525–539).

Dey, R., & Salem, F. M. (2017). Gate-variants of gated recurrent unit (gru) neural networks. In *2017 ieee 60th international midwest symposium on circuits and systems (mwscas)* (pp. 1597–1600).

Disha, R. A., & Waheed, S. (2022). Performance analysis of machine learning models for intrusion detection system using gini impurity-based weighted random forest (giwrf) feature selection technique. *Cybersecurity*, *5*(1), 1.

Dobilas, S. (2022). *Gru recurrent neural networks — a smart way to predict sequences in python.* Retrieved 03-03-2023, from https://towardsdatascience.com/gru-recurrent-neural-networks-a-smart-way-to-predict-sequences-in-python-80864e4fe9f6

Domínguez-Bolaño, T., Campos, O., Barral, V., Escudero, C. J., & García-Naya, J. A. (2022). An overview of iot architectures, technologies, and existing open-source projects. *Internet of Things*, 100626.

Huang, X. (2021). Network intrusion detection based on an improved long-short-term memory model in combination with multiple spatiotemporal structures. *Wireless Communications and Mobile Computing*, *2021*, 1–10.

Janarthanan, T., & Zargari, S. (2017). Feature selection in unsw-nb15 and kddcup'99 datasets. In *2017 ieee 26th international symposium on industrial electronics (isie)* (pp. 1881–1886).

Kandpal, A. (2018). *The lstm cell (long-short term memory cell).* Retrieved 04-23-2023, from https://towardsdatascience.com/gru-recurrent-neural-networks-a-smart-way-to-predict-sequences-in-python-80864e4fe9f6

Kasongo, S. M. (2023). A deep learning technique for intrusion detection system using a recurrent neural networks based framework. *Computer Communications*, *199*, 113–125.

McCandlish, S., Kaplan, J., Amodei, D., & Team, O. D. (2018). An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*.

MEDIA, N. P. . (2020). *Iot in transportation – 5 applications of iot technology in transportation.* Retrieved 04-26-2023, from https://www.nec.co.nz/market-leadership/publications-media/iot-in-transportation-5-applications-of-iot-technology-in-transportation/

Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., & Elovici, Y. (2018). N-baiot: Network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, *17*(3), 12–22.

Mohammed, A. J., Arif, M. H., & Ali, A. A. (2020). A multilayer perceptron artificial neural network approach for improving the accuracy of intrusion detection systems. *IAES International Journal of Artificial Intelligence*, *9*(4), 609.

Mohmand, M. I., Hussain, H., Khan, A. A., Ullah, U., Zakarya, M., Ahmed, A., … others (2022). A machine learning-based classification and prediction technique for ddos attacks. *IEEE Access*, *10*, 21443–21454.

Moustafa, N., & Slay, J. (2015). Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (milcis)* (pp. 1–6).

Moustafa, N., & Slay, J. (2016). The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. *Information Security Journal: A Global Perspective*, *25*(1-3), 18–31.

Navlani, A. (2018). *Adaboost classifier.* Retrieved 25-03-2023, from https://www.datacamp.com/tutorial/adaboost-classifier-python

of New Brunswick, U. (2017). *Intrusion detection evaluation dataset (cic-ids2017).* Retrieved 09-26-2022, from https://www.unb.ca/cic/datasets/ids-2017.html

Page, R. (2023). *Applications of internet of things (iot).* Retrieved 04-26-2023, from https://www.rfpage.com/applications-of-internet-of-things-iot/

Panigrahi, R., & Borah, S. (2018). A detailed analysis of cicids2017 dataset for designing intrusion detection systems. *International Journal of Engineering & Technology*, *7*(3.24), 479–482.

Powers, D. M. (2020). Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*.

Priyanka, & Kumar, D. (2020). Decision tree classifier: a detailed survey. *International Journal of Information and Decision Sciences*, *12*(3), 246–269.

Rashid, M. M., Kamruzzaman, J., Hassan, M. M., Imam, T., & Gordon, S. (2020). Cyberattacks detection in iot-based smart city applications using machine learning techniques. *International Journal of Environmental Research and Public Health*, *17*(24), 9347.

Rashid, M. M., Kamruzzaman, J., Imam, T., Kaisar, S., & Alam, M. J. (2020). Cyber attacks detection from smart city applications using artificial neural network. In *2020 ieee asia-pacific conference on computer science and data engineering (csde)* (pp. 1–6).

Reddy, G. T., Reddy, M. P. K., Lakshmanna, K., Kaluri, R., Rajput, D. S., Srivastava, G., & Baker, T. (2020). Analysis of dimensionality reduction techniques on big data. *Ieee Access*, *8*, 54776–54788.

Sha, K., Yang, T. A., Wei, W., & Davari, S. (2020). A survey of edge computing-based designs for iot security. *Digital Communications and Networks*, *6*(2), 195–202.

Shafiq, M., Tian, Z., Sun, Y., Du, X., & Guizani, M. (2020). Selection of effective machine learning algorithm and bot-iot attacks traffic identification for internet of things in smart city. *Future Generation Computer Systems*, *107*, 433–442.

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, *1*, 108–116.

Singh, S., Fernandes, S. V., Padmanabha, V., & Rubini, P. (2021). Mcids-multi classifier intrusion detection system for iot cyber attack using deep learning algorithm. In *2021 third international conference on intelligent communication technologies and virtual mobile networks (icicv)* (pp. 354–360).

Taud, H., & Mas, J. (2018). Multilayer perceptron (mlp). *Geomatic approaches for modeling land change scenarios*, 451–455.

Thakar, P. (2020). *The math behind machine learning algorithms.* Retrieved 04-24-2023, from https://towardsdatascience.com/the-math-behind-machine-learning-algorithms-9c5e4c87fff

Ullo, S. L., & Sinha, G. R. (2020). Advances in smart environment monitoring systems using iot and sensors. *Sensors*, *20*(11), 3113.

Vinayakumar, R., Alazab, M., Soman, K., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. *Ieee Access*, *7*, 41525–41550.

Vishwakarma, R., & Jain, A. K. (2020). A survey of ddos attacking techniques and defence mechanisms in the iot network. *Telecommunication Systems*, *73*(1), 3–25.

YADAV, S. (2019). *Intro to recurrent neural networks lstm | gru.* Retrieved 03-03-2023, from https://www.kaggle.com/code/thebrownviking20/intro-to-recurrent-neural-networks-lstm-gru

Zhang, K., Ni, J., Yang, K., Liang, X., Ren, J., & Shen, X. S. (2017). Security and privacy in smart city applications: Challenges and solutions. *IEEE Communications Magazine*, *55*(1), 122–129.

Appendices

Appendix A Python Source Code

Due to the size, we've included a few snippets of this thesis's Python source code in this section.

A.1 Data Preprocessing Sample Code Fragment

```python
import numpy as np  # for array
from numpy import array
import pandas as pd  # for csv files and dataframe
import matplotlib.pyplot as plt  # for plotting
import seaborn as sns  # plotting
import pickle  # To load data int disk
from scipy.sparse import coo_matrix
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import StandardScaler  # Standardizer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder  # One
    hot Encoder
from scipy.sparse import csr_matrix  # For sparse matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
    make_scorer  # Scoring functions
from sklearn.metrics import auc, f1_score, roc_curve, roc_auc_score  #
    Scoring fns
from sklearn.metrics import precision_score, recall_score,
    accuracy_score
import time
import os
%matplotlib inline
```

Listing A.1: Import most common library here

```python

# Reading datasets
dfs = []
for i in range(1,5):
    path = './UNSW-NB15_{}.csv'  # There are 4 input csv files
    dfs.append(pd.read_csv(path.format(i), header = None))
all_data = pd.concat(dfs).reset_index(drop=True)  # Concat all to a
    single df
# This csv file contains the names of all the features
df_col = pd.read_csv('./NUSW-NB15_features.csv', encoding='ISO-8859-1')
# Making column names lower case, removing spaces
```

```python
df_col['Name'] = df_col['Name'].apply(lambda x: x.strip().replace(' ',
    '').lower())
# Removing spaces and changing column names to lowercase
df_col['Name'] = df_col['Name'].apply(lambda x: x.strip().replace(' ',
    '').lower())
```

Listing A.2: Loading dataset and merging CSV files

```python
train, test = train_test_split(all_data, test_size=0.3, random_state=1)
```

Listing A.3: Dataset split in to train and test

```python
# Plotting the correlation matrix of the dataset
# Refer: https://towardsdatascience.com/feature-selection-correlation-
    and-p-value-da8921bfb3cf
method_1 = "pearson"
# correlation matrix
corr_matr = x_train.corr(method=method_1)
plt.figure(figsize=(12,12))
sns.heatmap(corr_matr, square=True)
plt.show()
```

Listing A.4: Identification of correlated data using person method

```python
#Reference: https://www.kaggle.com/code/prashant111/comprehensive-guide
    -on-feature-selection
#with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything
    other feature
def correlation_2(dataset, threshold):
    col_cor = set()  # Set of all the names of correlated columns
    corr_matrix_2 = x_train.corr().abs()
    for i in range(len(corr_matrix_2.columns)):
        for j in range(i):
            if abs(corr_matrix_2.iloc[i, j]) > threshold: # we are
    interested in absolute coeff value
                colname = corr_matrix_2.columns[i]  # getting the name
    of column
                col_cor.add(colname)
    return col_cor
```

Listing A.5: Removing highly correlated data

```python
# Standardizing the data
scaler = StandardScaler()
scaler = scaler.fit(x_train[num_col])
scalerT = StandardScaler()
scalerT = scalerT.fit(x_test[num_colT])
x_train[num_col] = scaler.transform(x_train[num_col])
```

```
x_test[num_colT] = scalerT.transform(x_test[num_colT])
```

Listing A.6: Standardization of train and test dataset

```
# Onehot Encoding for Train
service_ = OneHotEncoder()
proto_ = OneHotEncoder()
state_ = OneHotEncoder()
ohe_service = service_.fit(x_train.service.values.reshape(-1,1))
ohe_proto = proto_.fit(x_train.proto.values.reshape(-1,1))
ohe_state = state_.fit(x_train.state.values.reshape(-1,1))
# Remove the original categorical column
for col, ohe in zip(['proto', 'service', 'state'], [ohe_proto,
    ohe_service, ohe_state]):
     x = ohe.transform(x_train[col].values.reshape(-1,1))
     tmp_df = pd.DataFrame(x.todense(), columns=[col+'_'+i for i in ohe.
    categories_[0]])
     x_train = pd.concat([x_train.drop(col, axis=1), tmp_df], axis=1)
#Note: Use the same process for test data
# Making the train data sparse matrix
x_train_csr = csr_matrix(x_train.values)
col = x_train.columns
# Creating sparse dataframe with x_train sparse matrix
x_train = pd.DataFrame.sparse.from_spmatrix(x_train_csr, columns=col)
```

Listing A.7: One hot encoding

A.2 Models Setup Code

A.2.1 Classical and Ensemble Model Configuration

```
%%time
# reference: https://www.kaggle.com/code/waltermaffy/fruit-
    classification-pca-svm-knn-decision-tree
start = time.time()
decision_tree =DecisionTreeClassifier()
model_DT = decision_tree.fit(x_train,y_train)
end_train = time.time()
# prediction
y_pred_DT = model_DT.predict(x_test)
end_predict = time.time()
```

Listing A.8: Decision Tree

```
%%time
# reference:  https://www.datacamp.com/tutorial/adaboost-classifier-
    python
# Create adaboost classifer object
```

```python
start = time.time()
ada_boost = AdaBoostClassifier(n_estimators=50,
                         learning_rate=1,random_state=0)
# Train Adaboost Classifer
model_adaBoost = ada_boost.fit(x_train,y_train)
end_train = time.time()
#Predict the response for test dataset
y_pred_adaBoost = model_adaBoost.predict(x_test)
end_predict = time.time()
```

Listing A.9: AdaBoost

```python
%%time
start = time.time()
from sklearn.ensemble import RandomForestClassifier
# Create Random Forest classifier object
model_rfc = RandomForestClassifier(n_estimators=100, random_state=0)
# Train the model using the training sets
model_rfc.fit(x_train, y_train)
end_train = time.time()
# prediction
y_pred_rfc = model_rfc.predict(x_test)
end_predict = time.time()
```

Listing A.10: Random Forest

```python
%%time
start = time.time()
from sklearn.linear_model import LogisticRegression
#Fitmodel
model_logReg = LogisticRegression(random_state=0).fit(x_train,y_train)
end_train = time.time()
# prediction
y_pred_logReg=model_logReg.predict(x_test)
end_predict = time.time()
```

Listing A.11: Logistic Regression

```python
# Import necessary libraries for Gaussian Naive Bayes algorithm
from sklearn.naive_bayes import GaussianNB
%%time
start = time.time()
# Create Naive Bayes classifier object
model_gnb = GaussianNB()
# Train the model using the training sets
model_gnb.fit(x_train_gnb , y_train_gnb)
end_train = time.time()
# prediction
y_pred_gnb = model_gnb.predict(x_test_gnb)
```

```
end_predict = time.time()
```

Listing A.12: Gussian Naive Bayes

```
# Calculate the accuracy of the predictions, # Precision, # recall
    score  and f1 score
accuracy_DT = accuracy_score(y_test, y_pred_DT)
precision_DT= precision_score(y_test, y_pred_DT, average='weighted')
recall_DT= recall_score(y_test,y_pred_DT,average='weighted')
f1s_DT = f1_score(y_test, y_pred_DT,average='weighted')
#Note; this code only for the decision tree, repeat it for others
```

Listing A.13: Model evaluation metrics function

A.2.2 Deep Neural Network Model Configuration

```
%%time
from sklearn.neural_network import MLPClassifier
start = time.time()
model = MLPClassifier(hidden_layer_sizes = (20,10,),
                      activation='relu',
                      solver='adam',
                      batch_size=1024,
                      verbose=0).fit(x_train,y_train)
end_train = time.time()
y_predictions = model.predict(x_test) # These are the predictions from
    the test data.
end_predict = time.time()
```

Listing A.14: Multilayer perceptron With two hidden layer with scikit-learn

```
#Import libraries that will allow you to use keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU
from keras import metrics

# https://datascience.stackexchange.com/questions/45165/
# how-to-get-accuracy-f1-precision-and-recall-for-a-keras-model
from keras import backend as K
def recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall
def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision
def f1_scores(y_true, y_pred):
    p = precision(y_true, y_pred)
    r = recall(y_true, y_pred)
    return 2*((p*r)/(p+r+K.epsilon()))

#Build the feed-forward neural network model (2 layers with 20,10
    neurons )
from keras.optimizers import Adam
input_shape = x_train.shape[1]
def build_model_mk():
    model_mk = Sequential()
    model_mk.add(Dense(units=20, input_dim=input_shape, activation='
    relu'))
    model_mk.add(Dense(units=10, activation='relu'))
    model_mk.add(Dense(units=1, activation='sigmoid')) #for binary
    classification
    #Compile the model
    model_mk.compile(loss='binary_crossentropy',optimizer='adam',
                      metrics=['accuracy',f1_scores,precision, recall]
                 )
    return model_mk
#institate the model
model_mk = build_model_mk()
#fit the model
startM = time.time()
history_mk=model_mk.fit(x_train, y_train, epochs=150, batch_size=1024,
    verbose=2)
end_trainM = time.time()
```

Listing A.15: Multilayer perception using two hidden layer with Keras

```
#The GRU input layer must be 3D.
#The meaning of the 3 input dimensions are: samples, time steps, and
    features.
#reshape input data
X_train_array_GRU = array(x_train) #array has been declared in the
    previous cell
print(len(X_train_array_GRU))
X_train_reshaped_GRU = X_train_array_GRU.reshape(X_train_array_GRU.
    shape[0],1,input_shape)
#reshape output data
X_test_array_GRU=  array(x_test)
X_test_reshaped_GRU = X_test_array_GRU.reshape(X_test_array_GRU.shape
    [0],1,input_shape)
#Build the neural network model
def build_model_GRU():
    model_GRU = Sequential()
    model_GRU.add(GRU(units=20, return_sequences=True,input_shape=(1,
   input_shape)))
    model_GRU.add(GRU(units=10, return_sequences=True))
    model_GRU.add(Dense(units=1, activation='sigmoid')) #for binary
   classification
    # Compile the model
    model_GRU.compile(loss='binary_crossentropy',optimizer='adam',
                     metrics=['accuracy',f1_scores,precision, recall]
            )
    return model_GRU
#model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
   accuracy',recall_m, precision_m, f1_m]
#institate the model
model_GRU = build_model_GRU()
startG = time.time()
 #Fit the model on the dataset
history_GRU =model_GRU.fit(X_train_reshaped_GRU, y_train, epochs=150,
   batch_size=1024,verbose=2)
end_trainG = time.time()
# Evaluate the model on the test data using "evaluate"
startPG = time.time()
loss_GRU, accuracy_GRU, f1_scores_GRU, precision_GRU, recall_GRU =
   model_GRU.evaluate(X_test_reshaped_GRU, y_test)
end_predictG = time.time()
```

Listing A.16: GRU using two hidden Layers

```
#The LSTM input layer must be 3D.
#The meaning of the 3 input dimensions are: samples, time steps, and
   features.
#reshape input data
X_train_array_LSTM = array(x_train) #array has been declared in the
   previous cell
print(len(X_train_array_LSTM))
X_train_reshaped_LSTM = X_train_array_LSTM.reshape(X_train_array_LSTM.
   shape[0],1,input_shape)
#reshape output data
X_test_array_LSTM=  array(x_test)
X_test_reshaped_LSTM = X_test_array_LSTM.reshape(X_test_array_LSTM.
   shape[0],1,input_shape)
# input_shape = X_train.shape[1]
def build_model_LSTM():
    model_LSTM = Sequential()
    model_LSTM.add(LSTM(units=20, return_sequences=True,input_shape=(1,
   input_shape)))
    model_LSTM.add(LSTM(units=10, return_sequences=True))
    model_LSTM.add(Dense(units=1, activation='sigmoid')) #for binary
   classification
```

```
    #Compile the model
    model_LSTM.compile(loss= 'binary_crossentropy',optimizer='adam',
                     metrics=['accuracy',f1_scores,precision, recall]
                 )
    return model_LSTM
#institate the model
model_LSTM = build_model_LSTM()
#fit the model
startL = time.time()
history_LSTM=model_LSTM.fit(X_train_reshaped_LSTM, y_train, epochs=150,
    batch_size=1024,verbose=2)
end_trainL = time.time()
#Evaluate the neural network
startPL = time.time()
loss_LSTM, accuracy_LSTM, f1_scores_LSTM, precision_LSTM, recall_LSTM =
    model_LSTM.evaluate(X_test_reshaped_LSTM, y_test)
# loss, accuracy, f1s, precision, recall = model.evaluate(
   X_test_reshaped, y_test)
end_predictL = time.time()
```

Listing A.17: LSTM using two hidden Layers

```
%%time
startPG =time.time()
predictions_GRU = model_GR.predict(X_test_reshaped_GRU ).ravel()
endTimePG =time.time()
GRU_pre_time= endTimePG -startPG
print(GRU_pre_time)
```

Listing A.18: Time to predict compuation

```
# confusion matrix
GRU_TN, GRU_FP, GRU_FN, GRU_TP = confusion_matrix(y_test,
   predictions_GR).ravel()
# false positive rate
GRU_fpr = GRU_FP / (GRU_FP + GRU_TN)
print("False positive rate:",GRU_fpr )
## Print Result
# Note: Apply this process for the remaining model
print("Accuracy: "+ "{:.3%}".format(accuracy_GRU))
print("Recall: "+ "{:.3%}".format(recall_GRU))
print("Precision: "+ "{:.3%}".format(precision_GRU))
print("F1-Score: "+ "{:.3%}".format(f1_scores_GRU))
print("time to train: "+ "{:.3f}".format(end_trainG-startG)+" s")
print("time to predict: "+"{:.3f}".format(GRU_pre_time)+" s")
print("total: "+"{:.2f}".format( end_trainG-startG + GRU_pre_time)+" s"
   )
```

Listing A.19: False positive rate calculation and Print result

A.2.3 Correlation Heatmap for UNSW-NB15 and CICIDS2017 Dataset

Figures A.1 and A.2 show the correlation of our dataset features.

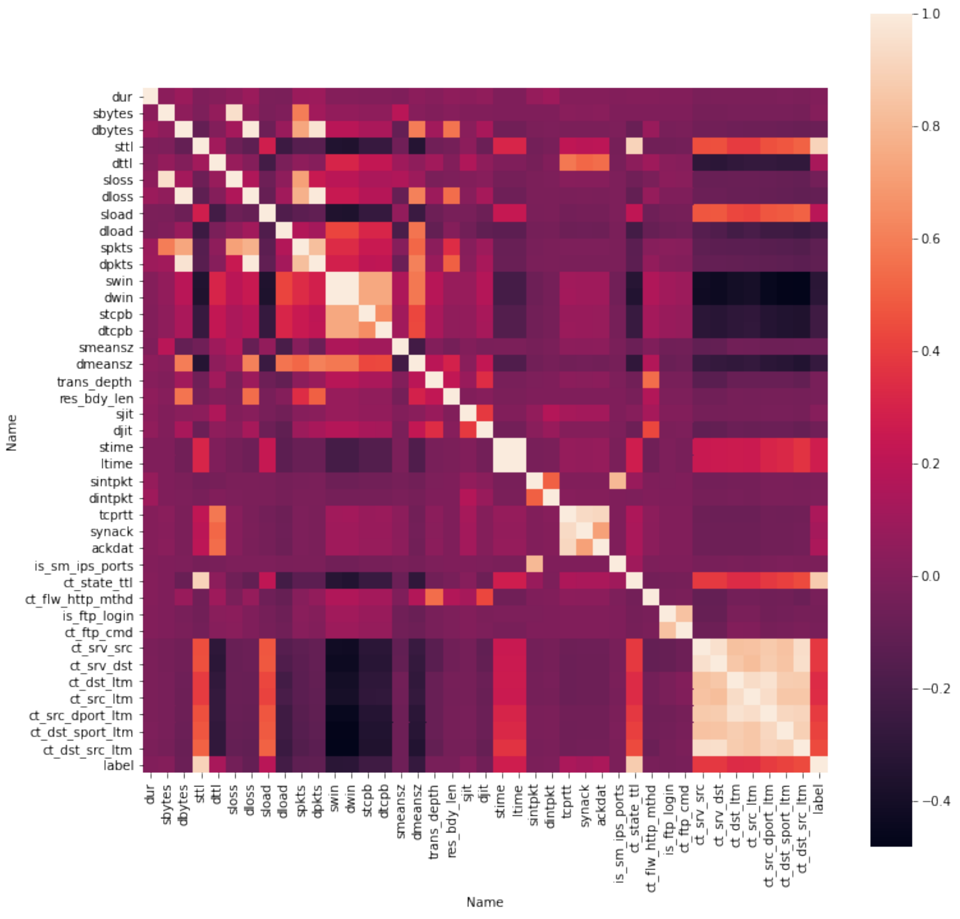


Figure A.1: UNSW-NB15 dataset features correlation heatmap

Listing A.20: For test

A.2.4 Deep Neural Network Model Performance Evaluation Using UNSW-NB15

The figures A.3,A.4, A.8, A.6, and A.5 show the graphical representation of the deep neural network models' results with different configuration setups of MPL, LSTM, and GRU.

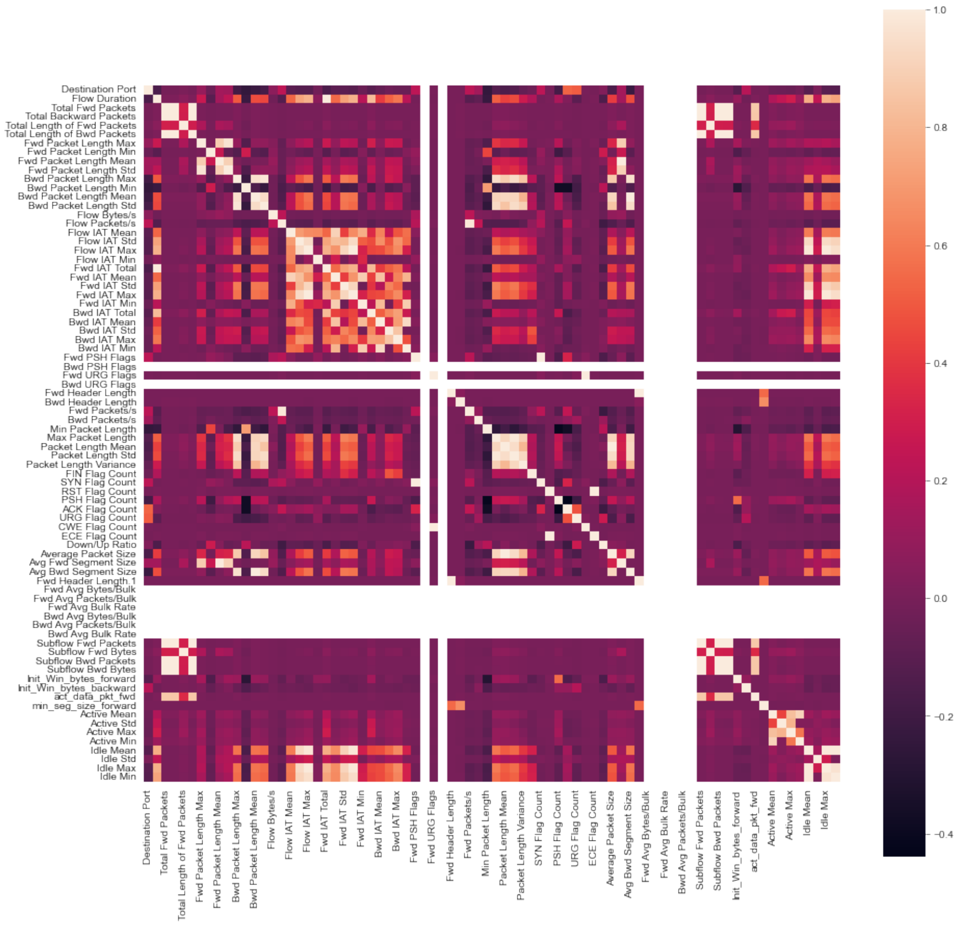


Figure A.2: CICIDS2017 dataset features correlation heatmap

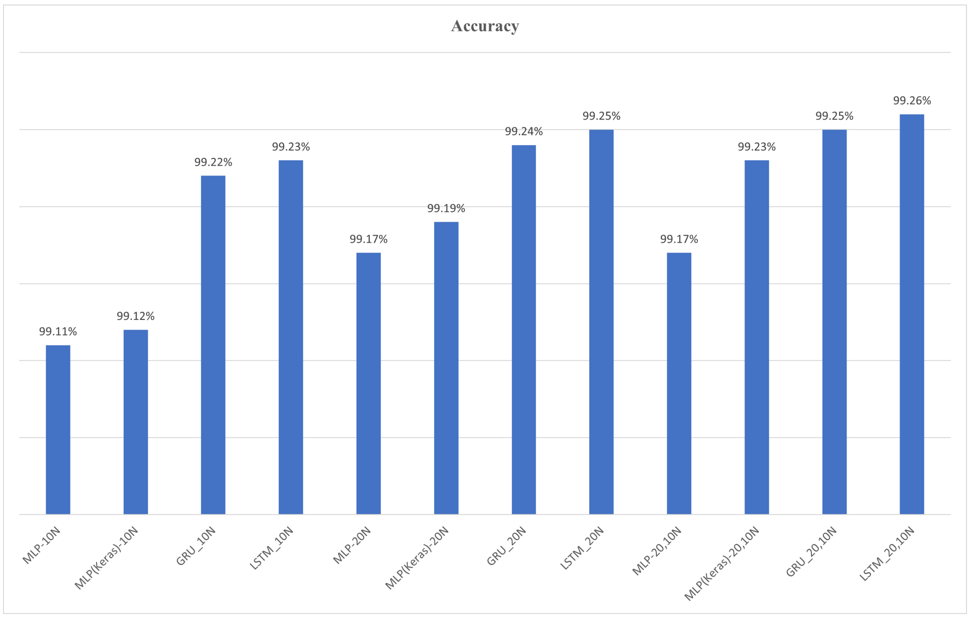


Figure A.3: Acuracy of MLP, MLP (Keras), LSTM and GRU on UNSW-NB15

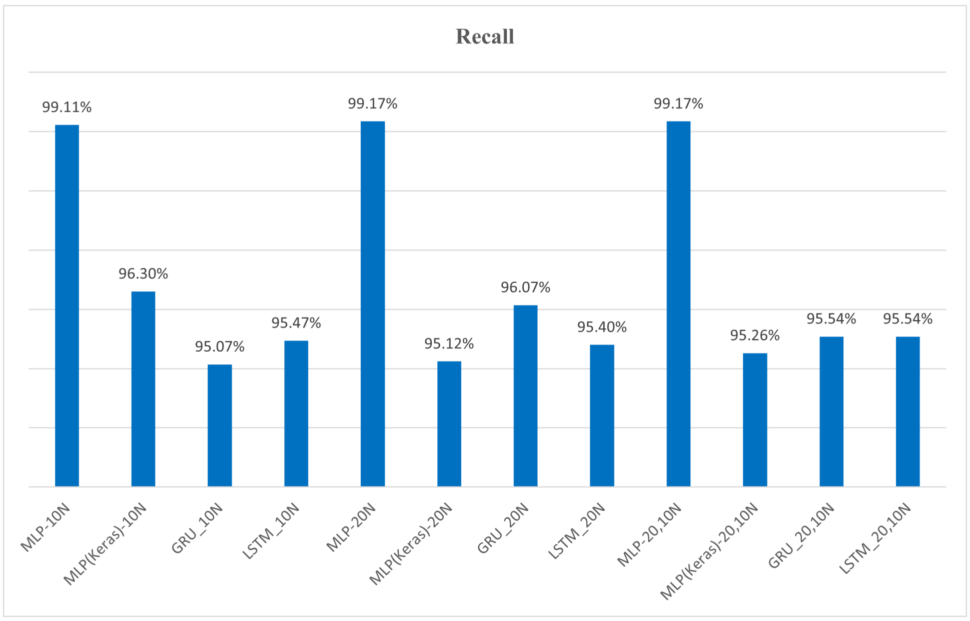


Figure A.4: Recall of MLP, MLP (Keras), LSTM and GRU on UNSW-NB15

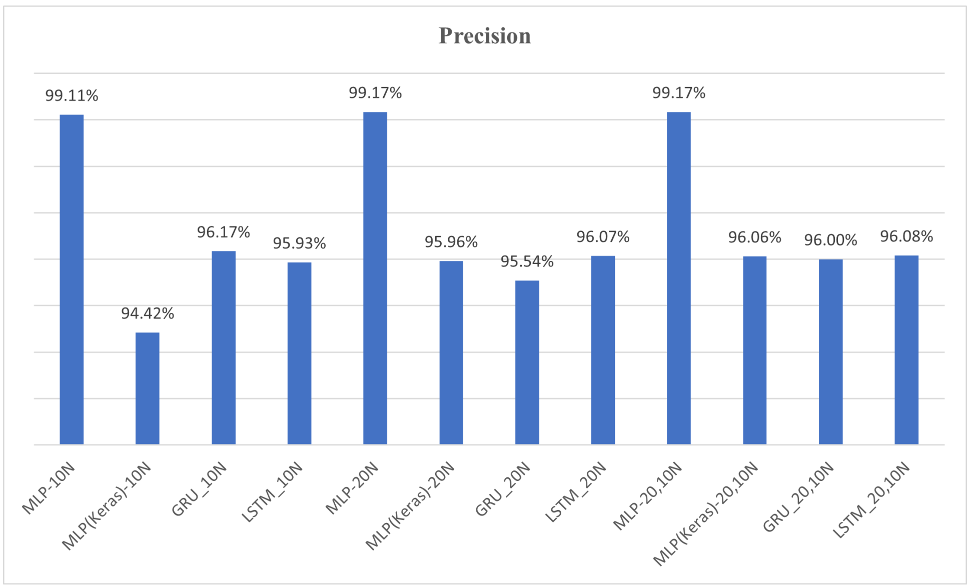


Figure A.5: Precision of MLP,MLP(Keras), LSTM and GRU on UNSW-NB15

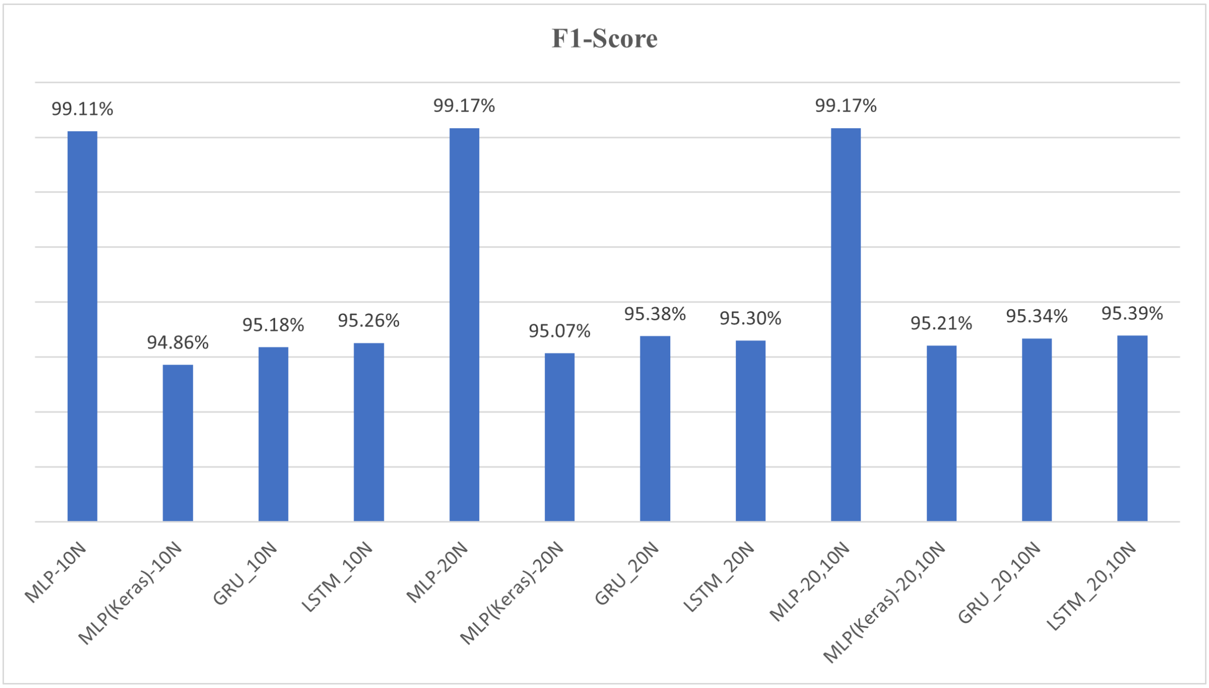


Figure A.6: F1-Score of MLP, MLP (Keras), LSTM and GRU on UNSW-NB15

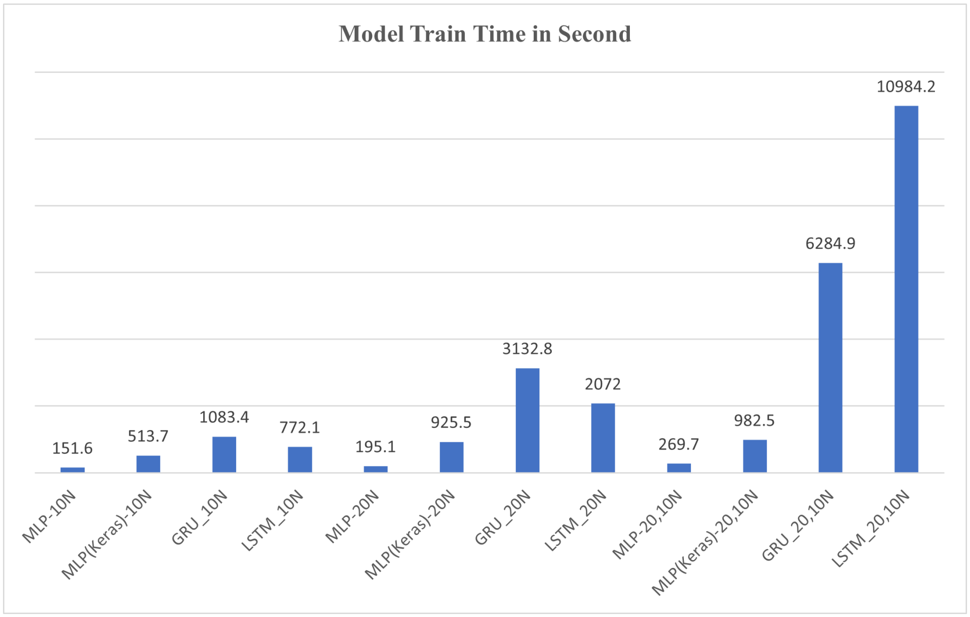


Figure A.7: Model Training Time of MLP, MLP (Keras), LSTM and GRU on UNSW-NB15

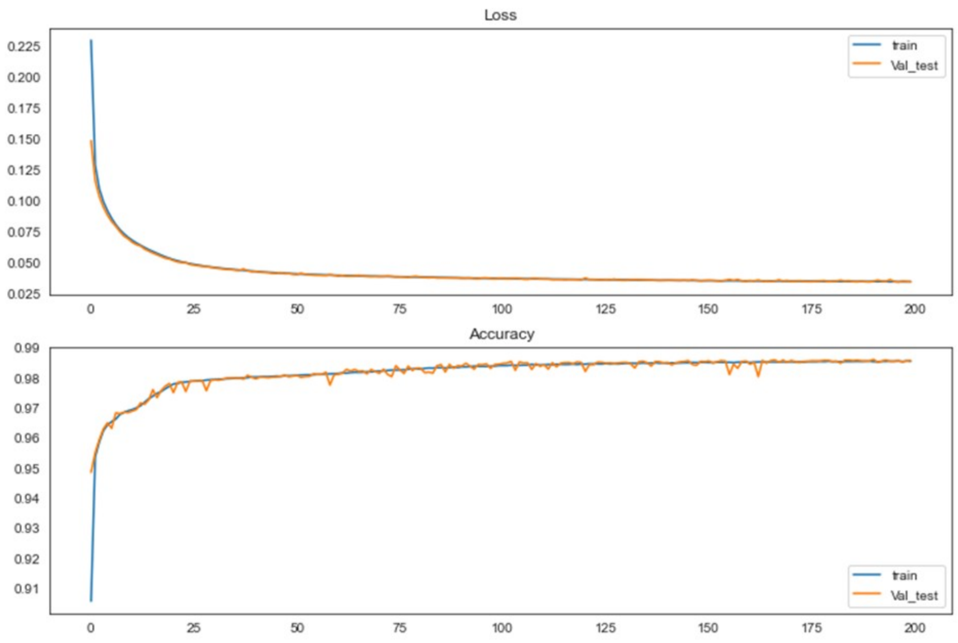


Figure A.8: Model Train Test Validation of LSTM on UNSW-NB15