DETECTION OF BOTNET ATTACK IN IoT USING MACHINE LEARNING

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

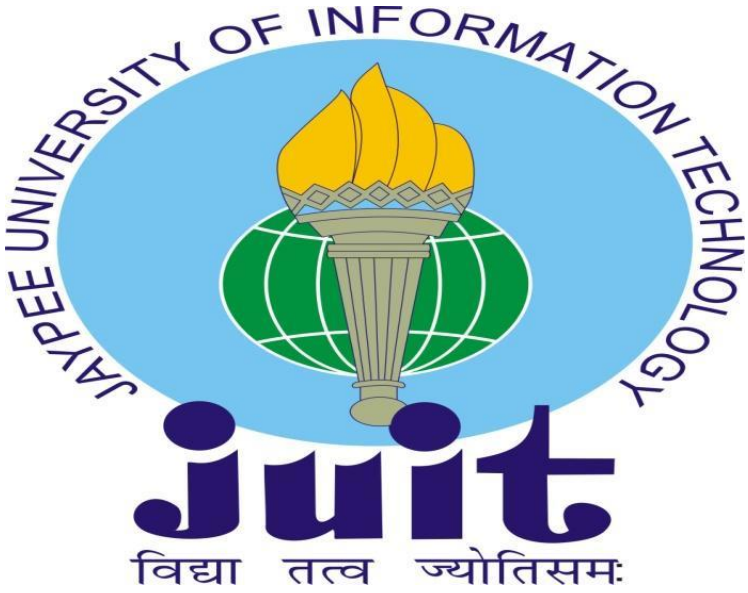
By

Vasundhara Pandey (191400) and Suryansh Mishra (191300)

Under the supervision of

Dr. Amol Vasudeva

to



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Certificate

Candidate's Declaration

I hereby declare that the work presented in this report entitled **" Detection of Botnet Attack in IoT using Machine Learning"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from July 2022 to May 2023 under the supervision of **Dr. Amol Vasudeva** Assistant Professor (Senior Grade) Computer Science and Engineering Dept.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)
Suryansh Mishra, 191300

(Student Signature)
Vasundhara Pandey, 191400

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature)
Dr. Amol Vasudeva
Assistant Professor (Senior Grade)
Computer Science and Engineering
Dated: 24 April 2023

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: …….

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: _____ __Department: _____ Enrolment No _____

Contact No. _____E-mail. _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

_____

_____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ……(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

<table>
<tr><td>Copy Received on</td><td rowspan="4">• All Preliminary Pages
• Bibliography/Images/Quotes
• 14 Words String</td><td rowspan="2"></td><td>Word Counts</td><td></td></tr>
<tr><td rowspan="2">Report Generated on</td><td>Character Counts</td><td></td></tr>
<tr><td rowspan="2"></td><td>Submission ID</td><td>Total Pages Scanned</td><td></td></tr>
<tr><td></td><td>File Size</td><td></td></tr>
</table>

Checked by
Name & Signature **Librarian**

……

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ACKNOWLEDGEMENT

First, I express my heartiest thanks and gratefulness to Almighty God for His divine blessing to make it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor **Dr. Amol Vasudeva, Assistant Professor (Senior Grade),** Department of CSE Jaypee University of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of "Machine Learning" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, and reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would also like to acknowledge my respective lab coordinators, Mr. Ravi Raina and Mr. Mohan Sharma for providing me the necessary resources that proved to be worthwhile in successfully completing my project.

I would like to express my heartiest gratitude to Dr. Amol Vasudeva, Department of CSE, for his kind help in finishing my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Suryansh Mishra 191300
Vasundhara Pandey 191400

TABLE OF CONTENT

List of Figures

List of Graphs

List of Tables

ABSTRACT

The Internet of Things (IoT) has changed the traditional approach to living our lives and replaced it with a lifestyle where technologies are part of our lives, sometimes even more than human beings. Everything we touch, see or feel is related to technology. Smartphones, smart cities, smart energy saving, smart homes, smart trains, etc are life-transforming changes that happened due to IoT. The technologies are enhancing with every passing second, but still, the full potential of IoT is yet to be achieved.

But with growing technology, threats related to them. The problem of compromised networks and malfunctioned systems are increasing rapidly. With the enhancement of technologies, these threats are also evolving at a fast pace and are more refined. These threats in computer language are called Bots.

Prevention and identification of these bots are becoming more important day-to-day. Several technologies and techniques have been designed for this purpose of prevention- Antivirus Software, Network sniffers for prevention, secure passwords, and periodic system check; for detection- Anti Botnet software, Honeypots & honeynets, Signature-based detection techniques, Anomaly-based detection techniques, etc. Though there are existing methodologies to deal with botnet threats yet there's always a chance for improvisation.

That's why we came up with this project where we are using the hot-pot technology of Machine Learning to detect Botnet attacks on your system.

CHAPTER 1: INTRODUCTION

1.1 Introduction

The phrase "Internet of Things", proposed by Kevin Ashton, in 1999 to highlight how data collecting via sensor technologies had limitless possibilities. By 2023, there will be more than 20 times as many smart devices as current IT positions due to the IoT inclusion in Gartner's Top 10 Strategic Technology Trends. The Gartner report predicts that the IoT will be increasingly used in the utility, healthcare, government, physical security, and automobile sectors.

Internet of Things (IoT) consists of a network of objects, called "things", that connect to other devices and systems over the internet through sensors, software, and other technologies. These gadgets vary from common domestic items to complex industrial machines. Because the IoT idea is still in its early stages, it lacks a comprehensive security infrastructure/mechanism, putting critical data at risk. To keep IoT entities, businesses, and individuals secure, modern security measures must be used on the IoT network. The most serious security issue in IoT is botnet-based DDoS attacks, in which hackers infect devices with scripts.

Botnets (abbreviations for "robot networks") are networks of compromised machines controlled by a centralized assailant, or "bot-herder." A bot is any machine under the herder's control. Combined illegal operations can be carried out by all devices in the botnet under the direction of one entity. Intruders are able to carry out large-scale operations with malware by taking advantage of botnets (many of which are made up of millions of bots). Because botnets are still controlled by foreign attackers, infected devices can be upgraded and modified on the fly. Therefore, bot herders can frequently rent access to segments of their botnets on the black market in exchange for substantial financial gains. The attacker, sometimes known as the Botmaster, spreads Trojans, malware, or both, increasing the number of bots on the network.

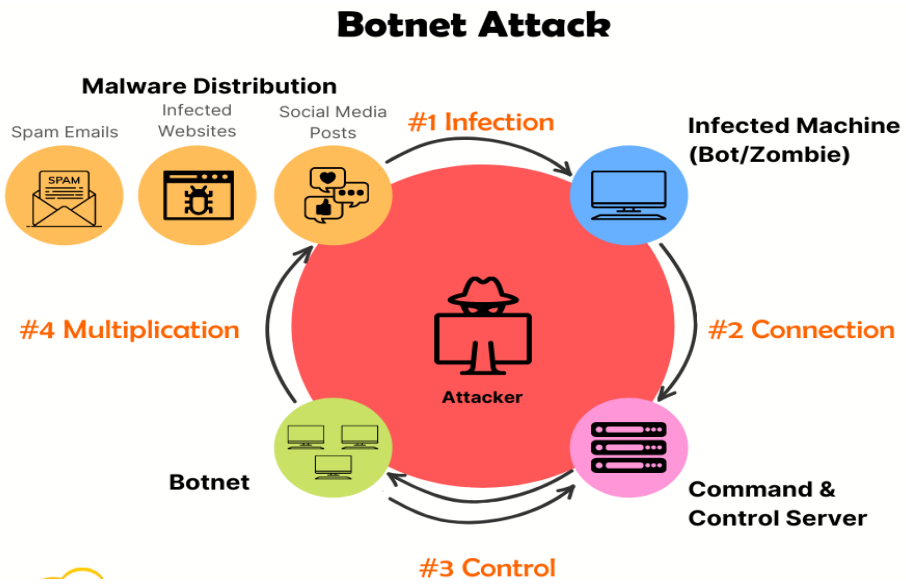


Fig 1.1 A Typical Botnet Attack Structure

Some common real-life botnet attacks include

Email spam- Spam botnets are among the largest in size, despite email being viewed as an outdated attack channel. In most cases, bots are used to send millions of spam messages, usually containing malware.

DDoS attacks use the enormous size of the botnet to flood a target network or server with requests, making it inaccessible to intended users. DDoS attacks target companies for personal or political reasons, or to extort money to stop the attack.

Financial attacks include botnets explicitly designed to steal cash from businesses and credit card information. Financial botnets, such as the ZeuS botnet, have been responsible for attacks that have stolen millions of dollars from multiple companies in a matter of minutes.

Botnets are distinguishable from other malware due to their distributed architecture. Botnets, like worms, may spread over millions of devices. In contrast to worms, zombie nodes in a botnet may collaborate and be managed by a central server. Botnets cannot be categorized as other forms of malware due to their dispersed architecture. Many publications attempt to summarize botnet phylogeny. The topology of the C&C infrastructure employed, the propagation mechanism, the exploitation approach, and the accessible set of operations used by the attacker are the key categorization areas of botnets.

C&C topology

"Command and Control" (C&C) servers are centralized systems that can send commands and receive output from a botnet. If an intruder wants to launch a DDoS attack, they can send specific commands to their botnet's command and control servers to instruct them to launch an attack against a specific target, and all packet sniffers interacting with the contacted C&C server would then launch a coordinated attack. Botnet C&C servers are often organized in one of four architectures, each of which has advantages and disadvantages: star, multiple servers, hierarchical and random.

The C&C server acts as the brain of the botnet, providing instructions to the bots on what actions to take, such as launching DDoS attacks, spreading malware, or conducting other malicious activities. The bots typically communicate with the C&C server using various communication protocols, such as HTTP, IRC, or P2P, to receive commands and report their status. This allows the botnet operator, also known as the botmaster, to remotely control and manipulate the bots for their nefarious purposes.

The C&C server is typically designed to be resilient and resistant to takedown efforts by law enforcement or security researchers. Botmasters often use sophisticated techniques to hide the location and identity of the C&C server, such as using proxy servers, domain name generation algorithms (DGA), and encryption to obfuscate communications. This makes the detection and disruption of the C&C server a challenging task.

Detecting and disrupting the C&C server is a critical step in mitigating botnet attacks. By identifying and blocking the communication between the bots and the C&C server, the bots lose their ability to receive commands and effectively neutralize the botnet. This requires advanced techniques, such as machine learning-based anomaly detection, network traffic analysis, and behavior-based heuristics, to identify patterns and characteristics associated with C&C communications and distinguish them from legitimate traffic.

In conclusion, the Command and Control (C&C) server is a critical element in botnet attacks, serving as the central hub for controlling and coordinating the activities of compromised devices. Detecting and disrupting the C&C server is crucial in mitigating botnet attacks and requires advanced techniques and collaboration among stakeholders. Efforts to combat botnet attacks continue to evolve as attackers employ sophisticated

tactics, highlighting the need for constant research and innovation in the field of botnet detection and mitigation.

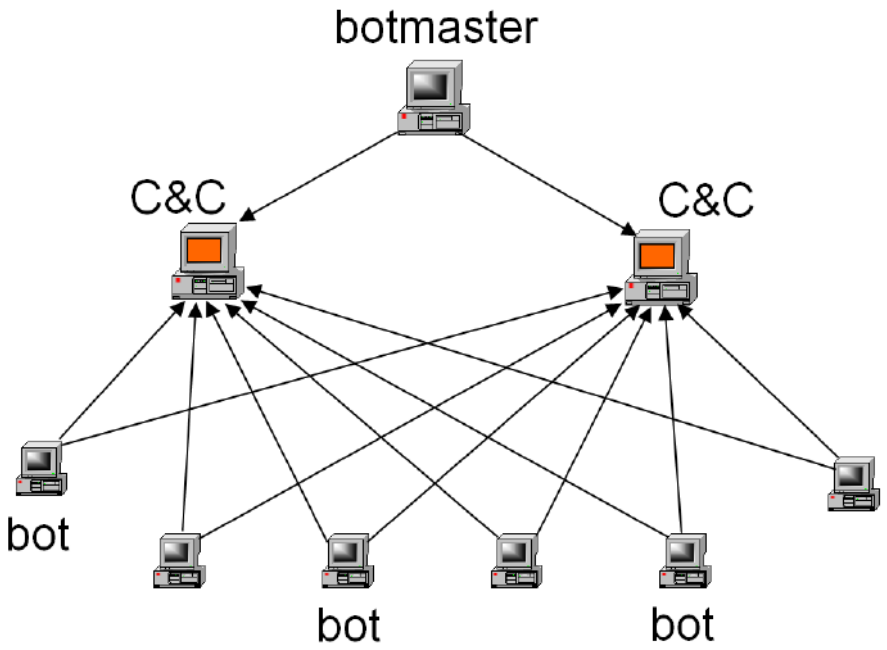


Fig 1.2 C&C Architecture

P2P topology

Peer-to-peer botnets differ from centralized command and control botnets in that they emphasize robustness through the deployment of a peer-to-peer network. In most other ways, peer-to-peer botnets are identical to centralized botnets. A peer-to-peer (P2P) network is a network in which devices are connected and share resources without delay instead of going via a server or authority that manages centralized resources. File-sharing is becoming a popular usage of P2P networks, also several P2P file-sharing apps are available. Some are eMule, uTorrent, and Deriv P2P. P2P networks are categorized based on their centralization and structure.

In a P2P botnet, bots communicate with each other using various communication protocols, such as TCP/IP, UDP, or HTTP, to exchange commands, updates, and other information. This allows the botnet to operate without a single point of failure, as the bots can dynamically switch between acting as clients and servers, making it challenging to disrupt the botnet by taking down a central C&C server.

P2P topology provides several advantages to botmasters. Firstly, it makes the botnet more resilient to takedown efforts, as there is no single point of failure that can be targeted for disruption. Even if some bots are taken down, the remaining bots can continue to communicate with each other and receive commands, making the botnet operational. Secondly, P2P botnets are more difficult to detect, as the communication patterns can resemble legitimate P2P traffic, making it harder to distinguish botnet traffic from legitimate traffic.

Detecting and mitigating P2P botnet attacks requires advanced techniques, such as machine learning-based anomaly detection, behavior-based heuristics, and network traffic analysis. These techniques involve monitoring and analyzing network traffic for patterns and characteristics associated with botnet communications, such as unusual communication patterns, high volumes of traffic to suspicious IP addresses or domains, and known botnet signatures. Additionally, efforts to disrupt P2P botnets may involve identifying and blocking known botnet domains or IP addresses, as well as collaborating with ISPs and other stakeholders to block or take down the botnet nodes.

As botnet attacks continue to evolve, constant research and innovation are needed to effectively combat this persistent and evolving threat.

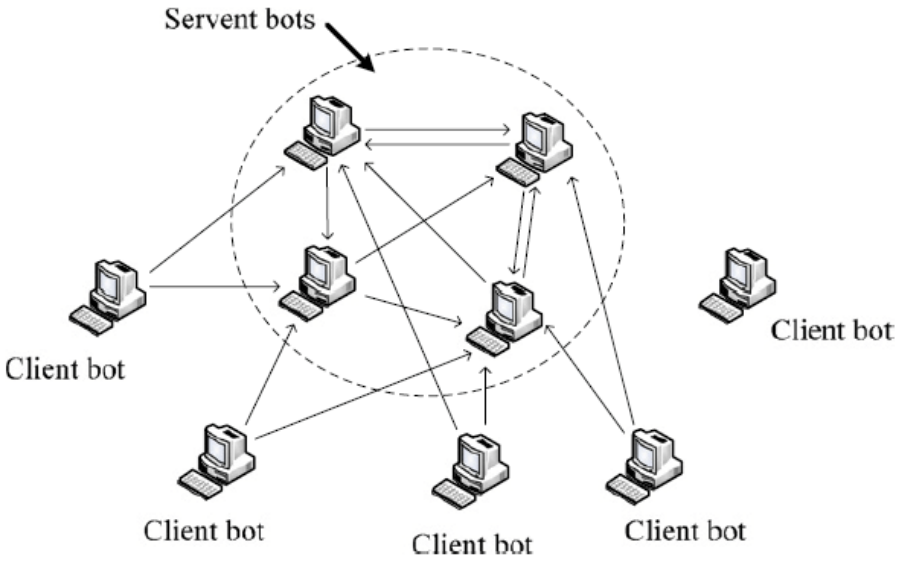


Fig 1.3 P2P Architecture

But the question is **why IoT devices became vulnerable to such botnet attacks?**

Hackers look for flaws in IoT devices or systems. If a hacker discovers a flaw, he or she will employ malware to infect machines and establish botnets, or he or she may organize assaults such as phishing to infect target devices with bot malware. With millions of unsecured IoT devices, we are exposed to a catastrophic attack that may significantly disrupt internet infrastructure and potentially bring it to a halt. By targeting known vulnerabilities in household equipment such as Wi-Fi routers, security cameras, and smart TVs, a hacker may proliferate bot servers. Assailants may use these vulnerable devices to provide a flood of traffic to certain sites, causing their servers to fail. Individuals whose devices are utilized in botnets are likewise at risk from these assaults. Many botnet victims are unaware that their devices are being utilized in this manner, exposing them to identity theft and even physical damage.

- ➔ Lack of security measures: Many IoT devices are designed and manufactured with a primary focus on functionality and cost, often neglecting proper security measures. This can include weak or default passwords, lack of regular security updates and patches, and limited or no encryption of data transmitted or stored on the device. These security weaknesses make it easier for botnet attackers to compromise IoT devices and gain control over them.

- ➔ Proliferation of IoT devices: The rapid proliferation of IoT devices in various industries and households has led to a significant increase in the attack surface for botnet attackers. With billions of IoT devices connected to the internet, including smart home devices, industrial control systems, healthcare devices, and more, the sheer volume and diversity of devices provide ample opportunities for attackers to find and exploit vulnerabilities.

- ➔ Lack of user awareness: Many IoT device users may not be aware of the potential security risks associated with these devices, and may not take necessary precautions to secure them. Users may not change default passwords, update firmware, or apply security patches, leaving their devices vulnerable to botnet attacks. Additionally, some IoT devices may lack user-friendly interfaces or instructions for securing the devices, further exacerbating the issue.

- ➔ Legacy and outdated devices: IoT devices often have a long lifespan, and many older devices may still be in use without receiving regular security updates or support from manufacturers. These legacy and outdated devices are more vulnerable to botnet attacks, as they may have known vulnerabilities that have not been patched or fixed, making them easy targets for attackers.

- ➔ Complexity of IoT ecosystem: The IoT ecosystem is complex, with multiple components such as sensors, gateways, communication protocols, and cloud services working together. This complexity can make it challenging to implement and maintain robust security measures across the entire ecosystem. Vulnerabilities in one component can potentially compromise the entire ecosystem and provide an entry point for botnet attacks.

- ➔ Economic and resource constraints: Some IoT devices, particularly those used in low-cost applications or in developing countries, may have limited computational resources, storage, or bandwidth, which may limit the implementation of robust security measures. Additionally, manufacturers of IoT devices may face economic constraints in investing in strong security measures due to cost considerations or lack of regulatory requirements.

Addressing these challenges requires a multi-faceted approach, including improved security measures in IoT device design and manufacturing, user education and awareness, regular updates and patches, and industry-wide standards and regulations to ensure the security of IoT devices and protect against botnet attacks.

There are various Botnet prevention techniques that exist. Botnets attack computers via worms or viruses that install the bot, or when users visit a malicious or untrustworthy website that exploits a software flaw and installs it.

- ➔ Periodic updates in OS
- ➔ Avoid downloading attachments from suspicious emails.
- ➔ Avoid P2P file sharing.
- ➔ Avoid malicious and suspicious links.
- ➔ Get Antivirus Software.
- ➔ Create secure passwords and keep updating them.
- ➔ Periodic system security check.

- ➔ Be aware of third-party applications.
- ➔ Network Intrusion Detection Systems (NIDS)
- ➔ Rootkit detection.
- ➔ Network sniffers.

Botnets are hard to spot since they employ little computational resources, making it difficult to detect. Botnet identification and tracking have been a key study area for network security experts in recent years. Various solutions have been presented, which may be divided into two categories. The first method is to use honeypots and honeynets, which can be considered active analysis. Focusing on passive network processing and assessment, the second strategy can be characterized as signature-based, DNS-based, anomaly-based, or mining-based.

Honeypots and Honeynets

A honeypot is described as "an environment in which vulnerabilities have been purposefully established in order to observe assaults and intrusions" (Pouget & Dacier, 2004). Honeynets of various sizes form a large-scale network. Honeynets based on Linux operating systems are typically selected due to their ability diversity and toolset features.

A honeypot is a single decoy system that is intentionally exposed to attackers, including botnets, to lure them into engaging with the system. The honeypot appears to be a legitimate system, but it is isolated from the main network and carefully monitored to detect any suspicious activities. For example, a honeypot may simulate a vulnerable IoT device, such as a smart camera or a smart thermostat, and expose it to the internet with known vulnerabilities. When a botnet attempts to exploit the vulnerabilities and infiltrate the honeypot, the security team can detect the attack and take appropriate actions to prevent further damage.

On the other hand, a honeynet is a network of decoy systems that mimics a complete network environment, including various types of devices and services. The honeynet is designed to attract and detect botnets that attempt to infiltrate the network and gain unauthorized access. The honeynet can simulate a small network or even an entire enterprise network, with different types of devices, operating systems, and services. When a botnet infiltrates the honeynet and attempts to propagate or communicate with command-

and-control servers, the security team can detect the activities and gather valuable information about the botnet's behavior and characteristics.

Honeypots and honeynets are effective in detecting botnet attacks because they provide a controlled environment for studying botnet behavior without posing any risk to the actual production systems or networks. They allow security experts to gather valuable intelligence on botnets, including their attack patterns, propagation techniques, and communication methods. This information can be used to develop effective countermeasures to prevent and mitigate botnet attacks in real production environments.

For example, a security team may set up a honeynet that mimics a smart home network with various IoT devices, such as smart cameras, smart thermostats, and smart speakers. When a botnet attempts to infiltrate the honeynet by exploiting vulnerabilities in these IoT devices, the honeynet can detect the attack and provide insights into the botnet's behavior, such as the types of devices it targets, the communication protocols it uses, and the commands it sends to the compromised devices. This information can then be used to develop patches, updates, or other preventive measures to protect real smart home networks from similar botnet attacks.

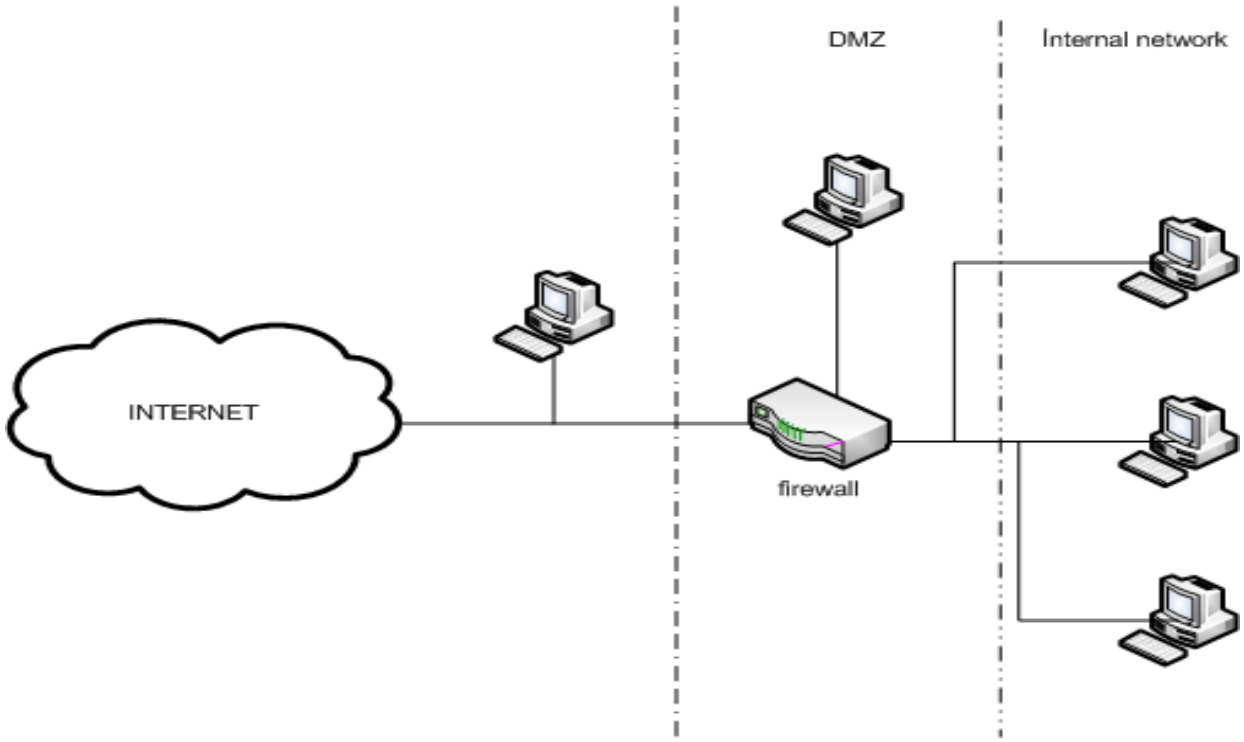


Fig 1.4 Potential Honeypot Traps

Signature-based Detection Technique

Malware signatures are commonly employed in malware detection and classification. Signatures of identified malware offer enormous power when it comes to classifying executables executing on a system. Snort and other rule-based intrusion detection systems identify established signatures of malware. They scan IoT traffic for signs of infiltration. The malware signature that is executable or the suspicious signatures of IoT device traffic created by malware can be used to identify malware. The botnets that are known to the world can also be detected using Signature-based detection. The consequences of this method are useless for unidentified bots.

Signature-based detection can be effective in detecting known botnet attacks that use well-known patterns or behaviors. However, it has limitations, as it relies on a database of known signatures, and may not be effective in detecting new or unknown botnets that do not match any existing signatures. Botnet operators can also evade signature-based detection by using encryption, obfuscation, or other techniques to hide their activities from signature-based detection mechanisms.

Furthermore, signature-based detection may also result in false positives or false negatives. False positives occur when legitimate network traffic or system behaviors are wrongly flagged as botnet activities, leading to unnecessary blockage or disruption. False negatives occur when botnet activities do not match any known signatures, and hence go undetected.

While signature-based detection can be a useful technique in detecting known botnet attacks, it has limitations and should be used in conjunction with other detection techniques to provide comprehensive botnet detection and prevention. Regular updates of the signature databases, combined with other advanced techniques such as behavior-based analysis, anomaly detection, and machine learning, can enhance the accuracy and effectiveness of botnet detection and mitigation efforts.

One example of signature-based detection is the use of antivirus software to detect and block botnets. Antivirus software maintains a database of signatures of known malware, including botnet-related malware, and scans files or network traffic for these signatures. When a file or network traffic matches a known botnet signature, the antivirus software can block the activity, quarantine the infected file, or alert the user or system administrator.

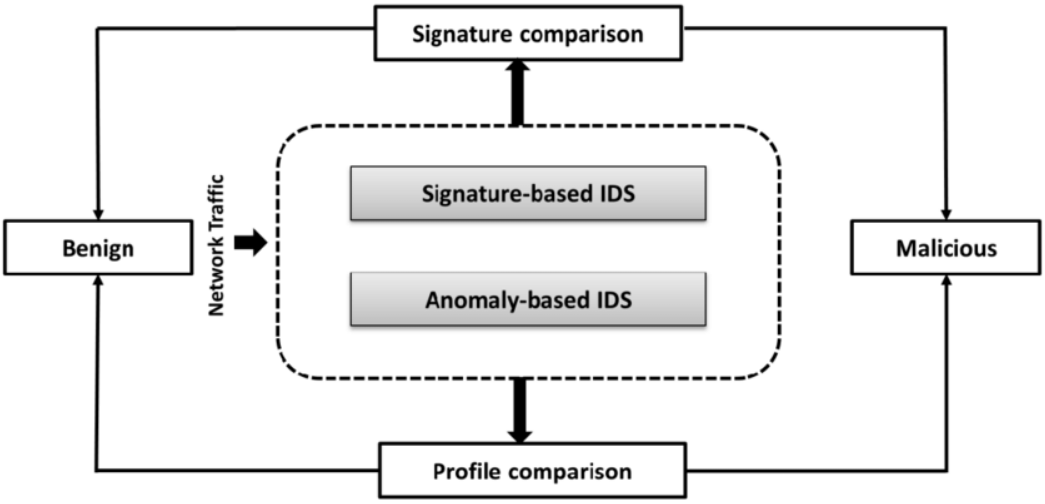


Fig 1.5 Signature-based and anomaly-based Detection

Anomaly-based Detection Technique

Botnet researchers spend a lot of time researching novel botnet detection algorithms based on network activity. Botnet detection based on network behavior anomalies attempts to detect bot activities based on network behavior anomalies such as unusual network latencies, netflow on unconventional and unoccupied ports, a high percentage of traffic for a semi-network, or strange structure behaviors that could reveal the presence of unauthorized entities in the network (Feily et al., 2009).

Anomaly-based detection is particularly effective in detecting unknown or zero-day botnet attacks that do not match any known signatures or patterns. It can also detect botnets that use sophisticated evasion techniques, such as encryption, obfuscation, or dynamic behavior changes, as anomalies in behaviors can still be detected even if the attack methods are changing.

However, anomaly-based detection also has limitations. It can generate false positives, as legitimate variations in network traffic or system behaviors may trigger alerts. For example, a sudden increase in network traffic due to legitimate activities such as a software update or a surge in user activity may be flagged as an anomaly. Fine-tuning the anomaly detection algorithms and establishing appropriate thresholds can help reduce false positives.

Anomaly-based detection is a valuable technique in detecting botnet attacks that do not match known patterns or signatures. It can provide an additional layer of defense against botnet attacks and can be used in conjunction with other detection techniques for comprehensive botnet detection and prevention. Regular monitoring, updating of baseline statistics, and tuning of anomaly detection algorithms can enhance the accuracy and effectiveness of this technique in detecting botnet attacks.

One example of anomaly-based detection is the use of statistical analysis to establish normal network traffic patterns. By analyzing historical data and establishing baseline statistics such as average traffic volume, protocol distribution, or packet size distribution, an anomaly detection system can then compare real-time network traffic against these baseline statistics. Any significant deviations from the established baseline can trigger an alert, indicating a potential botnet attack.

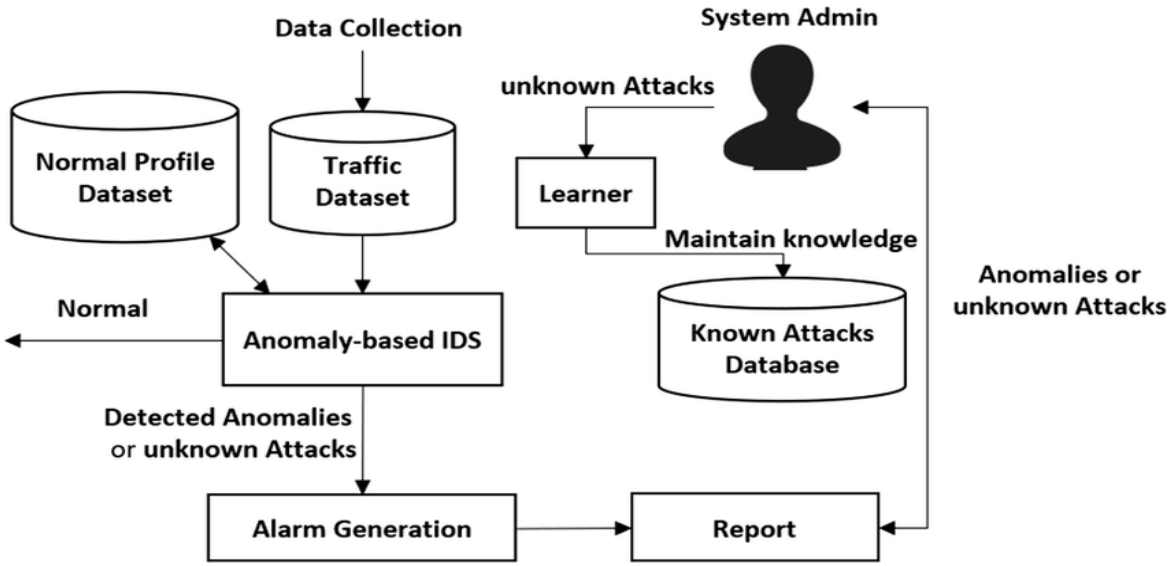


Fig 1.6 Anomaly-Based Detection System

DNS-based Detection Technique

DNS-based detection techniques are unobtrusively comparable to other anomaly-based detection algorithms. They are frequently predicated on the identification of abnormal DNS network traffic generated by bot machines. Thus, botnet DNS traffic may be detected by monitoring DNS operations and identifying irregular or unexpected DNS queries.

Kim et al. (Inhwan, Choi, & Lee, 2008) developed an approach for security counselors and administrators to detect botnets by giving significant visual information. The suggested approach is based on DNS traffic, which accounts for a little portion of overall network traffic. As a result, this approach is also suitable for real-time analysis.

DNS-based detection can also involve the use of threat intelligence feeds, which are databases of known malicious domain names or IP addresses associated with botnets. DNS traffic can be compared against these threat intelligence feeds to identify matches, indicating potential botnet activities. DNS-based detection can also involve the use of machine learning algorithms that analyze DNS traffic for patterns or behaviors that may indicate botnet activities. For example, machine learning algorithms can be trained on large datasets of normal DNS traffic and then used to identify deviations from the learned normal behaviors, such as unusual query patterns, domain names, or IP addresses associated with botnets.

However, DNS-based detection also has limitations. Botnets can employ various techniques to evade DNS-based detection, such as using encryption, domain generation algorithms (DGA) to generate random domain names, or hiding C&C communications within legitimate DNS traffic. DNS-based detection can also generate false positives, as legitimate activities such as legitimate DNS queries from misconfigured devices or legitimate use of CDNs (Content Delivery Networks) may exhibit similar patterns to botnet activities.

DNS-based detection is a valuable technique in detecting botnet attacks that rely on DNS for C&C communications. It can provide an additional layer of defense against botnet attacks and can be used in conjunction with other detection techniques for comprehensive botnet detection and prevention. Regular updates of threat intelligence feeds, fine-tuning of DNS analysis algorithms, and incorporating machine learning techniques can enhance the accuracy and effectiveness of DNS-based detection in detecting botnet attacks.

One example of DNS-based detection is the analysis of DNS query patterns. Botnets often generate a large number of DNS queries to communicate with their C&C servers, which may exhibit specific patterns such as high frequency, unusual domain names, or repetitive queries. DNS-based detection systems can analyze DNS query traffic and identify patterns that deviate from normal or expected DNS query behavior, and flag them as potentially indicative of a botnet attack.

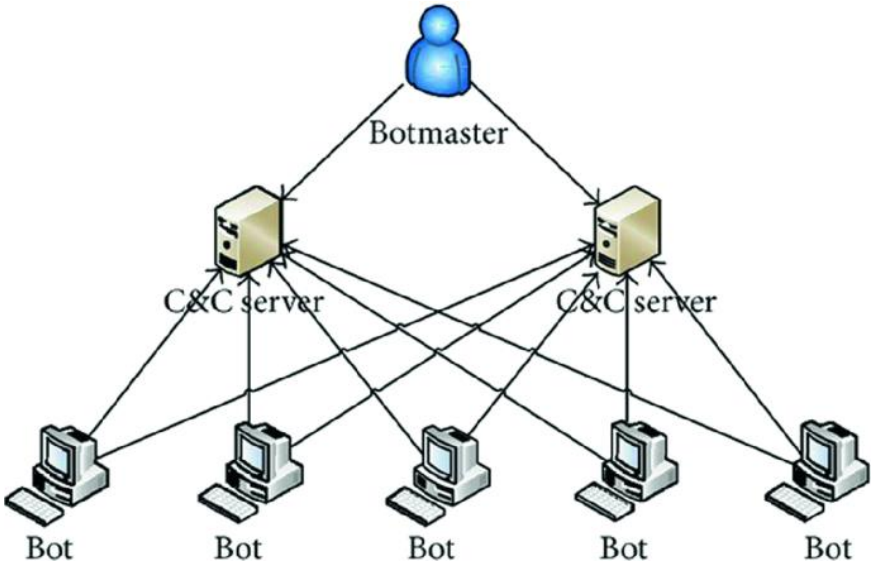


Fig 1.7 Botnet detection approach based on DNS

Mining-based Detection Technique

It is often difficult to distinguish C&C attacks in normal traffic activity. In this regard, machine learning-based data mining algorithms which help in the recognition of patterns are quite beneficial for detecting unexpected network patterns. Davis & Clark present an overview of existing pre-processing tasks for anomaly and mining-based intrusion detection algorithms. Strayer proposed a technique for detecting botnet C&C activity via a passive study of network flow data. Strayer's methodology works on the flow of characteristics such as duration, bytes/packet, bits/sec, and others. A new and effective solution for detecting bots through the structured graph is BotGrep developed by Nagaraja and his mates. Because of the decentralized C&C design of contemporary bot structures, identifying sub-graph networks is a highly valuable and convenient method of intrusion detection.

However, mining-based detection also has limitations. Botnets can employ various techniques to evade mining-based detection, such as using encryption, obfuscation, or polymorphic malware to disguise their activities. Additionally, mining-based detection may generate false positives or false negatives, as legitimate activities or legitimate software may exhibit similar patterns or behaviors to botnet activities.

Mining-based detection is a powerful technique in detecting botnet attacks that leverage data mining and machine learning algorithms to identify patterns, behaviors, or anomalies associated with botnet activities. It can provide valuable insights into botnet activities that may not be easily detectable by traditional methods. Regular updates of training data, fine-tuning of mining algorithms, and incorporating multiple detection techniques can enhance the accuracy and effectiveness of mining-based detection in detecting botnet attacks.

One example of mining-based detection is the analysis of network traffic for unusual patterns or behaviors. Botnets often generate a significant amount of network traffic for communication between infected devices and their command and control (C&C) servers. Mining-based detection techniques can analyze network traffic data, such as packet headers, payload content, or flow data, to identify patterns such as communication with suspicious IP addresses, use of non-standard ports, or abnormal data transfer rates, which may indicate botnet activities.

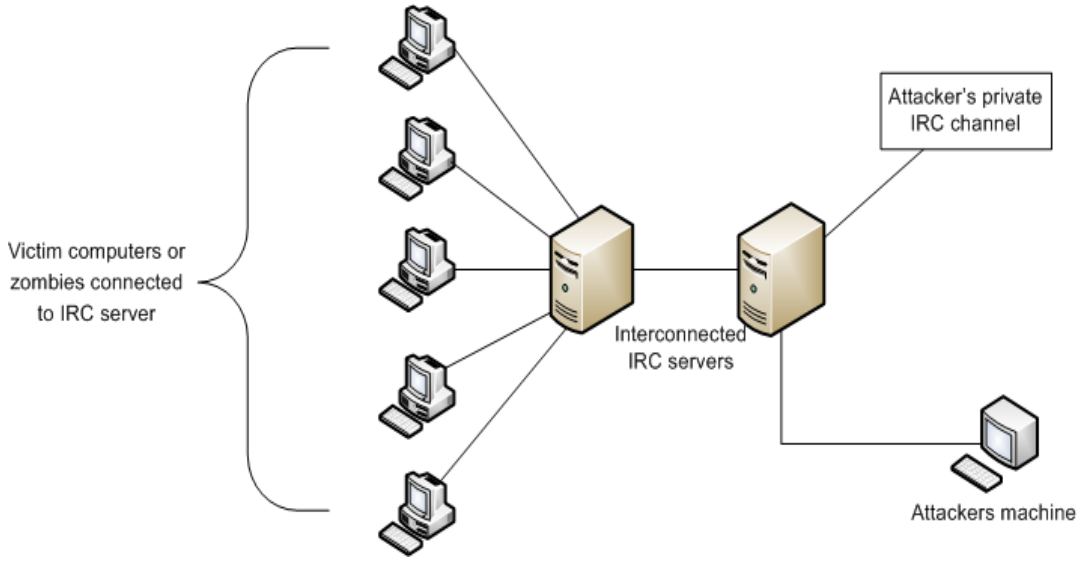


Fig 1.8 Botnet detection using Data Mining

Machine learning techniques & data-mining can be simply applied to network traffic data. Flow data is organized and connected, therefore it does not need extensive preparation. Furthermore, flow data suggests trends inside, making data mining techniques simple and effective for study. Consequently, flow-based approaches are more convincing since it just attends to discovering network flow information, which may be thought of as network flow meta-information without contents.

1.2 Problem Statement

Security dangers are emerging at a rapid rate as computers and technology progress. Botnets are one such security problem that demands much investigation and commitment to eradicate. The hacker numbers are growing rapidly exploiting gaps and defaults in corporate security systems and obtaining access to confidential files and data, increasing the threat of compromising cybersecurity. Cloud computing services are quickly expanding, allowing them to access data storage with no data overhead. However, botnet assaults include the server targeting information, resulting in faulty connections and denial of service. These assaults have become more polished and are still evolving. As a result, identifying and encountering them has become challenging.

The problem statement for this project is to develop an effective and efficient approach to detect botnet attacks in IoT environments using machine learning algorithms. This involves designing and implementing a solution that can analyze and process large amounts of data generated by IoT devices, such as network traffic, system logs, and device behavior, to identify patterns, behaviors, or anomalies that may indicate the presence of botnet activities. The solution should leverage machine learning algorithms, such as supervised or unsupervised learning, to train and deploy models that can accurately detect botnet attacks in real-time, while minimizing false positives and false negatives.

Some of the key challenges associated with this problem statement include dealing with the diverse characteristics of IoT devices, such as heterogeneous data formats, limited computational resources, and dynamic network environments. Another challenge is the dynamic nature of botnet attacks, as botnets continuously evolve and adapt to evade detection. Additionally, ensuring the privacy and security of IoT data during the detection process is crucial, as IoT devices may contain sensitive information.

The successful completion of this project will contribute to the field of cybersecurity by providing a proactive approach to detect and mitigate botnet attacks in IoT environments using machine learning algorithms. The developed solution can be integrated into existing IoT security frameworks to enhance the overall security posture of IoT deployments and protect against botnet attacks. Additionally, the project findings can serve as a foundation for further research and development in the area of botnet detection in IoT using machine learning and can be applied in various IoT domains, such as smart homes, industrial IoT, healthcare, and transportation, to safeguard IoT devices and data from malicious activities.

1.3 Objective

The purpose of this project is to find whether the device is under Botnet attack or not and can categorize it under different malware. The primary objective of the project is to develop an effective and efficient approach for detecting botnet attacks in IoT environments. This involves leveraging machine learning algorithms to analyze and process large amounts of data generated by IoT devices, such as network traffic, system logs, and device behavior, to identify patterns, behaviors, or anomalies that may indicate the presence of botnet activities. Specifically, the objectives of the project may include:

Designing and implementing a system for collecting and preprocessing IoT data: This may involve setting up data collection mechanisms to capture network traffic, system logs, and other relevant data from IoT devices, and preprocessing the data to prepare it for analysis.

Developing and training machine learning models: This may involve selecting appropriate machine learning algorithms, such as supervised or unsupervised learning, and training them using labeled or unlabeled data to build models that can accurately detect botnet attacks in IoT environments.

Evaluating the performance of the developed models: This may involve conducting experiments and evaluations to measure the accuracy, precision, recall, and other performance metrics of the developed models in detecting botnet attacks, and comparing the results against existing techniques or benchmarks.

Optimizing the models for real-time detection: This may involve optimizing the trained models for real-time or near real-time detection of botnet attacks, considering the resource-constrained nature of IoT devices and the need for timely response to cyber threats.

The successful achievement of these objectives will contribute to the advancement of the field of cybersecurity and IoT security, providing valuable insights and solutions for detecting and mitigating botnet attacks in IoT environments using machine learning algorithms.

In this project, we are using data mining and machine learning techniques, it is the most accurate detection technique to this date. But different machine learning techniques have different approaches for detecting a botnet attack. So here we are going to discuss various Machine learning techniques to get an alert for a Botnet Attack on a device. And also compare the accuracy of each model and classifier used to get a comparative and statistical

result so that we can have the best out of them to be used in the future for the same purpose.

1.4 Methodology

The methodology we are using in this project is Machine Learning and its classifying models.

Going through previous research on this topic, we found out that every thesis or journal has either chosen different datasets or they have been focusing on a particular type of Botnet and working on its detection through Machine Learning Algorithms. This made us think about what if a universal dataset could complete the requirements of all we need in just one place.

So the dataset we are using in this underlying research is **"Aposematic IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0)"**, which first became public in January 2020. This dataset was developed by capturing 20 malware and 3 benign executed in IoT devices traffic, which makes it a total of 23 captures(scenarios) of different traffic.

The purpose of this data set is to provide the community with two types of data sets: malware IoT traffic and secure IoT traffic. Added two additional columns for labels that describe network behavior for the flow of both malicious & benign traffic. This dataset additionally contains labels that describe the relationship between flows related to malicious or potentially harmful activity. This is how malware analysts and researchers are provided with such a piece of specific information.

Let us look at the libraries and platforms we have used for creating our project.

1.4.1 Python

Fig 1.9 Python logo

Python is a general-purpose, highly interactive, HLL-supported, and object-oriented programming language. Dynamically typed language, Python is also a garbage-collecting programming language. Guido van Rossum designed it around 1985-90. With source coding alike Perl, Python is available under the GNU GPL. Python is an enticing programming language for application development because it is simple to learn while being robust and adaptable.

Because of its syntax, dynamic typing, and nature as an interpreted language, Python is the finest language for scripting and rapid application development. Python supports a wide range of programming styles, including imperative, functional, and object-oriented. Python is not intended to be used for a specific goal, such as frontend web programming. It is recognized as a flexible programming language since it can be used with web, corporate, 3D CAD, and other applications. Because the variables are dynamically typed, we may assign an integer value by just writing a=10.

1.4.2 Jupyter Notebook

The platform that is used for generating the notebook corresponding to the project is Jupyter Notebook. The Jupyter Notebook is proposed under Project Jupyter. Providing an open-source platform, Project Jupyter is an interactive computing service for a handful of languages for programming. In 2014, Fernando Pérez and Brian Granger forked it out from IPython.

Fig 1.10 Jupyter Notebook logo

Creation and sharing of live code documents, equations, visualizations, etc, open-source Jupyter Notebook is free for everyone to use. The Jupyter project team is responsible for maintaining Jupyter Notebooks.

The IPython project had its own IPython notebook project, which gave birth to the Jupyter notebook. Supporting some primary coding languages such as Python, Julia, and R, it got the name Jupyter. Over 100 additional cores are currently available. Jupyter can be programmed in Python even if it comes from the kernel of IPython.

The Jupyter Notebook Application, a server-client software, allows notebook papers to be modified and launched from an internet browser. The Jupyter Notebook Application may be installed on distant servers and made available via the internet, or it can be used locally on a machine without requiring an online connection.

In addition to viewing, editing, and running notebook docs, the Jupyter Notebook Application has a "Dashboard", and a "control panel" accessing local files and allowing them to access notebook works or interrupted their kernels.

The NumPy and Pandas libraries are used for data processing in the initial stages.

1.4.3 NumPy

To handle arrays, the Python module NumPy is utilized. It also includes matrices, the Fourier transform, and routines for working with matrix multiplication. NumPy was created by Travis Oliphant in 2005. Because it is an open-source tool, you can use it for free. NumPy is an abbreviated form for Numerical Python.

Fig 1.11 NumPy Logo

NumPy extends Python with the computational capabilities of languages such as C and FORTRAN. Using Python's NumPy module, you may work with multidimensional arrays and matrices. It is useful for scientific or mathematical operations because of its efficiency and speed. NumPy also includes functionality for linear algebra and signal processing. As

a result, if you need to do mathematical operations on your data, NumPy is the library to use.

There exist some differences between arrays of Numpy & lists of Python. To begin, unlike Python lists, NumPy arrays contain several dimensions. Second, unlike Python lists, NumPy arrays are homogeneous. This means arrays of NumPy must all have similar type. Third, in terms of efficiency, NumPy arrays exceed Python lists. NumPy arrays may be created in a variety of ways. One way is to create an array from a Python list. After it has been formed, a NumPy array can be modified in a variety of ways. You can, for example, change the shape of an array or index into it to access its elements. NumPy arrays may also be used to perform arithmetic operations such as addition, multiplication, and division.

1.4.4 Pandas

Fig 1.12 Pandas Logo

Pandas is Python-based data analysis software. Wes McKinney founded Pandas in 2008 in response to a demand for a powerful and adaptable tool for mathematical modeling. Pandas is currently one of the most popular Python packages. It has a vibrant developer community. It is based on two essential Python libraries: NumPy for numerical operations and Matplotlib for qualitative data analysis. Pandas simplify access to numerous matplotlib and NumPy methods by acting as a wrapper for these libraries. For example, the pandas.plot() function combines numerous matplotlib routines into a single method, allowing you to plot a chart with minimal code.

The goal of pandas was to handle 2-D data. The pandas library has an in-built 2-dimensional data structure, DataFrame, which is similar to how the NumPy library consists of an in-built data structure known as an array with unique features & functionalities. This tool is where your data resides. To learn about data, you can clean, modify, and analyze it with pandas. In addition to being an important component of the data science toolkit, the pandas library is used in conjunction with other libraries in that collection.

1.4.5 Sklearn

Sklearn is a powerful and dependable Python machine-learning tool (Skit-Learn). It provides a number of powerful tools for statistical modeling and machine learning via a Python consistency interface, including classification, regression, clustering, and dimensionality reduction. This library is mostly written in Python and is based on NumPy, SciPy, and Matplotlib.

Fig 1.13 Sklearn Logo

It supports virtually all machine learning methods, including linear regression, logistic regression, k-means clustering, decision trees, and random forests. It is an open-source library that is provided commercially under the BSD license.

Getting towards the structure of the project, here we have 7 stages:

- Data Extraction
- Data Cleaning and Preprocessing
- Feature Engineering
- Dimensionality Reduction
- Train & Testing Split
- Execution of Classifiers
- Comparison & Result

The first stage is about extraction of the dataset from the local server to Jupyter's server. Then further we use the data that we want results on.

The second stage is probably the backbone of every data analytics project. The data cleaning and preprocessing help us normalize the data. This means any such value or data that is unfit for the model or could affect its performance is removed or filled in with an average value. These values could be null values or an outlier, which is taken care of in this step.

The third step of feature engineering is the most important step of all. As our dataset is very heavy and has a load of features we have to keep only those features that determine the model's work and performance. All those features that have either no or negligible effect on the model's accuracy are removed. In this project, it's very crucial to identify the important features so that our model could work perfectly and is not loaded much.

The fourth step is Dimensionality reduction. This is a process involved in Exploratory Data Analysis. Here we cut out some dimensions from the dataset that is not of much use. This step helps us in decreasing the vast dataset to only the precise dimension required.

The fifth step is very common in any Machine Learning project. The splitting of training and testing data should be done correctly for the model to behave properly. Usually, the training and testing dataset split equation is 80:20. The model is trained on the training dataset and its performance is evaluated from the testing dataset.

After this, comes the main step of the project, the Execution of Machine Learning classifiers. For this model, we are going to use a **Decision Tree Classification Algorithm, Support Vector Machine aka SVM Classifier, Gaussian Naive Bayes, and lastly CNN (Convolutional Neural Network)**. We will get to know about these classifiers later in the report.

These classifiers have their own accuracy and precision in this model. And so to find the best-performing classifier we will compare their results using graph analysis. Graph and visual analysis are done with help of two main libraries of Python: Seaborn and Matplotlib.

Matplotlib is a well-known graph charting package written in Python that is used for data science and machine learning tasks. **Seaborn** is a library that utilizes matplotlib as a basic library for plotting graphs but adds some more features to make the graphs more appealing and user pleasant.

Fig 1.14 Matplotlib Logo

Fig 1.15 Seaborn Logo

1.5 Organization

In this chapter we got through the overview of the project: what actually is a botnet attack; how and why it happens; the existing detection & prevention techniques; why it is needed for them to end or at least stop; and how are we gonna do that using current day technologies of Machine Learning. The organization of the report goes as follows.

In **Chapter 2 - Literature Survey,** there's gonna be a comparison and contrast of some existing research papers on this topic and see what different and better we are applying here.

In **Chapter 3 - System Development,** we are gonna study the design of our model, its development, how it is implemented, and last, we will analyze it.

In **Chapter 4 -Performance Analysis,** we have to go through the performance statistics of our implemented model, the results, and the output we got at various stages. We will also compare these results with the previously built theories and models.

In **Chapter 5 - Conclusion,** the project will be summarized, with whatever limitations or future scope we will have during or after the implementation of our model.

CHAPTER 2: LITERATURE SURVEY

Botnet attacks in the Internet of Things (IoT) have emerged as a significant cybersecurity challenge due to the proliferation of IoT devices and their vulnerabilities. Detecting botnet attacks in IoT is crucial to safeguarding the integrity, availability, and confidentiality of IoT systems and data. Machine learning algorithms have shown promising capabilities in detecting botnet attacks in IoT data, as they can learn from patterns and anomalies in the data to identify malicious activities. In this literature survey, we review the existing research on the detection of botnet attacks in IoT using machine learning algorithms.

Several studies have proposed various machine learning techniques for botnet detection in IoT. Wang et al. (2018) proposed a botnet detection framework for IoT devices using machine learning algorithms, including decision tree, random forest, and gradient-boosting classifiers. The authors used features such as packet size, protocol type, and port number from network traffic data to train the classifiers and achieved high detection accuracy. Raza et al. (2019) proposed an ensemble-based botnet detection approach using machine learning algorithms, including support vector machine, k-nearest neighbors, and logistic regression, along with feature selection techniques to identify malicious activities in IoT data. Their approach demonstrated improved detection performance compared to individual classifiers.

Deep learning algorithms, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have also been used for botnet detection in IoT. Tan et al. (2020) proposed a CNN-based approach for detecting botnet attacks in IoT by analyzing network traffic patterns. The authors achieved high accuracy in detecting botnet attacks by using the learned features from the CNN model. Singh et al. (2021) proposed an RNN-based approach for botnet detection in IoT by analyzing the behavior of IoT devices. The authors used sequential data from IoT devices' activities and achieved high accuracy in detecting botnet attacks using RNNs.

Ensemble learning techniques, such as stacking, bagging, and boosting, have also been employed for botnet detection in IoT. Yin et al. (2019) proposed a stacking-based ensemble approach for botnet detection in IoT using multiple base classifiers, including

decision tree, k-nearest neighbors, and support vector machine. The authors combined the outputs of the base classifiers to make the final decision and achieved improved detection performance compared to individual classifiers. Wang et al. (2020) proposed a boosting-based ensemble approach for botnet detection in IoT using multiple decision tree classifiers. The authors used adaptive boosting to combine the outputs of the decision tree classifiers and achieved high detection accuracy.

Feature selection and dimensionality reduction techniques have been widely used in botnet detection in IoT to reduce the complexity and improve the efficiency of machine learning algorithms. Yang et al. (2018) proposed a feature selection approach using information gain and recursive feature elimination to select the most relevant features for botnet detection in IoT. The authors achieved improved detection performance by reducing the feature space and improving the model's interpretability. Peng et al. (2019) proposed a principal component analysis (PCA)-based approach for dimensionality reduction in botnet detection in IoT. The authors used PCA to reduce the dimensionality of the feature space and achieved improved detection performance by capturing the most informative features.

The literature survey highlights that machine learning algorithms have been widely used for the detection of botnet attacks in IoT. Various techniques, including decision trees, support vector machines, neural networks, and ensemble learning methods, have been employed to detect botnet attacks in IoT data. Feature selection and dimensionality reduction techniques have been used to reduce the complexity and improve the efficiency of the detection models. Deep learning algorithms, such as CNNs and RNNs, have also shown promising capabilities in detecting

1. A novel feature for IoT botnet detection using classifier algorithms (Nguyen et al. 10)

Overview: This article improves on existing classification methods, is quick, detects IoT botnet families, for example, Mirai and Bashlite, and has a low FPR when benign files are exposed. The following are the main contributions of this paper: First, they introduced a unique PSI-rooted with a high-level subgraph-based feature for detecting IoT device botnets using a mix of processing deep learning and machine learning models. Second, they created a minimal number of characteristics with detailed behavioral descriptions, which take up less space and demand less processing time. Finally, they conducted comprehensive trials on various datasets. The assessment findings demonstrate the usefulness and resilience of PSI-rooted subgraph-based features for several machine algorithms, with a detection rate of more than 97%.

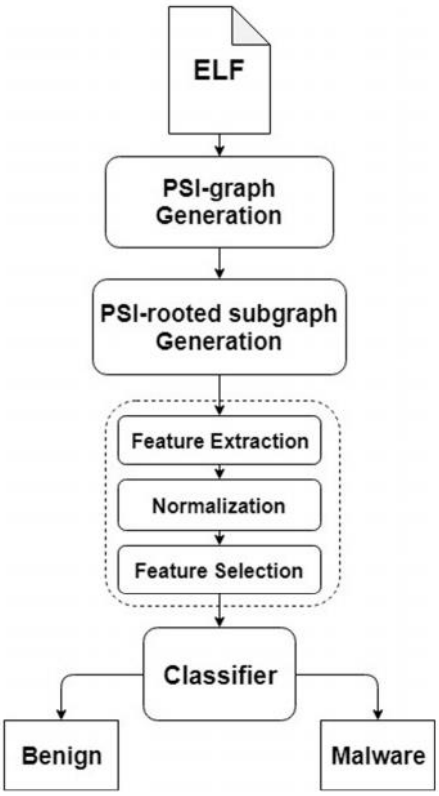


Fig 2.1 Overview of proposed model

Proposed Model and Methodology: They proposed an IoT botnet detection technique that uses PSI-rooted subgraphs as features to combine deep learning and machine learning. As shown in Fig.14, the suggested technique comprises four major

phases. First, extract the PSI graph from the ELF files. The PSI graphs are then traversed using a traversing technique to yield PSI-rooted subgraphs. Prior to the classification job, the raw data of PSI-rooted subgraphs is preprocessed so that machine learning algorithms can learn the data quickly. This process is divided into three stages: feature extraction to vectorize our data, normalization, and feature selection to improve the performance of learning algorithms. Finally, pick the best classifiers for identifying IoT botnet samples by applying different classifiers to the produced dataset.

The authors propose a methodology that involves the following stages:

Feature Engineering: The authors introduce a new feature called "IoT Botnet Feature" (IoTBF) that incorporates information related to IoT-specific characteristics, such as the number of connected devices, the type of devices, and the frequency of communication between devices. This feature is designed to capture the unique patterns of botnet attacks in IoT systems.

Classifier Algorithms: The authors use various machine learning algorithms, such as decision trees, support vector machines, and random forests, to train and test their model using the IoTBF feature. These classifiers are used to classify the data into a botnet or non-botnet traffic based on the patterns captured by the IoTBF feature.

Output: The proposed methodology involves several stages, including feature engineering and the use of machine learning classifiers. Feature engineering is a process of selecting and extracting relevant features from raw data to improve the accuracy of machine learning algorithms. The researchers have used novel feature engineering techniques to capture the unique characteristics of botnet traffic. The use of machine learning classifiers allows the system to learn from data and make accurate predictions.

Table 1. Comparison of processing time of different ML & DL classifiers

Comparison of processing time.

Classifier	Processing time (second)
Processing time with feature selection	
DT	1.84
RF	69.18
Bagging	144.64
kNN	12.83
SVM	237.78
Processing time without feature selection	
DT	18.49
RF	9305.21
Bagging	5225.02
kNN	19.60
SVM	1705.33

Table 2. Comparison of accuracy for traditional ML classifiers on IoT malware

Accuracy of conventional machine learning classifiers on IOT malware: a comparative summary.

Classifier	Accuracy (%)	
	Proposed method	Hamed et al. [43]
Random forest	98.8	92.37
SVM	99.3	82.21
kNN	97.8	94
Decision tree	97.8	92.36

Limitation/Future Scope: The authors acknowledge some limitations of their proposed methodology, such as the potential for false positives or false negatives in the detection results, the need for further validation on diverse datasets, and the potential for evolving botnet attack techniques that may require updates to the IoTBF feature. Future scope of research can involve exploring the combination of the IoTBF feature with other features or techniques for enhanced detection accuracy, evaluating the performance of the proposed methodology in real-world IoT environments, and considering the integration of anomaly-based or behavior-based detection methods for comprehensive botnet attack detection.

2. Botnet Attack Detection Using Machine Learning (Alshamkhany et al., 2020)

Overview: Security dangers are emerging at a rapid rate as computers and technology progress. Botnets are one such security problem that demands much investigation and commitment to eradicate. We employ machine learning to detect Botnet assaults in this study. Using the Bot-IoT and University of New South Wales (UNSW) datasets, 4 ML models based on 4 ML algorithms are created: Naive Bayes, K-Nearest Neighbor, Support Vector Machine, and Decision Trees. The decision trees model exhibited the greatest overall results in recognizing botnet invasions using 82,000 data from the UNSW-NB15 dataset, with 99.89% testing accuracy, 100% precision, 100% recall, and 100% F-score.

Proposed Model and Methodology: Among several available datasets this paper focused on the UNSW-NB15 dataset (43 features with DDos, Dos, as well as Reconnaissance, Backdoor, Worm, and Fuzzers labeled attacks). Data Extraction: The authors suggest collecting data from IoT devices, such as network traffic data or sensor data, to analyze botnet attacks. Data Cleaning and Preprocessing: The collected data is cleaned and pre-processed to remove noise, handle missing values, and normalize the data. Feature Engineering: Relevant features are extracted from the pre-processed data, which can include packet size, protocol type, port number, and other network traffic or sensor-related features. Dimensionality Reduction: To reduce the complexity of the data and improve the efficiency of the machine learning algorithms, dimensionality reduction techniques such as principal component analysis (PCA) or feature selection methods may be applied. Train and Testing Split: The pre-processed data is divided into training and testing datasets to train the machine learning models and evaluate their performance. Execution of Classifiers: Machine learning algorithms, such as decision trees, support vector machines, or neural networks, are trained on the training dataset and then used to predict the presence of botnet attacks in the testing dataset. Comparison of Results: The performance of different machine learning algorithms is compared based on metrics such as accuracy, precision, recall, and F1-score to determine the effectiveness of the proposed approach.

Dimensionality reduction and feature engineering were applied to this large dataset to reduce complexity and the preservation of variance in the data. For dimensionality

reduction, PCA was applied. In this work, the classifier takes the feature with the largest variance as input to ensure the selection of relevant features is in the best with the computational efficiency of the model.

This paper uses different classifiers of Python. For the purpose of evaluation of the best algorithm, confusion matrix, F-Score, Accuracy, and Precision are calculated. Classifiers used were: Gaussian Naive Bayes, kNN, SVM, and DT.

The dataset was split into training and testing parts before using these classifiers. The classifiers were trained using the training data. The classifiers were then evaluated to predict the labels using the testing dataset.

Output:

Table.3 Result of PCA

RESULTS OF APPLYING PCA ON THE DATASET USING DIFFERENT CLASSIFIERS

Classifier	Number of Components Chosen	Training Accuracy	Testing Accuracy
Decision Trees	43	99.28%	98.87%
Naïve Bayes	43	85.69%	85.91%
k-NN	42	90.5%	82.14%
SVM – rbf	20	99.9%	99.25%

Table.4 Accuracy of different classifiers

RESULTS OF APPLYING FEATU RE SELECTION ON THE DATASET USING DIFFERENT CLASSIFIERS

Classifier	Percentile of selected features	Number of selected features	Training Accuracy	Testing Accuracy
Decision Trees	67%	29	100%	100%
Naïve Bayes	67%	29	96.40%	96.39%
k-NN	72%	31	90.47%	82.23%
SVM– rbf	32%	14	99.85%	99.00%

Table.5 Performance comparison of classifiers

SUMMARY OF PERFORMANCE RESULTS FOR ALL THE COMPARED CLASSIFIERS

	Evaluation	Classifier - 5-fold validation			
		DT	NB Gaussian	SVM-rbf	KNN
Accuracy	Training accuracy	99.91%	97.10%	100%	91%
	Testing accuracy	99.89%	96.90%	98.80%	83.00%
Confusion Matrix	TP	7443	6986	22	6024
	FP	8	461	24	1346
	FN	10	53	0	1422
	TN	9006	8967	1954	7646
Precision	Normal traffic	100%	99%	100%	81%
	Attack traffic	100%	95%	99%	85%
	Macro avg	100%	97%	99%	83%
	Weighted avg	100%	97%	99%	83%
Recall	Normal traffic	100%	94%	48%	81%
	Attack traffic	100%	99%	100%	84%
	Macro avg	100%	97%	74%	83%
	Weighted avg	100%	97%	99%	83%
F1-score	Normal traffic	100%	96%	65%	81%
	Attack traffic	100%	97%	99%	85%
	Macro avg	100%	97%	99%	83%
	Weighted avg	100%	97%	82%	83%
Support	Normal traffic	7451	7447	46	7400
	Attack traffic	9016	9020	1954	9067
	Macro avg	16467	16467	2000	16467
	Weighted avg	16467	16467	2000	16467
	Accuracy	16467	16467	2000	16467

Limitation/Future Scope: The authors acknowledge some limitations of the proposed methodology, such as the potential imbalance in the dataset, the need for continuous updates and retraining of the machine learning models to adapt to evolving botnet attacks, and the potential for false positives or false negatives in the detection results. The future scope of research can involve exploring more advanced machine learning techniques, such as deep learning algorithms or ensemble methods, integrating additional data sources for more accurate detection, and considering the real-time or dynamic analysis of IoT data for more effective botnet attack detection.

3. Botnet Attack Detection by Using CNN-LSTM Model for Internet of Things Applications (Alkahtani and Aldhyani et al., 2021, 23)

Overview: This paper offered a deep learning-based strategy for mitigating the risks posed by DDoS attacks on IoT devices. Early identification of DDoS attacks can improve network security by accelerating attempts to unplug most IoT devices from Internet connections, preventing and limiting the growth of botnet attacks. We utilized the N-BaIoT dataset in this work, which was constructed from nine commercial IoT devices, including Danmini, Ennio, Ecobee, and Philips B120N/10, and was injected by two notable IoT attacks, BASHLITE and Mirai botnet attacks. The eBASHLITE attack has TCP flood, Junk, Scan, UDP flood, and COMBO sub-attacks, whereas the Mirai attack has ACK, SYN, Plain UDP, UDP flood, and Scan.

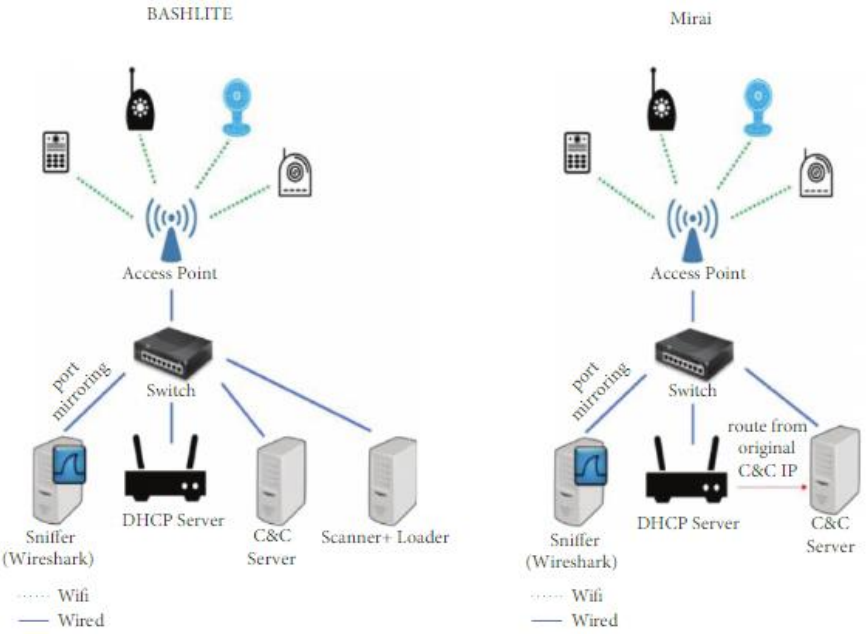


Fig 2.2 Setup for collecting botnet attack

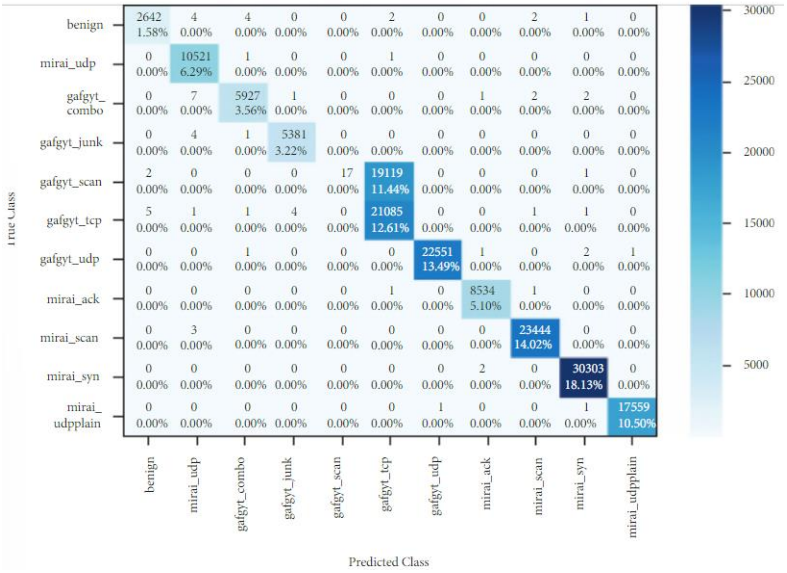
Proposed Model and Methodology: Five tests were carried out on various IoT platforms to analyze and examine the suggested solution. Using a network dataset taken from an IoT setup, machine learning, and deep learning techniques (CNN and LSTM) were used to identify botnet assaults. The datasets were separated into 20% testing data and 70% training data to validate.

The first experiment was on Doorbell Devices (Danminin and Ennio). For Danminin, precision, recall, and F-1 Score metrics are 93%, 91%, and 88% respectively. For Ennio, precision, recall, and F-1 Score metrics are 91%, 89%, and 85% respectively.

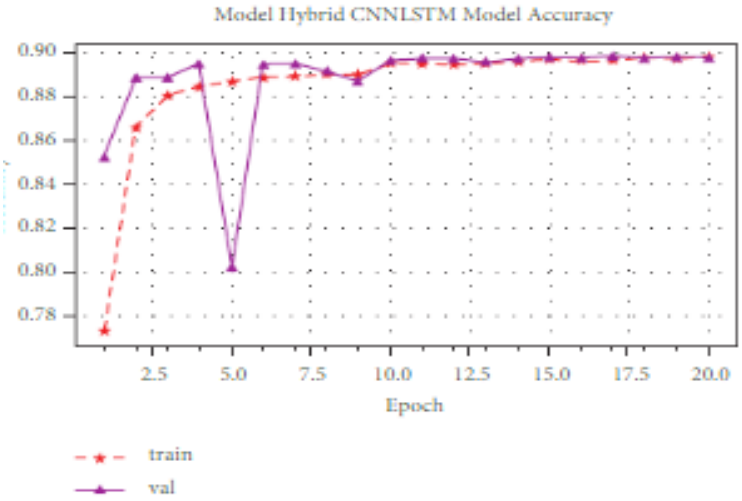
The second experiment was on Thermostat Devices. The evaluation metrics for a thermostat device were 94%, 89%, and 85% for precision, recall, and F-1 Score respectively. The accuracy of the CNN-LSTM proposed model increased from 80% to 88.53% with 20 epochs.

The third experiment occurred on Security WebCams. The weighted average of evaluation metrics for WebCams was 94% precision, 88% recall, and 84% F-1 Score. The performance of the model increased from 78% to 88%, while the training model loss is reduced from 20.0 to 0.16.

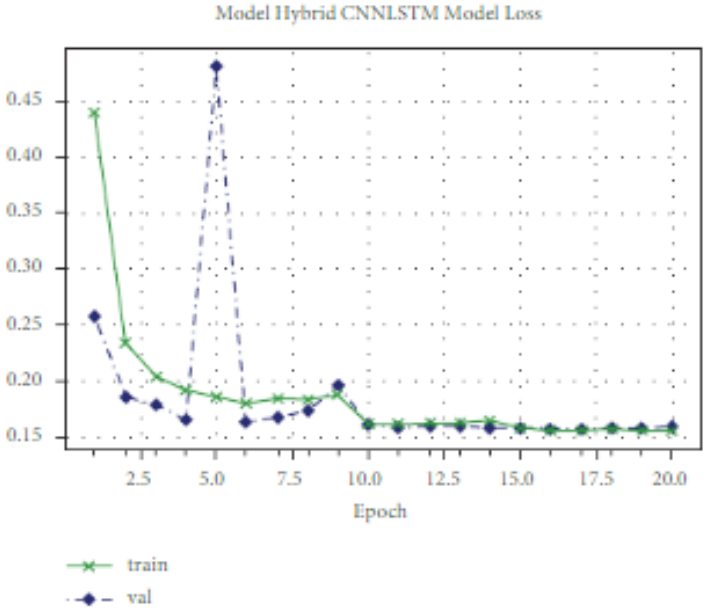
Output:



Graph 2.1 Confusion matrix for CNN_LSTM model on thermostat



Graph 2.2 Model Accuracy for CNN-LSTM



Graph 2.3 Model Loss for CNN-LSTM

Limitation/Future Scope: The authors acknowledge some limitations of their proposed methodology, such as the need for a large labeled dataset for training the CNN-LSTM model, the potential for overfitting due to the complexity of deep learning models, and the need for further evaluation on diverse IoT environments. The future scope of research can involve exploring the use of other deep learning techniques, such as attention mechanisms or transformers, for improved detection accuracy, incorporating multi-modal data sources for enhanced feature extraction, and evaluating the performance of the proposed methodology in real-world IoT applications with different types of botnet attacks and IoT devices.

4. N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders (Meidan et al., 2018, 10)

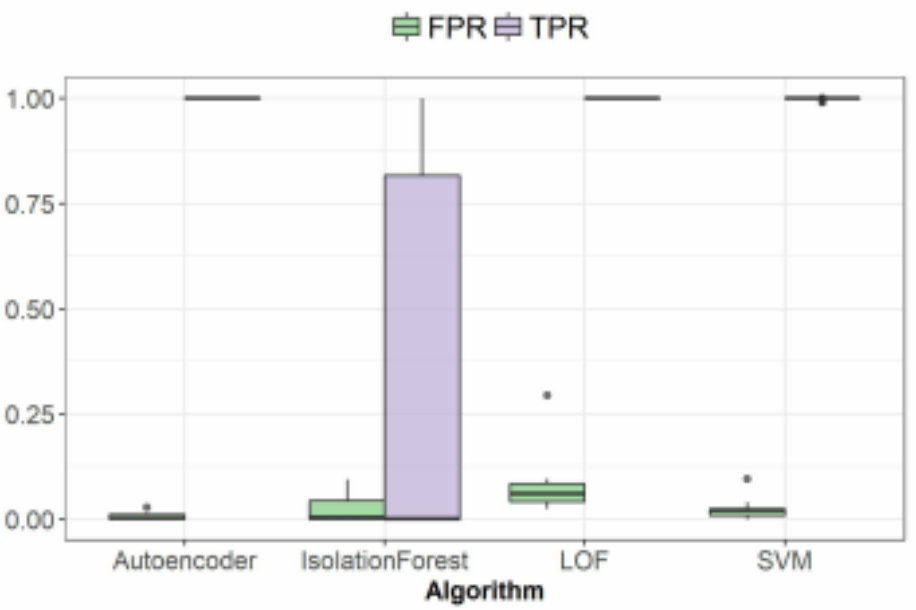
Overview: The author introduces & evaluates a novel network-based anomaly detection approach that takes network activity snapshots and employs deep autoencoders to detect unusual network traffic flowing from exploited IoT devices in this research. To test this system, Mirai and BASHLITE, two of the most notable IoT-based botnets, went after nine business IoT gadgets. The assessment results exhibited our proposed strategy's capacity to identify assaults started by compromised IoT gadgets that were essential for a botnet.

Proposed Model and Methodology: Deep autoencoders for every gadget, prepared on measurable qualities taken from harmless traffic information, are utilized in the proposed strategy for distinguishing IoT botnet attacks. When applied to new (possibly contaminated) IoT gadget information, distinguished irregularities might demonstrate that the gadget has been hacked. The vital stages of this procedure are as per the following: (1) information assortment, (2) highlight extraction, (3) preparing an inconsistency locator, and (4) consistent checking.

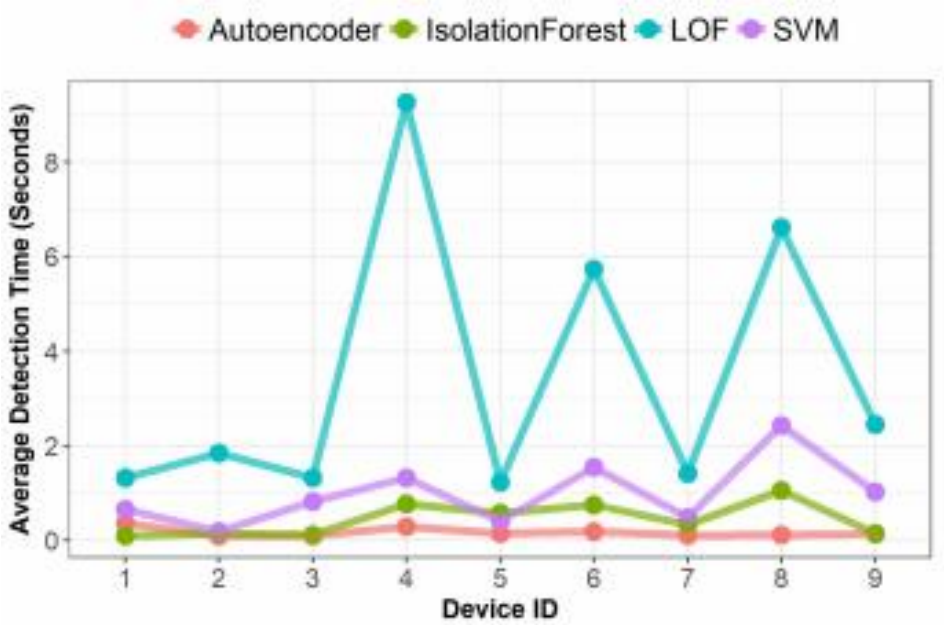
Each autoencoder has an info layer with a similar aspect as the quantity of elements in the dataset. Autoencoders effectively do dimensionality decrease inside, with the end goal that the code layer between the encoder(s) and decoder(s) packs and mirrors the information layer's significant properties. The examinations utilized four secret layers of encoders with diminishing sizes of 75%, half, 33%, and 25% of the element of the information layer. The following layers were decoders, which had a similar size as the encoders however in climbing requests (starting at 33%). We used some very similar (harmless) information to prepare three distinct calculations commonly utilized for irregularity recognizable proof during the autoencoder preparation and improvement stages: Neighborhood Exception Variable (LOF), One-Class SVM, and Disconnection Woods. Then, very much like the autoencoders, their hyperparameters were upgraded. At last, they did each of the former attacks all the while utilizing Mirai and BASHLITE's C&C servers. The qualities were then recovered from the unsafe information. Each harmless piece of DStst was connected to the related

malevolent piece of DStat, bringing about a solitary test dataset per IoT gadget with both harmless and vindictive occurrences.

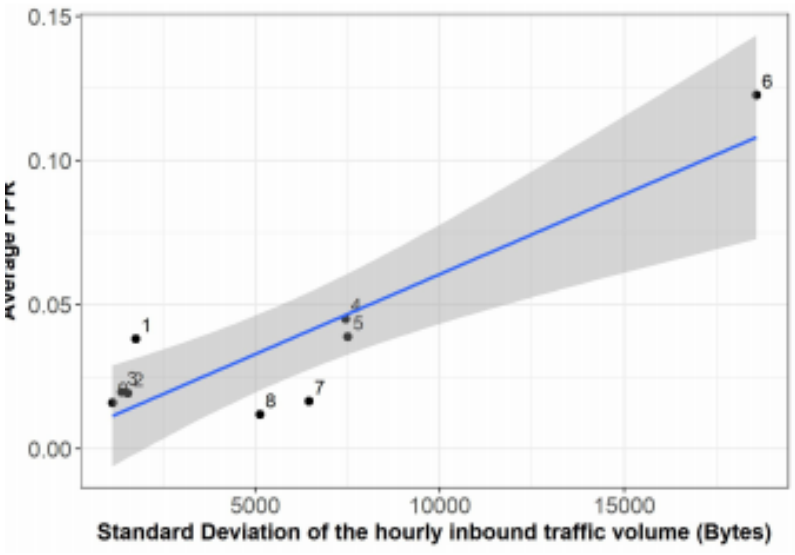
Output:



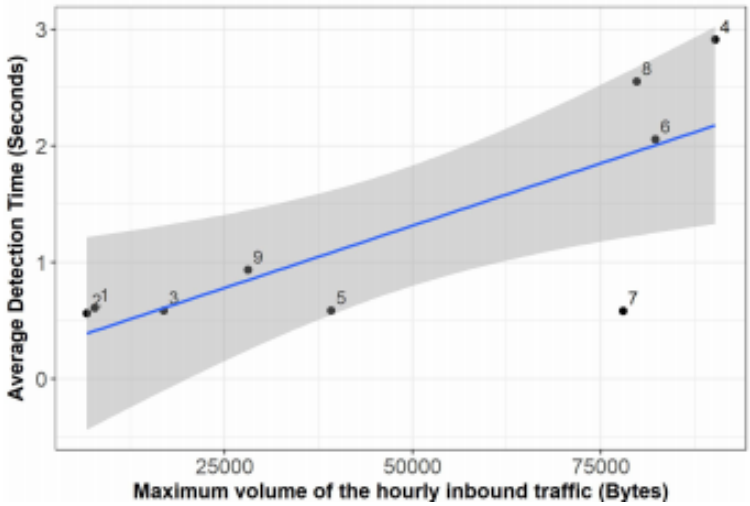
Graph 2.4 Accuracy of model



Graph 2.5 Computational time of model



Graph 2.6 Average FOR by traffic



Graph 2.7 Detection time by traffic

Limitation/Future Scope: The authors acknowledge some limitations of their proposed methodology, such as the need for a large labeled dataset for training the deep autoencoder, the potential for false positives or false negatives in the anomaly detection step, and the need for further evaluation on different types of botnet attacks and IoT environments. The future scope of research can involve exploring the use of other deep learning models or ensembles of models for improved detection accuracy, incorporating additional features or data sources for enhanced detection performance, and evaluating the performance of the proposed methodology in real-world IoT networks with diverse IoT devices and network configurations.

5. Botnet Detection Model Based on Artificial Intelligence (Peng et al., 2019, 7)

Overview: The study focuses on the development of a botnet detection model based on artificial intelligence (AI) techniques. Botnets are networks of compromised devices that are used by malicious actors to conduct various types of cyber attacks. Detecting botnet attacks is a critical task in ensuring the security of computer networks and protecting against potential cyber threats.

The overview of the study likely involves a brief introduction to the concept of botnets and their potential impact on cybersecurity. It may also highlight the limitations of traditional detection methods and the need for advanced techniques, such as AI, to effectively detect and mitigate botnet attacks. The overview may provide an overview of the proposed model, which could involve the use of machine learning algorithms, deep learning techniques, or other AI-based approaches for botnet detection. It may also discuss the motivation behind the selection of the proposed methodology and highlight the potential advantages of using AI techniques in botnet detection, such as improved accuracy, faster detection, and adaptability to new and emerging botnet threats.

Table.8 Top Ten TLDs Results

Normal Domain Name		DGA Domain Name	
TLD	**Ratio**	**TLD**	**Ratio**
com	62.42%	com	44.98%
org	8.38%	eu	19.25%
net	4.21%	net	7.86%
de	3.61%	biz	6.36%
uk	2.83%	org	4.00%
it	1.19%	ru	3.83%
fr	1.12%	cn	2.84%
jp	1.04%	info	1.94%
ca	0.94%	cc	1.07%
info	0.84%	pw	0.99%

Proposed Methodology: The proposed methodology in the study involves the use of AI techniques for botnet detection. This could include the use of machine learning algorithms, deep learning techniques, or a combination of both. The methodology may involve the following steps:

Data Collection: The researchers may collect a large dataset of network traffic data or other relevant data to train and test their botnet detection model. This dataset may include both benign and malicious traffic data.

Feature Extraction: The researchers may extract relevant features from the collected data, such as packet attributes, flow characteristics, or behavior patterns, to represent the data in a format suitable for machine learning or deep learning algorithms.

Model Training: The researchers may use the collected data and extracted features to train their AI-based botnet detection model. This could involve the use of supervised or unsupervised learning algorithms, where the model learns from labeled or unlabeled data, respectively.

Model Evaluation: The researchers may evaluate the performance of their botnet detection model using various metrics, such as accuracy, precision, recall, and F1-score, to assess its effectiveness in detecting botnet attacks.

Output: The output of the proposed botnet detection model could be a binary classification result, where the model classifies network traffic as either benign or malicious. The model may generate alerts or notifications when it detects potential botnet attacks, allowing network administrators to take appropriate actions to mitigate the threats.

Table.9 Cross Validation Result

Serial Number	Training Dataset		Validation Dataset		Number Rate of Training set to Testing Set	Accuracy	False Positive Rate	False Negative Rate
	Positive Sample	Negative Sample	Positive Sample	Negative Sample				
1	702236	701764	77764	78236	9:1	0.958	0.075	0.088
2	702007	701993	77993	78007	9:1	0.957	0.075	0.090
3	701748	702252	78252	77748	9:1	0.957	0.074	0.090
4	702032	701968	77968	78032	9:1	0.959	0.073	0.087
5	701942	702058	78058	77942	9:1	0.957	0.073	0.091
6	701781	702219	78219	77781	9:1	0.958	0.074	0.089
7	701959	702041	78041	77959	9:1	0.957	0.073	0.091
8	701992	702008	78008	77992	9:1	0.957	0.074	0.091
9	702242	701758	77758	78242	9:1	0.958	0.074	0.089
10	702061	701939	77939	78061	9:1	0.958	0.073	0.089

Limitation/ Future Scope: The study may discuss the limitations of the proposed botnet detection model, which could include challenges such as false positives, false negatives, scalability, and adaptability to new and evolving botnet threats. The researchers may also highlight the need for further research and development in the field of AI-based botnet detection, such as exploring new algorithms, incorporating additional data sources, or enhancing the model's performance in real-time or dynamic environments. The future scope of the study may also involve potential applications of the proposed model in practical scenarios, such as deployment in real-world networks, integration with existing security systems, or extension to other domains beyond IoT or connected computers.

CHAPTER 3: SYSTEM DESIGN AND DEVELOPMENT

3.1 Model Analysis and Design

As we already mentioned, the rapidly increasing advancement of IoT devices and their usage, all across the globe are increasing threats related to them. These cyber threats are of a lot more concern than any of us could have thought and even bigger threats for the world.

So, if cyber criminals are enhancing their attacking technologies for harming us, we should also be ready with whatever latest and best technology we have got. Without any question, that savior has to be Machine Learning and its powerful algorithms.

Here we are going to use the latest dataset created by Avast Software AIC Laboratory from network traffic experienced by IoT devices. They gave two datasets, a full IoT-23 dataset of 21 GB and a lighter version of IoT-23 of 8.8 GB. In this work, we are using the light version of the IoT-23 dataset. This dataset captures readings ranging from 2018 to 2019.

Using this dataset makes it unique and somewhat better than every other study and research previously done. This is a universal dataset with a combination of all important features from other used datasets.

The dataset is then pre-processed and cleaned, removing any useless value present in it such as null value or outliers, that could affect the performance of the model.

The feature selection is the most important here. Though it is a universal dataset for Botnet Attacks in IoT devices, its large size makes it difficult to compute as a whole. So we need to select the best features among them.

After that, normalization and dimensionality reduction will again help us to make the dataset handier.

Then we will be splitting the dataset into training and testing. The model is trained on the larger ratio of the training dataset and evaluated on the testing model.

The evaluation of the model depends upon the performance of various classifiers and their accuracy in the detection of the botnet and their classification.

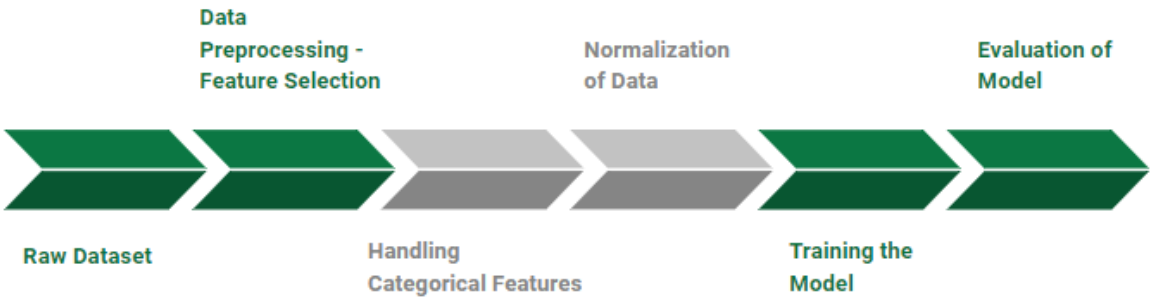


Fig 3.1 Flow of Model

3.2 Algorithm

Let's study the classifiers and the algorithm we have used in our model.

Decision Tree

A classification model is a tree in which every hub characterizes a test on a solitary component and each branch dropping from that hub compares to one of the likely qualities for that element. The decision Tree algorithm is a versatile and interpretable machine-learning technique for classification and regression tasks.

Working of Decision Tree Algorithm: Feature Selection: The algorithm selects the most important feature from the input features based on criteria like information gain, Gini impurity, or entropy. This is done by evaluating the impurity or randomness of the data at each feature and selecting the one that yields the maximum information gain.

Tree Building: The selected feature is used to split the data into subsets at each internal node of the tree. This process is repeated recursively until a stopping condition is met, such as reaching a maximum depth, having a minimum number of samples at a node, or achieving pure class labels at the leaf nodes.

Prediction: Once the tree is built, it can be used for prediction. New data is fed into the tree, and it follows the path of decision nodes based on the feature values until it reaches a leaf node, which represents the predicted outcome or class label for that data point.

Advantages of Decision Tree Algorithm: Interpretability: Decision Trees are easy to understand and interpret, making them suitable for explaining the decision-making process to stakeholders or non-technical users.

Handle Missing Values: Decision Trees can handle missing values in the input features by making decisions based on the available data.

Feature Importance: Decision Trees can provide information about the importance of different features in making accurate predictions, which can be useful for feature selection or feature engineering.

Non-linear Relationships: Decision Trees can capture nonlinear relationships between input features and the target variable, making them suitable for complex datasets.

Limitations of Decision Tree Algorithm: Overfitting- Decision Trees can be prone to overfitting, especially when the tree becomes too deep or when the dataset has noisy or irrelevant features. This can result in poor generalization performance on unseen data.

Lack of Robustness- Decision Trees can be sensitive to small changes in the input data, leading to different tree structures or predictions for slightly different datasets.

Decision Boundary- Decision Trees can create axis-parallel decision boundaries, which may not capture complex patterns in the data that require more flexible decision boundaries.

Class Imbalance- Decision Trees can have biased predictions when dealing with imbalanced datasets, as they tend to favor the majority class due to impurity-based splitting criteria.

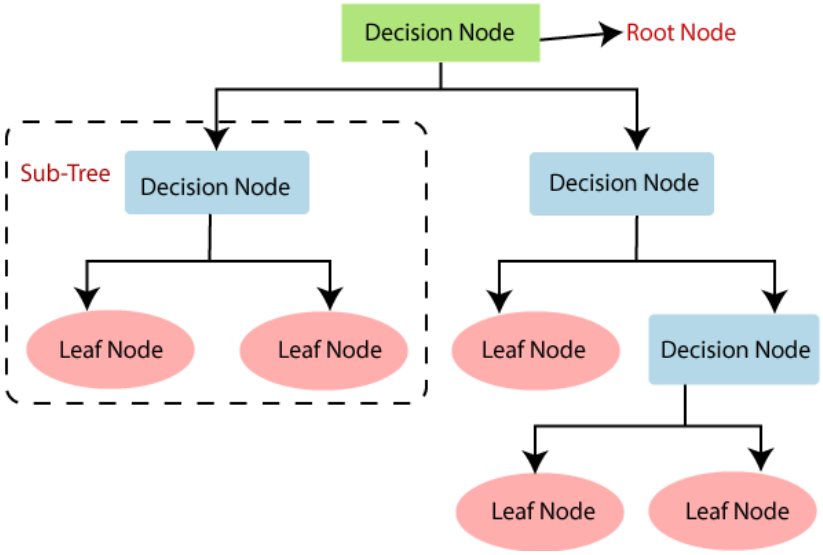


Fig 3.2 Decision tree Classifier

Support Vector Machine

It is using a nonlinear kernel, specifically, the radial basis function (RBF), to create a choice limit in light of tests from particular classes. The state of the choice is not set in stone by the part capability utilized as well as key hyper boundaries, for example, C, which controls the tradeoff between the perfection of the choice limit and the rightness of the grouping, and gamma, which characterizes the impact of the appropriation of the data of interest on the state of the chosen limit.

SVM with RBF kernel is a powerful algorithm for non-linear classification tasks and is widely used in various domains, including botnet detection in IoT. It leverages the RBF kernel to transform the input data into a higher-dimensional space, identifies support vectors to determine the optimal hyperplane, and uses the trained model for classification of new data points. Hyperparameter tuning is an important step to optimize the performance of the SVM model with the RBF kernel.

SVM with RBF kernel works as follows: Data Representation: The input data is represented as a set of feature vectors, where each feature vector contains multiple attributes or features that describe the characteristics of the data points. These feature vectors are used as input to the SVM algorithm.

Data Pre-processing: The input data is pre-processed to ensure that it is suitable for training the SVM model. This may involve tasks such as data normalization, handling missing values, and feature scaling to ensure that all features have similar scales and distributions.

Training Phase: The SVM model with RBF kernel is trained using a labeled dataset, which consists of input feature vectors and their corresponding class labels. The goal of the training phase is to find the optimal hyperplane that best separates the data points of different classes while maximizing the margin between the classes.

Kernel Trick: The RBF kernel function is used to transform the input feature vectors into a higher-dimensional space, where the data points are more likely to be linearly separable. The RBF kernel computes the similarity or distance between pairs of input feature vectors based on their Euclidean distance and assigns a weight to each pair.

Support Vectors: During the training phase, the SVM algorithm identifies a subset of data points called support vectors that lie closest to the decision boundary or margin. These

support vectors are used to determine the optimal hyperplane that maximizes the margin between the classes.

Classification Phase: Once the SVM model is trained, it can be used to classify new, unseen data points into their respective classes. The input feature vectors are transformed using the RBF kernel, and their class labels are predicted based on their position with respect to the learned decision boundary or margin.

Hyperparameter Tuning: SVM with RBF kernel has hyperparameters such as the regularization parameter (C) and the kernel parameter (gamma) that need to be tuned to optimize the performance of the model. This may involve techniques such as cross-validation or grid search to find the best values for these hyperparameters.

Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel is a popular machine learning algorithm used for classification and regression tasks. The RBF kernel is a type of kernel function that maps the input data into a higher-dimensional space to allow for non-linear separation of data points.

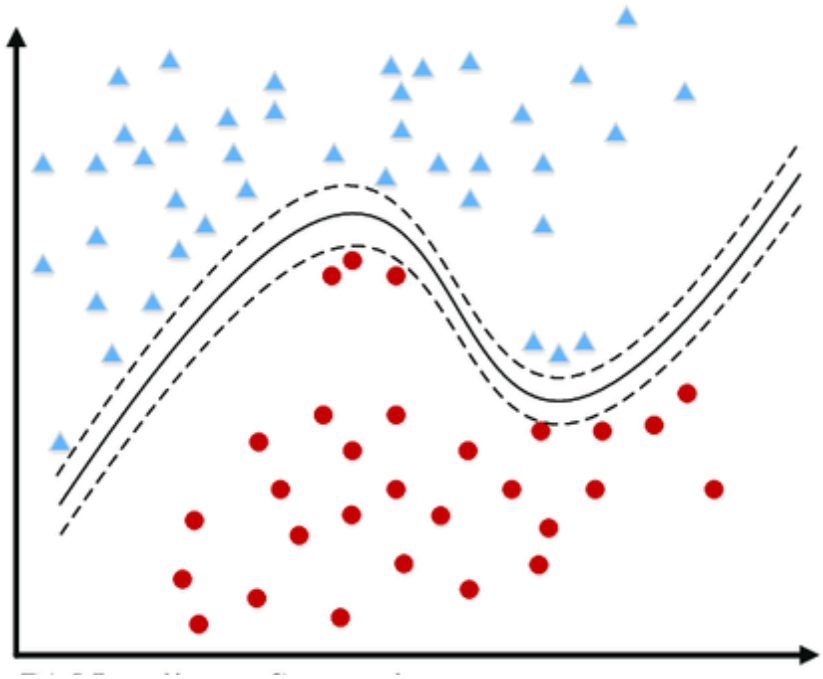


Fig 3.3 SVM with Non-Linear Kernel

Naive Bayes with Gaussian Probability

A probabilistic classifier that utilizes the Bayes hypothesis and expects restrictive freedom between the dataset's numerous qualities In view of the preparation set, NB computes the class likelihood. The Gaussian Naive Bayes algorithm assumes that the feature values in the dataset are normally distributed (i.e., they follow a Gaussian distribution), and it uses the Gaussian probability density function to model the distribution of each feature in each class. The algorithm is called "naive" because it makes the assumption that the features are conditionally independent given the class label, which means that the presence of one feature does not affect the presence of another feature.

Here are the main steps of the Gaussian Naive Bayes algorithm:

Data Preparation: The dataset is divided into features (predictor variables) and a target variable (class label). The feature values are assumed to be normally distributed.

Training Phase:

a. Calculate Class Prior Probabilities: The algorithm calculates the prior probabilities of each class by counting the frequency of each class label in the training dataset.

b. Estimate Mean and Variance: For each feature in each class, the algorithm estimates the mean and variance of the feature values using the training dataset. This is done separately for each class.

c. Calculate Gaussian Probability: The algorithm uses the estimated mean and variance of each feature in each class to calculate the Gaussian probability density function, which gives the probability of a given feature value belonging to a certain class.

Prediction Phase:

a. Calculate Class Posterior Probabilities: Given a new instance with feature values, the algorithm calculates the posterior probabilities of each class using the Gaussian probability density function for each feature in each class, along with the prior probabilities of each class.

b. Select Class Label: The algorithm selects the class label with the highest posterior probability as the predicted class label for the new instance.

Model Evaluation: The performance of the Gaussian Naive Bayes model is evaluated using appropriate evaluation metrics, such as accuracy, precision, recall, and F1-score, on a separate test dataset.

The Gaussian Naive Bayes algorithm is simple and computationally efficient, making it suitable for large datasets. However, its assumption of feature independence and the assumption of a Gaussian distribution for feature values may not always hold true in real-world datasets, which can affect its performance. Nevertheless, it can be a useful algorithm for classification tasks when these assumptions are reasonable, and it can be used in various domains, including text classification, spam detection, and medical diagnosis.

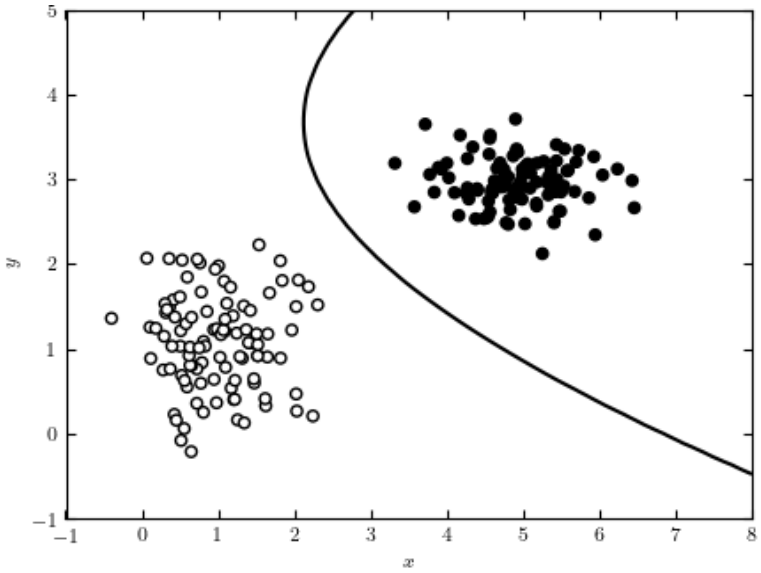


Fig 3.4 A simple Gaussian Naive Bayesian Plot

Convolutional Neural Network

CNN is a deep learning technique that is utilized to make a compelling picture characterization framework. Notwithstanding, the CNN model can likewise help with the development of effective safety efforts. The CNN strategy is similar to a customary brain network in that it has four essential layers: the information layer, convolutional layer, pooling layer, and completely associated layer. CNNs are designed to automatically learn features from raw pixel values in an image, without relying on handcrafted features. The key components of a CNN include convolutional layers, pooling layers, and fully connected layers.

Here are the main steps of the CNN algorithm:

Data Preparation: The input data consists of images, which are usually represented as matrices of pixel values. The dataset is divided into training, validation, and testing sets.

Convolutional Layers:

a. Convolution: Convolutional layers apply filters (also known as kernels) to the input image to extract local features such as edges, corners, and textures. This is done by sliding the filters over the input image and performing element-wise multiplication and summation.

b. Activation Function: An activation function is applied to the output of the convolution operation to introduce non-linearity into the model, allowing it to learn complex patterns.

Pooling Layers:

a. Pooling: Pooling layers downsample the feature maps generated by the convolutional layers, reducing the spatial dimensions and reducing the computational complexity of the model. Common pooling operations include max pooling and average pooling.

Fully Connected Layers:

a. Flattening: The feature maps from the convolutional and pooling layers are flattened into a 1D vector.

b. Fully Connected Layers: Fully connected layers are traditional artificial neural network layers where each neuron is connected to every neuron in the previous and next layers. These layers learn the high-level features and make predictions based on the learned features.

Output Layer: The final fully connected layer is connected to the output layer, which produces the predicted class probabilities using an appropriate activation function, such as softmax for multi-class classification.

Training Phase:

a. Forward Propagation: The input images are fed into the network, and the output is calculated through forward propagation.

b. Loss Calculation: The difference between the predicted class probabilities and the true labels is calculated using a suitable loss function, such as cross-entropy.

c. Backpropagation: The gradients of the loss with respect to the network parameters are computed and used to update the weights through backpropagation.

d. Optimization: An optimization algorithm, such as stochastic gradient descent (SGD) or Adam, is used to update the weights and minimize the loss.

Prediction Phase: The trained CNN is used to make predictions on new, unseen images by passing them through the trained network and selecting the class with the highest predicted probability as the final output.

Model Evaluation: The performance of the CNN model is evaluated using appropriate evaluation metrics, such as accuracy, precision, recall, and F1-score, on the validation or test dataset.

CNNs are powerful algorithms for image recognition tasks due to their ability to automatically learn features from raw pixel values. However, they can be computationally expensive and require a large amount of data for training.

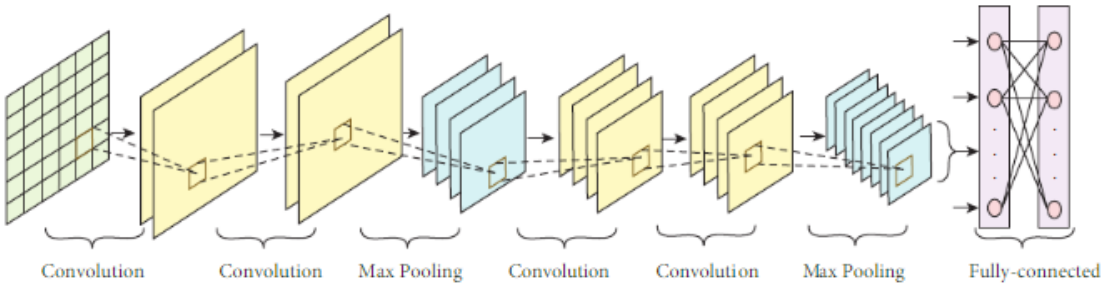


Fig 3.5 Architecture of a Convolutional Neural Network

CHAPTER 4: EXPERIMENTS AND RESULT ANALYSIS

4.1 System Developed and Experiments Performed

Let's analyze the structure of the model, there are various stages in the model. Let's get through each one briefly.

1. Data Extraction

   The IoT-23 dataset is made up of 23 distinct IoT network traffic recordings (called scenarios). These scenarios are separated into 20 network captures (pcap files) from infected IoT devices (each scenario will contain the name of the malware sample run) and three network grabs of legitimate IoT device data traffic. On each harmful scenario, we ran a particular malware sample on a Raspberry Pi, which utilized multiple protocols and did various activities. This dataset additionally includes labels to explain the relationship between flows connected to harmful or potentially malicious activity in order to give more specific information to network malware researchers and analysts.

   Attack: An attack can be defined as a flow that is attempted to damage a vulnerable device by behavior analysis. This label is for indicating that a device is been infected by an attack.

   Benign: This label is an indicator of the safe environment of a device. No suspicious activity was detected in the network.

   C&C: Indicates a link between a C&C server and attacked device. The C&C activity was discovered during the network malware capture examination because connections to the suspicious server are made on a regular basis, or our bot is extracting binaries from it, or some IRC-like or decoded orders are arriving and departing from it.

   DDoS: Indicates that the attacked device is carrying a Distributed Denial of Service assault. Because of the high amount of IoT network traffic routed to the same Internet Protocol address, these flows of traffic are spotted as part of a DDoS assault.

FileDownload: The label suggests that we are downloading a file to our infected device.

HeartBeat: This label describes that packets received during this connection are utilized by the C&C server to keep track of the attacked device.

Mirai: The label indicates that the bonds have Mirai botnet attributes. At the point when the streams have tantamount examples to the most predominant realized Mirai attacks, this mark is affixed.

Okiru: The mark denotes that the connections are part of the Okiru type of botnet.

PartOfAHorizontalPortScan: This mark demonstrates that the associations are used to do an even port output to accumulate data for future assaults.

Torii: this label denotes that the links are part of a Torii botnet.

	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	...	conn_state	local_orig	local_resp	missed_bytes	h
0	1545403848.981338	CDrkrSobGYxHhYfth	192.168.1.195	41040.0	185.244.25.235	80.0	tcp	http	1.477656	149	...	SF	-	-	2896.0	ShADa
1	1545403850.554283	CTWZQf2oJSvq6zmPAc	192.168.1.195	41042.0	185.244.25.235	80.0	tcp	-	3.147116	0	...	S0	-	-	0.0	
2	1545403857.781320	CvyynC4Sabj9BNXFRi	192.168.1.195	41042.0	185.244.25.235	80.0	tcp	http	1.305004	151	...	SF	-	-	5792.0	ShADa
3	1545403859.183341	CWYyiA2sgRijwk2jEd	192.168.1.195	41044.0	185.244.25.235	80.0	tcp	http	1.004605	148	...	SF	-	-	2896.0	ShAD
4	1545403860.282392	CYttPy2pqOIcen7UDh	192.168.1.195	41046.0	185.244.25.235	80.0	tcp	http	4.129647	148	...	SF	-	-	5792.0	ShAD
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
23137	1545490133.961380	C2F17zSUnGOcWzBa7	192.168.1.195	57110.0	185.244.25.235	6667.0	tcp	irc	32.840994	62	...	S3	-	-	0.0	Sh
23138	1545489934.221598	C93P4z4k5IRJD1rXJg	192.168.1.195	57092.0	185.244.25.235	6667.0	tcp	irc	36.290833	62	...	S3	-	-	0.0	Sh
23139	1545490198.463564	CXLZ3A2QY5E8weqpDk	192.168.1.195	123.0	147.251.48.140	123.0	udp	-	-	-	...	S0	-	-	0.0	
23140	1545490181.542213	CuXpFN3fWesWBXUhq1	192.168.1.195	123.0	82.113.53.40	123.0	udp	-	-	-	...	S0	-	-	0.0	
23141	1545490198.459568	Ct2Yhy4d33oL3yyZY9	192.168.1.195	123.0	89.221.210.188	123.0	udp	-	-	-	...	S0	-	-	0.0	

23142 rows × 21 columns

Fig 4.1 The IoT-23 Dataset

2. Data Cleaning and Preprocessing

Data cleaning and preprocessing are essential steps in the data analysis and machine learning pipeline. They involve the identification, correction, and transformation of data to ensure that it is accurate, consistent, and suitable for analysis. Data cleaning and preprocessing are crucial in order to address issues such as missing values, inconsistencies, errors, and outliers that may be present in the raw data.

The data cleaning and preprocessing help us normalize the data. This means any such value or data that is unfit for the model or could affect its performance is removed or filled in with an average value. These values could be null values or an outlier, which is taken care of in this step.

Data cleaning typically involves- *Removing or filling in missing values:* Missing values in the data can distort analysis and modeling results. Common techniques for handling missing values include imputation (i.e., filling in missing values with estimated values based on other data), deletion of rows or columns with missing values, or using algorithms that can handle missing data directly. *Handling inconsistencies and errors:* Inconsistent or erroneous data can lead to inaccurate analysis results. This may involve identifying and correcting inconsistencies in data values, resolving conflicting data, or removing duplicate data. *Handling outliers:* Outliers are data points that deviate significantly from the rest of the data and can skew analysis results. Identifying and handling outliers may involve removing them, transforming them, or imputing them based on domain knowledge.

Data pre-processing, on the other hand, involves transforming the data into a format that is suitable for analysis or modeling. This may include tasks such as- *Data normalization or scaling:* Data from different sources or with different units may need to be scaled or normalized to ensure that they are on the same scale and can be properly compared or combined. *Encoding categorical variables:* Categorical variables, such as gender or categorical labels, may need to be encoded into numerical values to be used in machine learning algorithms. *Feature extraction:* Extracting relevant features or variables from the raw data that are important for the analysis or modeling task at hand. *Data integration:* Combining data from multiple sources, resolving inconsistencies, and merging datasets to create a unified dataset for analysis or modeling.

	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	...	conn_state	local_orig	local_resp	missed_bytes	history	c
0	1.525880e+09	CDe43c1PtgynajGl6	192.168.100.103	60905	131.174.215.147	23	tcp	-	2.998796	0	...	S0	-	-	0	S	
1	1.525880e+09	CJaDcG3MZzvf1YVYI4	192.168.100.103	44301	91.42.47.63	23	tcp	-	-	-	...	S0	-	-	0	S	
2	1.525880e+09	CMBrup3BLXivSp4Avc	192.168.100.103	50244	120.210.108.200	23	tcp	-	-	-	...	S0	-	-	0	S	
3	1.525880e+09	CfHl9r3XMYtDQRrHnh	192.168.100.103	34243	147.7.65.203	49560	tcp	-	2.998804	0	...	S0	-	-	0	S	
4	1.525880e+09	C7USrA15nFVkniMqC5	192.168.100.103	34840	145.164.35.6	21288	tcp	-	-	-	...	S0	-	-	0	S	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99994	1.525917e+09	Cnxebd33nOXg4brQR8	192.168.100.103	43763	153.184.186.157	7205	udp	-	-	-	...	S0	-	-	0	D	
99995	1.525917e+09	CBbnGC4tdBYn2TVCN2	192.168.100.103	51645	44.30.32.248	48193	tcp	-	-	-	...	S0	-	-	0	S	
99996	1.525917e+09	C7lG5F4a2OgNDnyEc	192.168.100.103	34566	2.51.168.97	8080	tcp	-	-	-	...	S0	-	-	0	S	
99997	1.525917e+09	CTTeYw2v8lb7OmvUaa	192.168.100.103	43763	183.108.193.131	52028	udp	-	-	-	...	S0	-	-	0	D	
99998	1.525917e+09	CFDsvK15A8XVLw9IQ6	192.168.100.103	56403	31.210.235.245	23	tcp	-	2.999053	0	...	S0	-	-	0	S	

99999 rows × 21 columns

Fig 4.2 Dataset after data pre-processing

	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	...	conn_state	local_orig	local_resp	missed_bytes	histc
0	1525879832.01624	CDe43c1PtgynajGl6	192.168.100.103	60905.0	131.174.215.147	23.0	tcp	-	2.998796	0	...	S0	-	-	0.0	
1	1525879832.024985	CJaDcG3MZzvf1YVYI4	192.168.100.103	44301.0	91.42.47.63	23.0	tcp	-	-	-	...	S0	-	-	0.0	
2	1525879832.044975	CMBrup3BLXivSp4Avc	192.168.100.103	50244.0	120.210.108.200	23.0	tcp	-	-	-	...	S0	-	-	0.0	
3	1525879833.016171	CfHl9r3XMYtDQRrHnh	192.168.100.103	34243.0	147.7.65.203	49560.0	tcp	-	2.998804	0	...	S0	-	-	0.0	
4	1525879833.044906	C7USrA15nFVkniMqC5	192.168.100.103	34840.0	145.164.35.6	21288.0	tcp	-	-	-	...	S0	-	-	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99994	1532526102.004508	CMeH6R2aua5c5Dd65a	192.168.100.111	41762.0	221.182.209.127	23.0	tcp	-	-	-	...	S0	-	-	0.0	
99995	1532526102.00451	CvqGx33hsXDpDVXa1i	192.168.100.111	58758.0	208.50.139.48	23.0	tcp	-	-	-	...	S0	-	-	0.0	
99996	1532526102.004511	CC83RoUd9RLFuTL81	192.168.100.111	40400.0	40.95.136.51	23.0	tcp	-	-	-	...	S0	-	-	0.0	
99997	1532526102.004752	C4ISld2cuSukEEuQtk	192.168.100.111	27117.0	122.37.183.236	23.0	tcp	-	-	-	...	S0	-	-	0.0	
99998	1532526102.004756	C4U1azYmDx32faVY7	192.168.100.111	23227.0	189.62.234.179	23.0	tcp	-	-	-	...	S0	-	-	0.0	

1444674 rows × 21 columns

Fig 4.3 Dataset after data cleaning

3. Feature Engineering and Dimensionality Reduction

Our dataset is very heavy and has a load of features. We have to keep only those features that determine the model's work and performance. All those features that have either no or negligible effect on the model's accuracy are removed. In this project, it's very crucial to identify the important features so that our model could work perfectly and is not loaded much.

Feature engineering and dimensionality reduction are important techniques in data analysis and machine learning that involve transforming and reducing the number of features or variables in a dataset to improve the performance and interpretability of machine learning models.

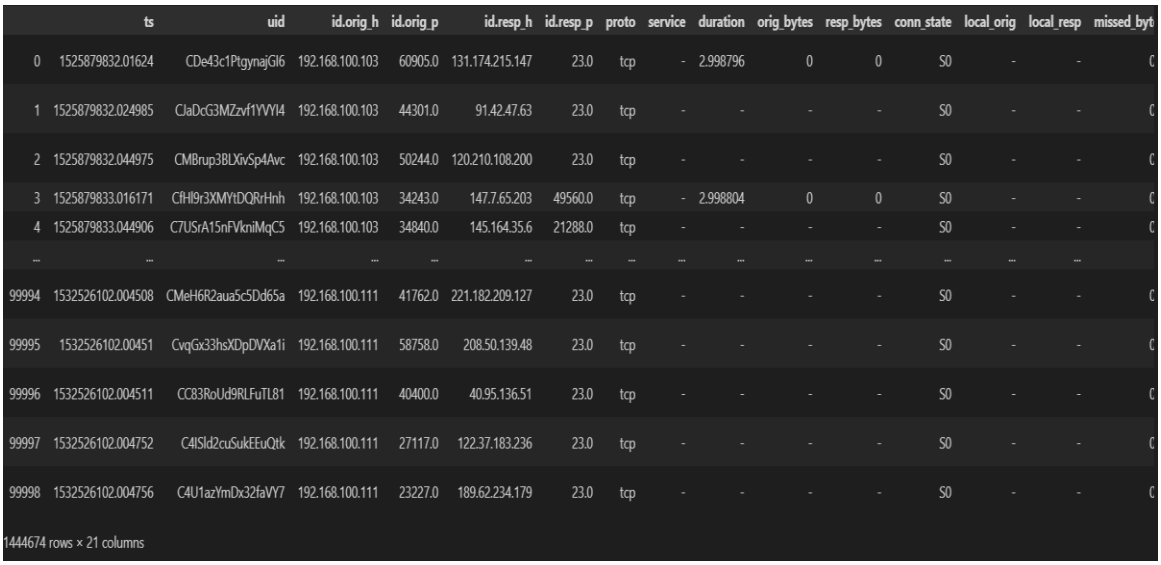
Feature engineering refers to the process of creating new features or variables from the existing raw data to improve the performance of machine learning algorithms. This may involve tasks such as: Feature extraction, Feature transformation, and Feature selection.

Next, Dimensionality reduction, on the other hand, involves reducing the number of features or variables in a dataset while retaining as much relevant information as possible. This can help to mitigate the "curse of dimensionality," which refers to the challenges and limitations associated with high-dimensional data. Dimensionality reduction techniques may include: Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Feature selection techniques like Recursive Feature Elimination (RFE), L1 regularization (Lasso), or Tree-based methods.

Feature engineering and dimensionality reduction are important techniques in data analysis and machine learning that can help to improve model performance, interpretability, and efficiency by reducing noise, removing irrelevant features, and mitigating the challenges of high-dimensional data.

	duration	orig_bytes	resp_bytes	missed_bytes	orig_pkts	orig_ip_bytes	resp_pkts	resp_ip_bytes	label	proto_icmp	proto_tcp	proto_udp	conn_state_OTH	conn_state_REJ
0	2.998796	0	0	0.0	3.0	180.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
1	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
2	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
3	2.998804	0	0	0.0	3.0	180.0	0.0	0.0	Benign	0	1	0	0	0
4	0	0	0	0.0	1.0	60.0	0.0	0.0	Benign	0	1	0	0	0
5	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
6	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
7	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
8	2.999300	0	0	0.0	3.0	180.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
9	2.993548	0	0	0.0	3.0	180.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
10	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
11	2.998807	0	0	0.0	3.0	180.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
12	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
13	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
14	0	0	0	0.0	1.0	60.0	0.0	0.0	Benign	0	1	0	0	0
15	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
16	2.999050	0	0	0.0	3.0	180.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
17	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
18	0	0	0	0.0	1.0	60.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0
19	2.999302	0	0	0.0	3.0	180.0	0.0	0.0	PartOfAHorizontalPortScan	0	1	0	0	0

Fig 4.4 Going through feature-engineering



	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	resp_bytes	conn_state	local_orig	local_resp	missed_byt
0	1525879832.01624	CDe43c1PtgynajGl6	192.168.100.103	60905.0	131.174.215.147	23.0	tcp	-	2.998796	0	0	S0	-	-	0
1	1525879832.024985	CJaDcG3MZzvf1YVYI4	192.168.100.103	44301.0	91.42.47.63	23.0	tcp	-	-	-	-	S0	-	-	0
2	1525879832.044975	CMBrup3BLXivSp4Avc	192.168.100.103	50244.0	120.210.108.200	23.0	tcp	-	-	-	-	S0	-	-	0
3	1525879833.016171	CfHl9r3XMYtDQRrHnh	192.168.100.103	34243.0	147.7.65.203	49560.0	tcp	-	2.998804	0	0	S0	-	-	0
4	1525879833.044906	C7USrA15nFVkniMqC5	192.168.100.103	34840.0	145.164.35.6	21288.0	tcp	-	-	-	-	S0	-	-	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99994	1532526102.004508	CMeH6R2aua5c5Dd65a	192.168.100.111	41762.0	221.182.209.127	23.0	tcp	-	-	-	-	S0	-	-	0
99995	1532526102.00451	CvqGx33hsXDpDVXa1i	192.168.100.111	58758.0	208.50.139.48	23.0	tcp	-	-	-	-	S0	-	-	0
99996	1532526102.004511	CC83RoUd9RLFuTL81	192.168.100.111	40400.0	40.95.136.51	23.0	tcp	-	-	-	-	S0	-	-	0
99997	1532526102.004752	C4ISld2cuSukEEuQtk	192.168.100.111	27117.0	122.37.183.236	23.0	tcp	-	-	-	-	S0	-	-	0
99998	1532526102.004756	C4U1azYmDx32faVY7	192.168.100.111	23227.0	189.62.234.179	23.0	tcp	-	-	-	-	S0	-	-	0

1444674 rows × 21 columns

Fig 4.5 Dataset after dimensionality reduction

4. Training & Testing Split

Training and testing split, also known as data partitioning, is a critical step in machine learning model development. It involves dividing the available dataset into separate subsets for training and testing the machine learning model.

The training set is used to train the machine learning model, while the testing set is used to evaluate the performance of the trained model. The training set typically constitutes a larger portion of the dataset, typically around 70-80% of the data, while the testing set constitutes the remaining portion, typically around 20-30% of the data.

Here, we are parting the preparation and testing data into a proportion of 80:20. The preparation dataset is dependably bigger in proportion so the model is prepared for pretty much every conceivable situation. Then, at that point, it is tried on testing information to assess the exhibition of the model.

The main purpose of the training and testing split is to evaluate the generalization performance of the machine learning model. By training the model on a separate subset of data and testing it on another independent subset of data, we can assess how well the model is likely to perform on unseen data in real-world scenarios.

The training set is used to train the model by feeding the input features to the model, allowing it to learn the underlying patterns and relationships in the data. The model then uses this learned knowledge to make predictions on the testing set, and the performance of the model is evaluated based on how well it predicts the target variable on the testing set. This evaluation helps to estimate the model's accuracy, precision, recall, F1-score, and other performance metrics.

Training and testing split is a crucial step in the machine learning model development process as it helps to assess the model's performance on unseen data, estimate its generalization performance, and ensure its reliability and accuracy in real-world scenarios.

```
X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes', 'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp', 'proto_tcp', 'proto_udp', 'conn_state_OTH'
Y = df['label']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=10, test_size=0.2)
```

Fig 4.6 Training & Testing Split

5. Execution of Classifiers

Decision Tree is a popular algorithm used for classification tasks. It works by recursively splitting the data based on feature values to create a tree-like structure that can be used for making decisions. To execute a Decision Tree classifier in botnet detection, the pre-processed dataset with labeled samples (i.e., botnet or non-botnet) is used to train the Decision Tree model. Once the model is trained, it can be used to classify unseen data samples as botnet or non-botnet based on the learned decision rules.

SVM is a powerful algorithm used for both classification and regression tasks. It works by finding the optimal hyperplane that best separates the data points of different classes. In botnet detection, SVM can be executed by training the model on the pre-processed dataset with labeled samples. The SVM model learns the optimal hyperplane that best separates the botnet and non-botnet samples in the

feature space, and can then be used to classify new data samples as botnet or non-botnet based on their position relative to the learned hyperplane.

Naive Bayes is a probabilistic algorithm used for classification tasks. It works based on the assumption of independence between the features, which allows for efficient and fast classification. In botnet detection, Naive Bayes can be executed by training the model on the pre-processed dataset with labeled samples. The Naive Bayes model learns the conditional probabilities of the features given the class labels (botnet or non-botnet), and it can be used to make predictions on new data samples based on these probabilities.

CNN is a deep learning algorithm specifically designed for image recognition tasks. In botnet detection, CNN can be used to analyze network traffic data, such as packet headers or payload information, as images. The CNN model is trained on the pre-processed dataset with labeled samples, and it learns the hierarchical features from the network traffic data to make predictions about whether a given traffic flow is indicative of a botnet activity.

	precision	recall	f1-score	support
Attack	0.99	0.99	0.99	798
Benign	0.95	0.55	0.70	39535
C&C	0.60	0.12	0.20	3102
C&C-FileDownload	1.00	0.78	0.88	9
C&C-HeartBeat	0.76	0.41	0.53	69
C&C-HeartBeat-FileDownload	1.00	0.67	0.80	3
C&C-Torii	0.00	0.00	0.00	7
DDoS	1.00	0.82	0.90	27856
FileDownload	0.60	1.00	0.75	3
Okiru	0.74	0.00	0.00	52607
PartOfAHorizontalPortScan	0.68	1.00	0.81	164946
accuracy			0.73	288935
macro avg	0.76	0.58	0.60	288935
weighted avg	0.76	0.73	0.65	288935

Fig 4.7 Classification report of Decision Tree Classifier

	precision	recall	f1-score	support
Attack	0.97	0.99	0.98	743
Benign	0.97	0.11	0.20	11473
C&C	0.00	0.00	0.00	1
C&C-HeartBeat	0.00	0.00	0.00	3
C&C-Torii	0.00	0.00	0.00	6
DDoS	0.00	0.00	0.00	4938
Okiru	0.00	0.00	0.00	9749
PartOfAHorizontalPortScan	0.68	1.00	0.81	53087
accuracy			0.69	80000
macro avg	0.33	0.26	0.25	80000
weighted avg	0.60	0.69	0.57	80000

Fig 4.8 Classification report of SVM Classifier

	precision	recall	f1-score	support
Attack	0.33	0.05	0.09	764
Benign	1.00	0.29	0.45	39944
C&C	0.66	0.11	0.18	2960
C&C-FileDownload	0.01	0.78	0.02	9
C&C-HeartBeat	0.00	0.70	0.01	60
C&C-HeartBeat-FileDownload	0.50	1.00	0.67	1
C&C-Torii	0.03	0.20	0.05	5
DDoS	1.00	0.82	0.90	27578
FileDownload	0.25	0.50	0.33	2
Okiru	0.21	1.00	0.35	52275
PartOfAHorizontalPortScan	1.00	0.00	0.00	165337
accuracy			0.30	288935
macro avg	0.45	0.50	0.28	288935
weighted avg	0.85	0.30	0.21	288935

Fig 4.9 Classification report of Gaussian Naive Bayes Classifier

```
Epoch 1/10
4515/4515 [==============================] - 26s 6ms/step - loss: 0.8835 - accuracy: 0.6902 - val_loss: 0.8618 - val_accuracy: 0.6929
Epoch 2/10
4515/4515 [==============================] - 24s 5ms/step - loss: 0.8592 - accuracy: 0.6936 - val_loss: 0.8614 - val_accuracy: 0.6934
Epoch 3/10
4515/4515 [==============================] - 24s 5ms/step - loss: 0.8590 - accuracy: 0.6942 - val_loss: 0.8605 - val_accuracy: 0.6934
Epoch 4/10
4515/4515 [==============================] - 24s 5ms/step - loss: 0.8597 - accuracy: 0.6936 - val_loss: 0.8606 - val_accuracy: 0.6934
Epoch 5/10
4515/4515 [==============================] - 24s 5ms/step - loss: 0.8576 - accuracy: 0.6944 - val_loss: 0.8611 - val_accuracy: 0.6934
Epoch 6/10
4515/4515 [==============================] - 24s 5ms/step - loss: 0.8586 - accuracy: 0.6937 - val_loss: 0.8600 - val_accuracy: 0.6934
Epoch 7/10
4515/4515 [==============================] - 24s 5ms/step - loss: 0.8577 - accuracy: 0.6943 - val_loss: 0.8602 - val_accuracy: 0.6934
Epoch 8/10
4515/4515 [==============================] - 24s 5ms/step - loss: 0.8578 - accuracy: 0.6942 - val_loss: 0.8602 - val_accuracy: 0.6934
Epoch 9/10
4515/4515 [==============================] - 24s 5ms/step - loss: 0.8582 - accuracy: 0.6939 - val_loss: 0.8604 - val_accuracy: 0.6934
Epoch 10/10
4515/4515 [==============================] - 25s 5ms/step - loss: 0.8583 - accuracy: 0.6937 - val_loss: 0.8602 - val_accuracy: 0.6935
```

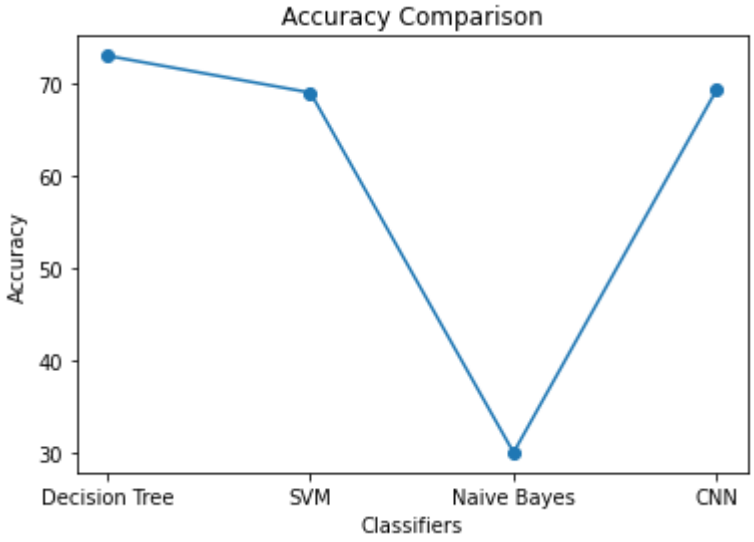
Fig 4.10 Classification report of Convolutional Neural Network

6. Comparison & Result

Let's compare the results we got from different classifiers' execution on the model.

Starting with a comparison of accuracy:

Decision Tree: 73%, SVM: 69%, Naive Bayes: 30%, CNN: 69.34%



Graph 4.1 Accuracy trends according to classifiers

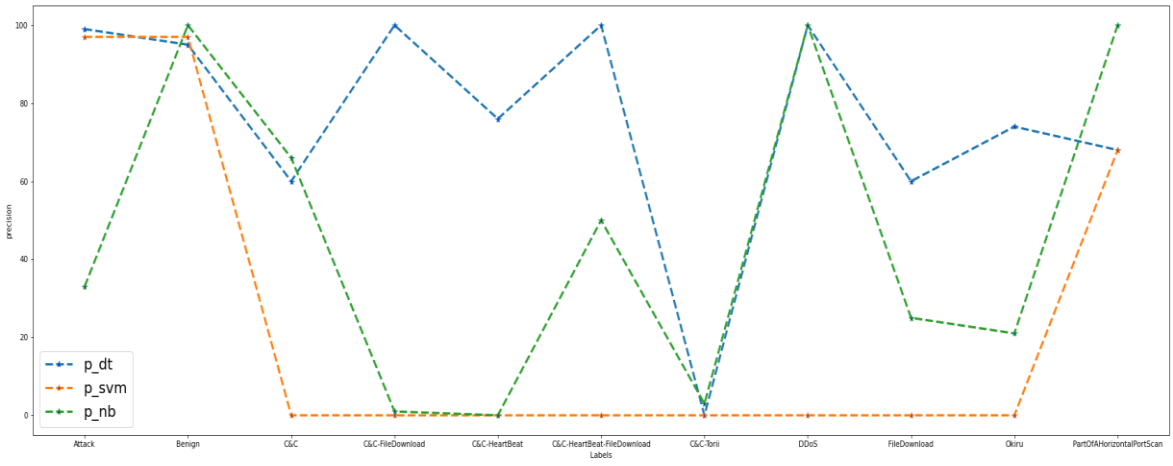
Next, let us look at the Precision of the model with reference to different classifiers and Labels.

Precision of the model measures the ability to classify positive samples in the model.

p_dt represents the precision trend for the Decision Tree.

p_svm represents the precision trend for SVM.

p_nb represents the precision trend for Naive Bayes.



Graph 4.2 Precision wrt Classifier and Labels

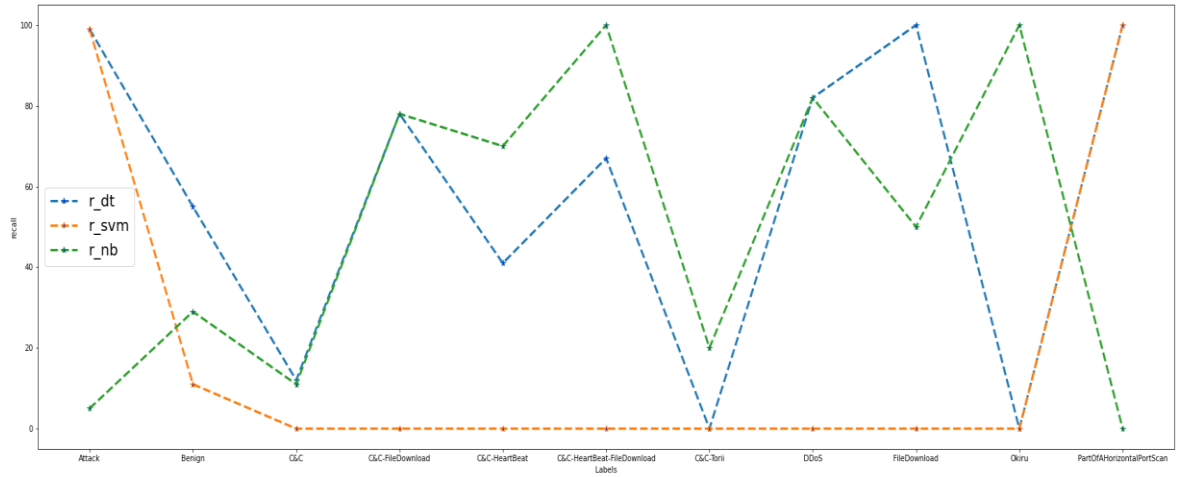
Next, let's see the Recall of the model with reference to different classifiers and Labels.

Recall of the model measures how many positive samples were correctly classified by the ML model.

r_dt represents the recall trend for the Decision Tree.

r_svm represents the recall trend for SVM.

r_nb represents the recall trend for Naive Bayes.



Graph 4.3 Recall wrt Classifier and Labels

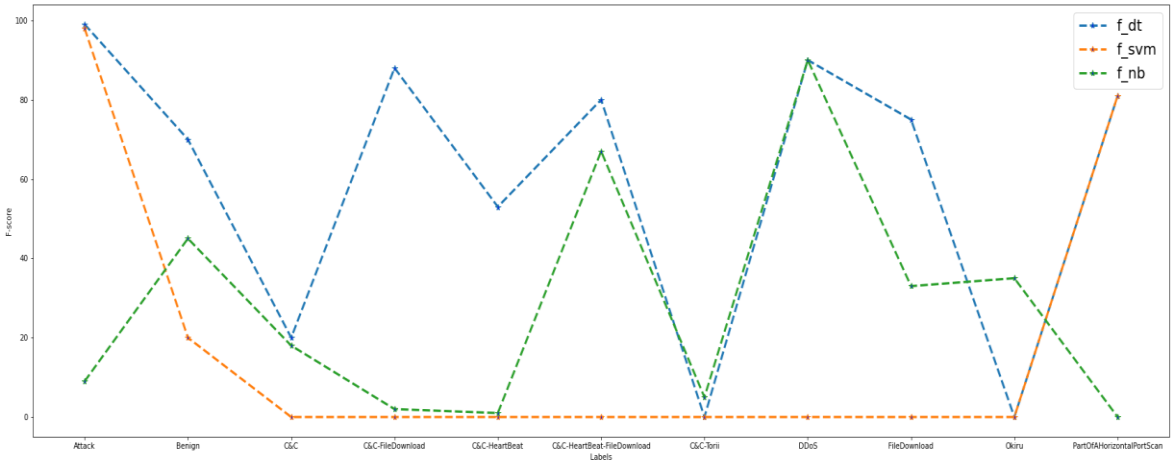
Finally, let's examine the F-1 Score of the model with reference to different classifiers and Labels.

F-1 Score of a model is the harmonic mean of precision and recall.

f_dt represents the precision trend for the Decision Tree.

f_svm represents the precision trend for SVM.

f_nb represents the precision trend for Naive Bayes.



Graph 4.4 F-1 Score wrt Classifier and Labels

4.2 Comparison of Results with Previous Theories

- In (Alshamkhany et al., 2020) the dataset used was the UNSW-NB15 dataset and its results were as follows:

Table.6 Comparison of UNSW-NB15 and IoT-23 evaluation metrics

	Decision Tree	NB Gaussian	SVM
Accuracy in UNSW-NB15	99.91%	97.10%	100%
Accuracy in IoT-23	73%	30%	69%
Precision in UNSW-NB15	100%	97%	99%
Precision in IoT-23	76%	85%	60%
Recall in UNSW-NB15	100%	97%	99%
Recall in IoT-23	73%	30%	69%
F-1 Score in UNSW-NB15	100%	97%	82%
F-1 Score in IoT-23	65%	21%	57%

The comparison of results in our project revealed that the accuracy of our model is lower compared to the performance reported in the referenced journal (Alshamkhany et al., 2020). However, it is worth mentioning that the UNSW-NB15 dataset used in our project specifically focuses on DDoS and DoS attacks, which may explain the relatively lower accuracy compared to the journal that might have used a different dataset with a broader range of attack types.

Despite the lower accuracy, our project suggests that further improvements could be made by exploring combinations of classifiers to enhance the performance of the model.

This aligns with the future work or limitation highlighted in the referenced journal (Alshamkhany et al., 2020) where a better dataset, more relevant to real-world scenarios, could potentially lead to improved model performance.

It is important to consider the context of the dataset used in the project, and how it may impact the model's performance. While the UNSW-NB15 dataset is widely used in the field of network security, it may not cover all possible attack scenarios, and using a dataset that includes a wider variety of attacks could potentially yield different results. Therefore, future research could involve exploring other datasets or real-world data to further validate and enhance the performance of the model.
In conclusion, while our model's accuracy may be lower compared to the referenced journal, it provides valuable insights into the limitations and potential areas of improvement for future research. By considering different classifiers and datasets, and addressing the specific challenges of the project, the model's performance could be further enhanced to better detect botnet attacks in real-world scenarios.

- In (Alkahtani & Aldhyani et al., 2021), the dataset used in the N-BaIoT dataset and the result comparison is as follows:

Table.7 Comparison of N-BaIoT and IoT-23

	CNN
Accuracy in N-BaIoT	80%
Accuracy in IoT-23	69.34%

Upon comparing our model with the CNN-LSTM model developed by Alkahtani and Aldhyani et al. (2021), it is evident that their model exhibits higher accuracy. However, it is important to note that the complexity of a CNN or LSTM model alone is quite significant, and the combination of both can present a formidable task.

One of the challenges with Deep Learning models, including CNNs and LSTMs, is that the research in this area is still evolving, and our understanding of these models is constantly evolving. As mentioned in the literature survey, a CNN model typically comprises multiple layers, making it impractical to retrain the model every time a modification is made. This poses difficulties in using Deep Neural Networks in real-life scenarios where changes occur frequently, as retraining such a large network can be time-consuming and resource-intensive.

While the accuracy and performance of the CNN-LSTM model may surpass our model, it is important to consider the practical challenges associated with using such complex models in dynamic environments. As the field of Deep Learning continues to advance, further research and developments may be required to fully leverage the potential of these models in real-world scenarios where changes occur frequently.

In conclusion, while the CNN-LSTM model may demonstrate superior accuracy, the practical challenges associated with the complexity of these models, including the need for frequent retraining, may pose limitations in their real-world application. Continued research and advancements in Deep Learning are necessary to fully harness the capabilities of these models in dynamic environments.

CHAPTER 5: CONCLUSION

5.1 Conclusion

The proliferation of Internet of Things (IoT) devices is projected to increase exponentially in the near future, making it crucial to ensure their security against cyber attacks such as Botnet Attacks. In this study, we have delved into the concept of IoT, explained the workings of Botnet Attacks in real-life scenarios, and highlighted the global impact of such attacks. We have also reviewed existing techniques for Botnet Attack detection and prevention, and identified their limitations and potential areas for improvement in future research. To address these limitations, we have proposed a machine learning-based model that employs classifiers to detect and classify botnets.

One of the key strengths of our study is the unique dataset we have utilized. Published as a universal dataset for botnet detection and study in 2020, it has not been processed previously and contains the best features required for accurate prediction of botnet attacks. The dataset also includes labels that indicate the type of botnet attack detected on the device, providing valuable information for effective mitigation measures.

We have employed four different machine learning classifiers for botnet detection in IoT devices: Decision Tree Classifier, Support Vector Machine with radial basis function (Non-linear kernel), Naive Bayes with Gaussian Probability, and Convolutional Neural Network. Through our experiments, we have found that the Decision Tree model exhibits the best performance with an accuracy of 73%, surpassing the other models. Theoretically, with the dataset used, our model can be leveraged to detect various types of malware activities.

As we move forward, our future work will focus on further improving the performance of our model on the dataset by exploring different classifiers and algorithms. We also plan to continually update and expand our dataset with new data to ensure its relevance and effectiveness in real-life scenarios. Additionally, we aim to validate the performance of our model on larger datasets, including the entire IoT-23 dataset, and compare its results with other universal datasets to assess its versatility and generalizability. Through these efforts, we aim to contribute to the advancement of botnet detection techniques and enhance the security of IoT devices against cyber attacks.

5.2 Future Scope

As we look towards our future work, there are several avenues for improvement and expansion of our model. Our immediate focus is on enhancing the performance of our model on the newly designed dataset. We plan to explore the use of different classifiers and combine them to create a new algorithm that may potentially yield higher accuracy in botnet detection.

To keep our dataset up-to-date and more robust, we plan to continuously improve it with new data as it becomes available. We also aim to validate the performance of our model on the entire IoT-23 dataset, which would provide a more comprehensive evaluation of its effectiveness. Additionally, we may consider incorporating other universal datasets to increase the variability in our dataset, thus making our model more adaptable to real-life scenarios.

In addition to the classifiers we have already used, such as Decision Tree, SVM, Naive Bayes, and CNN, we plan to explore the use of other classifiers such as Random Forest, kNN, and other supervised, unsupervised, and ensemble methods of machine learning. By comparing their results, we can identify the most effective classifiers and potentially create a new algorithm that leverages the strengths of multiple classifiers for improved performance.

Furthermore, we aim to test our machine learning model in real-time environments to evaluate its accuracy outside of controlled laboratory experiments. This will help us understand how well our model performs in real-world scenarios and how effectively it handles different types of threats, including both identified and unidentified ones.

In conclusion, our future work will focus on improving the performance of our model on the dataset, validating its effectiveness on larger datasets, exploring additional classifiers, and testing its performance in real-time environments. These efforts will contribute to the ongoing refinement and advancement of our botnet detection model for enhanced accuracy and practical applicability.

References

A) Book

[1] Elisan, C. C. (2012). *Malware, Rootkits & Botnets: A Beginner's Guide*. McGraw Hill LLC.

[2] Harley, D., Evron, G., Schiller, C., & Binkley, J. R. (2007). *Botnets: The Killer Web Applications*. Elsevier Science.

[3] Lee, W., Wang, C., & Dagon, D. (Eds.). (2007). *Botnet Detection-Countering the Largest Security Threat* (1st ed.). Springer New York, NY. https://doi.org/10.1007/978-0-387-68768-1

B) Journal

1. Afrifa, S., Varadarjan, V., Appiahene, P., Zhang, T., & Domfeh, E. A. (2023, February 16). Ensemble Machine Learning Techniques for Accurate and Efficient Detection of Botnet Attacks in Connected Computers. *MDPI Journal of Engineering, 2023*, *4*(1), 15. https://doi.org/ 10.6084/m9.figshare.21769658.v1. 10.3390/eng4010039
2. Alkahtani, H., & Aldhyani, T. H. H. (2021). Botnet Attack Detection by Using CNN-LSTM Model for Internet of Things Applications. *Hindawi Security and Communication Networks*, 1-23. N-BaIoT.
3. Alshamkhany, M., Alshamkhany, W., Mansour, M., Dhou, S., & Aloul, F. A. (2020, November). Botnet Attack Detection Using Machine Learning. *IEEE International Conference on Innovations in Information Technology*. UNSW-NB15. 10.1109/IIT50501.2020.9299061

4. Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., & Elovici, Y. (2018). N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders. *IEEE Pervasive Computing*, *17*(3), 12-22. N-BaIoT.
5. Nguyen, H.-T., Ngo, Q.-D., Nguyen, D.-H., & Le, V.-H. (2020). PSI-rooted subgraph: A novel feature for IoT botnet detection using classifier algorithms. *ICT Express*, *6*(2), 128-138.
6. Peng, J., Guo, Z., Fu, J., Cheng, Y., & Chen, C. (2019). Botnet Detection Model Based on Artificial Intelligence. *2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC)*, 7. 10.1109/dsc.2019.00080

Appendices

1. Dataset

The dataset used was **"Aposemat IoT-23"**. It is a labeled dataset of malicious and benign activity in IoT network traffic.

This dataset contains sub-datasets of malware activities: Malicious IoT network traffic scenario and Benign IoT traffic scenario.

#	Name of Dataset	Duration (hrs)	#Packets	#ZeekFlows	Pcap Size	Name
1	CTU-IoT-Malware-Capture-34-1	24	233,000	23,146	121 MB	Mirai
2	CTU-IoT-Malware-Capture-43-1	1	82,000,000	67,321,810	6 GB	Mirai
3	CTU-IoT-Malware-Capture-44-1	2	1,309,000	238	1.7 GB	Mirai
4	CTU-IoT-Malware-Capture-49-1	8	18,000,000	5,410,562	1.3 GB	Mirai
5	CTU-IoT-Malware-Capture-52-1	24	64,000,000	19,781,379	4.6 GB	Mirai
6	CTU-IoT-Malware-Capture-20-1	24	50,000	3,210	3.9 MB	Torii
7	CTU-IoT-Malware-Capture-21-1	24	50,000	3,287	3.9 MB	Torii
8	CTU-IoT-Malware-Capture-42-1	8	24,000	4,427	2.8 MB	Trojan
9	CTU-IoT-Malware-Capture-60-1	24	271,000,000	3,581,029	21 GB	Gagfyt
10	CTU-IoT-Malware-Capture-17-1	24	109,000,000	54,659,864	7.8 GB	Kenjiro
11	CTU-IoT-Malware-Capture-36-1	24	13,000,000	13,645,107	992 MB	Okiru
12	CTU-IoT-Malware-Capture-33-1	24	54,000,000	54,454,592	3.9 GB	Kenjiro
13	CTU-IoT-Malware-Capture-8-1	24	23,000	10,404	2.1 MB	Hakai
14	CTU-IoT-Malware-Capture-35-1	24	46,000,000	10,447,796	3.6G	Mirai
15	CTU-IoT-Malware-Capture-48-1	24	13,000,000	3,394,347	1.2G	Mirai
16	CTU-IoT-Malware-Capture-39-1	7	73,000,000	73,568,982	5.3GB	IRCBot
17	CTU-IoT-Malware-Capture-7-1	24	11,000,000	11,454,723	897 MB	Linux,Mirai
18	CTU-IoT-Malware-Capture-9-1	24	6,437,000	6,378,294	472 MB	Linux.Hajime
19	CTU-IoT-Malware-Capture-3-1	36	496,000	156,104	56 MB	Muhstik
20	CTU-IoT-Malware-Capture-1-1	112	1,686,000	1,008,749	140 MB	Hide and Seek

Fig.32 Summarised malicious IoT-23 scenario

#	Name of Dataset	HTTP	DNS	DHCP	TELNET	SSL	SSH	IRC	Not recognized by Zeek	Name
1	CTU-IoT-Malware-Capture-34-1	12	192	2	-	-	-	1,641	21,298	Mirai
2	CTU-IoT-Malware-Capture-43-1	16	204	-	-	-	-	-	67,321,589	Mirai
3	CTU-IoT-Malware-Capture-44-1	11	-	-	-	-	-	-	226	Mirai
4	CTU-IoT-Malware-Capture-49-1	19	6	1	-	-	-	-	5,410,535	Mirai
5	CTU-IoT-Malware-Capture-52-1	14	4	1	-	-	-	-	19,781,359	Mirai
6	CTU-IoT-Malware-Capture-20-1	-	592	-	-	-	-	-	2,617	Torii
7	CTU-IoT-Malware-Capture-21-1	-	1,924	-	-	-	-	-	1,362	Torii
8	CTU-IoT-Malware-Capture-42-1	33	1,680	1	-	2	-	-	2,710	Trojan
9	CTU-IoT-Malware-Capture-60-1	-	-	2	-	-	-	-	3,581,026	Gagfyt
10	CTU-IoT-Malware-Capture-17-1	4	11,902	-	-	-	-	-	54,647,949	Kenjiro
11	CTU-IoT-Malware-Capture-36-1	-	751	2	-	-	-	-	13,644,345	Okiru
12	CTU-IoT-Malware-Capture-33-1	228	80	2	-	-	-	-	54,454,281	Kenjiro
13	CTU-IoT-Malware-Capture-8-1	-	-	-	-	-	-	-	10,403	Hakai
14	CTU-IoT-Malware-Capture-35-1	36	1,479	2	-	9	-	-	10,446,261	Mirai
15	CTU-IoT-Malware-Capture-48-1	11	2	-	-	-	-	-	3,394,325	Mirai
16	CTU-IoT-Malware-Capture-39-1	14	2,308	-	-	6	538	914	73,565,201	IRCBot
17	CTU-IoT-Malware-Capture-7-1	-	7	1	-	-	-	-	11,454,706	Linux,Mirai
18	CTU-IoT-Malware-Capture-9-1	55	1,162	-	-	-	-	-	6,377,076	Linux.Hajime
19	CTU-IoT-Malware-Capture-3-1	-	1	3	-	-	5,898	6	150,195	Muhstik
20	CTU-IoT-Malware-Capture-1-1	3,238	1	1	-		-	-	1,005,507	Hide and Seek

Fig.33 Application layer breakdown of the Malicious Scenarios

#	Name of Dataset	Duration(~hrs)	#Packets	#ZeekFlows	Pcap Size	Device
21	CTU-Honeypot-Capture-7-1	1.4	8,276	139	2,094 KB	Somfy Door Lock
22	CTU-Honeypot-Capture-4-1	24	21,000	461	4,594 KB	Philips HUE
23	CTU-Honeypot-Capture-5-1	5.4	398,000	1,383	381 MB	Amazon Echo

Fig.34 Summarised Benign scenario

#	Name of Dataset	HTTP	DNS	DHCP	TELNET	SSL	SSH	IRC	Not recognized by Zeek	Device
1	CTU-Honeypot-Capture-7-1	-	54	15	-	1	-	-	60	Somfy Door Lock
2	CTU-Honeypot-Capture-4-1	54	191	2	-	-	-	-	205	Philips HUE
3	CTU-Honeypot-Capture-5-1	157	521	156	-	86	-	-	454	Amazon Echo

Fig.35 Application layer breakdown of the Benign scenario

Table.8 Label configuration file

Id	Field	Bro field number	Data	Comparator	Label	Type
1	id.resp_h	5	176.32.33.171	eq	C&C	Malicious
2	id.resp_p	6	37215	eq	Okiru	Malicious
3	id.resp_p	6	666	eq	HeartBeat	Malicious
4	id.resp_p	6	80	eq	PartOfAHorizontalPortScan	Malicious
5	conn_state	12	S0	eq	PartOfAHorizontalPortScan	Malicious
6	id.resp_p	6	52869	eq	PartOfAHorizontalPortScan	Malicious
7	conn_state	12	S0	eq	PartOfAHorizontalPortScan	Malicious
8	id.resp_p	6	8081	eq	PartOfAHorizontalPortScan	Malicious

Link to the dataset files:

https://mcfp.felk.cvut.cz/publicDatasets/IoT-23-Dataset/IndividualScenarios/CTU-IoT-Malware-Capture-34-1/

2. Code Snippets

```
!pip install pandas --quiet
!pip install sklearn --quiet
!pip install numpy --quiet
!pip install seaborn --quiet
!pip install matplotlib --quiet
```

Fig.36 Imported Libraries

```
capture_34 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-34-1/bro/conn.log.labeled"
capture_43 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-43-1/bro/conn.log.labeled"
capture_44 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-44-1/bro/conn.log.labeled"
capture_49 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-49-1/bro/conn.log.labeled"
capture_52 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-52-1/bro/conn.log.labeled"
capture_20 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-20-1/bro/conn.log.labeled"
capture_21 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-21-1/bro/conn.log.labeled"
capture_42 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-42-1/bro/conn.log.labeled"
capture_60 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-60-1/bro/conn.log.labeled"
capture_17 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-17-1/bro/conn.log.labeled"
capture_36 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-36-1/bro/conn.log.labeled"
capture_33 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-33-1/bro/conn.log.labeled"
capture_8 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-8-1/bro/conn.log.labeled"
capture_35 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-35-1/bro/conn.log.labeled"
capture_48 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-48-1/bro/conn.log.labeled"
capture_39 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-39-1/bro/conn.log.labeled"
capture_7 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-7-1/bro/conn.log.labeled"
capture_9 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-9-1/bro/conn.log.labeled"
capture_3 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-3-1/bro/conn.log.labeled"
capture_1 = "opt/Malware-Project/BigDataset/IoTScenarios/CTU-IoT-Malware-Capture-1-1/bro/conn.log.labeled"
```

Fig.37 Used dataset

```
df_c=pd.concat(frames)
```

Python

```
df_c
```

Python

	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	...	conn_state	local_orig	local_resp	missed_bytes	histc
0	1525879832.01624	CDe43c1PtgynajGl6	192.168.100.103	60905.0	131.174.215.147	23.0	tcp	-	2.998796	0	...	S0	-	-	0.0	
1	1525879832.024985	CJaDcG3MZzvf1YVYl4	192.168.100.103	44301.0	91.42.47.63	23.0	tcp	-	-	-	...	S0	-	-	0.0	
2	1525879832.044975	CMBrup3BLXivSp4Avc	192.168.100.103	50244.0	120.210.108.200	23.0	tcp	-	-	-	...	S0	-	-	0.0	
3	1525879833.016171	CfHl9r3XMYtDQRrHnh	192.168.100.103	34243.0	147.7.65.203	49560.0	tcp	-	2.998804	0	...	S0	-	-	0.0	
4	1525879833.044906	C7USrA15nFVkniMqC5	192.168.100.103	34840.0	145.164.35.6	21288.0	tcp	-	-	-	...	S0	-	-	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99994	1532526102.004508	CMeH6R2aua5c5Dd65a	192.168.100.111	41762.0	221.182.209.127	23.0	tcp	-	-	-	...	S0	-	-	0.0	
99995	1532526102.00451	CvqGx33hsXDpDVXa1i	192.168.100.111	58758.0	208.50.139.48	23.0	tcp	-	-	-	...	S0	-	-	0.0	
99996	1532526102.004511	CC83RoUd9RLFuTL81	192.168.100.111	40400.0	40.95.136.51	23.0	tcp	-	-	-	...	S0	-	-	0.0	
99997	1532526102.004752	C4ISld2cuSukEEuQtk	192.168.100.111	27117.0	122.37.183.236	23.0	tcp	-	-	-	...	S0	-	-	0.0	
99998	1532526102.004756	C4U1azYmDx32faVY7	192.168.100.111	23227.0	189.62.234.179	23.0	tcp	-	-	-	...	S0	-	-	0.0	

Fig.38 Dataset Pre-processing

```
df_c['label'].value_counts()
```

```
-   Malicious   PartOfAHorizontalPortScan      578916
(empty)   Malicious   PartOfAHorizontalPortScan   247023
-   Malicious   Okiru                          163016
-   Benign   -                                 146275
-   Malicious   DDoS                           138777
(empty)   Malicious   Okiru                    99674
(empty)   Benign   -                           51534
(empty)   Malicious   C&C                      8229
-   Malicious   C&C                            6871
(empty)   Malicious   Attack                   3814
-   Malicious   C&C-HeartBeat                  227
(empty)   Malicious   C&C-HeartBeat            122
-   Malicious   Attack                         101
-   Malicious   C&C-FileDownload               43
-   Malicious   C&C-Torii                      30
-   Malicious   FileDownload                   13
-   Malicious   C&C-HeartBeat-FileDownload     8
-   Malicious   C&C-Mirai                      1
Name: label, dtype: int64
```

Fig.39 Total labeled malicious attack

```
df['label'].value_counts()
```

```
PartOfAHorizontalPortScan      825939
Okiru                          262690
Benign                         197809
DDoS                           138777
C&C                            15100
Attack                         3915
C&C-HeartBeat                  349
C&C-FileDownload               43
C&C-Torii                      30
FileDownload                   13
C&C-HeartBeat-FileDownload     8
C&C-Mirai                      1
Name: label, dtype: int64
```

Fig.40 Label count for Decision Tree

```
from sklearn import tree
clf_tree=DecisionTreeClassifier()
clf_tree.fit(X_train,Y_train)
fig,ax=plt.subplots(figsize=(15,20))
tree.plot_tree(clf_tree,fontsize=2.5)
plt.show()
```

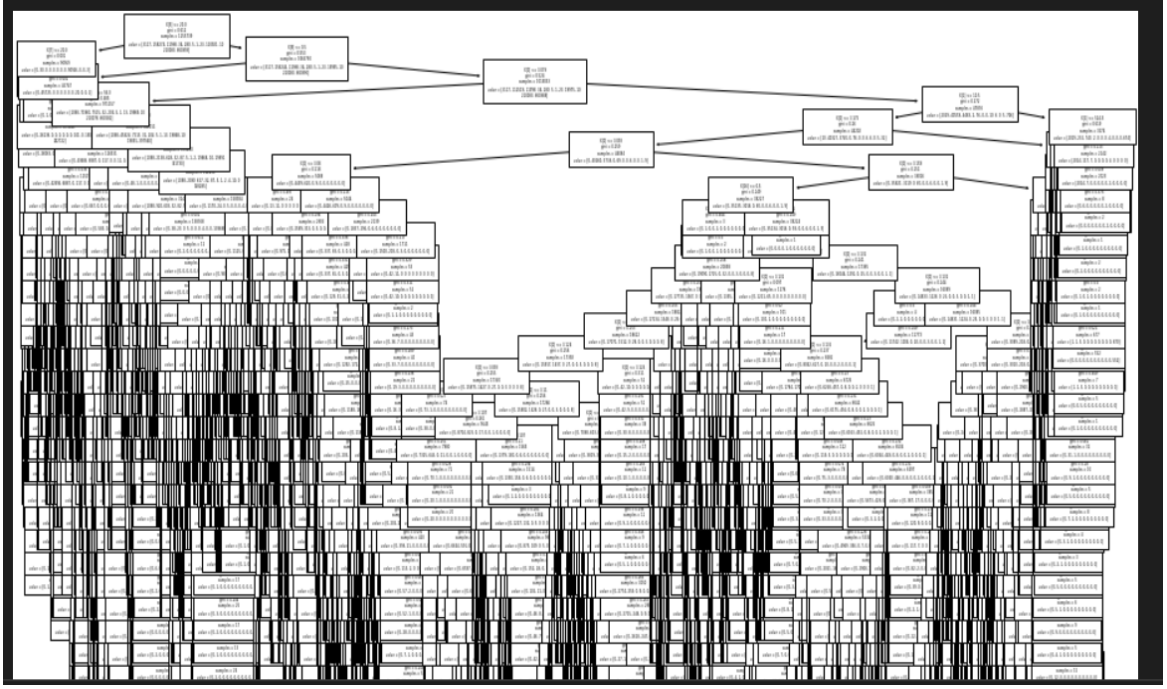


Fig.41 Decision Tree Classifier visualization

```
df['label'].value_counts()
```

PartOfAHorizontalPortScan	825939
Okiru	262690
Benign	197809
DDoS	138777
C&C	15100
Attack	3915
C&C-HeartBeat	349
C&C-FileDownload	43
C&C-Torii	30
FileDownload	13
C&C-HeartBeat-FileDownload	8
C&C-Mirai	1

Name: label, dtype: int64

```
start = time.time()
print('program start...')
print()

clf = GaussianNB().fit(X_train, Y_train)
print()
print(clf.score(X_test, Y_test))
print()

y_pred = clf.fit(X_train, Y_train).predict(X_test)
print(y_pred)
print()

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')
```

```
program start...


0.30113001194040184

['Okiru' 'Okiru' 'Okiru' ... 'Okiru' 'Okiru' 'Okiru']

program end...

time cost:
5.830923318862915 seconds
```

Fig.42 Label count for GNaive Bayes

Fig.43 Model score for GNaive Bayes

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
normalized_x = scaler.transform(X)
normalized_x
```

```
array([[8.16450401e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

Fig.44 Data-preprocessing for GNaive Bayes

```
df['label'].value_counts()
```

```
PartOfAHorizontalPortScan    265875
Benign                        56943
Okiru                         48352
DDoS                          24959
Attack                         3814
C&C-Torii                        30
C&C-HeartBeat                    20
C&C                               7
Name: label, dtype: int64
```

Fig.45 Label count for SVM

```
start = time.time()
print('program start...')
print()

SVM_classifier = SVC(C=1.0, cache_size=1500, verbose=True).fit(X_train, Y_train)
print()
print(SVM_classifier.score(X_test, Y_test))
print()

y_pred = SVM_classifier.predict(X_test)
print(y_pred)
print()

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')
```

```
program start...

[LibSVM]
0.6879875

['PartOfAHorizontalPortScan' 'PartOfAHorizontalPortScan'
 'PartOfAHorizontalPortScan' ... 'PartOfAHorizontalPortScan'
 'PartOfAHorizontalPortScan' 'PartOfAHorizontalPortScan']

program end...

time cost:
5849.032120704651 seconds
```

Fig.46 Model score for SVM

```
df['label'].value_counts()
```

```
PartOfAHorizontalPortScan    825939
Okiru                        262690
Benign                       197809
DDoS                         138777
C&C                           15100
Attack                         3915
C&C-HeartBeat                   349
C&C-FileDownload                 43
C&C-Torii                        30
FileDownload                     13
C&C-HeartBeat-FileDownload        8
C&C-Mirai                         1
Name: label, dtype: int64
```

Fig.47 Label count for CNN

```
scaler = MinMaxScaler()

scaler.fit(X)
```

```
MinMaxScaler()
```

```
normalized_x = scaler.transform(X)

normalized_x
```

```
array([[8.16450401e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.04174057e-05, 5.73121586e-10, 8.57558209e-08, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

Fig.48 Scaling & Normalization for CNN

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2000)              50000
_________________________________________________________________
dense_1 (Dense)              (None, 1500)              3001500
_________________________________________________________________
dropout (Dropout)            (None, 1500)              0
_________________________________________________________________
dense_2 (Dense)              (None, 800)               1200800
_________________________________________________________________
dropout_1 (Dropout)          (None, 800)               0
_________________________________________________________________
dense_3 (Dense)              (None, 400)               320400
_________________________________________________________________
dropout_2 (Dropout)          (None, 400)               0
_________________________________________________________________
dense_4 (Dense)              (None, 150)               60150
_________________________________________________________________
dropout_3 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_5 (Dense)              (None, 12)                1812
=================================================================
Total params: 4,634,662
Trainable params: 4,634,662
Non-trainable params: 0
_________________________________________________________________
```

Fig.49 Model summary of CNN