WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

IoT Botnet Detection using Deep Learning and Machine Learning Techniques for Network Traffic Analysis

Tanusree Reddy Gavinnolla[1], Shashank Aluru[2], Sai Vaibhav Medavarapu[3], Santhoshi R[4], Abhinav Goyal[5], Kommaraju Manasa Chowdary[6], Balamurugan K[7], Harsh Garg[8].

CVR College of Engineering[1], Vellore Institute of Technology, Amaravati, India[2], JNTUH[3], PSG College of Technology[4], BITS Pilani, Dubai Campus[5], Gandhi Institute of Technology and Management, Hyderabad (GITAM)[6], Rajalakshmi Institute of Technology[7], Delhi Technological University[8].

ABSTRACT:

The growth of the Internet of Things (IoT) has presented significant challenges for Botnet intrusion detection systems (IDS). IDS, which operates primarily at the network layer of IoT architecture, plays a crucial role in securing these networks. A botnet in the IoT context consists of a network of compromised devices (such as routers) that have been infected with malware and are controlled by a botmaster. Detecting and mitigating botnet attacks in IoT systems is challenging, particularly in maintaining high detection accuracy while reducing false alarms, especially as botnet attacks become more prevalent. These attacks can take various forms, from spam attacks to Distributed Denial of Service (DDoS), initiated by numerous infected devices. To address this, the botnet detection system first collects traffic data, converts this information into useful features, and applies deep learning techniques to identify malicious activity from compromised IoT devices. This paper proposes an IDS that can distinguish between normal and attack traffic in botnet IoT networks. It employs classification techniques, including decision tree and random forest machine learning algorithms, alongside a one-dimensional convolutional neural network (CNN) for botnet detection.

Introduction :

This Section introduces the key concepts related to our analysis.

IoT - Internet of Things

Smart devices are increasingly interconnected, with each device communicating with other connected systems in environments designed to automate both household and industrial tasks. These devices provide users with valuable sensor data, benefiting individuals, companies, and other targeted entities. These devices are grouped into four categories: consumer, business, and commercial.

In industrial settings, IoT devices are commonly used in factories and manufacturing units. Most IoT devices, particularly sensors, are deployed to monitor production lines and other industrial operations. Data from various sensors is transmitted to applications, ensuring that critical processes run smoothly. These sensors can also predict changes in components, helping to prevent unnecessary losses.

Botnet

A botnet is a network of interconnected devices, including personal computers, servers, handheld platforms, and IoT devices, that are infected and controlled by a single piece of malware, often without the device owner's knowledge. These compromised devices are remotely controlled by malicious attackers, such as cybercriminals, and are used for various tasks, although these harmful activities remain hidden from the user.

In 2020, botnets accounted for 25.6% of all website traffic, marking an increase of 6.2% compared to the previous year, according to the Imperva Bad Bot Report for 2021. Even more concerning, advanced persistent bots were responsible for 57.1% of all problematic bot traffic in 2020. This trend indicates that the use of botnets by cybercriminals has become increasingly sophisticated. Fraudsters employ botnets in various ways depending on the nature of the business.

Botnet in IoT

To control botnets, malicious actors use a management and control server to infect connected devices with malware. Once an intruder gains access to a device, all vulnerable devices within the network are at risk of being compromised. Some well-known active botnets include:

Mirai

The Mirai botnet remains active, according to a Fortinet analysis released in August 2018. Mirai is one of the most active botnets, particularly in the second half of that year. Since the release of its source code two years prior, the Mirai botnet has developed additional capabilities, such as turning compromised devices into malware proxies and crypto miners. As reported by Fortinet, Mirai continues to target both known and undisclosed vulnerabilities, expanding its range of attacks.

Echobot

Echobot, identified in early 2019, is a variant of the Mirai botnet, spreading through at least 26 vulnerabilities. Like other botnets, it exploits unpatched IoT devices, but it also targets flaws in enterprise applications such as Oracle WebLogic and VMware SD-WAN. Echobot was recognized by Palo Alto Technologies, which concluded that its main objective is to assist larger botnets in executing more powerful Distributed Denial of Service (DDoS) attacks.

Emotet, Gamut, and Necurs

The primary purpose of the Emotet, Gamut, and Necurs botnets is to flood networks with large volumes of junk traffic in a short time, delivering harmful payloads or convincing victims to take specific actions. Cisco's "Email: Click with Caution" study indicates that each of these botnets has its own specialized method of attack. Emotet, Gamut, and Necurs are known for their significant role in distributing spam emails and ransomware.

Intrusion Detection System

Botnets are typically managed by a centralized command server. Disrupting this server and tracing the resulting traffic to compromised devices should theoretically be straightforward. An Intrusion Detection System (IDS) monitors network traffic for any unusual behavior and notifies administrators upon detection. IDSs analyze devices for malicious activities or policy violations. Alerts are sent to administrators or centralized security event management (SIEM) systems, which aggregate data from multiple sources and use filtering mechanisms to highlight potential threats.

Machine Learning in IoT Botnet Detection

Several machine learning algorithms, including K-Nearest Neighbor, Naive Bayes, and Multilayer Perceptron Artificial Neural Network, have been used to develop models for IoT botnet detection, trained on the BoT-IoT dataset. The best technique was selected based on accuracy and the area under the receiver operating characteristic curve (AUC). These machine learning methods were integrated with feature engineering and synthetic minority oversampling to address class imbalances in datasets. The performance of these techniques was compared across both class-imbalanced and balanced datasets.

Deep Learning in IoT Botnet Detection

Deep Learning (DL) has gained significant traction and is used to create sophisticated models by incorporating multiple data processing layers into a hierarchical structure. DL's capability to automatically discover optimal features from raw data through successive nonlinear transformations is a key aspect of its success. This technology has been effectively applied in various sectors, including research, industry, and government, showing highly promising results across a wide range of applications.

Scope of the Research

The rapid proliferation of IoT devices has led to increased vulnerability to malicious third-party attacks. To address these challenges, robust security measures, including network intrusion detection systems (IDS) and network forensic systems, need to be effectively developed. A well-structured and representative dataset is essential for training and validating these systems' credibility. Although many network datasets exist, they often lack sufficient data on Botnet scenarios. To overcome this gap, Nickolaos Koroniotis et al. proposed the Bot-IoT dataset, which includes both legitimate and simulated IoT network traffic, along with various attack types. This dataset also features a realistic testbed environment designed to address the limitations of capturing comprehensive network data, accurate labeling, and incorporating recent and complex attack types. The Bot-IoT dataset has been assessed using various statistical and machine learning methodologies to ensure its reliability against benchmark datasets. This work lays the groundwork for botnet detection in IoT-specific networks. In this Research, the goal is to implement machine learning and deep learning models to detect botnet attacks through IoT network traffic. Several previous studies have used machine learning and deep learning algorithms on the Bot-IoT dataset to tackle the detection problem.

Problem Statement

With the significant increase in IoT device usage, ensuring the security of these devices is crucial. A large and practical dataset of synthetic IoT-based data provides ample opportunities for enhancing botnet detection security in IoT devices before real-world issues arise. Therefore, the problem statement for this Research is: *"To develop a deep learning-based intrusion detection system (IDS) for detecting botnet attacks in the Internet of Things networks."*

SECTION 2: LITERATURE REVIEW :

In this Section, various implementations and research work related to IoT botnet intrusion detection systems are discussed.

Implementation of Botnet Detection

Detection of IoT Botnet Attacks using Universal Features Set

An approach towards detecting IoT botnet attacks involved the use of a graphical analysis model. In this method, attributes from PSI graphs were analyzed to detect malware on IoT platforms. A total of 12 graph-theoretic features were extracted from the designed PSI graph. These features were then used to represent different attack types. Machine learning and deep learning models were applied to match the labeled attacks from the dataset, resulting in high detection accuracyrds a Graph-Theoretic Method for IoT Botnets Detection**

This approach also focused on using a graphical analysis model for detecting botnet attacks in IoT networks. By extracting 12 graph-theoretic features from PSI graphs, the system was able to analyze malware attacks on the IoT platform. These extracted features were used to represent different types of attacks, which were subsequently matched with the labeled attacks from the dataset. The study found that machine learning and deep learning methods could be successfully applied to detect IoT botnet attacks with good accuracy

Machine Learning in IoT Botnet Detection

In the domain of IoT botnet detection, machine learning (ML) plays a crucial role in identifying anomalies and detecting attacks. Various machine learning algorithms have been proposed to improve intrusion detection systems (IDS) for IoT networks. Below are some of the key approaches and methodologies in using machine learning for IoT botnet detection.

IoT Botnet Analysis Algorithms Learning

The development of intelligent IDS requires self-learning systems that can detect intrusions and adapt to new communication patterns. For an IDS to be effective, it needs an adequate training set to learn from. Researchers have explored six different machine learning algorithms to assess their performance in IoT botnet detection: Decision Tree, Random Forest Gini, Support Vector Machine, Logistic Regression, Random Forest Information Gain (IG), and Gaussian Naive Bayes. The study, using the Matplotlib library, visualized the accuracy of these algorithms, with Random Forest IG achieving the highest

accuracy of 92.63%, while Gaussian Naive Bayes showed the lowest accuracy of 76.63%. This highlights the effectiveness of Random Forest IG in IoT botnet detection, offering a promising solution for real-time IDS.

Supervised Detection of Internet of Things Botnet Attacks

A novel approach to anomaly detection in IoT networks involves using supervised machine learning techniques to detect zero-day attacks. Zero-day attacks are previously unknown threats that the system has not encountered before. To tackle this, researchers applied clustering algorithms to differentiate between attack types and improve detection accuracy. This technique showed promising results by enhancing the IDS's ability to detect new, unseen attacks, making it a valuable approach for securing IoT networks against evolving threats.

Unsupervised Machine Learning for IoT Botnet Detection

In addition to supervised methods, unsupervised machine learning algorithms have been explored for identifying network anomalies. These types of attacks, which are harder to detect and cause significant damage, are increasingly prevalent in IoT environments. One such approach involves using Autoencoder-based unsupervised classifiers, which have been shown to be effective in predicting network activity and detecting anomalies in industrial IoT networks. These Autoencoders outperform several supervised ML classifiers in recognizing new and unusual attacks, making them a viable option for detecting previously unknown IoT botnet attacks.

Botnet Attack Detection at the IoT Edge Based on Sparse Representation

Edge devices in IoT networks often have limited processing power and storage, which makes them challenging to secure. To address this issue, a diagnostic technique has been proposed for detecting IoT botnet attacks at the IoT edge. The goal is to quickly identify and isolate affected IoT devices, thereby limiting the impact of the attack. The approach uses a sparse representation model, where the decision threshold is set based on benign training cases only. A single hidden-layer autoencoder is employed to implement the sparse representation approach. Experimental results showed that this method outperformed other detection strategies in terms of the F1-score, demonstrating its potential for effective botnet attack detection in resource-constrained IoT devices.

Identification of IoT Botnets Based on Static and Dynamic Vector Properties

The identification of IoT botnets can be significantly enhanced by combining both static and dynamic analysis. Static analysis involves examining the executable files without running them, allowing for the retrieval of header sections and other relevant features directly from the file. This method provides insights into the malware's structure. On the other hand, dynamic analysis requires running the executable programs and observing their behavior, such as API calls, network activity, registry changes, and memory consumption logs. This method can be especially useful for detecting obfuscated or disguised malware.

Both analysis techniques come with their advantages and limitations. While static analysis excels in revealing the structure of malware, dynamic analysis is better equipped to handle malware that uses obfuscation techniques. By combining both methods, a hybrid approach can be developed that leverages the strengths of both static and dynamic analysis to improve malware detection accuracy. A recent study proposed such a hybrid approach, where the PSI graph and SCG plot features are fused into a single hybrid feature using the early fusion process. The approach was tested on a real-world dataset and produced promising results. However, the hybrid method faces challenges when dealing with obfuscated malware, particularly in extracting the PSI graph from IoT botnet samples using decompilation techniques Deep Learning in IoT Botnet Detection: A Bibliography**

Botnet Attack Detection Using Deep Learning and Network Flow

One of the approaches for detecting IoT botnet attacks involves using deep learning techniques based on network flow frameworks. This method is particularly designed to protect smart city applications, such as traffic control and water treatment systems, that rely on IoT devices. The framework can handle large volumes of data in a Big Data environment. Deep learning methods outperform traditional machine learning algorithms in this context, as they allow for the analysis of the complete payload data, differentiating between legitimate and botnet attack traffic. This deep learning approach is efficient and provides exceptional performance in detecting IoT botnet activities.

Intrusion Detection Solution for IoT Botnet Attacks*

A comparison of different deep learning algorithms for identifying various IoT network attacks caused by botnets has revealed that Convolutional Neural Networks (CNNs) are the most effective at intrusion detection. CNNs outperformed other deep learning methods such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRUs) in terms of accuracy, prediction time, and lower loss rates. The research demonstrated that CNNs offer higher accuracy and reliability in detecting IoT botnet attacks. The study concludes that CNNs hold significant promise for autonomous intrusion detection systems in IoT environments .

IoT Intrg LSTMs-AE-Based Deep Learning

In large-scale enterprise environments with numerous IoT devices connected via Wi-Fi, deep learning faces several challenges, such as handling the continuous flow of big data. In response to these challenges, researchers proposed a solution based on Long Short-Term Memory (LSTM) networks combined with autoencoders (LSTMs-AE) for online IoT intrusion detection. The model processes temporal features of the data, with the autoencoder learning to minimize reconstruction errors and identify outliers. The approach was tested on an IoT dataset with attack data from the Kitsune team and demonstrated high accuracy, F1 scores, and good performance on the AUC (Area Under Curve) indicator. However, the performance of the system is affected when there are too few negative samples in the dataset .

N-BaIoT (Network-Based IoT tification Using Deep Autoencoders)

Given the increasing number of IoT devices within organizational networks, the challenge of detecting botnet attacks becomes more complex. To address this, researchers proposed N-BaIoT, a network-based solution that uses deep learning techniques to identify botnet attacks. N-BaIoT leverages deep autoencoders trained on statistical data from benign IoT traffic to understand each device's normal behavior. The autoencoders are used to detect deviations from typical patterns, indicating the presence of botnet activities. This method provides a promising centralized and automated solution for

detecting compromised IoT devices in large networks .

Proposed System :

In the proposed system, we aim to enhance IoT botnet detection through an advanced approach combining deep learning and machine learning techniques. The primary focus is to create an autonomous intrusion detection system (IDS) capable of effectively identifying and differentiating between various types of IoT network attacks. The system's performance is evaluated in terms of prediction time, loss rate, accuracy, and F1 score, ensuring that it not only detects botnet activities but also does so efficiently and with minimal false positives.

Deep Learning Approach: 1D CNN vs 2D CNN

The proposed system utilizes Convolutional Neural Networks (CNN), a powerful deep learning technique, to detect IoT botnet attacks. A notable aspect of the design is the choice of 1D CNN over 2D CNN for this application. While 2D CNNs are widely used in image processing tasks, 1D CNNs have shown superior performance for sequential data such as network traffic. In this context, the 1D CNN is better suited to handle the nature of IoT traffic data, which is typically represented in a linear sequence of network packets or flow features. The key advantage of using 1D CNNs lies in their ability to efficiently process time-series data or other sequential data formats that describe network behavior, which is more typical in botnet detection tasks. By focusing on detecting patterns in the temporal sequence of network traffic, the 1D CNN provides better accuracy, faster convergence, and less computational overhead compared to a 2D CNN.

The 1D CNN architecture is designed to capture the relationships between different features in a single time-series or feature vector, allowing it to focus on the sequential patterns that emerge in attack scenarios. This decision is crucial as the nature of IoT botnet attacks often involves subtle, sequential changes in network traffic, making 1D CNNs particularly adept at identifying such patterns.

Machine Learning Classifiers for Comparison

In addition to deep learning techniques, the system also incorporates machine learning algorithms for comparison and further refinement of the detection process. Specifically, Random Forest and Decision Tree classifiers are employed to assess their effectiveness in botnet attack detection.

- **Random Forest Classifier**: Random Forest, an ensemble learning method, is used to create a set of decision trees, where each tree is trained on a random subset of the dataset. By averaging the predictions of multiple decision trees, Random Forest helps reduce overfitting and increases the robustness of the model. It is particularly well-suited for handling large and complex datasets, which is common in IoT environments where network traffic can be highly variable.
- **Decision Tree Classifier**: The Decision Tree algorithm is a simple yet powerful method for classification. It constructs a tree-like model where each node represents a decision based on the value of a feature. It works well for datasets with clear decision boundaries but may be prone to overfitting if not properly tuned. By comparing Decision Trees with Random Forests, we can evaluate whether a simpler model can offer competitive performance in terms of accuracy and detection time.

The output from these classifiers is compared based on multiple performance metrics, including accuracy, F1 score, and prediction time. The goal is to evaluate which model provides the best trade-off between precision (minimizing false positives) and recall (maximizing true positives) in the context of IoT botnet detection.

Cross-Validation for Improved Model Reliability

To ensure the robustness and generalization ability of the proposed models, we integrate cross-validation into the training process. Cross-validation is a statistical method used to estimate the skill of machine learning models on unseen data. Specifically, **k-fold cross-validation** is employed, where the dataset is divided into 'k' subsets, and the model is trained on 'k-1' subsets while the remaining subset is used for validation. This process is repeated for each fold, ensuring that every data point is used for both training and testing.

This technique helps mitigate the risk of overfitting, which is particularly important in IoT environments where attack patterns can be highly diverse and unseen in prior training data. By using k-fold cross-validation, we can obtain a more reliable estimate of model performance and avoid issues related to model overfitting or bias due to the specific data partitioning.

In addition, a **super-run approach** is employed, which optimizes the training process by tuning hyperparameters such as learning rates and regularization terms. This hyperparameter optimization ensures that the models are not only trained to maximize accuracy but also to minimize loss, improving both prediction speed and detection reliability.

Evaluation and Comparison

The final step involves comparing the performance of the proposed system against previously completed tasks and benchmarks. The evaluation focuses on multiple metrics, including:

1. **Accuracy**: Measures the percentage of correct predictions (both true positives and true negatives) made by the model. This is the primary indicator of how well the model can classify IoT traffic as either benign or malicious.
2. **F1 Score**: The F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's ability to correctly identify botnet attacks while minimizing false alarms.
3. **Prediction Time**: This metric assesses the time it takes for the model to make predictions. In IoT environments, where real-time detection is crucial, low prediction time is essential for ensuring that malicious activities are detected promptly.
4. **Loss Rate**: The loss rate quantifies how much the model's predictions deviate from the true values. A lower loss rate indicates that the model is making more accurate predictions.

System Design :

This Section outlines the implementation steps for the proposed IoT botnet detection system, detailing the data selection, model creation, and evaluation processes.

The system architecture follows a structured approach consisting of multiple stages to ensure effective botnet detection. Below is a summary of the key steps:

Step 1: Data Selection and Exploratory Data Analysis (EDA)

A subset of **5% of the dataset** (3.6 million entries) was selected from a large dataset containing over 70 million entries. The dataset includes both **normal traffic** and various attack types, including **spying**, **DDoS**, **DoS**, and **theft** attacks. EDA is performed to understand the distribution of features and attack patterns.

Step 2: Cleaning and Selecting Targets

The dataset was cleaned by removing redundant columns, converting data types to **float**, and replacing **NaN** values with zeros. It was split into a **70% training** and **30% testing** dataset. The target variable, "Attack," was selected to indicate whether the traffic is normal or malicious.

Step 3: Creating Base Model and Analysis

The base model used a **2D CNN**, which achieved an accuracy of **99.94%**. However, a **1D CNN** was tested and improved performance to **99.998%** with a loss rate of **0.56%**, indicating better suitability for time-series data like network traffic.

Step 4: Creating Classification Model

Two machine learning models were selected for comparison: **Random Forest** and **Decision Tree**. These models are known for their effectiveness in classification tasks and serve as baselines for comparison with the deep learning-based approach.

Step 5: K-Fold Cross-Validation Method

The **K-fold cross-validation** method was applied to assess the model's performance across different data splits. This technique ensures the model is robust and avoids overfitting, improving generalization.

Step 6: Accuracy Comparison with Respect to Time

One of the key challenges of deploying botnet detection systems in real time is **training time**. The proposed model's **training time** was significantly reduced compared to previous approaches, making it more feasible for **real-time IoT botnet detection**.

Implementation :

In this Section, we provide an in-depth explanation of the implementation steps followed for the proposed architecture, from data preprocessing to model evaluation. The implementation consists of three major phases: **data cleaning and preprocessing**, **base model selection and training**, and **hyperparameter tuning and validation**. After implementing the model, we evaluate its performance using a testing dataset, and perform comparison with previous work to assess its efficiency.

First Stage Implementation

The first stage focuses on preparing the dataset for the machine learning model. This step involves cleaning the raw data, handling missing values, and encoding categorical variables to make the dataset compatible with machine learning algorithms.

Data Selection and Cleaning

Since the dataset used for the Research was very large, containing over **70 million entries**, we decided to work with only **5%** of the data, which amounted to **3.6 million entries**. The dataset was a mixture of normal network traffic and attack traffic, with the following attack types:

- **Spying**: Service analysis and OS fingerprinting
- **DDoS**: TCP, UDP, and HTTP
- **DoS**: TCP, UDP, and HTTP
- **Theft**: Keylogging and data exfiltration

This selection ensured that we had a balanced representation of both normal and malicious traffic.

Next, we removed irrelevant and redundant columns from the dataset, such as **state number**, **protocol number**, and **flag count**. These columns were deemed unnecessary for the prediction of botnet attacks. Additionally, miscellaneous columns, including **sadder** and **dadder**, as well as targets like **category** and **subcategory**, were removed to avoid unnecessary noise in the data. The cleaned dataset was left with **39 columns**, which were relevant to the model's predictions.

Handling Missing Data

Before proceeding with the analysis, we checked for any missing (NaN) values in the dataset. Missing data can adversely affect the performance of machine learning models, so all **NaN values** were replaced with **zeros**. This ensured that the model was trained on complete, consistent data.

Encoding Categorical Data

One critical step in preparing the dataset was encoding categorical variables. The dataset included columns with categorical data that could not be directly processed by machine learning algorithms, so they had to be converted into numerical representations. To achieve this, we applied **one-hot encoding**.

One-Hot Encoding :

One-hot encoding is a technique that transforms categorical variables into binary vectors, where each unique category value is represented as a separate column. In this method, for each category of a feature, a new binary column is created. A value of **1** indicates that the instance belongs to that category,

and a value of **0** indicates that it does not.
For instance, if the **protocol type** feature contains values like **TCP**, **UDP**, and **HTTP**, one-hot encoding would create three new columns: **TCP**, **UDP**, and **HTTP**. Each row will have a **1** in the column corresponding to the protocol type it represents, and **0** in the other columns.
This encoding was applied to all relevant categorical columns, ensuring that the dataset could be processed by the deep learning model.
Finally, after applying one-hot encoding, all **object data type columns** were converted to **float data types**, making the dataset ready for machine learning algorithms.

Second Stage Implementation

Once the data was cleaned and preprocessed, the next phase of implementation involved splitting the dataset into training and testing subsets, selecting a base model, and performing hyperparameter tuning.

Splitting the Dataset

The first step in the second stage was splitting the dataset into two portions: a **training set** (77% of the data) and a **testing set** (33% of the data). This split allows the model to be trained on a large portion of the data while reserving a separate portion for testing the model's performance.
Once the dataset was split, we divided the data into features (**X**) and the target label (**Y**). The **target label** in this case was the **attack column**, which indicates whether the traffic is a botnet attack or normal traffic.

Model Training and Hyperparameter Tuning

To optimize the model, hyperparameter tuning was conducted to ensure the best performance. Hyperparameters, such as the learning rate, number of layers, and activation functions, were selected based on experimentation and iterative optimization.
To accomplish this efficiently, we used **K-fold cross-validation**.

K-Fold Cross-Validation :

K-fold cross-validation is a powerful technique that combines training and validation in a systematic way. The dataset is divided into **K subsets** or "folds." The model is trained on **K-1** of the folds and validated on the remaining fold. This process is repeated **K times**, each time with a different fold used for validation.
This technique is preferred over a simple train-test split because it ensures that every data point is used for both training and validation, which results in a more reliable estimate of model performance. Moreover, it reduces the risk of overfitting and ensures the model is robust and generalizes well to unseen data.
In our implementation, we used **5-fold cross-validation**. This method not only tunes the hyperparameters but also evaluates the model's performance across different subsets of the data, leading to a more stable and accurate performance estimate.

Third Stage Implementation

In the final phase of the implementation, we selected the best-performing model, evaluated its performance on the test data, and compared it to other models and previous work to assess its efficiency and effectiveness in real-world botnet detection scenarios.

Base Model Selection: Convolutional Neural Networks (CNN)

Based on prior research, **Convolutional Neural Networks (CNNs)** were chosen as the base model for attack detection. CNNs are effective in image-like data, but their application to time-series data or sequential data (such as network traffic) has shown promising results as well. In our case, a **1D CNN** was used, as it is more effective than a **2D CNN** for sequential or time-series data.
The model was trained using the training dataset, and its accuracy was measured. After applying the 1D CNN architecture, we achieved a remarkable **accuracy of 99.998%** and a **loss rate of 0.56%**. This result was very satisfying, as it significantly outperformed the 2D CNN model used in previous studies, which had achieved an accuracy of **99.94%**.

Model Evaluation

To evaluate the model's performance on the test data, we used **several metrics**, including:

- **Accuracy**: The percentage of correct predictions made by the model.
- **Precision and Recall**: These metrics help evaluate the model's ability to detect botnet attacks (true positives) and avoid false positives and false negatives.
- **F1 Score**: The harmonic mean of precision and recall, providing a balanced measure of the model's performance.

The final model showed excellent results, achieving high accuracy, precision, recall, and F1 score, confirming its effectiveness in detecting botnet attacks in IoT networks.

Hyperparameter Tuning with Grid Search :

To further improve the performance of the CNN model, **grid search** was employed to perform an exhaustive search over the hyperparameter space. This process involved testing different combinations of hyperparameters, such as the number of layers, learning rate, and filter size in the CNN, to find the optimal configuration that yielded the best performance on the test data.
The grid search was combined with **K-fold cross-validation**, ensuring that the model was evaluated thoroughly across different data subsets.

Results and Comparison :

Once the final model was trained and tuned, its performance was evaluated and compared to previous approaches for botnet detection in IoT networks. The results showed that our model outperformed prior models in terms of accuracy, training time, and loss rate.
Moreover, the model showed promise for real-time IoT botnet detection, addressing one of the main challenges in botnet detection: **the time required for model training and inference**. With an optimized training process, the proposed solution is better suited for real-time applications.

Algorithm for Implementation

The procedure followed in this Research to implement the machine learning models is outlined below. The algorithm involves dataset splitting, training, testing, and model evaluation.

1. **Mix the Occasional Dataset:**
    - Start by mixing the dataset randomly to ensure that the model does not learn from any specific sequence of data, leading to better generalization.
2. **Split the Dataset into X Groups:**
    - Divide the dataset into **X** number of subsets (or folds) for cross-validation. This enables us to evaluate the model's performance on multiple partitions of the data.
3. **For Each Unique Group:**
    - For each unique group (or fold):
        1. Take the current group as the **test dataset**.
        2. The remaining data (from other groups) is used as the **training dataset**.
        3. Fit the model on the training dataset.
        4. Test the model on the test dataset and record the evaluation score.
4. **Analyze the Model's Performance:**
    - After evaluating the model on each fold, calculate the overall performance metrics (e.g., accuracy, precision, recall) by averaging the results obtained from each fold.
5. **Data Sample Utilization:**
    - Importantly, every observation in the dataset is assigned to a distinct group and remains in that group throughout the process. This ensures that each data point is used once in the test set and multiple times in the training set, providing a balanced training approach.

Third Stage Implementation

In the third stage, different machine learning algorithms, including **Decision Tree**, **Random Forest**, and **1D Convolutional Neural Network (CNN)**, were employed for the model comparison. The goal was to analyze how each model performed on the dataset and compare their results.

Decision Tree Algorithm

The **Decision Tree** algorithm is one of the most fundamental machine learning algorithms used for classification and regression. It splits the data into branches based on the most significant feature(s) that maximize the homogeneity of the target variable within the sub-nodes.

Working Principle:

- The decision tree recursively partitions the dataset at each node. It uses various algorithms to select the feature and the threshold that will maximize the purity of the resulting sub-nodes.
- The process is done in a **top-down** manner, starting from the root and creating splits until stopping conditions (e.g., depth of the tree, minimum samples per leaf) are met.

Key Concepts:

- **Greedy Algorithm:** Decision trees use a greedy approach, meaning that at each step, the tree chooses the split that provides the best immediate result. This may not always lead to the optimal solution globally.
- **Homogeneity of Sub-Nodes:** The decision tree algorithm aims to create sub-nodes that are as homogeneous as possible, meaning that the instances within each sub-node should ideally belong to the same class.
- **No Backtracking:** Once a decision is made at a node, there's no backtracking. It's a straightforward decision-making process based on the current data.

Advantages:

- Simple and easy to interpret.
- Handles both numerical and categorical data well.
- Can model non-linear relationships.

Disadvantages:

- Prone to overfitting, especially with complex data and large trees.
- Sensitive to noisy data.

Random Forest Algorithm

The **Random Forest** algorithm is an ensemble method that builds multiple decision trees and merges their results to improve accuracy and reduce overfitting. The key idea behind random forests is to create an ensemble of **uncorrelated decision trees** to improve the robustness and accuracy of

predictions.

Working Principle:

- **Bootstrap Sampling (Bagging):** Each tree in the random forest is trained on a different subset of the data, obtained through **bootstrap sampling**, meaning that each tree receives a random sample of the data (with replacement).
- **Random Feature Selection:** In addition to sampling the data, random forests introduce randomness in the feature selection process. At each node, instead of considering all features, only a random subset of features is considered for splitting. This reduces the correlation between trees.

The key advantage of random forests is that they aggregate the predictions of individual trees. This collective decision-making helps to correct the errors of individual trees and prevent overfitting.

Advantages:

- Less prone to overfitting compared to a single decision tree.
- Handles large datasets well and maintains high accuracy.
- Can handle both classification and regression tasks.

Disadvantages:

- More complex and computationally expensive than decision trees.
- Less interpretable compared to a single decision tree due to the large number of trees in the ensemble.

Wisdom of Crowds:

Random forests are inspired by the **wisdom of crowds** principle, where the diversity of individual decision trees helps the overall model to make better predictions. Just like how different investors (stocks and bonds) come together to form a well-balanced portfolio, individual trees in a random forest contribute to a more accurate and generalized prediction.

1D Convolutional Neural Network (CNN)

In addition to traditional machine learning algorithms, a **1D Convolutional Neural Network (CNN)** was implemented to analyze the dataset. CNNs are widely used in image processing but have also proven to be effective in time-series analysis and sequential data problems, making them a good fit for analyzing network traffic data.

Working Principle:

- **Convolution Layers:** The 1D CNN uses convolutional filters to scan through the data, detecting patterns such as peaks, trends, and other critical features in the sequence.
- **Pooling Layers:** Pooling layers follow convolution layers and reduce the dimensionality of the data, helping the model to learn the most significant features while discarding irrelevant information.
- **Fully Connected Layers:** After the convolution and pooling layers, the output is passed through fully connected layers, which help in the final classification based on the learned features.

Advantages:

- Excellent at learning hierarchical patterns and extracting relevant features from complex data.
- Can handle noisy data well, especially in sequential datasets like network traffic.
- Capable of real-time processing if trained efficiently.

Disadvantages:

- Requires significant computational resources for training, especially on large datasets.
- Hyperparameter tuning can be time-consuming and complex.

Comparison of Models

After implementing and training the **Decision Tree**, **Random Forest**, and **1D CNN** models, we compared their performance on the test dataset. The models were evaluated based on:

- **Accuracy:** The percentage of correctly predicted instances.
- **Precision and Recall:** These metrics were used to evaluate how well the models identified botnet attacks and distinguished them from normal traffic.
- **F1 Score:** The harmonic mean of precision and recall, providing a balance between the two.
- **Training Time:** The amount of time taken to train the models.

The results indicated that while the **Random Forest** model performed excellently with a high accuracy rate, the **1D CNN** model surpassed it in terms of precision and recall, achieving better overall performance in detecting botnet attacks.

The **Decision Tree**, although simple and interpretable, performed less effectively compared to the other two models, especially in terms of generalization and accuracy.

Convolutional Neural Network - 1D Array

In this Research, we used a **1D Convolutional Neural Network (CNN)** to analyze the dataset. A **1D CNN** is a type of deep learning model that is particularly effective for sequential data or time-series data, where the input is structured in a single dimension (such as time or any other sequential feature).

A convolution layer in a CNN for 1D data works by convolving the input sequence with filters (kernels) to extract spatial features. These filters slide over the input data, performing element-wise multiplication and summation. Each filter is applied to the entire sequence, producing feature maps that highlight

important patterns or structures within the data.

CNN 1D Architecture for the Model

The architecture for the **1D CNN** used in this Research is designed as a **sequential model**. The model consists of several layers, each serving a specific purpose in feature extraction, transformation, and classification.

CNN 1D Model Design:

- **Conv1D Layer:**
  - **64 filters**: These filters convolve with the input sequence, extracting features like trends, peaks, and patterns that may be crucial for classification.
  - **Kernel Size of 2**: The kernel size determines the number of time steps or features the filter will consider at each convolution step. A kernel size of 2 is chosen to capture local patterns in the sequence.
- **ReLU Activation Function**:
  - This activation function introduces non-linearity, allowing the model to learn more complex patterns. It replaces all negative values with zero, ensuring that the model focuses on positive features.
- **MaxPooling1D Layer:**
  - A **1x64 pool size** is used to downsample the feature maps, reducing the dimensionality of the data and highlighting the most important features from the convolution step. This helps reduce overfitting by preventing the model from learning too many irrelevant details.
- **Flatten Layer:**
  - The output from the pooling layer is flattened into a 1D array, which makes it suitable for the fully connected layers that follow.
- **Dense Layers:**
  - **Two Dense Layers**: The first dense layer has **100 units** with a **ReLU activation function**, which processes the flattened features.
  - The second dense layer is the output layer with **1 unit**, and the **Softmax activation function** is used, making it suitable for multi-class classification problems.

Hyperparameter Tuning

The training model was optimized using the **Adam optimizer**, which is a popular optimization algorithm in deep learning, known for adapting the learning rate during training and ensuring fast convergence.

Hyperparameters Tuned:

- **Learning Rate:** Set to **0.001**. The learning rate controls how much the model adjusts the weights during training. A lower value like 0.001 ensures stable and gradual training.
- **Batch Size:** Set to **30**, which determines how many samples are processed before the model updates its weights. This value was chosen based on experimentation.
- **Epochs:** Set to **3**, which is the number of times the entire dataset is passed through the model during training. Three epochs were used as a balance between performance and training time.
- **Weight Constraints:** Set to **0**, meaning no specific constraints were applied to the weights during training.

Activation Function:

- **Sigmoid Activation Function**: This function is used for binary classification tasks, but in the case of multi-class classification, **Softmax** is used in the output layer to produce probability distributions over the classes.

Model Training and Evaluation

- **Adam Optimizer**: The Adam optimizer was chosen because of its ability to adjust the learning rate during training, thus improving convergence speed. It works well with sparse gradients and noisy data, making it suitable for complex tasks like botnet detection.
- **Grid Search Technique**: To fine-tune the hyperparameters, a **grid search** technique was utilized. This method systematically evaluates different combinations of parameters (like learning rate, batch size, number of filters, etc.) and finds the optimal set that maximizes model performance.

Cross-validation:

- **Cross-validation** was used to evaluate the model. This involves splitting the dataset into multiple subsets, training the model on different folds, and evaluating the performance on the remaining fold. The final model performance was averaged across all folds to ensure generalizability.

Best Model Parameters:

After performing the grid search and cross-validation, the following hyperparameters were identified as optimal for the **1D CNN** model:

- **Batch Size**: 30
- **Number of Epochs**: 3
- **Learning Rate**: 0.001
- **Weight Constraints**: 0

These parameters helped achieve the best balance between training time and model accuracy.

Accuracy Comparison

In this section, we compare the performance of our proposed models with their respective base models. The proposed models include **DT-OWN** (Decision

Tree), **RF-OWN** (Random Forest), and **CNN1D-OWN** (1D Convolutional Neural Network). The base models are represented as **DT-BASE** (Decision Tree), **RF-BASE** (Random Forest), and **CNN2D-BASE** (2D Convolutional Neural Network), as defined in the base paper.

MODELS	ACC	PRE	F1	TRAINING TIME
DT- OWN	0.99996	0.99998	0.9163	10.402
RF- OWN	0.99999	0.99605	0.9790	1133.942
CNN 1D- OWN	0.99987	0.4999	0.9999	257.634
DT-BASE	0.99999	0.99999	0.99999	42.048
RF- BASE	0.99999	0.99999	0.99999	884.666
CNN 2D-BASE	0.99935	—	—	1395

Table : Accuracy and Time Result Comparison

Discussion:

- **Random Forest (RF-OWN)** in our model achieved the highest accuracy compared to the Decision Tree and CNN 1D models. This suggests that ensemble learning methods like Random Forest are more robust in terms of classification accuracy for this particular dataset.
- **Decision Tree (DT-OWN)** had the lowest training time, reflecting its relatively simple structure compared to the other models. It is known for being a fast and efficient algorithm but may not capture as complex patterns as Random Forest or CNN.
- **CNN 1D** took **84 seconds per epoch** to train, which was less than the **138 seconds per epoch** for the CNN 2D model in the base paper. This shows that using a 1D convolutional structure for this type of data may be computationally more efficient without sacrificing too much performance.
- **Epoch Comparison:** The base model, CNN 2D, required **15 epochs** for training, whereas our CNN 1D model only required **3 epochs**, indicating that the 1D architecture may be more efficient in capturing relevant patterns with fewer iterations.

Conclusion:

- **Random Forest** outperformed the other models in terms of accuracy.
- **Decision Tree** was the most time-efficient model but achieved lower accuracy.
- **CNN 1D** provided a good balance of accuracy and computational efficiency, with fewer epochs and faster training time than the 2D CNN model from the base paper.

Confusion Matrix

The **Confusion Matrix** is a tool used to evaluate the performance of classification models by comparing the predicted class labels with the actual labels. It helps identify the number of correct and incorrect predictions made by the model for each class.
The confusion matrix tracks the following metrics:

- **True Positives (TP):** Correct predictions where the model accurately predicts the positive class (e.g., Attack).
- **True Negatives (TN):** Correct predictions where the model accurately predicts the negative class (e.g., Non-Attack).
- **False Positives (FP):** Incorrect predictions where the model incorrectly labels a non-attack sample as an attack.
- **False Negatives (FN):** Incorrect predictions where the model incorrectly labels an attack sample as non-attack.

For this work, we used the label **"Attack"** to represent the positive class. The confusion matrix analysis helps in understanding the types of errors the model makes, which is crucial for improving the classification accuracy, especially for the **Attack** label.

Key Observations:

- A higher number of **True Positives (TP)** indicates that the model is successfully identifying the **Attack** label, which is important for security detection tasks like botnet detection.
- A lower number of **False Negatives (FN)** ensures that fewer attack instances are missed, which is critical in applications where detecting attacks is paramount.
- A lower number of **False Positives (FP)** ensures that the model is not falsely flagging normal behavior as attacks, reducing unnecessary alarms.

Conclusion :

As the proliferation of IoT devices continues to increase rapidly, the need for enhanced security measures has become paramount. The objective of this study was to detect botnet attacks on IoT devices through network frequency analysis using the Bot-IoT dataset. In previous works, the CNN 2D model demonstrated impressive accuracy, reaching 99.94%. However, our implementation of the CNN 1D model achieved even higher accuracy, with 99.998%, marking a significant improvement in detecting botnet attacks.

In addition to the higher accuracy, our CNN 1D model also exhibited a notable reduction in training time, requiring only **884 seconds** compared to the **1133 seconds** needed by the CNN 2D model. This demonstrates that the 1D convolutional model is not only more efficient in terms of performance but also faster in processing the data, making it a more practical solution for real-time applications in IoT security.

Overall, our work highlights the potential of using CNN 1D architectures for botnet detection in IoT networks, offering both superior accuracy and faster training times, which is crucial for scalable and real-time security solutions in IoT environments. The results of this study suggest that further optimization and application of deep learning models, particularly CNN 1D, could significantly enhance security measures for IoT devices against evolving threats like botnet attacks.

REFERENCES :

1. CSO Online. (n.d.). *What is a botnet? When armies of infected IoT devices attack*. Retrieved from https://www.csoonline.com/article/3510733/what-is-a-botnet-when-armies-of-infected-iot-devices-attack.html
2. Idrissi, I., Boukabous, M., Azizi, M., Moussaoui, O., El Fadili, H. (2021). Toward a deep learning-based intrusion detection system for IoT against Botnet attacks. *IAES International Journal of Artificial Intelligence*, 10(1), 110-120. https://doi.org/10.11591/ijai.v10.i1.pp110-120
3. Garg, U., Kaushik, V., Panwar, A., Gupta, N. (2021). Analysis of Machine Learning Algorithms for IoT Botnet. In *Proceedings of the 2nd International Conference for Emerging Technology (INCET)*, Belgaum, India, 21–23 May 2021; pp. 1–4.
4. Sriram, R., Vinayakumar, M., Alazab, M., KP, S. (2020). Network Flow-based IoT Botnet Attack Detection using Deep Learning. *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 189-194. https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162668
5. Hussain, F., Abbas, S.G., Fayyaz, U.U., Shah, G.A., Toqeer, A., Ali, A. (2020). Towards a Universal Features Set for IoT Botnet Attacks Detection. In *Proceedings of the 2020 IEEE 23rd International Multitopic Conference (INMIC)*, Bahawalpur, Pakistan, 5–7 November 2020; pp. 1–6. *Bot-iot, 2018*. Retrieved from https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot_iot.php
6. Ngo, Q.D., Nguyen, H.T., Tran, H.A., Pham, N.A., Dang, X.H. (2021). Toward an approach using graph-theoretic for IoT botnet detection. In *2021 2nd International Conference on Computing, Networks and Internet of Things*, pp. 1-6.
7. Alazzam, H., Alsmady, A., Shorman, A.A. (2019). Supervised Detection of IoT Botnet Attacks. In *Proceedings of the Second International Conference on Data Science, E-Learning and Information Systems, DATA '19*, Dubai, United Arab Emirates, 2–5 December 2019.
8. Bhatia, R., Benno, S., Esteban, J., Lakshman, T., Grogan, J. (2019). Unsupervised machine learning for network-centric anomaly detection in IoT. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big Data, Machine Learning, and Artificial Intelligence for Data Communication Networks*, ACM, New York, NY, USA, pp. 42–48.
9. Tzagkarakis, C., Petroulakis, N., Ioannidis, S. (2019). Botnet attack detection at the IoT edge based on sparse representation. In *2019 Global IoT Summit (GIoTS)*, IEEE, pp. 1–6.
10. Ngo, Q.D., Nguyen, H.T., Tran, H.A., Nguyen, D.H. (2020). IoT Botnet detection based on the integration of static and dynamic vector features. In *Proceedings of the ICCE 2020—2020 IEEE 8th International Conference on Communications and Electronics*, Phu Quoc Island, Vietnam, 13–15 January 2020; pp. 540–545.
11. 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.
12. Xu, Y., Tang, Y., Yang, Q. (2020). Deep Learning for IoT Intrusion Detection based on LSTMs-AE. In *Proceedings of the 2nd International Conference on Artificial Intelligence and Advanced Manufacture*, ACM, pp. 64–68.
13. Hussain, F., Abbas, S.G., Fayyaz, U.U., Shah, G.A., Toqeer, A., Ali, A. (2020). Towards a Universal Features Set for IoT Botnet Attacks Detection. In *Proceedings of the 2020 IEEE 23rd International Multitopic Conference (INMIC)*, Bahawalpur, Pakistan, 5–7 November 2020; pp. 1–6.
14. 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.