THESIS ABSTRACT

Master of Science in Applied Computer Science
Cybersecurity Option

Adventist University of Africa

School of Postgraduate Studies

Title: COMBINING META-HEURISTIC TECHNIQUE AND NEURAL NETWORKS TO DETECT INTERNET OF THINGS NETWORK ATTACKS

Researcher: Steve SAMBA

Primary Adviser: Lossan BONDE, PhD

Date Completed: March 2023

The Internet of Things (IoT) refers to ordinary objects equipped with wearable sensors and batteries that can communicate over the internet and perform predefined actions. These devices are evading our everyday lives in many ways. It is now possible to sense temperature, and heart rate with a smartphone, while cloud applications can monitor security systems or smart home equipment. Consequently, IoT networks have simplified life.

However, the growing popularity of Internet of Things devices poses security concerns that need attention. For instance, attackers may target IoT networks for several reasons, including a quest for personal, medical, or financial information and espionage. In certain circumstances, these attacks can have severe repercussions for people's lives. Others may face extortion, damaged reputation, impersonation, fraud, or financial damages.

Detection is critical for defending IoT networks and avoiding the negative repercussions of cyberattacks. Detection consists of identifying assaults before they occur. Numerous writers have examined the security of IoT networks and proposed promising solutions based on machine learning.

This research investigated how efficient could combining Neural Networks and Metaheuristic technique be in detecting IoT network attacks. To address that concern this study proposed a novel method that integrates neural networks for attack classification and Particle Swarm Optimization, a metaheuristic strategy for feature selection and hyperparameter tuning.

The outcomes of the suggested strategy using two different IoT data sets, namely the BaIoT and the CICIDS 2017 datasets yielded accuracy scores of 98% and 99.95% for multiclass classification. The binary categorization was nearly flawless. Furthermore, this study revealed the potential of CNN, MLP and FFNN when dealing with classification problems for IoT environments. The study also highlighted interesting future venues for improving IoT network security, such as deployment, training models with higher quality datasets, or even tweaking more parameters.

Adventist University of Africa

School of Postgraduate Studies

COMBINING META-HEURISTIC TECHNIQUE AND NEURAL NETWORKS TO DETECT INTERNET OF THINGS NETWORK ATTACKS

A thesis

presented in partial fulfillment

of the requirements for the degree

Master of Science in Applied Computer Science
Cybersecurity Option

by

Steve Samba

March 2023

COMBINING META-HEURISTIC TECHNIQUE AND NEURAL NETWORKS TO DETECT INTERNET OF THINGS NETWORK ATTACKS

A thesis

presented in partial fulfillment

of the requirements for the degree

Master of Science in Applied Computer Science
Cybersecurity Option

by

Steve Samba

APPROVAL BY THE COMMITTEE:

Primary Adviser Lossan Bonde, PhD	Program Director, MPH Lossan Bonde, PhD
Secondary Adviser Paul-Marie Moulema, PhD	Head, Department of Applied Sciences Prof. Daniel Ganu, DrPH
External Examiner Awodele Oludela, PhD	Dean, School of Postgraduate Studies Lossan Bonde, PhD

Extension Site:

Date: March 2023

To my parents Dieudonne and Seraphine SAMBA, who have always been my sources of inspiration and unwavering support. Your love and encouragement have been the driving force behind my academic journey and I am deeply grateful for everything you have done for me. This dedication is a small token of my appreciation for all that you have given me. Thank you for believing in me and for always being there. I dedicate this thesis to you with love and admiration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IoT	Internet of Things
AUA	Adventist University of Africa
API	Application Programming Interface
CIA	Confidentiality-Integrity-Availability
IT	Information Technology
CNN	Convolutional Neural Network
DNS	Domain Name Server
PSO	Particle Swarm Optimization
FFNN	Feedforward Neural Network
FTP	File Transfer Protocol
GA	Genetic Algorithm
Hp	Hewlett Packard
USD	United States Dollar
IDE	Integrated Development Environment
IDS	Intrusion Detection System
DoS	Denial of Services
DDoS	Distributed Denial of Services
IoT	Internet of Things
SCA	Side Chanel Attack
KNN	K-Nearest Neighbour
AQU	Aquiler Optimizer
LSTM	Long Short-Term Memory

MLP	Multi-Layer Perceptron
NN	Neural Network
RNN	Recurrent Neural Network
SMOTE	Synthetic Minority Oversampling Technique
SQL	Structured Query Language
CART	Classification and Regression Trees
SSH	Secure Socket Header
SVM	Support Vector Machine
ARM	Advance RISC Machine
ML	Machine Learning
LR	Logistic Regression
RF	Random Forest
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WSN	Wireless Sensor Network
IP	Internet Protocol
RAM	Random Access Memory
TB	Terabyte
GPU	Graphical Processing Unit
GB	Gigabyte
ReLU	Rectifier Linear Unit
SeLU	Scale Exponential Linear Unit

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all those who have contributed to the completion of this thesis. First and foremost, I would like to thank my advisor, Dr Lossan BONDE, for his guidance, support, and encouragement throughout my academic journey. Your wisdom and expertise have been valuable and I am grateful for the time and effort you have invested in me. I am thankful also to my secondary advisor Dr Paul MOULEMA for his precious advices throughout the research. I would also like to extend my thanks to the AUA IT Department for providing me with the computing resources to carry out my experiments.

I am grateful to my friends Daniel KAMGA, Patrick Moye, Fabrice Olinga, Jaures Fondja for their support, and endless encouragement. Finally, I would like to acknowledge the support and love of Lavedrine MIAKATSINDILA, Zhuri SAMBA, Owen SAMBA, Arthur & Seguy SAMBA, Octavia SAMBA, Centhia SAMBA, and Gael THONG who have been there for me every step of the way. Your unwavering belief in me has been a constant source of inspiration and motivation. Thank you to all who have played a role in this journey, and I dedicate this thesis to you with gratitude and appreciation.

CHAPTER 1

INTRODUCTION

Background

The Internet of Things (IoT) refers to common objects equipped with wearable sensors and batteries that can communicate over the Internet and perform predefined actions. According to Bassi et al., the term “Internet of Things” was first coined by Kevin Ashton, a British technology pioneer, in 1999 [1]. He envisioned a world where everyday objects would be connected to the Internet and have the ability to communicate with each other. Over the next few years, the development of sensors, low-power microcontrollers, and wireless communication technology made it possible to bring this vision to life. As technology has matured, the focus has shifted from simply connecting devices to the Internet to using the data collected from those devices to automate processes, increase efficiency, and deliver new services. “Smart” equally describes things like smartphones, smart homes, smart cars, smart cities, or any physical object equipped with sensors, software, and connectivity.

The Internet of Things has changed the way we live because it allows us to manage our homes with a few clicks using smartphones. From adjusting the thermostat to turning on the lights or controlling security cameras, smart home devices allow us to manage our homes from anywhere, anytime. Wearable devices and home sensors allow health professionals to monitor the condition of patients outside the hospital. Such technologies enable 24/7 treatment while freeing up resources for patients who require immediate and direct care. Autonomous vehicles,

often known as self-driving or driverless, can operate without human intervention or control. These cars navigate and perform actions on the road using a combination of sensors, cameras, radar and artificial intelligence. In addition, IoT is being used to make cities more efficient, safer and more sustainable. For example, smart traffic lights can adjust their timing based on real-time traffic data, reducing traffic congestion and improving air quality. Smart grids can monitor, control and optimize the generation, transmission and distribution of electricity.

IoT devices are present in almost all key industries, from retail, healthcare and finance to supply management, industry and urban planning. IoT is undoubtedly gaining considerable popularity over time due to its remarkable efficiency. According to Statista [2], the number of connected devices exceeded one billion in 2006 and exceeded ten billion by 2012. This number has continued to grow, surpassing 18 billion in 2018. With major tech firms such as Google, Apple and Microsoft entering the market, IoT has attracted considerable interest and investment. Today, the Internet of Things is one of the most disruptive technologies of our time, with the potential to transform industries from healthcare and transportation to manufacturing and energy.

According to Statista [2], the number of IoT devices worldwide is expected to grow from 9.8 billion devices in 2020 to more than 22 billion by 2030, with global spending expected to reach 1 trillion dollars by 2023. Home technology and media devices such as smartphones are the primary use cases for IoT devices, with the number of IoT devices expected to exceed 17 billion by 2030 [2]. Autonomous cars, IT infrastructure, asset tracking and monitoring, and smart grids are some of the other use cases that will include more than one billion IoT devices by 2030. Healthcare also represents an opportunity for IoT technology, with the market share of IoT-enabled

health products expected to reach $267 billion by 2023.Despite their widespread adoption, IoT devices are still extremely vulnerable to cyber threats.

Problem Statement

The rapid ubiquity of IoT devices and their vulnerability to attacks creates new security concerns that can affect the reputation, privacy, and finances of businesses and individuals. During the recent COVID-19 pandemic, more people were working remotely and using personal devices (smartphones, laptops, tablets) to share and access company data. This situation increased opportunities for attackers to exploit vulnerabilities in IoT environments to gain unauthorized access to sensitive information and conduct sophisticated attacks.

In 2020, JSOF Lab researchers [3] discovered a series of 19 zero-day vulnerabilities in low-level TCP/IP software libraries integrated into many IoT devices. The Ripple20 vulnerability [3] affects hundreds of millions (or more) of devices and includes a variety of remote code execution vulnerabilities. Attackers could exploit them over the Internet from outside the network and even from devices not connected to the Internet. Affected companies ranged from individuals to Fortune 500 multinationals such as HP, Schneider Electric, Intel, Rockwell Automation, Caterpillar, and Baxter.

In 2021, the same team of researchers discovered seven vulnerabilities in the DNS software used by many IoT devices [4]. Dubbed “DNSPooq,” the flaws could allow attackers to execute malware and take complete control of compromised equipment remotely. It has targeted companies such as Android/Google, Comcast, Cisco, Redhat, Netgear and Ubiquiti. In 2021 researchers also identified 1.5 billion attacks on IoT devices by malicious actors intending to access personal and corporate data, mine cryptocurrency, and build botnets [5]. Other attacks included Denial of

Services (DoS), Distributed Denial of Services (DDoS) and ransomware. Cyber threats have expanded from targeting and damaging traditional devices such as computers, networks and smartphones to people, cars, trains, planes, power grids and anything with embedded chips. Many of these are connected to corporate networks, complicating security.

These attacks violate the Confidentiality-Integrity-Availability (CIA) triad and thus can have severe consequences on people and companies. Moreover, projections estimate that the global cost of cybercrime will reach 10.8 trillion USD by 2025, representing the third economy after the United States and China [6]. Rapid detection is critical for preventing, mitigating, and avoiding such assaults and their detrimental repercussions.

Intrusion Detection Systems (IDS) are tools or programs that monitor network traffic to discover malicious or suspicious activity within the network [7]. They help identify unauthorized access coupled with unknown or suspicious traffic that could compromise the Confidentiality-Integrity-Availability (CIA) triad of network infrastructures. In other words, an IDS examines traffic or events against a pre-configured set of rules or signatures and alerts administrators if they detect any activity that matches those rules. Once an IDS detects a potential threat, it will alert administrators through several means, such as email, text messages, or a dashboard interface.

However, building an IDS for IoT requires acknowledging the limitations of IoT devices like storage and computing power [7]. Hence, the ideal IDS for IoT networks must possess some built-in abilities such as scalability, robustness portability, high detection rate, low false alarm rate, and low resource consumption [8]. The proposed system should be light enough to minimize computing time and

resources while guaranteeing high detection accuracy. Research conducted on intrusion detection systems for the internet of things showed the potential of machine learning to build solid security solutions [1].

Machine learning algorithms used to develop intrusion detection systems for IoT contexts include Support Vector Machine (SVM), K-nearest neighbors (KNN), genetic algorithms, clustering, neural networks, and fuzzy logic [9] through learning approaches such as supervised, unsupervised, reinforcement learning, and deep learning [1]. However, these machine learning-based IDS failed to adapt and protect IoT environments. As for illustration, related research showed that while there is a tradeoff between performance and the reduction of false positive rate in some detection techniques, the type of attacks detected and the detection accuracy are still a concern in other approaches [10]. Furthermore, some datasets used to train detection algorithms, such as the Knowledge Discovery in Databases (KDD-99) or Network Security Laboratory – Knowledge Discovery in Databases (NSL-KDD) dataset, are obsolete, meaning they do not account for new types of attacks, such as zero-day vulnerabilities [1].

As a result, this work aims at building a more robust detection mechanism for the internet of things by combining deep learning and metaheuristic. The idea of combining neural networks and metaheuristics came from [8] in a review on machine learning-based IDS for IoT researchers suggested that combining Neural networks with meta-heuristics algorithms to build a more powerful IDS could significantly overcome the limitations above.

Neural Networks are deep learning techniques that mimic the human brain and identify complex patterns even when dealing with incomplete or massive information [11]. Some use cases of deep learning algorithms include pattern recognition, image

classification, speech recognition, computer vision, and face alignment. Some characteristics of neural networks include robustness, fault-tolerance, flexibility, real-time, ability to deal with heterogeneous data and scalability [11].

In contrast, meta-heuristic algorithms are nature-inspired algorithms that aim to find the optimal solution to a complex problem in a reasonable time. Meta-heuristics are computationally intelligent and can handle complicated optimization issues by reducing computing costs, classification, and storage [12]. They are computationally intelligent and can cope with complex optimization problems by enhancing computation needs [12].

Although deep learning possesses extraordinary potential, it desperately requires larges historical datasets and intensive training to perform adequately. Training time increases computing resources and does not always guarantee the quality of the model. Feature selection and hyperparameter tuning are critical when developing computationally efficient machine learning models [13], [14]. Feature selection aims to reduce the size of the dataset by identifying and discriminating irrelevant or redundant features that can decrease the quality of models [15], While hyperparameter tuning is the process of selecting the optimal set of hyperparameters for a machine learning algorithm.

Hence, this research intends to design an anomaly-based intrusion detection system for IoT environments that will combine a Meta-Heuristic technique, Particle Swarm Optimization and neural networks. Particle swarm optimization is used for feature selection and hyperparameter tuning, whereas neural networks are employed for attack classification. The resulting model can perform binary and multi-class categorization with a high accuracy score and a low rate of false positive rate alerts on one hand while minimizing the resources used on the other. Furthermore, a series of

experiments demonstrate that the research effectively contributes to securing IoT networks.

Research Question

How effective could combining meta-heuristic techniques with neural networks be in detecting IoT network attacks?

Support Questions

1. What is the detection rate of a combined meta-heuristics and Neural Networks solution compared to best neural networks solution?
2. What is the percentage of false alarms of a combined meta-heuristics and neural network solution compared to best neural networks solution?
3. What are the different types of attacks detected by a combined meta-heuristics and neural network compared to best neural networks solution?
4. What is the resource consumption ratio of a combined meta-heuristics and Neural Networks compared to best neural networks solution?

To answer to those questions, we developed and compared five different types of neural network architectures on two IoT datasets. Furthermore, we used particle swarm optimization as metaheuristic technique to reduce data dimensionality and tune model hyperparameters. The best model is compared to existing works.

Study Objectives

The objectives of this study are as follow:

1. **Minimize false positives:** The combination of neural networks and meta-heuristics can reduce false positive alarms, thus minimizing the need for manual intervention and reducing the risk of missed attacks.
2. **Increased efficiency:** The system can reduce the computational burden on the network by using efficient algorithms while maintaining high accuracy. By combining the strengths of neural networks and meta-heuristics, the system provides a comprehensive defense against cyberattacks while offering efficient resource utilization.

Significance of the Study

The benefits of this study are at many levels:

1. Secure data: IoT devices generate and transmit large amounts of sensitive data, which can be vulnerable to cyberattacks. By protecting IoT networks, organizations can ensure their data is secure and confidential while avoiding reputational and financial damage.

2. Maintain device functionality: Cyberattacks can cause IoT devices to malfunction, leading to downtime and loss of productivity. By protecting IoT networks, organizations can maintain the functionality of their devices and ensure that they continue to operate smoothly. This allows companies to avoid service disruptions while maintaining business continuity.

3. Prevent unauthorized access: IoT devices can be vulnerable to unauthorized access, which can compromise the security of the network and the data it transmits. By protecting IoT networks, organizations can prevent unauthorized access and maintain the integrity of their systems.

4. Compliance: Many industries have regulations and standards that require organizations to protect their networks and data from cyberattacks. By protecting IoT networks, organizations can maintain compliance and avoid legal and financial penalties.

5. Build trust: Organizations that protect their IoT networks can demonstrate to their customers, partners, and other stakeholders that they are committed to security and privacy. This can build trust and enhance the reputation of the organization.

6. Financial impact: with the cost of cyberattack increasing more and more, early attack detection and mitigation can result in cost saving for organizations and individuals.

In summary, detecting IoT networks cyberattacks provides numerous benefits, including securing data, avoiding data breaches, maintaining device functionality, avoiding service disruptions, preventing unauthorized access, compliance with regulations, and increasing trust.

Scope and Limitations

The research designed a detection system for the Internet of Things networks. The proposed system combined various neural networks for attack detection while metaheuristics served to optimise the datasets and neural network architectures. The study considered neither intrusion response or attack mitigation nor real-time

deployment. Since operations like hyperparameter tuning required considerable computing resources and time, the study considered a limited number of hyperparameters. Moreover, the study retained five neural network architectures for attack detection, and PSO was the metaheuristic used for feature selection and hyperparameter tuning.

Other limitations of the research included adversarial attacks. To avoid detection, attackers might purposefully modify the input data or exploit flaws in the neural network model. Adapting and improving neural networks to fight against adversarial assaults can be difficult, and extra defences, such as adversarial training or resilient topologies, may be required.

Operational Definition of Terms

Adversarial machine learning attack: this is a sort of attack in which the goal is to cause the machine learning model to generate inaccurate predictions or classifications by altering the input data in a way that humans cannot detect.

Artificial intelligence: a branch of computer science and engineering that focuses on developing machines and software programs that can perform tasks that normally require human intelligence, such as visual perception, speech recognition, decision making, and natural language processing.

Availability: Availability means that data and systems are accessible and usable when needed. This means that authorized users can access data and systems in a timely and efficient manner.

Confidentiality: is an information security principle that guarantees that data may only be accessed and seen by authorized persons.

Integrity: Principle that ensures that data is accurate, complete, and trustworthy. This means that data has not been tampered with, modified or deleted in an unauthorized way.

Semi-supervised learning: is a learning technique combining supervised and unsupervised learning aspects. A model is trained on both labelled and unlabelled data in this technique, with the goal of utilizing the structure in the unlabelled data to improve the model's performance on the labelled data.

Supervised learning: is a machine learning approach that involves training a model using labelled data, which means that each data point relates to a predefined objective or outcome.

Unsupervised learning: involves training a model on unlabelled data, which means there are no known objectives or outputs. The model's purpose is to discover patterns or structure in the data that may be utilized for tasks like clustering, dimensionality reduction, and anomaly detection.

Zero-day-vulnerability: is a software flaw that is unknown to the developer or the general public. The term “zero-day” alludes to the notion that the developer has 0 days to fix the vulnerability before attackers may exploit it.

CHAPTER 2

LITERATURE REVIEW

This literature review aims to establish whether or not the problem to address by this research – Combining meta-heuristic techniques and Neural Networks to detect IoT network attacks – is of interest and fills an existing gap. This research assumes that combining meta-heuristics techniques and neural networks may improve the security of IoT networks. Therefore, the first step was to determine what previous research says about it and identifying any existing gaps.

This literature informs the reader how previous research has dealt with this specific problem and where the present study fits in. The literature study is arranged as follows: the first part discusses the method for identifying and selecting relevant material, the next section summarizes IoT network attacks. The following section provides an overview of solutions discovered in the literature that are based on neural networks, meta-heuristics, or a combination of the two. The chapter ends with a summary of the main gaps identified and henceforth the reason of this research.

Review Method

To identify and select relevant literature that address the issue of detecting attacks in IoT network, the research followed a three steps process: search, selection and analysis of relevant literature. Tools used to search, select and analyze relevant publications included google scholar, semantic scholar, Mendeley, Zotero and Microsoft office.

Step 1 Search

A complete and reliable search of scientific sources requires employing relevant search terms or keywords, a search engine, and a collection of scientific databases. Thus, the search process is about databases and keywords that help to the discovery relevant materials. Table 1, summarizes the search phrases, search engines, and matching search results for this investigation.

Table 1. Summary of Preliminary Search for Literature

Database	Keywords	All Results		Reviewed Results	
		All date	2017 and +	All date	2017 and +
Google scholar	IoT networks attacks (Anywhere)	214,000	92,200	12,000	10,500
Google scholar	IoT networks attacks (In title)	985	896	42	4
Semantic Scholar	IoT networks attacks (Anywhere)	194000	112000	15300	13700
Semantic scholar	IoT networks attacks (In title)	6810	6340	1060	1040
Google scholar	NN, attacks and IoT	46600	17600	7080	6740
Google scholar	Metaheuristics and Machine Learning	66700	16900	5870	4600
Semantic scholar	NN, attacks and IoT	477	468	109	109
Semantic Scholar	Metaheuritics and Machine Learning	1700	1300	170	165
Google scholar	Neural networks, Metaheuristics, attacks and IoT	3770	3620	612	587
Semantic Scholar	Neural networks, Metaheuristics, attacks and IoT	695	683	168	168

To find more specific publications related to detection approaches in IoT, a progressive search was carried out, beginning with publications dealing with IoT attacks. After that, we narrow down our search to research publications concerning detection techniques using neural networks, metaheuristics or a combination of the two. Titles that included exactly ‘'neural network, meta-heuristics, attacks and IoT"

and published between 2017 and today were hard to find, one of the reasons could be that metaheuristics is a broad term that refers to a set nature inspired technique.

Step 2 Selection

The targeted sources for the research are scholarly peer-reviewed articles dealing with neural networks /and metaheuristic techniques for detecting threats in IoT networks. The sources considered for this research comprised journals, conference publications, thesis, dissertations and book chapters. For quality considerations, web articles, essays and novels were not considered. Other selection factors include the year of publication.

Journal papers, conference proceedings, theses, and dissertations from 2017 and beyond were included. Short articles of fewer than five pages, essays, and novels were excluded. As a result of search and selection steps, 106 papers were identified for further analysis.

Step 3 Study

We subdivided this phase into two parts. The first part analyses the title of the paper and abstract to determine their relevance to the study. The second phase involved cross-referencing and backwards searching. Works identified during the selection phase were inspected and pertinent articles were retrieved from their references. This process raised the number of sources to consider from 106 to 116 distributed as follow in **Error! Reference source not found.**.

Table 2. Repartition of the Literature

Problematics addressed	Number of papers
Internet of Things (IoT) Attacks	28
Neural Networks for Detecting IoT Attacks	76
Combining Neural Networks and Metaheuristic for Detecting IoT Attacks	12

All the articles gathered were processed in a spreadsheet, recording information such as the paper number, title, authors, publication year, venue, publishing type (conference or journal), and research kind (new method, modification, hybrid, comparisons and analysis, or survey).

Internet of Things Network Attacks

Hassija et al. [16] considered a four-layer IoT architecture and exposed vulnerabilities and attacks that may occur at each layer. Findings revealed that attacks such as node capturing, malicious code injection, false data injection, Side-Channel Attack (SCA), eavesdropping and interference, sleep deprivation, and booting attack might affect the sensing layer.

Threats at the network layer include Denial of Services (DOS)/ Distributed Denial of Services (DDOS), routing attacks, phishing, data transmission, and access. Attacks at the middleware-layer concerned man-in-the-middle, SQL injection, signature wrapping, and flooding. Malicious code injection, sniffing, data theft, access control attacks, service disruption, and reprogramming assaults can all occur at the application layer of IoT. The study also considered vulnerabilities at the gateway layer like end-to-end encryption, secure onboarding and firmware updates.

Similarly Krishna et al. [17], surveyed IoT threats and attacks on IoT considering three different architectures: a five-layer (perception layer, network layer, service layer, operation layer, application layer), a seven-layer (perception layer, abstraction layer, network layer, transport layer, computing layer, operation layer, application layer) and the existing three-layer (perception layer, network layer, application layer) architecture. An exhaustive list of possible attacks at each layer and their impact on IoT as shown in Table 3, as illustrated in [17].

Table 3. Summary of Attacks per Layer

Layer	Attacks
Perception	Eavesdropping, Injection attacks, Sybil Attack, Side-Channel, Attack, Node cloning, Exhaustion attack
Abstraction	Illegal access, Man-in-the-middle, spoofing, protocol attacks, sleep deprivation, DoS/DDoS
Network	Hello flood, hole attacks, RPL exploits
Transport	Desynchronization, session hijacking, DoS/DDoS
Computing	Phishing, SQL-injection, software modification, DoS/DDoS
Operation	Man-in-the-middle, firmware attack, unsecure onboarding, software attack
Application	Malware attacks, injection attacks, DDoS, flooding, spoofing

Khanam et al. [18] went a step further by giving a taxonomy of the most recent and significant IoT security threats. The study contrasted parameters such as kind of device (low-end vs high-end), attack location (internal vs external), attack technique (active vs passive), damage level, and protocol attack (protocol deviation, protocol disruption). Even though the research focused on a three-layer architecture (application, network, and transport), it gave clues on attacks that might harm IoT at each layer. Additionally, the study highlighted attacks that might affect multiple layers like the man-in the middle attack, DoS/DDoS, side channel and cryptanalysis.

Following the same trend, Butun et al. [19] investigated IoT and Wireless Sensor Networks (WSNs) attacks and classified them into two types: passive and active attacks. Passive cyberattacks involve attackers concealing themselves with camouflage and tapping communications to obtain data. Passive assaults, which include eavesdropping, node malfunctioning, node tampering/destruction, and traffic analysis, primarily target data confidentiality. On the other hand, active cyberattacks aim not just for data secrecy but also data integrity and availability. In this case, attackers intend to gain illegal access to and use resources, as well as disrupt communications. Examples of such attacks include injection attacks, Dos/DDoS, spoofing, jamming, flooding, hole attacks, and Sybil attacks. More practically,

Stellios et al. [20] surveyed IoT-enabled attacks concerning critical IoT infrastructures such as smart power grids, intelligent transportation systems, smart homes, industrial control systems like Supervisory Control and Data Acquisition (SCADA) and e-health services. The research considered real-world incidents or attacks experimented on and published by researchers.

The study looked at genuine attack paths that allow, exploit, or intensify vulnerabilities of IoT devices. Interestingly, regardless of the sector examined in the study, the success of the attacks depended on one or more of the following factors: the location of the attack, the exploitation of communication interfaces (network, physical), and hidden functionalities provided by IoT devices.

However, literature also suggested ways to address these attacks to better secure IoT networks using neural networks. Despite the fact that there were multiple IoT-related attacks listed above, other forms of attacks, such as zero-day vulnerabilities and adversarial machine learning, were omitted and so require further attention.

Neural Networks for Detecting Internet of Thing Attacks

HaddadPajouh et al. [21] proposed a signature-based detection technique using a recurrent neural network. The proposed method was capable of analyzing ARM-based IoT application operation codes and detecting new malware. They detected unknown malware with 98,18% accuracy utilizing 2-layer neurons and specifically Long Short-Term Memory (LSTM), a form of Recurrent Neural Network (RNN) as the classifier Results were then compared with other Machine learning (ML) approaches like KNN, SVM, Decision tree, and Multi-Layer Perceptron (MLP) and showed that the proposed technique surpasses the others. However, this technique was

tested on a small sample of data and can only detect malware. Other parameters, such as false alarm rate and computation time, were not factored in.

Smys et al. [22] also concerned with the security of IoT, suggested a hybrid detection method using a convolution neural network. This technique combined LSTM for feature selection and Convolutional Neural Network (CNN) for intrusion detection. This technique takes advantage of the computation efficiency of LSTM to extract valuable features from the dataset to train the model. Although the data used was not very significant, they achieved an accuracy of 98,6%. This model achieves better than regular Recurrent Neural Networks (RNN) in terms of accuracy, detection time, and recall. However, the type of attacks detected is not specified.

Derhab et al. [23] presented an intrusion detection technique for IoT based on Temporal Convolutional Neural Network (TCNN). TCNN is a variant of CNN that uses causal convolutions. The proposed model considered the limitations of IoT devices like storage, variety of devices and computing power. To achieve an efficient system, the authors performed two prior tasks before classification to reduce the computational complexity of the model: space reduction and feature transformation. Feature space reduction consists of a preprocessing phase to remove noise from the dataset. Feature transformation is the process by which numerical and categorical features are transformed.

The proposed model was benchmarked with other machine learning techniques like Logistic Regression (LR), Random Forest (RF) and deep learning techniques like CNN and LSTM. The findings demonstrate that TCNN can achieve 99.99% accuracy in multiclass classification; nevertheless, it takes longer to train than CNN. One limitation of this study is that 97% of the data utilized to evaluate this approach was made up of DoS and DDoS assaults, implying that other attacks were

underrepresented. To circumvent this constraint, the authors used a technique known as the synthetic minority sampling technique (SMOTE), which generates a synthetic sample of the minority class. Even if it partially solves the oversampling problem, it does not mimic the legitimate traffic behavior of IoT networks. Further testing on diverse datasets is not available to evaluate the scalability of the model.

Combining Neural Networks and Metaheuristic Algorithms for Detecting Internet of Things Attacks

Fatani et al. [24] proposed a new approach that combined the benefits of deep learning and swarm intelligence to select the best features in the datasets to train the machine learning model. Their method involved firstly a CNN-based feature extraction phase and secondly a features selection phase using a modified Aquila optimizer (AQU) on the extracted features to remain with the most relevant. The model's performance was assessed on four well-known datasets CICIDS2017, NSL-KDD, BoT-IoT, and KDD99 and compared with other metaheuristic techniques.

The Metaheuristic algorithms selected for comparison include the Firefly Algorithm (FFA), Particle Swarm Optimization (PSO), Whale Optimization Algorithm (WOA), Moth Flame Optimization (MFO), traditional TSO, multiverse optimization algorithm (MVO), Bat algorithm, and Grey Wolf Optimizer (GWO). The results showed that AQU performs better regarding the accuracy, precision, recall, and F1-score for binary and multiclass classification with respectively 99.997% and 99.910%. However, datasets used for evaluating the model are outdated like KDD99, NSL-KDD.

Bacanin et al. [25] developed a novel training algorithm to improve hyperparameters tuning of the CNN-based IDS using a modified Particle Swarm Optimization (PSO) method. The enhanced PSO increased the exploitation and

exploration abilities of the conventional PSO algorithm. The objective was to detect intrusion accurately in IoT networks and decrease computing costs. The proposed algorithm optimized weight and biases of a fully connected multi-layer perceptron in the CNN. The parameters retained by the enhanced PSO were then used to optimally train the CNN model to improve performance.

To evaluate the performance of this technique, the authors used two network intrusion detection datasets named UNSW-NB15 and Bot-IoT. The experiment results suggested that the proposed network intrusion detection system has the best performance compared to other state-of-the-art schemes. However, the data used to train the model are not enough representative of IoT environments whereby network traffic is huge. Moreover, this work did not consider reducing data dimensionality to gain in accuracy and performance.

Metaheuristics have also been employed as a standalone technique to identify threats in IoT networks. Hajisalem and Babaie [26] implemented an effective IDS for IoT contexts by integrating Artificial Bee Colony (ABC) and Artificial Fish Swarm (AFS). These two metaheuristic algorithms are used for feature selection and extraction to guarantee low computation complexity and time cost. The suggested approach was evaluated on two datasets, and the results revealed that it outperformed previous strategies in terms of accuracy in detecting normal and anomalous traffic. To help to discriminate malicious traffic from normal ones CART techniques were used to generate if-then rules with the selected features.

The issue with this method is that specifying all attack scenarios using if-then rules may be time-consuming. New attack variations, such as zero-day attacks, are developing as attackers become more creative.

Gap Analysis

Finally, the amount of relevant material demonstrates that the proposed study is worthwhile. In summary, there is an abundant literature on attacks on IoT networks. Similarly, the literature also provided countermeasures that uses machine learning techniques and/or metaheuristics to detect IoT network attacks. The predominant methods utilized in the literature to develop attacks classifiers were Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM), which produced remarkable outcomes. However, the datasets utilized in several of the research were either very tiny or old. Some systems, on the other hand, were only capable of detecting a small number of attacks. Furthermore, researchers started lately investigating the use of a combination of neural networks and metaheuristics to identify IoT assaults. Even though further experiments with various meta-heuristics methodologies and optimal settings for neural networks are needed to judge the performance of such a tandem, earlier findings appear promising.

As a result, this research proposed to overcome the limits of previous works by using Neural Networks for attacks detection along with particle swarm optimization for dimensionality reduction and hyperparameters tuning. The study considered recent datasets (no older than 2017) comprising at least one million observations and a variety of attacks (at least five).

CHAPTER 3

METHODOLOGY

This research was both exploratory and experimental. The study initially investigated the problem of detecting attacks in IoT networks by identifying attacks, countermeasures, and weaknesses of those solutions. This study was also experimental because, based on the findings gathered in the exploratory phase, different neural networks configurations were implemented, compared and evaluated. The resultant model is robust enough to overcome the limits of previous works regarding performance. The rest of the chapter firstly presents the conceptual framework of the experiment, the various theories, methods, and tools needed to implement the proposed solution, then finally discusses the datasets.

Conceptual Framework

The research proposed a three steps process as framework to assess the efficiency of combined neural networks along with metaheuristic in detecting attacks in IoT networks. Step 1consisted to acquire the datasets, pre-process them make predictions and evaluate the models. Initially the hyperparameters of the models are picked empirically. The second step is about applying PSO on pre-process dataset for feature selection. During this step relevant attributes are extracted from the dataset and trained with the previous models. Models are evaluated again to catch any gain from previous measurements. Finally, during the last step, following the step 2, the models hyperparameters are optimized using. The operation results in new

hyperparameters that were used to train models again on the engineer data. Models are evaluated again to perceive any improvement, as seen in Figure 1.

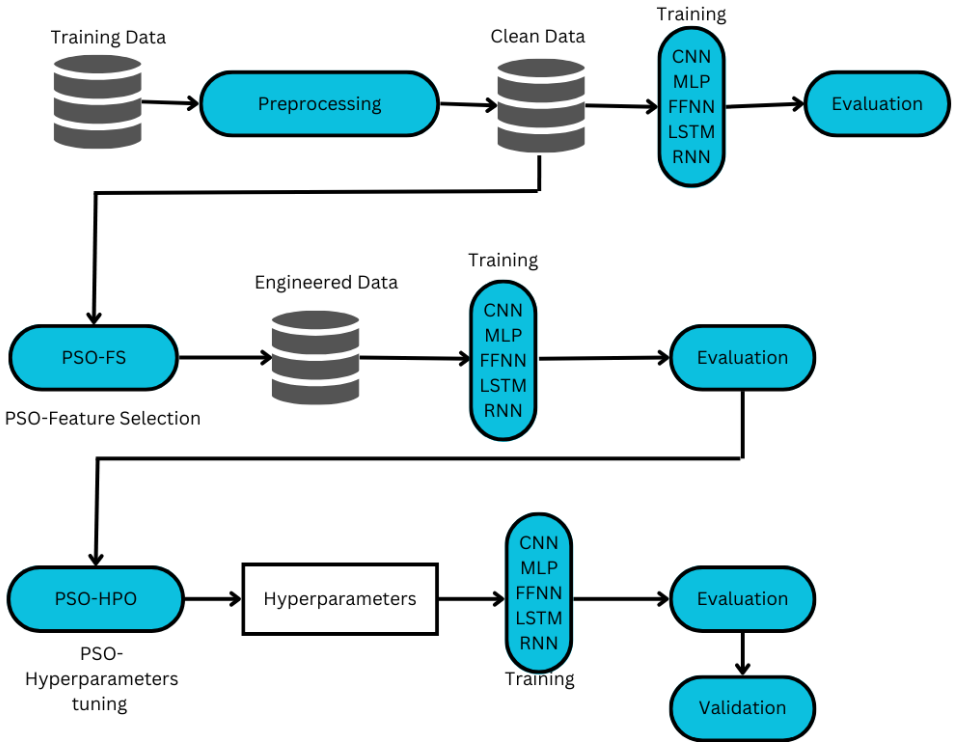

Figure 1. Design of the Research Framework

After all the training the same methodology is apply on the CICIDS2017 dataset and compare to previous works.

1. **Training Datasets**: The first step in implementing an efficient solution for IoT networks consist of to acquire the data described above.

2. **Data Pre-processing**: During this step, data are explored, and cleaned using libraries like Pandas and NumPy.

3. **Training:** Using TensorFlow and Keras, five distinct Neural Network classifier will be built: Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Feedforward Neural Network (FFNN), and Multi-layer Perceptron (MLP).

4. **Feature selection using PSO:** The next step will consist of using Particle Swarm Optimizer (PSO) for data dimensionality reduction to remove unnecessary features and save training resources.
5. **Hyperparameters Optimization:** Furthermore, Particle Swarm Optimizer (PSO) will be used to fine-tune the hyperparameters of each model to retain the values that produce the highest accuracy.
6. **Evaluation:** Models will be reviewed and compared to one another at each stage.
7. **Validation**: Finally, the best model among the five is assessed to previous works using criteria like accuracy, precision, f1-score, and recall.

Theories, Methods, and Tools

Theories

Artificial Neural Networks (ANN). Neural Networks or Artificial Neural Networks are structures comprised of densely interconnected adaptive simple processing elements called artificial neurons or nodes that can perform massively parallel computations for data processing and knowledge representation [49,50]. Neural networks reflect the behavior of the human brain, allowing computer programs to recognize patterns and solve complex real-world problems.

Research on ANN has experienced three main periods of extensive activity. Later in the 1940s, McCulloch and Pitts [27] proposed the first computer model of neural activity in cells of the human nervous system. Their research revolved around the functioning of a simple neuron shown in **Error! Reference source not found.**. Inside the artificial neuron, the sum of each x_i input multiplied by a scaling factor or weight w_i is generated.

The weights represent the strength of the synaptic link: positive weights represent euphoric or excitatory effects and negative weights represent inhibitory effects. If the result of the sum is greater than a certain threshold value then the cell returns a positive value (usually +1); otherwise, the output is negative (usually −1) or zero. In general, the model follows neurobiological behavior: neurons produce

nonlinear responses when provided stimulation by a given input. In particular, McCulloch and Pitts [27] proposed an activation function, which explains the nonlinearity of the model, called the hard threshold function (Figure 2).

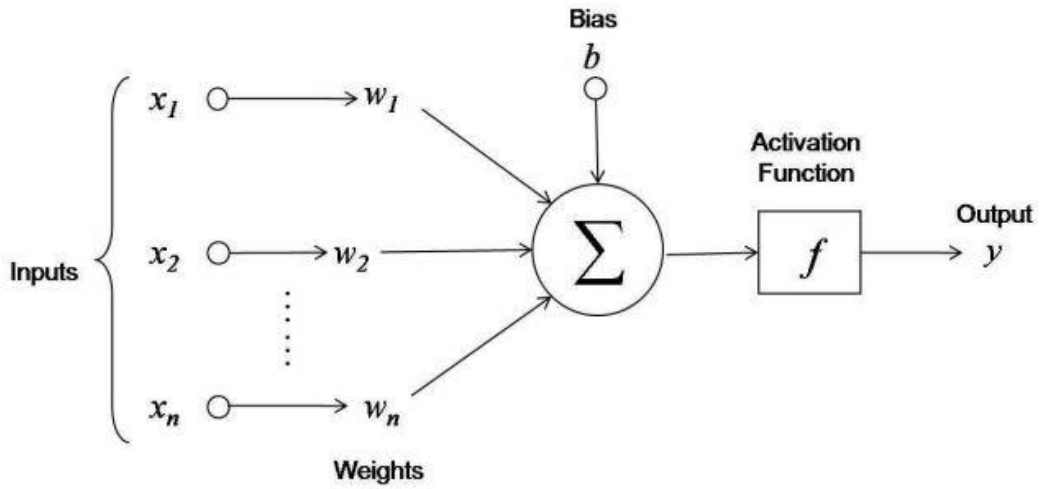

Figure 2. Warren McCulloch and Walter Pitts Neural Network Design

The mathematical expression of a formal neuron is as follows;

Let us call this function Base Function, *u*. In this case, the Base Function is a weighted sum where *b* is a threshold or bias and w_j are the synaptic weights.

(1) $u = b + \sum_{i=1}^{n} w_i x_i$

The output is obtained by applying a non-linear activation function to the Base Function of a formal neuron:

(2) $y = f(u) = f(b + \sum_{i=1}^{n} w_i\, x_i)$

The activation function can have several forms based on the application:

- Exponential function

(3) $f(x) = \frac{1}{1+e^{-\alpha x}}$, α>0

- Hyperbolic function

(4) $f(x) = \frac{1-e^{-\alpha x}}{1+e^{-\alpha x}}$, $\alpha>0$

- Gaussian function

(5) $f(x) = ae^{-\frac{x^2}{\sigma^2}}$

Hebb's rule is a postulate proposed by Donald Hebb in 1949 [28]. It is a learning rule that defines how neural activity impact neuronal connections. Hebb postulated that when a neuron's input and output were both active at the same time, the value of the synaptic link would increase. This strengthens the most frequently used network connections, simulating the biological process of habit and learning via repetition. The classical Hebb's rule indicates "neurons that fire together, wire together [28].

Perceptron. One of the first neural networks was the perceptron. Frank Rosenblatt invented it in 1957 at the Cornell Aeronautical Laboratory [29]. He was motivated by Warren McCulloch and Walter Pitts' [27] prior work on the artificial neuron. A perceptron consists of a set of neurons connected together to form a network. Each of these neurons can receive an input signal either from other units in the network or from the environment, and responds by generating the corresponding output signal, which may be transmitted, through connections, to a selected set of output units [29]. A perceptron is a Feedforward neural network that separates data into two distinct classes.

Typically, a neural network is a layer-based architecture, with the input layer receiving input data, one or more hidden layers processing the data, and the output layer delivering model predictions. There are several varieties of neural networks,

each with its architecture and tailored to handle specific issues. For this study, the following neural network types are considered.

Convolutional neural network. Convolutional Neural Network (CNN) extracts spatial information from input data and pools them to produce a hierarchical representation of the input using convolutional layers. It comprises an input layer, an output layer, and many hidden layers as shown in Figure 3 [30]. The following hyper-parameters define the configuration of a 1D-CNN:

1. Number of hidden layers.
2. Filter (kernel) size in each layer.
3. Subsampling factor in each layer.
4. The choice of pooling and activation operators.

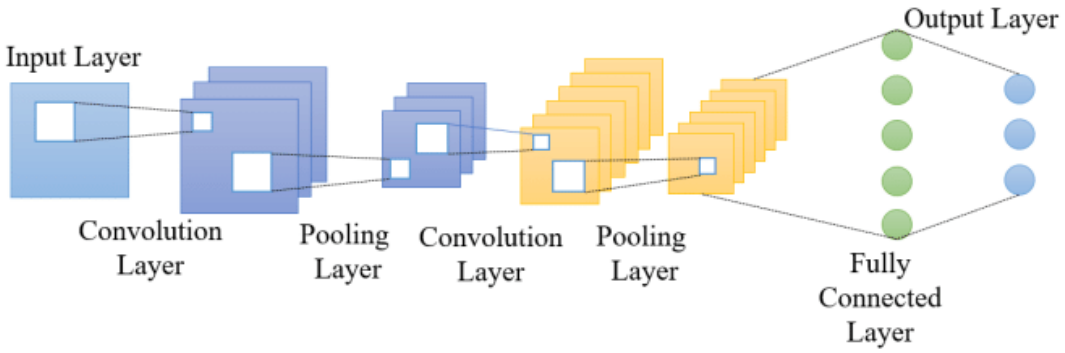

Figure 3. Convolutional Neural Network Architecture

Recurrent neural network. Recurrent Neural Network (RNN) employs feedback loops to let information to persist between time steps and has memory-like capabilities that allow it to record long-term dependencies in the input. RNNs' basic architecture includes input, output, and hidden units, with the hidden units performing all of the calculations to generate the outputs via weight modification. Figure 4 depicts a simple RNN design with two hidden layers [30].

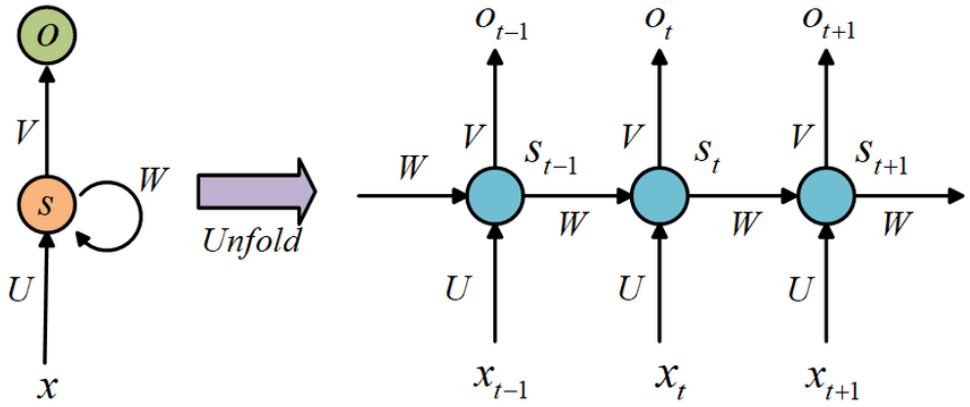

Figure 4. Recurrent Neural Network Architecture

Long-Short Term Memory. Long-Short Term Memory (LSTM) networks replace hidden layers with memory blocks. Each memory block possesses at least one memory cell and is regulated by gates. These gates control the flow of information in the cell, both incoming and outgoing. If the information stored is no longer needed, a forget-gate is positioned between an input gate and an output gate to reset the state of the linear unit. These gates are basic sigmoid threshold units with values ranging from 0 to 1, where zero signifies “nothing passes through”, and one implies “everything is allowed through”. Figure 5 shows one cell in a simple LSTM network [31].

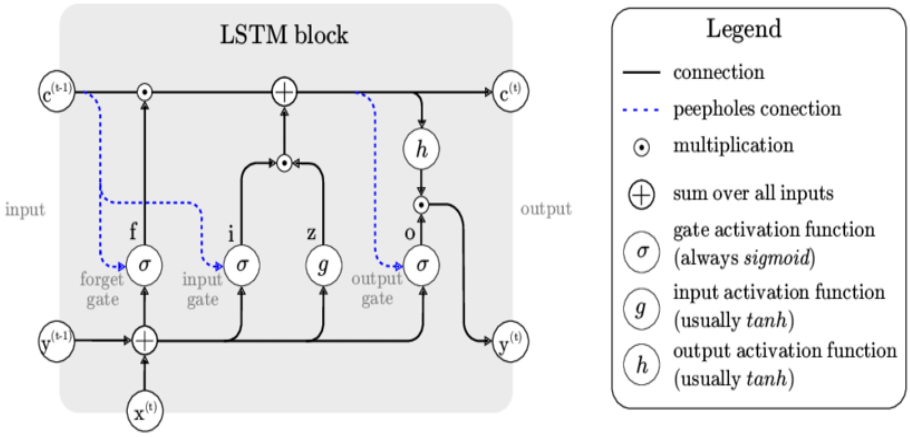

Figure 5. LSTM Cell

Feedforward neural network In Feedforward Neural Network(FFNNs), information flows solely forward, from the input nodes through the hidden nodes to the output nodes. Hidden layers are not mandatory in classic FFNNs. Single-layer perceptron is the simplest form of FFNNs. Figure 6 describes the workflow of a FFNN [32]; data enters the input layer and is multiplied by the weights. The sum of weighted input values is compared to get a certain threshold (normally set to zero). If the total of the values is more than the threshold, the output value is often 1, and if the sum is less than the cut-off value, the output value is -1.

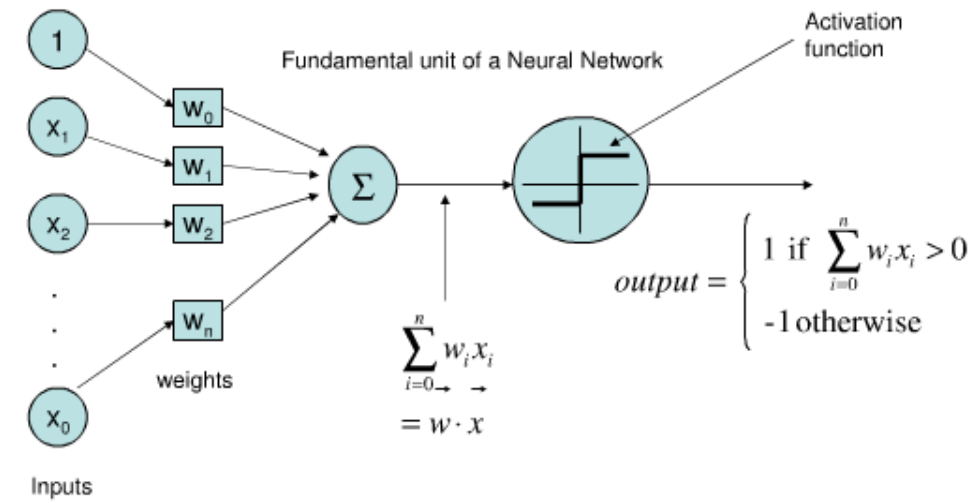

Figure 6. Simple Feedforward Neural Network

Multi-layer perceptron. Figure 7 contains input and output layers, along with one or more hidden layers consisting of numerous neurons layered on top of each other [33]. While neurons in a Perceptron must have an activation function that specifies a threshold, such as Rectified linear Unit (ReLU) or sigmoid, the neurons in a Multi-Layer Perceptron (MLP) can employ any arbitrary activation function.

Backpropagation is a learning method that enables the MLP to continuously update the network's weights to minimize the cost function.

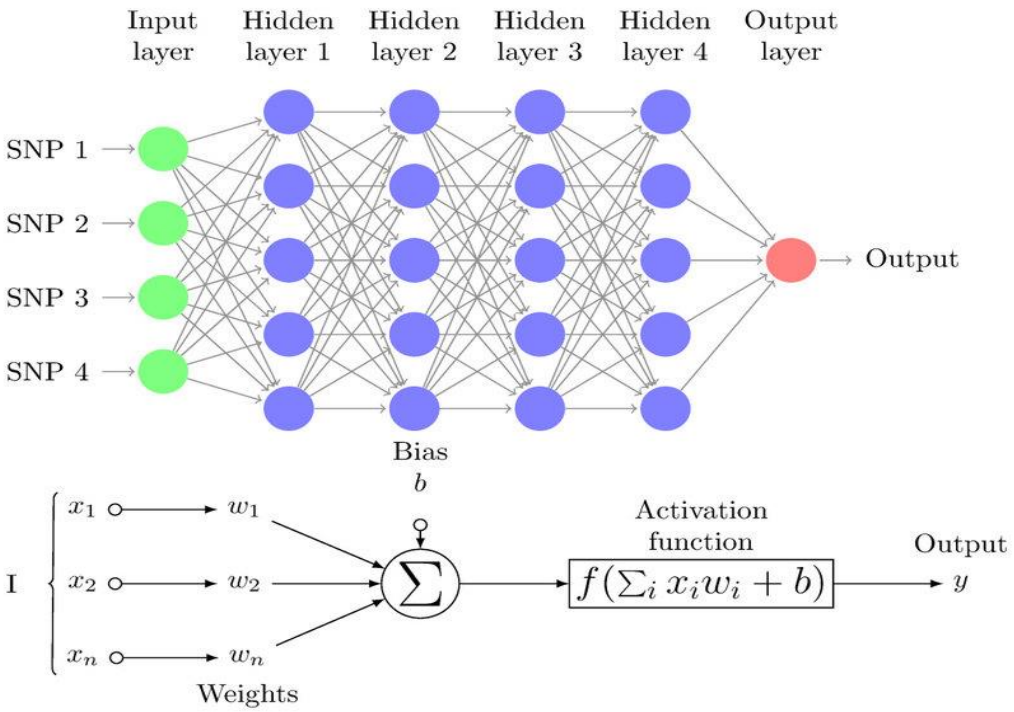

Figure 7. Multi-Layer Perceptron Architecture

The following is the MLP learning procedure: Data passes via the input layer before proceeding to the output layer. This process is referred to as forward propagation. Then, depending on the output, compute the error, which is the difference between the projected and actual outcome. The goal is to limit the errors to a minimum. The fault is then transmitted backward while considering each network weight in the following phase. Following the model's update, repeat the three techniques indicated above across many epochs to find optimal weights. Finally, the predicted class is obtained by passing the output through a threshold function.

Setting up the various neural networks employed in this study is a complex process. As a result, we must decide on the numerous characteristics to be

experimentally considered: What number of filters to consider? What are the dimensions of these filters? Which window size for each pooling layer? How many hidden neurons should we use for each new fully connected layer? How many epochs are necessary to train the models?

Metaheuristics algorithms. Optimization is crucial in every decision-making process, whether in business or engineering [51]. It entails utilizing mathematical functions to make the best possible choice. Based on their nature, optimization techniques can be classified into deterministic and probabilistic or stochastic. Deterministic algorithms always provide the same output for the same input. They are utilized for cases when precise precision and predictability are required, such as cryptography or simulations. Stochastic algorithms, on the other hand, may handle large and complex problems and find solutions faster than deterministic algorithms. Probabilistic algorithms are employed in scenarios that allow for approximations and statistical judgements like machine learning. The study restricts its investigations to probabilistic approaches.

In general, there are two types of probabilistic algorithms: heuristics and meta-heuristics, with only slight distinctions between them. Heuristic means “to find” or “to discover by trial and error” [52]. Heuristic algorithms tend to find good solutions to an optimization problem rather than the best solutions. This suggest that heuristic algorithms work most of the time, but not all the time [52]. A further improvement over the heuristic algorithms is the so-called meta-heuristics algorithms. Meta means beyond or high level, and they generally perform better than simple heuristics [34]. Moreover, all meta-heuristics algorithms use a certain tradeoff of randomization and local search. Therefore, metaheuristics algorithms intend to be suitable for global optimization.

The metaphors of the metaheuristics available today are taken mainly from disciplines like biology, physics [35], [36], computation, psychology, and chemistry [34]. Other metaphors included interior design [37], sports [38], music [39], politics [40], economic systems, and military [41].

A Brief History of Meta-heuristics is an annotated chronological list of meta-heuristic methods. The decades of the 1960s and 1970s were crucial in the evolution of metaheuristics algorithms [42]. John Holland and his students at the University of Michigan pioneered the development of Genetic Algorithms (GA) in 1975 [43].

Genetic Algorithm (GA), in essence, are search methods based on abstractions of Darwinian evolution and natural selection of biological systems, expressed in terms of mathematical operators. Since then, genetic algorithms have grown in popularity for solving a several optimization problems, including the travelling salesman problem, vehicle routing problems, image processing, data mining, and many others. The 1980s and 1990s were also an exciting moment for metaheuristic algorithms, with the introduction of Simulated Annealing (SA) in 1983, an optimization technique pioneered by S. Kirkpatrikm, Gellat, and M. P. Vecchi inspired by the annealing process of metals [44]. Things became much more thrilling as the twenty-first century progressed.

Lorenzo et al. [45] created the Harmony Search (HS) method in 2001 to tackle optimization problems in domains like water distribution, transportation modelling, and scheduling. Other algorithms, such as Cuckoo Search (CS), Bat Algorithm (BA), Firefly Algorithm (FA), and Fireworks Algorithm (FWA), Particle Swarm Optimization (PSO) have shown efficiency in providing quality results [42]. Recent meta-heuristics, such as Teaching Learning-Based Optimization (TLBO),

Biogeography-Based Optimization (BBO), and Bacterial Foraging Optimization Algorithm (BFOA), also produced impressive results [42].

Particle Swarm Optimization (PSO). Particle Swarm Optimization (PSO) is a population-based optimization technique influenced by bird flocking and fish schooling. The algorithm operates by keeping track of a population of potential solutions, known as particles, and moving them around in the search space using basic mathematical principles that consider the locations and velocities of the other particles in the population. The algorithm aims to identify the optimal solution to the optimization issue by having the particles "swarm" around the optimal solution discovered thus far. PSO is an algorithm that is easy to implement and often used to solve optimization issues across several industries like finance, engineering and machine learning.

The PSO algorithm has several parameters to set before running the algorithm. Here is a brief explanation of some of the most important parameters:

1. Swarm size: This parameter specifies the number of particles in the swarm. Increasing the swarm size can increase the exploration ability of the algorithm, but it may also increase the computation time.
2. The maximum number of iterations: This parameter specifies the maximum number of iterations the algorithm will run. Increasing the number of iterations can improve the accuracy of the solution, but it may also increase the computation time.
3. Inertia weight: This parameter controls the influence of the previous velocity on the current velocity of each particle. A high inertia weight allows particles to explore the search space more freely, while a low inertia weight helps particles converge to the optimal solution more quickly.
4. Cognitive coefficient: This parameter controls the influence of each particle's personal best solution on its movement. A high cognitive coefficient emphasizes the particle's personal experience, while a low cognitive coefficient emphasizes the swarm's collective experience.
5. Social coefficient: This parameter controls the influence of the best solution found by the swarm on each particle's movement. A high social coefficient emphasizes

the swarm's collective experience, while a low social coefficient emphasizes the particle's personal experience.

6. Velocity limits: This parameter specifies the maximum velocity that each particle can achieve. It is often used to prevent particles from moving too far away from the search space, which can lead to slower convergence or even divergence of the algorithm.

Methods

Data pre-processing. Data preprocessing aims at converting raw data into a usable, intelligible format. Raw data is frequently inconsistently formatted, contains human mistakes, or may be incomplete, as seen in Figure 8.

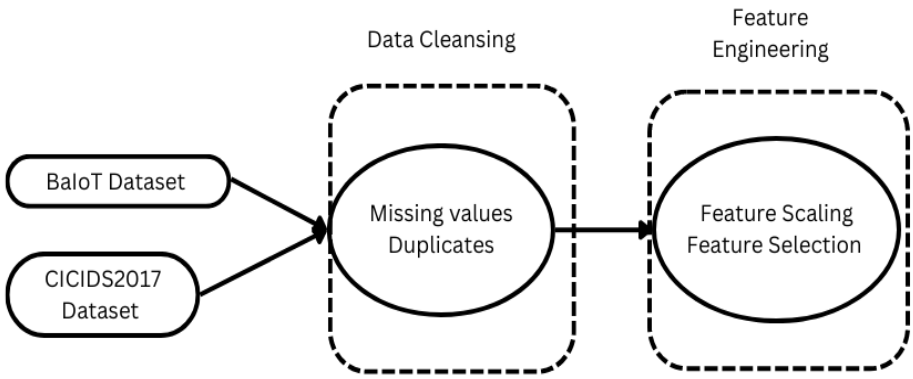

Figure 8. Data Pre-processing Pipeline

Data scientists solve such challenges through data preprocessing, which makes datasets more comprehensive and efficient for data analysis. It is an important step that can influence the performance of machine learning models since it speeds up knowledge discovery from datasets. It involves the following steps:

Data cleansing. Techniques for cleaning datasets used in the research include the following:

1. Identify and sort out missing and not a number (NaN) data: Dealing with missing and NaN data in the research consisted of handling empty rows plus rows containing null or infinite values. Once discovered, the missing rows and NaN values were either discarded or imputed with likely values.

2. Identify and remove duplicates: In the study, dealing with duplicate data required several steps to guarantee that the data was accurate and reliable. The first step is to find duplicates in the dataset by looking for identical or nearly identical entries.

Once detected, the duplicates are investigated to see if they are redundant or provide unique information critical to the research. If duplicates are judged redundant, they are deleted from the dataset to avoid bias or errors in the study. But, if the copies contain unique information, a choice on how to treat the data is made. This may involve merging the duplicate entries or choosing the most complete or accurate version.

Feature engineering. As previously stated, feature engineering refers to strategies used by data scientists to arrange data in ways that make training and interpretations more efficient. Among these techniques are the following:

1. Feature scaling or normalization: Multiple variables fluctuate on separate scales, or one changes linearly while another change exponentially. Feature scaling is a crucial preprocessing step that involved transforming numerical features into a standardized range. It is a significant step because many machine learning algorithms are sensitive to the scale of the input features. Normalization was performed in the study to scale data between 0 and 1. Scaling ensures all attribute values are on the same scale, allowing the algorithms to work more effectively.
2. Feature selection: Feature selection involves selecting the most relevant and informative features from a dataset having the greatest impact on the outcome variable. The goal of feature selection is to improve the accuracy and, efficiency of models by reducing the dimensionality of the dataset and eliminating irrelevant or redundant features. The research used PSO for the feature selection process.

In PSO-based feature selection, each particle represents a subset of attributes that can potentially be a feasible solution. The fitness of each particle is evaluated using a performance metric such as accuracy. The algorithm then updates the positions and velocities of the particles based on the fitness values and the positions of the particles with the best fitness values. The process of feature selection using PSO is iterative and stops when a stopping criterion is met, such as a maximum number of iterations or when the performance of the selected features stops improving. Thus, the particle with the highest score is the one to fit in the model.

Hyperparameters tuning using PSO. Through a series of iterations, the PSO algorithm can identify the optimal set of hyperparameters that maximizes the performance of the models on the given datasets. Due to limited computing resources,

the study considered few parameters for tuning namely batch size, the number of epochs, learning rate, hidden-layer size, and patience.

1. The batch size determines how many samples are processed before the weights are updated during training. The small batch size can result in faster convergence but may also lead to a noisy gradient, while a large batch size can provide a smoother gradient but may require more memory.
2. The number of epochs determines the number of times the model iterates over the entire dataset during training. Too few epochs can lead to underfitting, while too many epochs can lead to overfitting.
3. The learning rate controls the step size at each iteration during optimization. If the learning rate is too high, the model performance can result in a large error rate and poor model performance. On the other hand, if the learning rate is too low, the model may converge too slowly, requiring more iterations and time to train. This can result in a longer training time and higher computational cost.
4. Hidden-layer size determines the number of neurons in each hidden layer of a neural network. A larger hidden-layer size can provide the model with more capacity to capture complex patterns but can also increase the risk of overfitting.
5. Patience determines the number of epochs to wait before stopping training if the model's performance does not improve. Setting a suitable patience value can help prevent the model from overfitting or stopping training too early.

Performance measurement. ***Confusion matrix.*** Performance measurement is essential in evaluating the accuracy and usefulness of the machine learning models used in the research. A common way to measure performance is by using a confusion matrix. A confusion matrix Figure 9 is a table that summarizes the performance of a classification model by showing the number of true positive, false positive, true negative, and false negative predictions made by the model. Using predicted with actual values it is possible to compute metrics such as accuracy, precision, recall, and F1-score. The confusion matrix gives a comprehensive view of the model's performance and identifies areas for improvement.

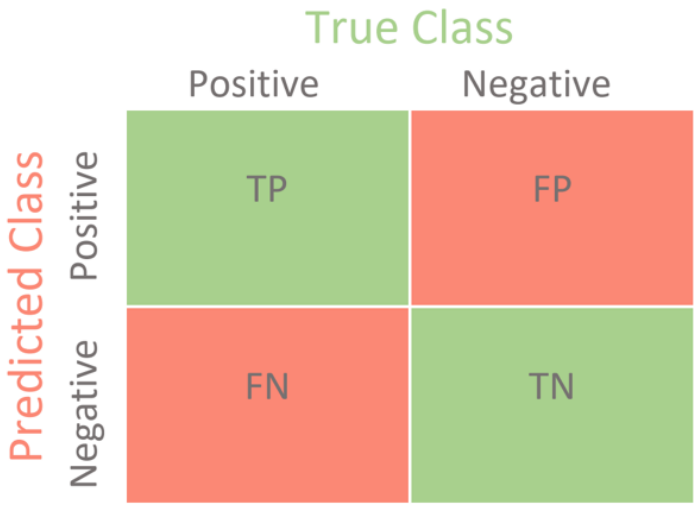

Figure 9. Confusion Matrix

Understanding some of the performance evaluation jargon linked with such a tool may help better comprehend what a confusion matrix is:

1. True Positive TP: Cases in which the model predicted a positive value and the actual result is True. As example, you anticipated that someone would be contagious, and he is.

2. True Negative TN: Represents circumstances in which the predicted negative is true. For example, suppose you anticipated that someone is not contagious and he is not.

3. False Positive FP (type 1): The model predicted True, but it was incorrect. For example, suppose you anticipated that someone is contagious, but he is not.

4. False Negative FN (type 2 error): This error type represents circumstances in which the model predicted false, but the actual output was True. For example, you expected that someone would not be contagious, yet he is.

Metrics. Some metrics that help us better understand machine learning models are as follows.

Accuracy. The model's accuracy evaluates how frequently it predicts correctly. The Accuracy formula is the sum of True Positive and True Negative divided by the total of all entries in the confusion matrix.

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

Precision. is the percentage of values that our model predicts will be Positive and are really Positive. In other words, precision indicates how much we may rely on the model when it forecasts a class as Positive.

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

Recall. The Recall metric reflects the model's prediction accuracy for the positive class; intuitively, it represents the model's ability to locate all Positive classes in the dataset. The recall is calculated by dividing the proportion of True Positive elements by the total number of positively categorized classes.

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

F1-Score. The F-measure or F1-score represents the harmonic mean of the precision and the recall. The F1-score formula may be view as a weighted average of Precision and Recall, with a top score of 1 and a worst score of 0.

$$F - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Oversampling and under-sampling. Oversampling and under-sampling are methodologies used in machine learning to balance the class distribution when dealing with unbalanced datasets. Oversampling involves replicating instances from the minority class [53] or producing synthetic observations to increase the size of the minority class and balance the class distribution. Under-sampling is the practice of deleting rows from the majority class to lower its size and balance the distribution of

the class. Both oversampling and under-sampling are employed to balance the class distribution in datasets.

Chawla et al. [46] suggested Synthetic Minority Over-sampling Technique (SMOTE) was the technique used to balance the class distribution in unbalanced datasets. SMOTE works by picking instances from the minority class, computing the k-nearest neighbours for each sample, and then producing additional synthetic samples.

Tools

Here we provide a small description of the tools used in the research.

1. **Python**: is a popular programming language, used for data science and machine learning. It offers a large and active community plus a diverse set of libraries and tools. It is also the language used in experiments.

2. **TensorFlow**: is a Google open-source software library used for designing and training neural networks. It is among the most widely used deep-learning frameworks nowadays.

3. **Keras**: is a Python-based high-level neural network API that works on top of TensorFlow. It provides an easy-to-use interface for creating and training neural networks.

4. **Pandas**: is a Python package used for data manipulation and analysis. It contains numerous data analysis tools as well as data structures for effectively storing and processing huge datasets.

5. **Scikit-learn:** is a well-known library used in machine learning for model selection and assessment. It provided tools for benchmarking models.

6. **Scikit-optimize**: provides many optimization algorithms and tools for hyperparameter tuning. It is used for PSO hyperparameter tuning.

7. **Matplotlib**: is a plotting library for Python used to create charts, graphs, and other visual representations in research. Seaborn is a Python data visualization library based on Matplotlib that provides a comprehensive interface for drawing statistical visuals.

8. **Imbalanced-learn**: is a Python library that provides a range of techniques for dealing with imbalanced datasets in machine learning. It provides tools for resampling datasets, generating synthetic data, and implementing cost-sensitive learning algorithms.

9. **Jupyter-notebook**: Open-source IDE used to write code and run all experiments.

Datasets

In this section, we present the data collected for this research. The data described below are popular in the community and were available and accessible freely.

N-BaIoT

The BaIoT dataset [47] overcomes the scarcity of public botnet datasets, particularly for IoT networks. Researchers focused on two of the most prevalent IoT botnet families to create this dataset: BASHLITE and Mirai. BaIoT provides traffic data collected from nine commercial IoT devices infected with Mirai and BASHLITE. Initially, the dataset intended to differentiate between benign and malicious traffic data using anomaly detection algorithms. However, because the malicious data can be separated into ten attacks carried out by the two botnets, the dataset may also be utilized for multi- class classification: 10 attack classes + 1 benign class. The table below summarizes the various forms of cyberattacks recorded in the BaIoT dataset, as well as their frequency.

Table 4. BaIoT Dataset Summary

Attack Type	Description	Frequency
Benign Traffic	Normal traffic	555,932
BashLite Scan	Scanning the network for vulnerable devices	255,111
BashLite Junk	Sending spam data	261,789
BashLite UDP	UDP flooding	859,850
BashLite Combo	Sending spam data and opening a connection to a specified IP address and port	515,156
BashLite TCP	TCP flooding	946,366
Mirai Scan	Scanning the network for vulnerable devices	539,979
Mirai Ack	Ack flooding	643,821
Mirai UDP	UDP flooding	1,229,999
Mirai Syn	Syn flooding	733,299
Mirai UDPplain	Variation of UDP flooding	523,304
TOTAL		7,062,606

CICIDS2017

CICIDS2017 dataset [48] is an IoT-based dataset created in 2017 at the University of New Brunswick. It was designed using actual traffic data from host running multiple operating systems (Microsoft, Ubuntu, MacOS). It contains benign traffic and frequent attacks such as FTP brute force, SSH brute force, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS. It also includes the results of a network traffic analysis performed with CICFlowMeter, which included labelled flows based on the time stamp, source and destination IPs, source and destination ports, protocols, and attacks**.**

The data collection process lasted five days, beginning at 9 a.m. on Monday, July 3, 2017, and finishing at 5 p.m. on Friday, July 7, 2017 [48]. On Monday, researchers captured regular traffic and the above-mentioned attacks were recorded on Tuesday, Wednesday, Thursday, and Friday. The dataset description is shown in Table 5.

Table 5. CICIDS2017 Dataset Description

Attack Types	Frequency
Benign Traffic	2,096,481
SQL injections	21
Cross Site Scripting (XSS)	652
Distributed denial of services (DDoS)	128,016
Bot	1,953
Port Scan	90,819
Brute-force	1,470
Infiltration	36
SSH patator	3,219
FTP patator	5,933
Heartbleed	11
Denial of Service Goldeneye	10,286
Denial of Service Hulk	172,849
Denial of Service Slowhttptest	5,228
Denial of service Slowloris	5,385
TOTAL	2,830,743

Conclusion

This chapter described in detail the methodology used to conduct this research. It also covers the design of our solution, in addition to the datasets and methods utilized to achieve to build attacks classifiers. Furthermore, this part presented a performance assessment mechanism to analyse the significance of this thesis.

CHAPTER 4

RESULTS AND DISCUSSIONS

This chapter describes the experiment, its results, and interpretation. It is organized as follows: The first section presents the system setup utilized for the experimentations, while the second section discusses the data pre-processing step and its outcomes. The third section of this chapter goes through the configuration settings of the five neural networks that were employed. Section four discusses feature selection using PSO, including the technique and parameters utilized. Section five and six comprise respectively the Hyperparameter Tuning process and the assessment against previous research.

System Setting

A typical computer would not have been enough for the experiment of this work due to the large number of computations required and the volume of data utilised. Model training and hyperparameter tuning are computationally demanding processes that need processing power and memory. The tests were performed on a computer equipped with an Intel Xeon E5-260 v4 @ 2.10GHz 32 cores CPU, 64 GB of RAM, 2.0 TB of disk space, and Ubuntu 20.04 as the operating system. The Deep learning structures were created with Python, TensorFlow-GPU package, and the Keras neural network framework discussed previously in the tool's subsection. The imbalanced-learn tool [52], an open-source Python program, was used to balance the dataset for better performance.

Data Pre-Processing

The assembling of the BaIoT dataset was a tedious task. The dataset includes traffic from 9 different devices, half of the attacks were missing from some devices. Because each assault is divided into folders and benign traffic is a single file that

applies to all devices, I utilized Pandas to combine the attack and benign data files. After assembling the dataset, the next step consisted of to inspect the data. The BaIoT dataset contained many duplicates and after the cleaning phase contains three million rows, many. The inspection also revealed that the dataset was not balanced with some classes outnumbering others.

The imbalanced-learn technique helped equalize the data to have an even proportion of data between each class. A temporary column was added on the dataset for binary classification. Additionally, any categorical values were transformed into numeric values before training. The pre-processed dataset consisted of 12 target variables (10 attack classes, one benign class, and one class for binary classification) and 115 features. Finally, the data is split into training and test sets using an 80-20 split ratio. 80% of the data is for training, while the remaining 20% is for testing.

Pre-processing the CICIDS2017 dataset consisted of first assembling it. The dataset contains network traffic recorded over five days. It provides 2.8 million rows with 78 numerical features and 15 output classes. Among 15 classes, there is one benign class and 14 types of attack instances. The data pre-processing step consisted of cleaning, label grouping, and normalization of the data. Columns with 0 were removed and row having infinity or NaN values were set to -1.

At this stage also, is done undersampling and oversampling. The CICIDS2017 is unbalanced, meaning one traffic type outweighs the others. This will cause the model to be biased towards a type of traffic. A temporary column is also added for binary classification. Then pre-processed data was also split into a training set, and a test set according to the 80:20 ratio. Figures 10 and 11 below give a snapshot of the datasets before and after balancing classes for binary classification.

In summary, the pre-processing of the data for this experiment consisted of first cleaning those data by removing duplicates, NaN and zero values. Then data undersampling and oversampling finally comes standardization and feature encoding.

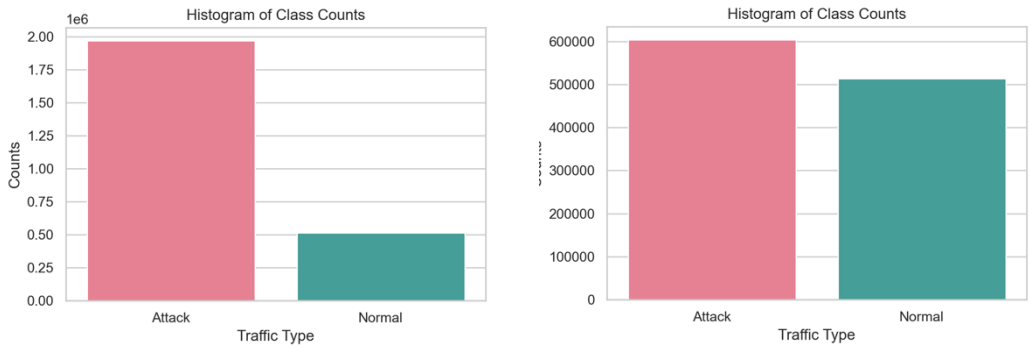

Figure 10. BaIoT Dataset Before and After Undersampling for Binary Classification

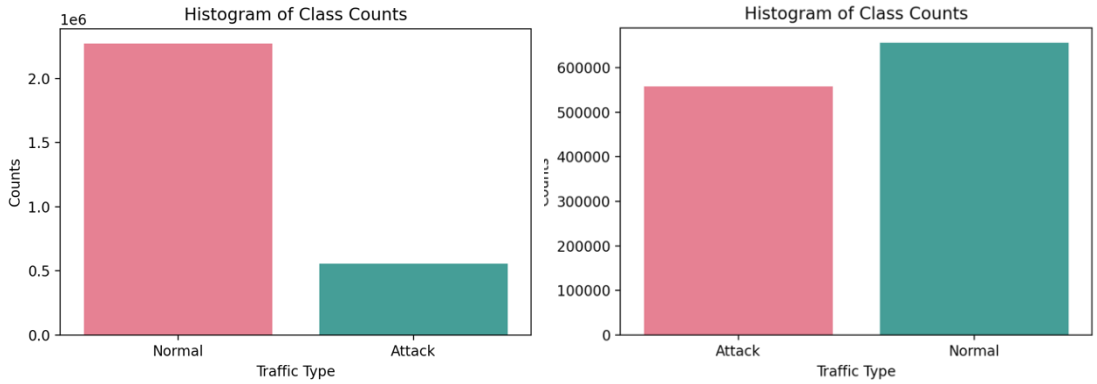

Figure 11. CICIDS Before and After Undersampling for Binary Classification

Training

The training step involved implementing several classifiers and training them on pre-processed datasets. As a result, this section discusses the setups of the various algorithms used to build classifiers. Additionally, the training was divided into two sections, the first of which consisted of experimenting with the approach provided in

this research on the BaIoT dataset and the second of which consisted of validating it on the CICIDS2017 dataset (and comparing it to previous works).

It is important to keep in mind that an assessment is performed during the training phase after generating the basic models, then after the feature selection phase, and lastly after the Hyperparameters tuning. This is to confirm that feature selection and hyperparameter optimization or tuning do really improve classifier performance. Setting up the parameters of the different classifiers was an important task to get a good model and the results are summarized in **Error! Reference source not found.**.

Table 6. Hyperparameters Used for Neural Networks

Algorithm	Parameter	Value(s)
CNN	Hidden-layer size	64
	Pooling layer	3
	Activation	[Selu, Softmax]
	Dropout	0.2
	Epochs	20
	Learning rate	0.01
	Batch_size	64
	Dense Layer	[64,100,2]
RNN	Hidden-layer size	20
	Activation	[Selu, Selu]
	Epochs	20
	Batch_size	64
	Learning rate	0.01
MLP	Hidden-layer size	5
	Max-iteration	50
	Activation	Selu
FFNN	Hidden-layer size	20
	Activation	[Selu, Sigmoid]
	Epochs	30
	Batch_size	16
	Learning rate	0.001
LSTM	Hidden-layer size	20
	Batch_size	16
	Dropout	0.5
	Patience	2
	Activation	Selu

The hyperparameters for each model are selected empirically. For all the classifiers, I set Scaled Exponential Linear Unit (SelU) as activation function, combined in some cases with Sigmoid. Moreover, early stopping and Dropout are used during the training phase to prevent overfitting. After setting up all the models comes the training phase.

As previously stated, accuracy, precision, F1-score, recall and runtime are the metrics considered to evaluate each model. Results of experiments are consigned in Tables 7, 8 and 9 for binary classification, multiclass classification and the accuracy scores for binary and multiclass classification, respectively. As shown in the results, three out the five models outperformed in binary classification CNN, MLP and FFNN with an accuracy score of 1.00.

Also, CNN, MLP, and FFNN appear very promising for multiclass classification. They scored 0.96, 0.96, and 0.94 for accuracy, respectively. LSTM and RNN are struggling with both binary and multiclass classification with respective accuracy scores of 0.5, 0.53 and 0.44, 0.46. It might mean that the model is unsuitable for the case or that the hyperparameters need finetuning. Moreover, MLP is the most efficient in terms of computing resources consumed both during training and predicting time with a total runtime of 16 minutes 59 seconds. Moreover, **Error! Reference source not found.** also confirmed that the false alarm rate for LSTM and RNN is very high as compared to the others.

Table 7. Metrics for Binary Classification

Approach	Class	Precision	Recall	F1-score	Training time	Predict time	Total Runtime
MLP	Normal	1.00	1.00	1.00	16 min 29s	30s	16 min 59s
	Attack	1.00	1.00	1.00			
FFNN	Normal	1.00	1.00	1.00	41 min 23s	32s	43 min 55s
	Attack	1.00	1.00	1.00			
CNN	Normal	0.99	1.00	0.98	59 min 01s	1 min 47s	60 min 48s
	Attack	1.00	0.98	0.99			
LSTM	Normal	0.54	1.00	0.70	3 hr 17min	18 min 02s	3 hr 35min 02s
	Attack	0.00	0.00	0.00			
RNN	Normal	0.54	0.99	0.70	2 hr 37min	1 min 28s	2 hr 38 min 28s
	Attack	0.00	0.00	0.00			

Table 8. Comparison of Metrics for Multiclass Classification

Approach	Class	Precision	Recall	F1-score	Training time	Predict time	Total Runtime
MLP	Benign	0.86	0.82	0.84			
	Gafgyt_combo	1.00	0.94	0.97			
	Gafgyt_junk	0.00	0.00	0.00			
	Gafgyt_scan	1.00	1.00	1.00	47 min 01s	2 min 50s	49 min 50s
	Gafgyt_tcp	0.97	1.00	0.99			
	Gafgyt_udp	1.00	1.00	1.00			
	Mirai_ack	1.00	0.98	0.99			
	Mirai_scan	0.68	0.74	0.71			
	Mirai_syn	1.00	1.00	1.00			
	Mirai_udp	1.00	1.00	1.00			
	Mirai_udpplain	0.53	1.00	0.69			
FFNN	Benign	0.96	0.63	0.76			
	Gafgyt_combo	0.76	0.98	0.86			
	Gafgyt_junk	0.47	1.00	0.64	55 min 13s	11 min 10s	66 min 23s
	Gafgyt_scan	1.00	0.99	1.00			
	Gafgyt_tcp	0.99	0.86	0.92			
	Gafgyt_udp	0.99	1.00	1.00			
	Mirai_ack	0.97	0.98	0.97			
	Mirai_scan	0.57	0.95	0.71			
	Mirai_syn	0.98	0.99	0.98			
	Mirai_udp	1.00	1.00	1.00			
	Mirai_udpplain	0.00	0.00	0.00			
CNN	Benign	0.98	0.92	0.95			
	Gafgyt_combo	1.00	1.00	1.00			
	Gafgyt_junk	0.47	1.00	0.64	3 hrs 40 min 12s	18 min 44s	3 hrs 60 min 45s
	Gafgyt_scan	0.99	1.00	1.00			
	Gafgyt_tcp	0.97	1.00	0.98			
	Gafgyt_udp	1.00	1.00	1.00			
	Mirai_ack	1.00	0.98	0.99			
	Mirai_scan	0.92	0.97	0.94			
	Mirai_syn	1.00	0.93	0.96			
	Mirai_udp	1.00	1.00	1.00			
	mirai_udpplain	0.02	0.00	0.00			
LSTM	Benign	0.00	0.00	0.00			
	Gafgyt_combo	0.00	0.00	0.00			
	Gafgyt_junk	0.00	0.00	0.00	7 hr 17min	35 min	7 hr 42 min 02s
	Gafgyt_scan	0.00	0.00	0.00			
	Gafgyt_tcp	0.00	0.00	0.00			
	Gafgyt_udp	0.00	0.00	0.00			
	Mirai_ack	0.00	0.00	0.00			
	Mirai_scan	0.00	0.00	0.00			
	Mirai_syn	0.00	0.00	0.00			
	Mirai_udp	0.54	0.70	0.63			
	Mirai_udpplain	0.00	0.00	0.00			
RNN	Benign	0.00	0.00	0.00			
	Gafgyt_combo	0.00	0.00	0.00			
	Gafgyt_junk	0.00	0.00	0.00	4 hr 1min	10 min 59s	4 hr 12 min 05s
	Gafgyt_scan	0.00	0.00	0.00			
	Gafgyt_tcp	0.00	0.00	0.00			
	Gafgyt_udp	0.00	0.00	0.00			
	Mirai_ack	0.00	0.00	0.00			
	Mirai_scan	0.00	0.00	0.00			
	Mirai_syn	0.00	0.00	0.00			
	Mirai_udp	0.46	1.00	0.63			
	Mirai_udpplain	0.00	0.00	0.00			

Table 9. Comparison of Accuracy Score for Binary and Multiclass Classification

Approach	Accuracy		False Positive rate	
	Binary	*Multiclass*	*Binary*	*Multiclass*
MLP	1.00	0.96	0.001	0.02
FFNN	1.00	0.94	0.002	0.05
CNN	1.00	0.96	0.001	0.02
LSTM	0.54	0.43	8.7	12.01
RNN	0.54	0.46	0.8	0.25

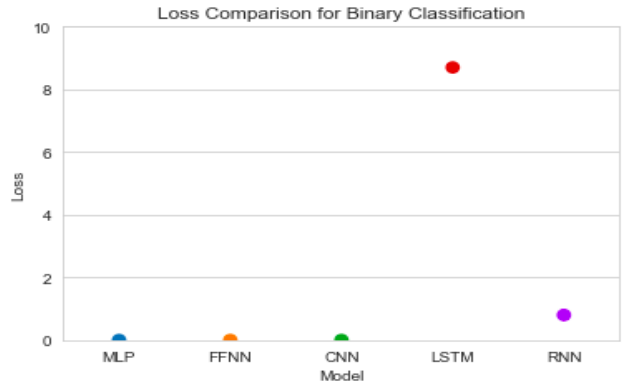

Figure 12. Loss Comparison for Binary Classification

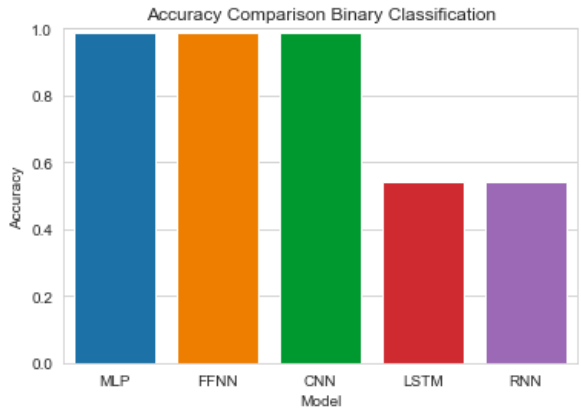

Figure 13. Graphic of Accuracy for Binary Classification

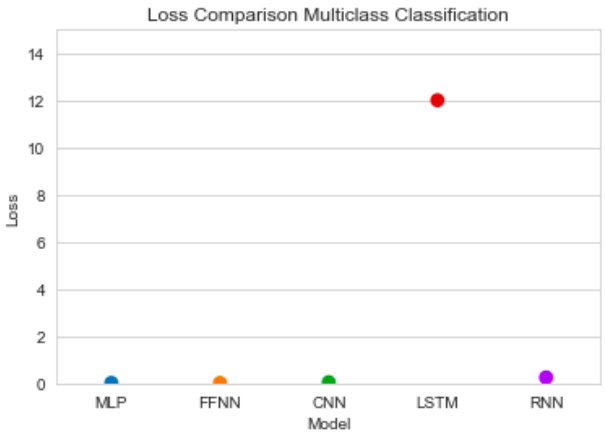

Figure 14. Loss Comparison for Multiclass Classification

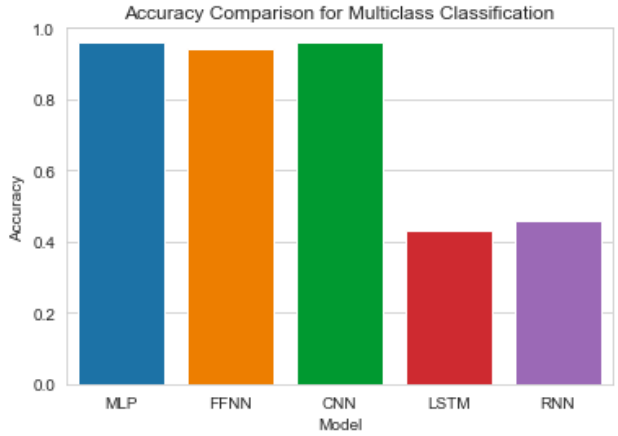

Figure 15. Graphic of Accuracy for Multiclass Classification

Feature Selection Using PSO

Feature selection is the process of assessing a dataset and keeping the attributes or inputs that ensure high accuracy. This section describes the PSO feature selection results used in the study and the benefits it has on models' performance.

Following pre-processing, the data is divided into train and test sets and fed into the PSO algorithm. The BaIoT dataset initially contains 114 attributes and over seven million rows. After the pre-processing phase, it has three million rows and 114

features. The CICIDS2017 dataset, at start contained 71 features and over two million rows. After the pre-processing phase the number of rows was around 1.7 million. The feature selection process began by assessing the significance of each attribute and its relationship to the target variable using XGboost. Figures 16 and Figure 17show the significance scores for each feature in the dataset. It defines how closely the attribute is connected to the desired output. The graphs demonstrate that not all features are important to such a problem.

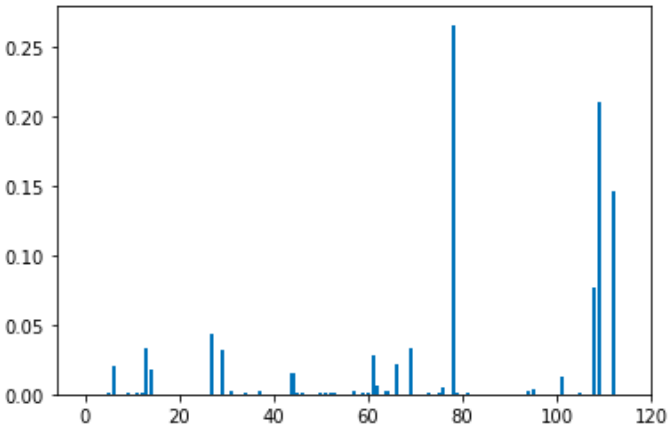

Figure 16. BaIoT Dataset Feature Importance using XGboost

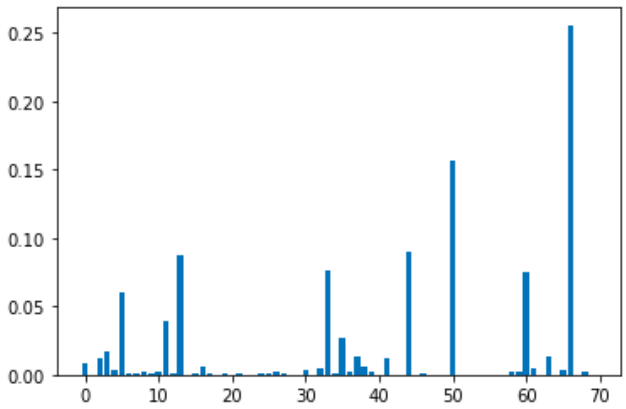

Figure 17. CICIDS 2017 Feature Importance using XGboost

Following the assessment phase, the feature selection and extraction stage using PSO begins. The PSO algorithm is given in the appendix, and the parameters utilized to implement it are summarized in the table below. A total of 58 of the 114 features in the BaIoT dataset were considered as the most important after numerous calculations. Similarly, the number of features in the CICIDS2017 has decreased to 17. Table 10 shows the summary of the PSO parameters.

Table 10. Summary of PSO Parameters

Parameter	Value
Cognitive factor	2
Social factor	2
Inertia weight	0.5
Population size	20
Number of iterations	30

Following feature selection, the models were trained again to detect any improvements in performance. The results reveal that training and prediction time do actually decrease while accuracy somewhat increases. As a consequence, all of the training and prediction times are significantly reduced. Total runtime decreased by half on CNN, MLP, FFNN, and LSTM, for example. Also, there is a gain in accuracy and a reduction in loss. CNN remains the best with a 0.98 accuracy score, while MLP and FFNN remain at 0.96 and 0.94 with improved precision. Interestingly enough runtime dropped considerably for some classifiers after the feature extraction phase.

Table 11. Metrics for Binary Classification After PSO Feature Selection

Approach	Class	Precision	Recall	F1-score	Training time	Predict time	Total Runtime
MLP	Normal	1.00	1.00	1.00	7 min 46s	15s	8 min 01s
	Attack	1.00	1.00	1.00			
FFNN	Normal	1.00	1.00	1.00	39 min 23s	32s	39 min 55s
	Attack	1.00	1.00	1.00			
CNN	Normal	1.00	1.00	1.00	50 min 01s	1 min 38s	51 min 39s
	Attack	1.00	1.00	0.99			
LSTM	Normal	0.56	1.00	0.78	3 hr 03min	16 min 22s	3 hr 19 min 22s
	Attack	0.02	0.00	0.00			
RNN	Normal	0.55	1.00	0.78	2 hr 37min	1 min 28s	2 hr 36 min 28s
	Attack	0.00	0.00	0.00			

Table 12. Metrics for Multiclass Classification After PSO Feature Selection

Approach	Class	Precision	Recall	F1-score	Training time	Predict time	Total Runtime
MLP	Benign	0.88	0.88	0.88			
	Gafgyt_combo	1.00	0.94	0.97			
	Gafgyt_junk	0.00	0.00	0.00	26 min 40s	1 min 41s	28 min 21s
	Gafgyt_scan	1.00	1.00	1.00			
	Gafgyt_tcp	0.97	1.00	0.99			
	Gafgyt_udp	1.00	1.00	1.00			
	Mirai_ack	1.00	0.98	0.99			
	Mirai_scan	0.70	0.85	0.80			
	Mirai_syn	1.00	1.00	1.00			
	Mirai_udp	1.00	1.00	1.00			
	Mirai_udpplain	0.64	1.00	0.69			
FFNN	Benign	0.97	0.70	0.79			
	Gafgyt_combo	0.77	0.98	0.86			
	Gafgyt_junk	0.63	1.00	0.64	49 min 05s	10min 06s	59 min 01s
	Gafgyt_scan	1.00	0.99	1.00			
	Gafgyt_tcp	0.99	0.86	0.92			
	Gafgyt_udp	0.99	1.00	1.00			
	Mirai_ack	0.97	0.98	0.97			
	Mirai_scan	0.60	0.95	0.73			
	Mirai_syn	0.98	0.99	0.98			
	Mirai_udp	1.00	1.00	1.00			
	Mirai_udpplain	0.00	0.00	0.00			
CNN	Benign	0.98	0.94	0.96			
	Gafgyt_combo	1.00	1.00	1.00			
	Gafgyt_junk	0.50	1.00	0.66	3 hrs 32 min 26s	16 min 19s	3 hrs 48 min 45s
	Gafgyt_scan	0.99	1.00	1.00			
	Gafgyt_tcp	0.97	1.00	0.98			
	Gafgyt_udp	1.00	1.00	1.00			
	Mirai_ack	1.00	0.98	0.99			
	Mirai_scan	0.98	0.97	0.94			
	Mirai_syn	1.00	0.93	0.96			
	Mirai_udp	1.00	1.00	1.00			
	mirai_udpplain	0.02	0.00	0.00			
LSTM	Benign	0.32	0.02	0.21			
	Gafgyt_combo	0.00	0.00	0.00			
	Gafgyt_junk	0.00	0.00	0.00	7 hr 07 min	32 min 02s	7 hr 39 min 02s
	Gafgyt_scan	0.00	0.00	0.00			
	Gafgyt_tcp	0.00	0.00	0.00			
	Gafgyt_udp	0.00	0.00	0.00			
	Mirai_ack	0.00	0.00	0.00			
	Mirai_scan	0.00	0.00	0.00			
	Mirai_syn	0.00	0.00	0.00			
	Mirai_udp	0.54	0.70	0.63			
	Mirai_udpplain	0.00	0.00	0.00			
RNN	Benign	0.28	0.00	0.33			
	Gafgyt_combo	0.00	0.00	0.00			
	Gafgyt_junk	0.00	0.00	0.00	3 hr 28 min	9 min 59s	3 hr 38 min 28s
	Gafgyt_scan	0.00	0.00	0.00			
	Gafgyt_tcp	0.00	0.00	0.00			
	Gafgyt_udp	0.00	0.00	0.00			
	Mirai_ack	0.00	0.00	0.00			
	Mirai_scan	0.00	0.00	0.00			
	Mirai_syn	0.00	0.00	0.00			
	Mirai_udp	0.48	1.00	0.69			
	Mirai_udpplain	0.00	0.00	0.00			

Table 13. Accuracy for Binary and Multiclass Classification after PSO Feature Selection

Approach	Accuracy		Loss	
	Binary	*Multiclass*	*Binary*	*Multiclass*
MLP	1.00	0.96	0.001	0.02
FFNN	1.00	0.94	0.001	0.04
CNN	1.00	0.98	0.001	0.019
LSTM	0.54	0.43	8.7	12.01
RNN	0.54	0.46	0.07	0.228

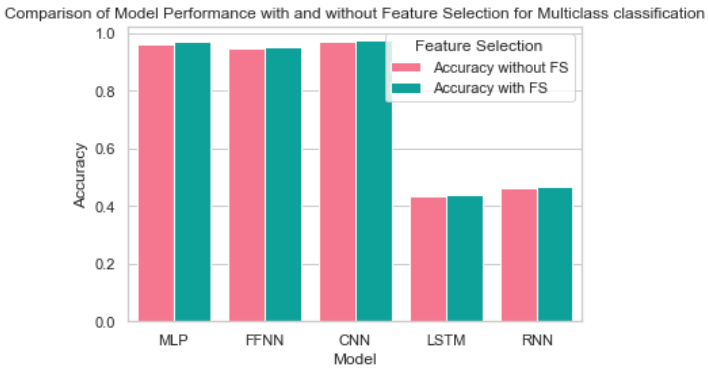

Figure 18. Accuracy Before and After PSO Features Selection

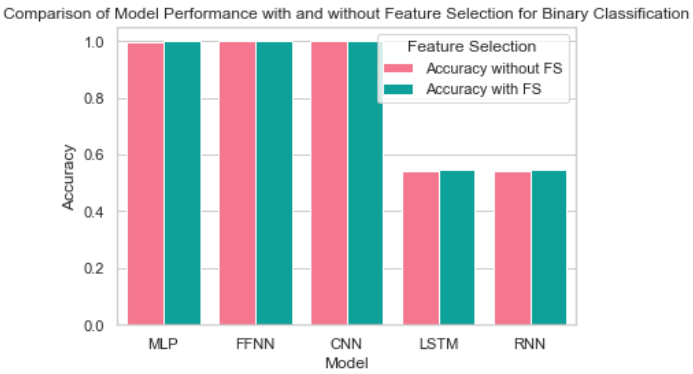

Figure 19. Accuracy Before and After PSO Feature Selection for Multiclass

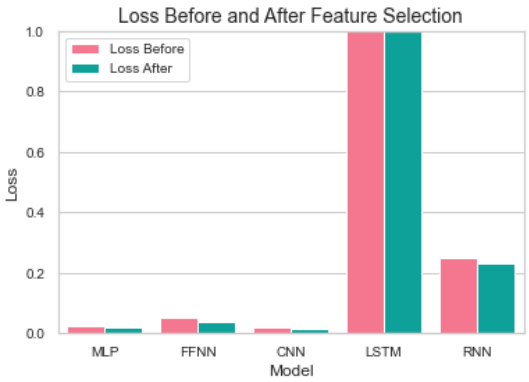

Figure 20. Losses Before and After PSO Feature Selection for Binary Classification

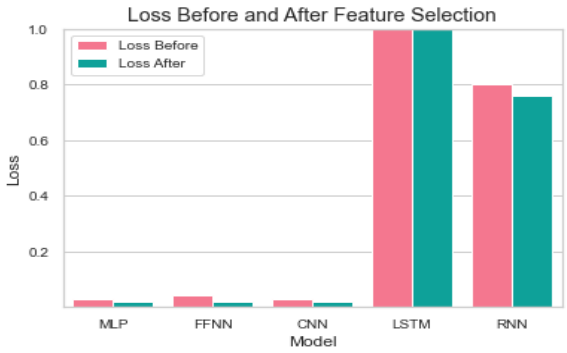

Figure 21. Losses Before and After PSO Feature Selection for Multiclass Classification

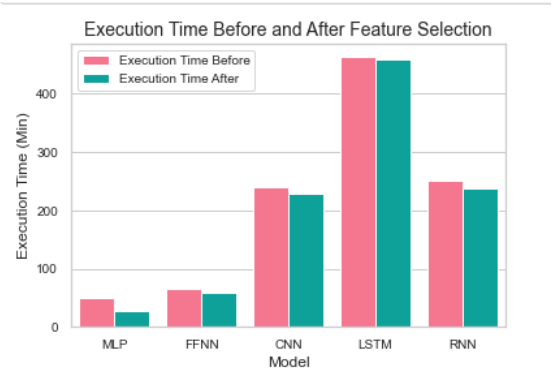

Figure 22. Runtime Before and After PSO Feature Selection for Multiclass Classification

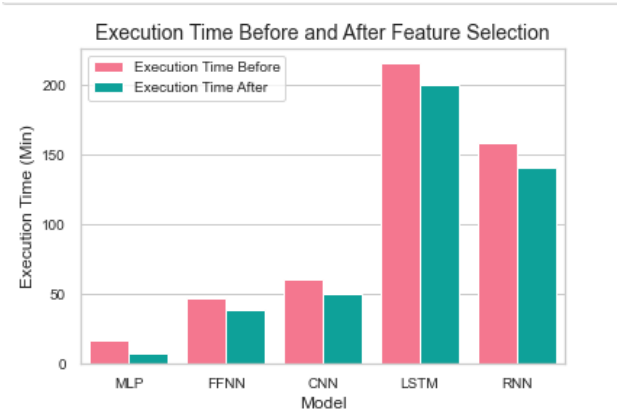

Figure 23. Runtime Before and After PSO Feature Selection for Binary Classification

Hyperparameter Tuning using PSO

The process of searching for the optimal parameter to train the models is known as hyperparameter tuning. PSO was utilized in this work to do hyperparameter optimization. It was implemented using the Scikit-Optimize Python module, which is described in the methods section. It is a time-consuming procedure that is dependent on the number of parameters to be evaluated as well as the volume of data used to train the model. This procedure also depends to the algorithm to optimize. For example, hyperparameter optimization for the CNN took longer than MLP. Due primarily to restricted resources, the study only investigated a subset of the hyperparameters discussed in the preceding chapter.

In practice, I established upper and lower bounds for each numerical parameter, and the PSO algorithm attempts to minimize error while training models with those hyperparameters. The goal is to keep the set of Hyperparameters that provide the highest accuracy. However, optimizing the model using the training set could yield overfitting. Therefore, to prevent such behaviour, I split the initial dataset into 60% for training, 20% for hyperparameter tuning and 20% for testing.

Table 14 summarizes the hyperparameters retained for each model and their respective values. Results showed that models such as MLP, CNN and FFNN preferred SelU as an activation function for this problem. The batch size for FFNN increased from 18 to 18 as well as the learning rate and hidden layer size. For CNN, the number of epochs changed to 34, and the learning rate became 0.03.

Table 14. Summary of PSO Hyperparameters Tuning

Algorithm	Parameter	Value(s)
CNN	Hidden-layer size	64
	Pooling layer	3
	Activation	[Selu, Softmax]
	Dropout	0.2
	Epochs	34
	Learning rate	0.03
	Batch_size	65
	Dense Layer	[64,100,2]
RNN	Hidden-layer size	20
	Activation	[Selu, Selu]
	Epochs	20
	Batch_size	64
	Learning rate	0.01
MLP	Hidden-layer size	7
	Max-iteration	12
	Activation	Selu
FFNN	Hidden-layer size	25
	Activation	[Selu, Sigmoid]
	Epochs	30
	Batch_size	18
	Learning rate	0.005
LSTM	Hidden-layer size	17
	Batch_size	33
	Dropout	0.5
	Patience	3
	Activation	Selu

After tuning the hyperparameters of each classifier, models are trained and evaluated again to appreciate any gain. As a result, there is an improvement concerning accuracy and runtime. Table 15 below proves that tuning hyperparameters improved models' accuracy, precision, recall and f-score. MLP, FFNN and CNN are

still performing incredibly well for binary classification having all metrics close to 1.00. There is also an improvement concerning multiclass categorization as seen in **Error! Reference source not found.**. Figures 23 and 24 show a comparison of initial models to models enhanced by PSO feature selection and PSO feature selection along with hyperparameter optimization, respectively. Figures 25 and 26 show the gain in terms of loss during the categorization. While Figures 27 and 28 indicate that there is an improvement in runtime execution as compared to the initial models. As can be seen, accuracy, time, and loss improved sequentially following feature selection and hyperparameter adjustment.

Table 15. Metrics for Binary Classification After PSO Feature Selection and Hyperparameters Optimization

Approach	Class	Precision	Recall	F1-score	Training time	Predict time	Total Runtime
MLP	Normal	1.00	1.00	1.00	8 min s	30s	8 min 01s
	Attack	1.00	1.00	1.00			
FFNN	Normal	1.00	1.00	1.00	39 min 23s	32s	39 min 55s
	Attack	1.00	1.00	1.00			
CNN	Normal	0.99	1.00	0.99	50 min 01s	1 min 38s	51 min 39s
	Attack	1.00	0.99	0.99			
LSTM	Normal	0.56	1.00	0.75	3 hr 03min	16 min 22s	3 hr 19 min 22s
	Attack	0.02	0.00	0.00			
RNN	Normal	0.55	0.99	0.76	2 hr 37min	1 min 28s	2 hr 36 min 28s
	Attack	0.00	0.00	0.00			

Table 16. Metrics for Multiclass Classification after PSO Feature Selection and Hyperparameters Optimization

Approach	Class	Precision	Recall	F1-score	Training time	Predict time	Total Runtime
MLP	Benign	0.88	0.88	0.88			
	Gafgyt_combo	1.00	0.94	0.97			
	Gafgyt_junk	0.00	0.00	0.00	26 min 40s	1 min 41s	28 min 21s
	Gafgyt_scan	1.00	1.00	1.00			
	Gafgyt_tcp	0.97	1.00	0.99			
	Gafgyt_udp	1.00	1.00	1.00			
	Mirai_ack	1.00	0.98	0.99			
	Mirai_scan	0.70	0.85	0.80			
	Mirai_syn	1.00	1.00	1.00			
	Mirai_udp	1.00	1.00	1.00			
	Mirai_udpplain	0.64	1.00	0.69			
FFNN	Benign	0.97	0.70	0.79			
	Gafgyt_combo	0.77	0.98	0.86			
	Gafgyt_junk	0.63	1.00	0.64	49 min 05s	10min 06s	59 min 01s
	Gafgyt_scan	1.00	0.99	1.00			
	Gafgyt_tcp	0.99	0.86	0.92			
	Gafgyt_udp	0.99	1.00	1.00			
	Mirai_ack	0.97	0.98	0.97			
	Mirai_scan	0.60	0.95	0.73			
	Mirai_syn	0.98	0.99	0.98			
	Mirai_udp	1.00	1.00	1.00			
	Mirai_udpplain	0.00	0.00	0.00			
CNN	Benign	0.98	0.94	0.96			
	Gafgyt_combo	1.00	1.00	1.00			
	Gafgyt_junk	0.50	1.00	0.66	3 hrs 32 min 26s	16 min 19s	3 hrs 48 min 45s
	Gafgyt_scan	0.99	1.00	1.00			
	Gafgyt_tcp	0.97	1.00	0.98			
	Gafgyt_udp	1.00	1.00	1.00			
	Mirai_ack	1.00	0.98	0.99			
	Mirai_scan	0.92	0.97	0.94			
	Mirai_syn	1.00	0.93	0.96			
	Mirai_udp	1.00	1.00	1.00			
	mirai_udpplain	0.02	0.00	0.00			
LSTM	Benign	0.32	0.02	0.21			
	Gafgyt_combo	0.00	0.00	0.00			
	Gafgyt_junk	0.00	0.00	0.00	7 hr 07 min	32 min 02s	7 hr 39 min 02s
	Gafgyt_scan	0.00	0.00	0.00			
	Gafgyt_tcp	0.00	0.00	0.00			
	Gafgyt_udp	0.00	0.00	0.00			
	Mirai_ack	0.00	0.00	0.00			
	Mirai_scan	0.00	0.00	0.00			
	Mirai_syn	0.00	0.00	0.00			
	Mirai_udp	0.54	0.70	0.63			
	Mirai_udpplain	0.00	0.00	0.00			
RNN	Benign	0.28	0.00	0.33			
	Gafgyt_combo	0.00	0.00	0.00			
	Gafgyt_junk	0.00	0.00	0.00	3 hr 28 min	9 min 59s	3 hr 38 min 28s
	Gafgyt_scan	0.00	0.00	0.00			
	Gafgyt_tcp	0.00	0.00	0.00			
	Gafgyt_udp	0.00	0.00	0.00			
	Mirai_ack	0.00	0.00	0.00			
	Mirai_scan	0.00	0.00	0.00			
	Mirai_syn	0.00	0.00	0.00			
	Mirai_udp	0.48	1.00	0.69			
	Mirai_udpplain	0.00	0.00	0.00			

Table 16 (continued)

Approach	Class	Precision	Recall	F1-score	Training time	Predict time	Total Runtime
MLP	Benign	0.86	0.82	0.84			
	Gafgyt_combo	1.00	0.94	0.97	23 min 29s	30s	23 min 59s
	Gafgyt_junk	0.00	0.00	0.00			
	Gafgyt_scan	1.00	1.00	1.00			
	Gafgyt_tcp	0.97	1.00	0.99			
	Gafgyt_udp	1.00	1.00	1.00			
	Mirai_ack	1.00	0.98	0.99			
	Mirai_scan	0.68	0.74	0.71			
	Mirai_syn	1.00	1.00	1.00			
	Mirai_udp	1.00	1.00	1.00			
	Mirai_udpplain	0.53	1.00	0.69			
FFNN	Benign	0.96	0.70	0.80			
	Gafgyt_combo	0.76	0.98	0.86			
	Gafgyt_junk	0.47	1.00	0.64	46 min 23s	32s	46 min 55s
	Gafgyt_scan	1.00	0.99	1.00			
	Gafgyt_tcp	0.99	0.86	0.92			
	Gafgyt_udp	0.99	1.00	1.00			
	Mirai_ack	0.97	0.98	0.97			
	Mirai_scan	0.57	0.95	0.71			
	Mirai_syn	0.98	0.99	0.98			
	Mirai_udp	1.00	1.00	1.00			
	Mirai_udpplain	0.00	0.00	0.00			
CNN	Benign	0.98	0.92	0.95			
	Gafgyt_combo	1.00	1.00	1.00			
	Gafgyt_junk	0.70	1.00	0.70	55 min 01s	1 min 44s	56 min 45s
	Gafgyt_scan	0.99	1.00	1.00			
	Gafgyt_tcp	0.97	1.00	0.98			
	Gafgyt_udp	1.00	1.00	1.00			
	Mirai_ack	1.00	0.98	0.99			
	Mirai_scan	0.92	0.97	0.94			
	Mirai_syn	1.00	0.93	0.96			
	Mirai_udp	1.00	1.00	1.00			
	mirai_udpplain	0.02	0.00	0.00			
LSTM	Benign	1.00	0.93	1.00			
	Gafgyt_combo	0.00	0.00	0.00			
	Gafgyt_junk	0.00	0.00	0.00	3 hr 17min	18 min 02s	3 hr 35min 02s
	Gafgyt_scan	0.00	0.00	0.00			
	Gafgyt_tcp	0.00	0.00	0.00			
	Gafgyt_udp	0.00	0.00	0.00			
	Mirai_ack	0.00	0.00	0.00			
	Mirai_scan	0.00	0.00	0.00			
	Mirai_syn	0.00	0.00	0.00			
	Mirai_udp	0.55	0.74	0.63			
	Mirai_udpplain	0.00	0.00	0.00			
RNN	Benign	0.00	0.00	0.00			
	Gafgyt_combo	0.00	0.00	0.00			
	Gafgyt_junk	0.00	0.00	0.00	2 hr 37min	1 min 28s	2 hr 38 min 28s
	Gafgyt_scan	0.00	0.00	0.00			
	Gafgyt_tcp	0.00	0.00	0.00			
	Gafgyt_udp	0.00	0.00	0.00			
	Mirai_ack	0.00	0.00	0.00			
	Mirai_scan	0.00	0.00	0.00			
	Mirai_syn	0.00	0.00	0.00			
	Mirai_udp	0.48	1.00	0.65			
	Mirai_udpplain	0.00	0.00	0.00			

Table 17. Accuracy for Binary and Multiclass Classification after PSO Feature Selection and Hyperparameters Optimization

Approach	Accuracy		False Alarm Rate	
	Binary	*Multiclass*	*Binary*	*Multiclass*
MLP	1.00	0.96	0.001	0.02
FFNN	1.00	0.94	0.001	0.04
CNN	1.00	0.98	0.001	0.019
LSTM	0.54	0.43	8.7	12.01
RNN	0.54	0.46	0.07	0.228

Figure 24. Accuracy After PSO Features selection and HPO For Binary Classification

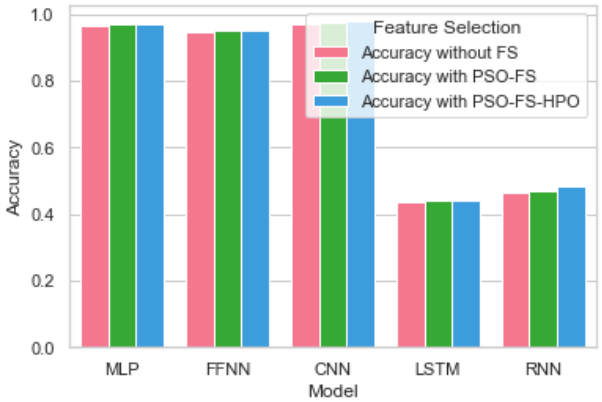

Figure 25. Accuracy After PSO Features selection and HPO for Multiclass Classification

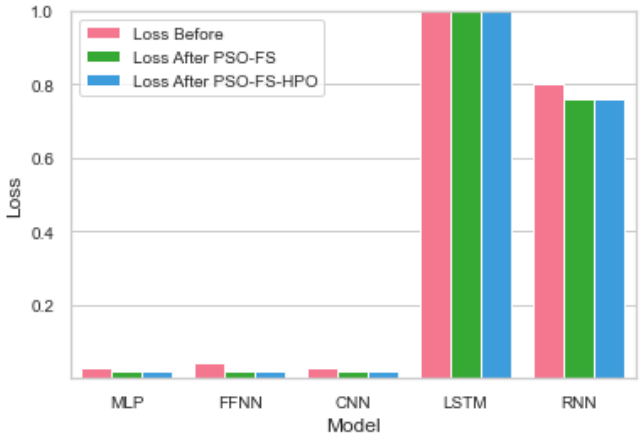

Figure 26. Loss After PSO Features selection and HPO for Multiclass Classification

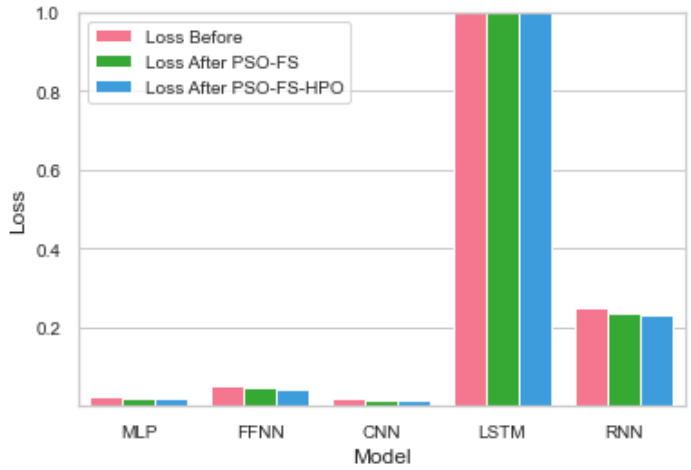

Figure 27. Loss After PSO Features selection and HPO For Binary Classification

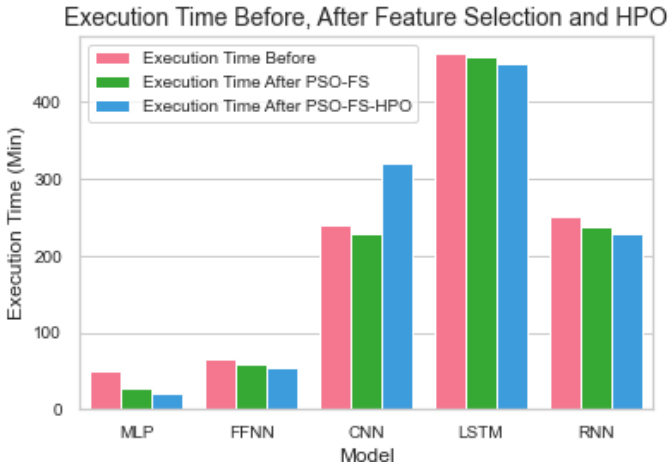

Figure 28. Runtime After PSO Features Selection and HPO for Multiclass Classification

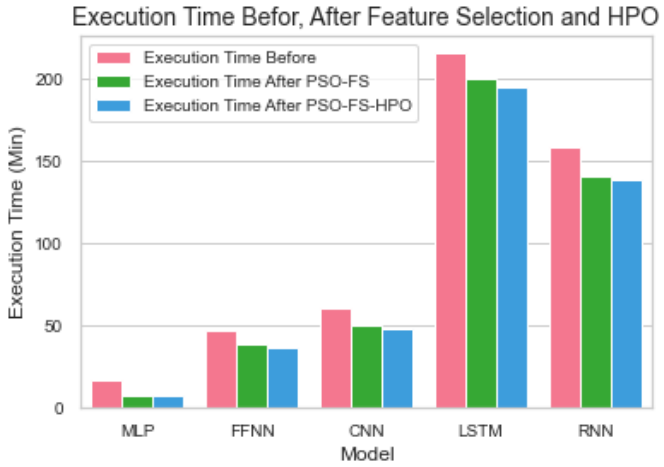

Figure 29. Runtime PSO Features Selection and HPO For Binary Classification

Evaluation

This section discusses the outcomes of the comparison between the suggested technique and past efforts. The evaluation entails testing this work using the CICIDS2017 dataset and comparing it to previous studies. Only the best neural networks were saved and utilized for training and comparison. Meanwhile, this phase

was a bit tough since choosing the best does not guarantee that it would still perform well on this dataset.

To narrow down the choices, I considered CNN and MLP. Previous experiments revealed that, while PSO-CNN has significant promise for both binary and multiclass classification, it takes longer to train and requires more computational resources. PSO-MLP, on the other hand, produced an excellent performance for binary classification and adequate performance for multiclass classification in a reasonable amount of time. As a result, I decided to use both PSO-CNN and PSO-MLP on the CICIDS2017 datasets to investigate if the tendencies observed on the BaIoT dataset will be replicated here.

An approach mentioned in the literature review section performed well in identifying IoT attacks [34]. On the CICIDS2017 dataset, the research earned 99.997% for binary and 99.91% for multiclass classification. That study's method combines CNN and Aquila Optimizer (AQU). Benchmarking entailed applying the technique described in this study to the CICIDS. It implies before CNN or MLP classification, PSO feature selection and PSO hyperparameter tuning.

As a result, Tables 18 and 19 show that PSO-CNN can accurately characterize attacks, with an accuracy score of 99.998 for binary and 99.95 for multiclass classification. As seen in the figures, the technique proposed in this study outperformed earlier work in binary and multiclass classification. When compared to other techniques, PSO-CNN had the highest accuracy. In terms of runtime, PSO-MLP still outperforms PSO-CNN. Sadly, the authors of [34] did not offer measurements for resources utilized.

Table 18. Performance for Binary Classification

Approach	Accuracy	Precision	Recall	F1-score
AQU-CNN	99.997	99.997	99.997	99.997
PSO-MLP	99.997	99.997	99.997	99.996
PSO-CNN	99.998	99.998	99.998	99.998

Table 19. Performance for Multiclass Classification

Approach	Accuracy	Precision	Recall	F1-score
AQU-CNN	99.911	99.910	99.910	99.888
PSO-MLP	98.992	98.990	98.990	98.996
PSO-CNN	99.950	99.955	99.955	99.940

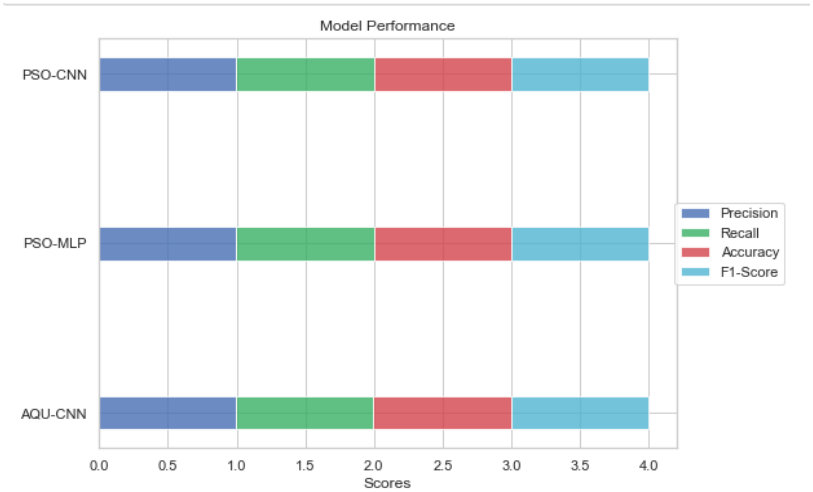

Figure 30. PSO-CNN, PSO-MLP, AQU-CNN metrics for Binary Classification

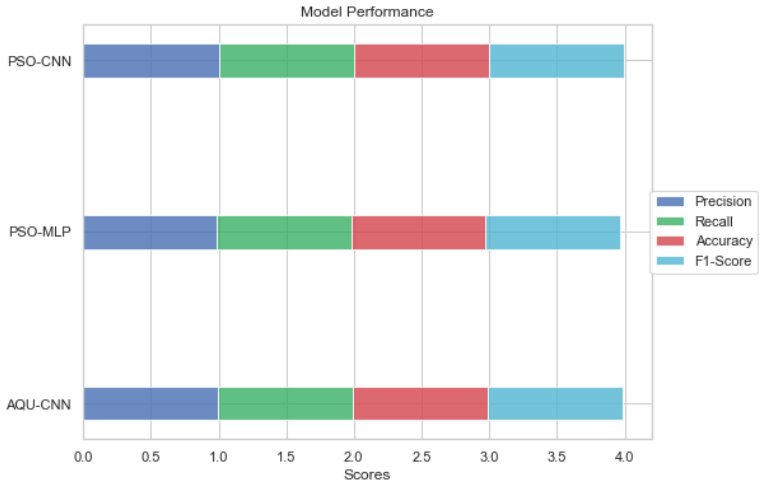

Figure 31. PSO-CNN, PSO-MLP, AQU-CNN Metrics for Multiclass Classification

CHAPTER 5

CONCLUSIONS, PERSPECTIVES AND RECOMMENDATIONS

Conclusion

This research investigated the effectiveness of a combined neural network and metaheuristic to identify IoT network assaults. Based on the results of experiments conducted on two IoT datasets, it can be concluded that combining a convolutional neural network (CNN) as attack classifier and Particle swarm optimization (PSO) for data dimensionality reduction and model hyperparameter optimization is very effective in detecting attacks in IoT networks.

PSO-CNN excelled in multiclass classification, scoring 98.96% on the BaIoT dataset and 99.95% on the CICIDS 2017, with false positive rates of 0.019 and 0.008 respectively. Results also revealed that three of the five models examined, PSO-CNN, PSO-MLP, and PSO-FFNN, produced stunning results on binary classification, with an accuracy score of 99.998% and a false alarm rate of around 0.001. Moreover, the study confirmed that selecting the most relevant features in the dataset and selecting the appropriate hyperparameters for the models improves performance.

Although more alternative ways must be examined, this work proved that removing unnecessary characteristics and tweaking model hyperparameters may significantly reduce computation time while retaining a high detection rate with minimal false alarms. This study also shows that artificial intelligence and swarm intelligence may contribute to building robust cybersecurity solutions for IT systems in general and, as well as tailored solutions for IoT networks in particular.

Contribution to the Field

The most significant contribution of this research is the development of an efficient detection method for IoT networks that combines neural networks with PSO. The approach utilized in this study distinguishes itself from others in that optimization is performed at two levels: the dataset and the model.

Perspectives

Datasets

Access to high-quality datasets for researchers remains a barrier in IoT contexts. This study dealt with datasets that contained unrepresented classes, therefore I had to employ an oversampling strategy that did not accurately represent real traffic. More quality models can be built and trained if the community has access to more accurate network traffic data. Therefore, making quality datasets public might be an intriguing future direction.

More Parameters for Tuning

This research focused on a small number of hyperparameters to tune. Considering more hyperparameters for tuning is another potential future endeavour. Deep learning models, such as the ones utilized in this study, can include anything from a few to hundred hyperparameters. These settings can have a significant influence on model performance. As a result, selecting them carefully may result in better results, particularly for IoT networks.

Reinforcement Learning

Reinforcement learning entails rewarding the system for correct predictions and penalizing it for incorrect predictions. An agent may learn the regular behaviour of IoT devices and thus identify abnormal traffic. Developing a system that can learn

benign and malicious traffic independently can be a remarkable accomplishment. The benefit of using this learning method is that the system may learn new attack patterns without prior training. This can help detect attacks like zero-day attacks. However, the quality of the traffic on which the agent is trained will heavily influence the quality of the model.

Deployment

Applying this concept in a real-world environment is also a promising option. The solution might help IoT network administrators since it can be deployed on a device like the Raspberry Pi and placed within the network. Furthermore, a dashboard application might provide alerts and recommend tailored fixes.

This study provides hints for further research on a realistic implementation of the PSO-CNN on Raspberry Pi. The only instrument required here is a Raspberry Pi 4 with sufficient capacity for the memory card. The first step will be to install the operating system and Python libraries like TensorFlow Lite on Raspberry. The PSO-CNN is copied and saved as a file on the raspberry pi. The model can then be trained on sample data. Figure 31 depicts a possible deployment strategy for the PSO-CNN model on a Raspberry Pi.

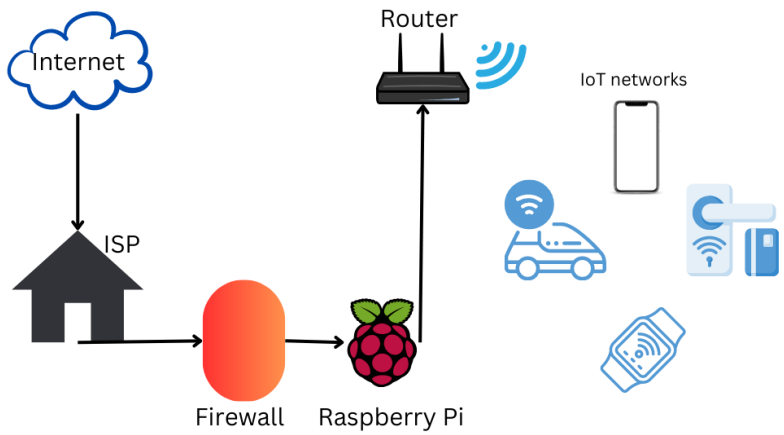

Figure 32. Deployment Strategy

It is critical to note that, depending on the complexity of the model, actions like quantizing or pruning will be needed to minimize the size of the model. After the model's performance on sample data is satisfactory, deployment in a production environment for inference (testing on unseen data) is possible. A dashboard program to gather traffic, pre-process it, and deliver it to the Raspberry Pi for analysis and prediction will complete the deployment of the solution. Real-time monitoring is also feasible, but it necessitates considering criteria like the deployment environment, system latency, and incoming traffic processing.

Recommendations

Here are some recommendations the research is giving for African countries to fully benefits of the advantages of IoT and avoid attacks.

African governments should raise awareness about the importance of IoT security among users. Also, provide education and training programs to promote best practices for securing IoT devices, networks, and data. Areas for training could consider IoT security, threat detection, incident response, and risk management for example.

African governments should also establish and enforce regulatory frameworks for IoT security in Africa, including standards, certifications, and compliance requirements.

African leaders should collaborate with industry, academia, and international institutions to develop African regulations that address the challenges of IoT security in Africa.

REFERENCES

[1] A. Bassi *et al.*, *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model*. New York, NY: Springer, 2013.

[2] Statista, "IoT connected devices worldwide 2019-2030," *Statista*, Feb. 02, 2022. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[3] "DNSPOOQ," *JSOF*, Nov. 12, 2020. https://www.jsof-tech.com/disclosures/dnspooq/

[4] JSOF, "Ripple20," *JSOF*, Jul. 07, 2020. https://www.jsof-tech.com/disclosures/ripple20/

[5] T. Seals, "IoT Attacks Skyrocket, Doubling in 6 Months," *Threat Post*, Mar. 07, 2021. https://threatpost.com/iot-attacks-doubling/169224/

[6] S. Morgan, "Cybercrime To Cost The World $10.5 Trillion Annually By 2025," *Cybercrime Magazine*, Apr. 02, 2018. https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/

[7] A. Khraisat and A. Alazab, "A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges," *Cybersecurity*, vol. 4, no. 1, p. 18, Mar. 2021.

[8] S. Hajiheidari, K. Wakil, M. Badri, and N. J. Navimipour, "Intrusion detection systems in the Internet of things: A comprehensive investigation," *Computer Networks*, vol. 160, pp. 165–191, Sep. 2019.

[9] K. A. P. da Costa, J. P. Papa, C. O. Lisboa, R. Munoz, and V. H. C. de Albuquerque, "Internet of Things: A survey on machine learning-based intrusion detection approaches," *Computer Networks*, vol. 151, pp. 147–157, Mar. 2019.

[10] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, Jul. 2019.

[11] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, Nov. 2018.

[12] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, "Chapter 10 - Metaheuristic Algorithms: A Comprehensive Review," in *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*, A. K. Sangaiah, M. Sheng, and Z. Zhang, Eds. Cambridge, MA: Academic Press, 2018, pp. 185–231. doi: 10.1016/B978-0-12-813314-9.00010-4.

[13] S. Nematzadeh, F. Kiani, M. Torkamanian-Afshar, and N. Aydin, "Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: A bioinformatics study on biomedical and biological cases," *Computational Biology and Chemistry*, vol. 97, p. 107619, Apr. 2022.

[14] F. S. Gharehchopogh, I. Maleki, and Z. A. Dizaji, "Chaotic vortex search algorithm: metaheuristic algorithm for feature selection," *Evolutionary Intelligence*, vol. 15, no. 3, pp. 1777–1808, Mar. 2021.

[15] P. Agrawal, H. F. Abutarboush, T. Ganesh, and A. W. Mohamed, “Metaheuristic Algorithms on Feature Selection: A Survey of One Decade of Research (2009-2019),” *IEEE Access*, vol. 9, pp. 26766–26791, 2021.
[16] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, “A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures,” *IEEE Access*, vol. 7, pp. 82721–82743, Jun. 2019.
[17] R. R. Krishna, A. Priyadarshini, A. V. Jha, B. Appasani, A. Srinivasulu, and N. Bizon, “State-of-the-Art Review on IoT Threats and Attacks: Taxonomy, Challenges and Solutions,” *Sustainability*, vol. 13, no. 16, p. 9463, Aug. 2021.
[18] S. Khanam, I. B. Ahmedy, M. Y. Idna Idris, M. H. Jaward, and A. Q. Bin Md Sabri, “A Survey of Security Challenges, Attacks Taxonomy and Advanced Countermeasures in the Internet of Things,” *IEEE Access*, vol. 8, pp. 219709–219743, Nov. 2020.
[19] I. Butun, P. Österberg, and H. Song, “Security of the Internet of Things: Vulnerabilities, Attacks, and Countermeasures,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 616–644, 2020, doi: 10.1109/COMST.2019.2953364.
[20] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, “A Survey of IoT-Enabled Cyberattacks: Assessing Attack Paths to Critical Infrastructures and Services,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3453–3495, Jul. 2018.
[21] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, “A deep Recurrent Neural Network based approach for Internet of Things malware threat hunting,” *Future Generation Computer Systems*, vol. 85, pp. 88–96, Aug. 2018.
[22] S. Smys, A. Basar, and H. Wang, “Hybrid Intrusion Detection System for Internet of Things (IoT),” *Journal of ISMAC*, vol. 2, pp. 190–199, Sep. 2020.
[23] A. Derhab, A. Aldweesh, A. Z. Emam, and F. A. Khan, “Intrusion Detection System for Internet of Things Based on Temporal Convolution Neural Network and Efficient Feature Engineering,” *Wireless Communications and Mobile Computing*, vol. 2020, p. e6689134, Dec. 2020.
[24] A. Fatani, A. Dahou, M. A. A. Al-qaness, S. Lu, and M. Abd Elaziz, “Advanced Feature Extraction and Selection Approach Using Deep Learning and Aquila Optimizer for IoT Intrusion Detection System,” *Sensors*, vol. 22, no. 1, p. 140, Dec. 2021.
[25] N. Bacanin, T. Bezdan, E. Tuba, I. Strumberger, and M. Tuba, “Optimizing Convolutional Neural Network Hyperparameters by Enhanced Swarm Intelligence Metaheuristics,” *Algorithms*, vol. 13, no. 3, p. 67, Mar. 2020.
[26] V. Hajisalem and S. Babaie, “A hybrid intrusion detection system based on ABC-AFS algorithm for misuse and anomaly detection,” *Computer Networks*, vol. 136, pp. 37–50, May 2018.
[27] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943.
[28] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. Mahwah, N.J: Psychology Press, 2002.
[29] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, pp. 386–408, Nov. 1958.

[30] J. Zhu *et al.*, “Electric Vehicle Charging Load Forecasting: A Comparative Study of Deep Learning Approaches,” *Energies*, vol. 12, no. 14, p. 2692, Jan. 2019.
[31] G. Van Houdt, C. Mosquera, and G. Nápoles, “A review on the long short-term memory model,” *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5929–5955, Dec. 2020.
[32] K. Verma and P. Singh, “An Insight to Soft Computing based Defect Prediction Techniques in Software,” *International Journal of Modern Education and Computer Science*, vol. 7, pp. 52–58, Sep. 2015.
[33] M. Pérez-Enciso and L. M. Zingaretti, “A Guide for Using Deep Learning for Complex Trait Genomic Prediction,” *Genes*, vol. 10, no. 7, p. 553, Jul. 2019.
[34] K. Hussain, M. N. Mohd Salleh, S. Cheng, and Y. Shi, “Metaheuristic research: a comprehensive survey,” *Artificial Intelligence Review*, vol. 52, no. 4, pp. 2191–2233, Jan. 2018.
[35] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, “GSA: A Gravitational Search Algorithm,” *Information Sciences*, vol. 179, no. 13, pp. 2232–2248, Jun. 2009.
[36] H. Abedinpourshotorban, S. Mariyam-Shamsuddin, Z. Beheshti, and D. Jawawi, “Electromagnetic field optimization: A physics-inspired metaheuristic optimization algorithm,” *Swarm and Evolutionary*, vol. 26, pp. 8–22, Feb. 2016.
[37] A. H. Gandomi, “Interior search algorithm (ISA): A novel approach for global optimization,” *ISA Transactions*, vol. 53, no. 4, pp. 1168–1183, Jul. 2014.
[38] E. Osaba, F. Diaz, and E. Onieva, “Golden ball: a novel meta-heuristic to solve combinatorial optimization problems based on soccer concepts,” *Applied Intelligence*, vol. 41, no. 1, pp. 145–166, Feb. 2014.
[39] Z. W. Geem, J. H. Kim, and G. V. Loganathan, “A New Heuristic Optimization Algorithm: Harmony Search,” *Simulation*, vol. 76, no. 2, pp. 60–68, Feb. 2001.
[40] E. Atashpaz-Gargari and C. Lucas, “Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition,” presented at the 2007 IEEE Congress on Evolutionary Computation, Singapore, 2007, pp. 4661–4667. [Online]. Available: https://doi.org/10.1109/cec.2007.4425083
[41] Tummala. S. L. V. Ayyarao *et al.*, “War Strategy Optimization Algorithm: A New Effective Metaheuristic Algorithm for Global Optimization,” *IEEE Access*, vol. 10, pp. 25073–25105, Feb. 2022, doi: https://doi.org/10.1109/access.2022.3153493.
[42] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, Perth, WA, Australia, 1995, vol. 4. [Online]. Available: https://doi.org/10.1109/icnn.1995.488968
[43] J. H. Holland, “Genetic algorithms,” *Scholarpedia*, vol. 7, no. 12, p. 1482, Dec. 2012.
[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
[45] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, “Particle swarm optimization for hyper-parameter selection in deep neural networks,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, NY, 2017, pp. 481–488. [Online]. Available: https://doi.org/10.1145/3071178.3071208
[46] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002.

[47] Y. Meidan *et al.*, "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, Aug. 2018.
[48] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, Funchal, Madeira, Portugal, 2018, pp. 108–116. [Online]. Available: https://doi.org/10.5220/0006639801080116
[49] Hecht-Nielsen, R. (1989). *Neurocomputing*. Addison-Wesley Longman Publishing Co., Inc..
[50] Schalkoff, R. J. (1997). *Artificial neural networks*. McGraw-Hill Higher Education.
[51] Chong, E. K., & Zak, S. H. (2013). *An introduction to optimization* (Vol. 75). John Wiley & Sons.
[52] Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research*, *18*(1), 559-563
[53] Li, X., Wang, L., & Sung, E. (2008, August). AdaBoost with SVM-based component classifiers. *Engineering Applications of Artificial Intelligence*, *21*(5), 785–795. https://doi.org/10.1016/j.engappai.2007.07.001