Project Report

On

**Intrusion Detection in Computer Networks Using CNN**

Thesis submitted in partial fulfillment of the requirements for the award of degree of

**Bachelor of Engineering**

In

**Computer Science and Engineering**

By

**PUNNA AKHIL 1608-20-733-086**

**MOHAMMED ABUBAKER 1608-20-733-102**

**GOKARLA NAVEEN 1608-20-733-085**

Under the guidance of

**Dr. L. K. INDUMATHI**

**Associate Professor**



**Department of Computer Science and Engineering**

**Matrusri Engineering College**

**Accredited by NBA & NAAC**

(Affiliated to Osmania University, Approved by AICTE)

Saidabad, Hyderabad-500059

2023-2024

**Department of Computer Science and Engineering**

**Matrusri Engineering College**

**Accredited by NBA & NAAC**

(Affiliated to Osmania University, Approved by AICTE)

Saidabad, Hyderabad-500059

2023-2024



**CERTIFICATE**

This is to Certify that a Project report entitled **“****Intrusion Detection in Computer Networks Using CNN**” is being submitted by Punna Akhil **(**1608-20-733-086**),** Mohammed Abubaker  **(**1608-20-733-102**),** andGokarla Naveen **(**1608-20-733-085**)** in partial fulfillment of the requirement of the award for the degree of Bachelor of Engineering in “Computer Science and Engineering”O.U., Hyderabad during the year 2023-2024 is a record of bonafide work carried out by them under my guidance. The results presented in this thesis have been verified and are found to be satisfactory.

**Project Guide H.O.D.**

**Dr. L. K. Indumathi Dr. P. Vijayapal Reddy**

**Associate Professor, Professor & Head,**

**Dept of CSE Dept. of CSE**

**External Examiner(s)**

**Department of Computer Science and Engineering**

**Matrusri Engineering College**

**Accredited by NBA & NAAC**

(Affiliated to Osmania University, Approved by AICTE)

Saidabad, Hyderabad-500059

2023-2024



**DECLARATION**

We, Punna Akhil (1608-20-733-086), Mohammed Abubaker Siddiq (1608-20-733-102), Gokarla Naveen (1608-20-733-085), hereby certify that the project report entitled “**Intrusion Detection in Computer Networks Using CNN**” is submitted in the partial fulfillment of the requirement for the award of the degree of Bachelor of Engineering in Computer Science and Engineering.

This is a record of the bonafide work carried out by us under the guidance of Dr. L. K. Indumati, Associate Professor, Matrusri Engineering College, Saidabad, Hyderabad. The Results embodied in this report have not been reproduced/copied from any source. The results embodied in this report have not been submitted to any other University or Institute for the award of any other degree or diploma.

**Punna Akhil (1608-20-733-086)**

**Mohammmed Abubaker Siddiq (1608-20-733-102)**

**Gokarla Naveen (1608-20-733-085)**

**Dept of CSE, MECS**

**ACKNOWLEDGEMENT**

This project consumed huge amount of work, research and dedication. Still implementation would not have been possible if we did not have support of my Project Guide, Project Coordinator, Head of the Department and Principal. Therefore, we like to extend our sincere gratitude to all of them.

We are grateful to our project guide **Dr. L. K. Indumathi**, Associate Professor, for provision of expertise, technical support and guidance in the implementation.

We wish to express our gratitude to project coordinators for their indefatigable inspiration, constructive criticisms and encouragement throughout this dissertation work.

We would like to express our sincere thanks to the Professor and Head of the Department, **Dr. P. Vijaya Pal Reddy**, for permitting us to do this project.

We would like to express our gratitude to **Dr. D. Hanumantha Rao**, principal of Matrusri Engineering College who permitted to carry out this project as per the academics.

We would like to thank CSE Department for providing us this opportunity to share and contribute our part of work to accomplish the project in time and all the teaching and support staff for their steadfast support and encouragement.

Nevertheless, we express our gratitude towards our families and colleagues for their kind cooperation and encouragement which helped us in completion of this project.

**Punna Akhil (1608-20-733-086)**

**Mohammmed Abubaker Siddiq (1608-20-733-102)**

**Gokarla Naveen (1608-20-733-085)**

**Dept. of CSE, MECS**

**ABSTRACT**

The widespread integration and connectivity of computer systems have become indispensable for enhancing daily operations. However, this increased interconnectivity also exposes vulnerabilities that surpass human control. To counter these weaknesses, cybersecurity measures are imperative for ensuring secure communication.

Given the ever-evolving landscape of computer networks (CN), the protection and integrity of data are paramount. The escalating sophistication of cyber threats necessitates the deployment of intrusion detection systems (IDS) to actively monitor network activities, identify malicious behaviour, and respond swiftly to mitigate potential risks.

An intrusion detection system is a security tool employed to assess the normality or abnormality of records by analysing network traffic. Its primary purpose is to prevent unauthorised access to networks. Achieving the capability to discern anomalies requires utilising a varied combination of feature selections and classifiers from machine learning techniques. IDS utilises network traffic, system logs, and other data sources to detect patterns or behaviours deviating from established norms.

**Table of Contents**

Abstract………………………………………………………………………………………v

[1 Introduction 2](#_Toc168589719)

[1.1 Motivation 3](#_Toc168589720)

[1.2 Existing systems 4](#_Toc168589721)

[1.2.1 Signature-Based Systems: 4](#_Toc168589722)

[1.2.2 Anomaly-Based Systems: 4](#_Toc168589723)

[1.2.3 Network Intrusion Detection Systems (NIDS): 4](#_Toc168589724)

[1.2.4 Host Intrusion Detection Systems (HIDS): 4](#_Toc168589725)

[1.2.5 Machine Learning-Based Systems: 4](#_Toc168589726)

[1.2.6 Hybrid Systems: 5](#_Toc168589727)

[1.2.7 Cloud-Based Intrusion Detection: 5](#_Toc168589728)

[1.3 Objectives 5](#_Toc168589729)

[1.3.1 Enhance Cybersecurity Resilience: 5](#_Toc168589730)

[1.3.2 Adaptability to Dynamic Threats: 5](#_Toc168589731)

[1.3.3 Proactive Threat Mitigation: 6](#_Toc168589732)

[1.3.4 Optimize False Positive/Negative Rates: 6](#_Toc168589733)

[1.3.5 Integration of Diverse Data Sources: 6](#_Toc168589734)

[1.3.6 Evaluate and Benchmark Performance: 6](#_Toc168589735)

[1.3.7 Knowledge Contribution to Cybersecurity Field: 6](#_Toc168589736)

[1.3.8 User-Friendly Implementation: 7](#_Toc168589737)

[1.3.9 Scalability and Resource Efficiency: 7](#_Toc168589738)

[1.3.10 Continuous Improvement and Updates: 7](#_Toc168589739)

[1.4 Problem Statement 7](#_Toc168589740)

[1.5 Proposed Methadology 8](#_Toc168589741)

[1.5.1 Data Collection and Preprocessing: 8](#_Toc168589742)

[1.5.2 Model Architecture Design: 8](#_Toc168589743)

[1.5.3 Model Training and Optimization: 8](#_Toc168589744)

[1.5.4 Model Evaluation and Validation: 9](#_Toc168589745)

[1.5.5 Model Deployment and Testing: 9](#_Toc168589746)

[1.5.6 Documentation and Reporting: 9](#_Toc168589747)

[2 Literature review 11](#_Toc168589748)

[2.1 An Improved CNN Approach for Network Intrusion Detection System 11](#_Toc168589749)

[2.2 Host Based Intrusion Detection System with Combined CNN/RNN Model 13](#_Toc168589750)

[2.3 Deep Learning Approach for Intelligent Intrusion Detection System 15](#_Toc168589751)

[2.4 A Deep Learning Approach for Network Intrusion Detection System 17](#_Toc168589752)

[2.5 Intrusion Detection System (IDS): Anomaly Detection using Outlier Detection Approach 18](#_Toc168589753)

[2.6 Survey on Intrusion Detection System using Machine Learning Techniques 20](#_Toc168589754)

[2.7 Modeling intrusion detection system using hybrid intelligent systems 21](#_Toc168589755)

[3 Analysis 25](#_Toc168589756)

[3.1 Strengths of the Project: 25](#_Toc168589757)

[3.2 Challenges: 26](#_Toc168589758)

[3.3 Opportunities: 26](#_Toc168589759)

[3.4 Future Directions: 27](#_Toc168589760)

[3.5 UNSW NB-15 Dataset 27](#_Toc168589761)

[3.5.1 Dataset Composition: 27](#_Toc168589762)

[3.5.2 Data Collection: 27](#_Toc168589763)

[3.5.3 Dataset Characteristics: 28](#_Toc168589764)

[3.5.4 Types of Attacks: 28](#_Toc168589765)

[3.5.5 Dataset Size and Availability: 28](#_Toc168589766)

[3.5.6 Use in Research: 28](#_Toc168589767)

[3.5.7 Importance in Machine Learning: 28](#_Toc168589768)

[3.6 Convolutional Neural Networks (CNNs) 29](#_Toc168589769)

[3.6.1 Convolutional Layers: 29](#_Toc168589770)

[3.6.2 Pooling Layers: 29](#_Toc168589771)

[3.6.3 Activation Functions: 29](#_Toc168589772)

[3.6.4 Fully Connected Layers: 30](#_Toc168589773)

[3.6.5 Training with Backpropagation: 30](#_Toc168589774)

[3.6.6 Hierarchical Feature Learning: 30](#_Toc168589775)

[3.6.7 Transfer Learning: 30](#_Toc168589776)

[3.7 Multilayer Perceptron (MLP) 30](#_Toc168589777)

[3.7.1 Feedforward Architecture: 30](#_Toc168589778)

[3.7.2 Neurons and Activation Functions: 31](#_Toc168589779)

[3.7.3 Training with Backpropagation: 31](#_Toc168589780)

[3.7.4 Weight Initialization and Optimization: 31](#_Toc168589781)

[3.7.5 Non-linear Mapping: 31](#_Toc168589782)

[3.7.6 Universal Approximation Theorem: 31](#_Toc168589783)

[3.8 Flask integration 32](#_Toc168589784)

[3.8.1 Web-based User Interface (UI): 32](#_Toc168589785)

[3.8.2 Model Deployment and Prediction: 32](#_Toc168589786)

[3.8.3 File Upload and Processing: 32](#_Toc168589787)

[3.8.4 API Endpoints for Model Interaction: 32](#_Toc168589788)

[3.8.5 Error Handling and Feedback: 32](#_Toc168589789)

[3.8.6 Scalability and Deployment: 33](#_Toc168589790)

[3.8.7 Security Considerations: 33](#_Toc168589791)

[4 Design 35](#_Toc168589792)

[4.1 System architecture 36](#_Toc168589793)

[4.2 UML 36](#_Toc168589794)

[4.2.1 Use case diagram 37](#_Toc168589795)

[4.2.2 Class Diagram 38](#_Toc168589796)

[5 Implementation 40](#_Toc168589797)

[5.1 Data Preparation and Preprocessing 40](#_Toc168589798)

[5.1.1 UNSW NB-15 dataset: 40](#_Toc168589799)

[5.1.2 Data Cleaning and Preprocessing: 40](#_Toc168589800)

[5.1.3 Removing unique and highly correlated attributes: 40](#_Toc168589801)

[5.1.4 Standardizing numerical features: 40](#_Toc168589802)

[5.1.5 Encoding categorical attributes: 41](#_Toc168589803)

[5.1.6 Splitting Data: 41](#_Toc168589804)

[5.2 Model Development and Training 41](#_Toc168589805)

[5.2.1 CNN-MLP Architecture Design: 41](#_Toc168589806)

[5.2.2 Model Training: 42](#_Toc168589807)

[5.3 Flask Integration for User Interface (UI) 44](#_Toc168589808)

[5.3.1 Direct Input: 44](#_Toc168589809)

[5.3.2 CSV File Upload: 44](#_Toc168589810)

[5.4 Post-Training Steps and Deployment 44](#_Toc168589811)

[5.4.1 Save Model and Preprocessing Tools: 44](#_Toc168589812)

[5.4.2 Handle Class Imbalance: 44](#_Toc168589813)

[6 Testing 47](#_Toc168589814)

[7 Results 51](#_Toc168589815)

[8 Conclusion and Future scope 54](#_Toc168589816)

[8.1 Conclusion 54](#_Toc168589817)

[8.2 Future scope 55](#_Toc168589818)

[8.2.1 Advanced Model Architectures: 55](#_Toc168589819)

[8.2.2 Feature Engineering and Selection: 55](#_Toc168589820)

[8.2.3 Class Imbalance Handling: 55](#_Toc168589821)

[8.2.4 Real-time Monitoring and Deployment: 55](#_Toc168589822)

[8.2.5 Continuous Model Improvement: 56](#_Toc168589823)

[8.2.6 Integration with Security Operations: 56](#_Toc168589824)

[8.2.7 Cross-Domain Applications: 56](#_Toc168589825)

[9 References 58](#_Toc168589826)

**List of figures**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Fig No.** | **Name of the Figure** | **Page No** |
| 1 | Figure 4.1 | System architecture | 31 |
| 2 | Figure 4.2 | Use case diagram | 32 |
| 3 | Figure 4.3 | Class diagram | 33 |
| 4 | Figure 5.1 | Standardization | 35 |
| 5 | Figure 5.2 | Encoding | 36 |
| 6 | Figure 5.3 | Splitting data | 36 |
| 7 | Figure 5.4 | Model architecture | 37 |
| 8 | Figure 5.5 | Training model | 37 |
| 9 | Figure 5.6 | Inputs | 38 |
| 10 | Figure 5.7 | Saving model | 38 |
| 11 | Figure 5.8 | Oversampling | 38 |
| 12 | Figure 6.1 | Accuracy over epochs | 40 |
| 13 | Figure 6.2 | Class wise detection rate | 40 |
| 14 | Figure 6.3 | Confusion matrix | 41 |
| 15 | Figure 7.1 | home.html | 44 |
| 16 | Figure 7.2 | Predict using input data | 44 |
| 17 | Figure 7.3 | Prediction using .csv file | 44 |
| 18 | Figure 7.4 | Results of .csv file prediction | 45 |
| 19 | Figure 7.5 | Features of model | 45 |

**CHAPTER 1**

**INTRODUCTION**

# INTRODUCTION

In the context of computer security, an intrusion refers to any unauthorized access, entry, or activity within a computer system or network. It involves an entity—whether an individual, program, or process—gaining access to resources or data without proper authorization. Intrusions can take various forms, including hacking, exploiting vulnerabilities, spreading malware, or engaging in other malicious activities. The goal of an intrusion may range from unauthorized data access and theft to disrupting the normal functioning of systems or networks. Protecting against intrusions is a fundamental aspect of cybersecurity, involving the implementation of security measures to prevent, detect, and respond to unauthorized access or activities.

Intrusions in computer networks occur for various reasons, including financial gain through cybercrime, state-sponsored espionage, hacktivism for ideological purposes, theft of intellectual property, disruption through denial-of-service attacks, identity theft, malware distribution, insider threats, vulnerability exploitation, and unintentional actions such as misconfigurations. Mitigating these risks requires implementing robust cybersecurity measures like regular updates, access controls, encryption, and user education on security best practices.

Intrusions into computer systems can result in severe consequences, including data breaches with the potential exposure of sensitive information, financial losses from theft and fraudulent activities, reputation damage due to compromised security, disruption of services leading to downtime, identity theft through the misuse of stolen personal data, loss of intellectual property impacting competitiveness, legal consequences and regulatory penalties, compromised system integrity with potential long-term security challenges, loss of productivity during incident response, increased security costs, and psychological impact on individuals affected by the breach. Mitigating these effects requires robust cybersecurity measures, incident response planning, and a proactive approach to prevent unauthorized access and activities.

Intrusion Detection Systems (IDS) employ two primary approaches to identify and respond to security threats: Signature-Based and Anomaly-Based detection. Signature-Based IDS relies on a database of known attack patterns, comparing observed activity with predefined signatures to detect familiar threats. Anomaly-Based IDS establishes a baseline of normal behavior and flags deviations, identifying potential intrusions based on unusual activity. A hybrid approach often combines both methods for a more comprehensive defense, leveraging the strengths of each. Recent advancements in machine learning have further enhanced IDS capabilities, allowing for adaptive learning and improved accuracy in detecting both known and unknown threats. The choice of approach depends on specific security needs and the desired balance between sensitivity and adaptability in the face of evolving cybersecurity risks.

## Motivation

The motivation behind developing an intrusion detection system using a Convolutional Neural Network (CNN) architecture followed by Multilayer Perceptron (MLP) layers stems from the critical need to enhance cybersecurity measures in today's networked environments. With the proliferation of connected devices and the increasing sophistication of cyber threats, traditional methods of detecting and preventing intrusions are proving insufficient.

By leveraging machine learning techniques, specifically CNN-MLP models, we aim to achieve several key objectives. Firstly, this project seeks to improve the accuracy and efficiency of intrusion detection by automatically analyzing and classifying network traffic data. Unlike rule-based or signature-based approaches, which may struggle to adapt to evolving threats, machine learning models can learn patterns and behaviors indicative of different types of intrusions.

Another motivation is to address the challenge of handling large volumes of network data effectively. CNNs excel in extracting hierarchical features from raw data, making them well-suited for processing complex network traffic. Moreover, the integration of MLP layers allows for deeper learning and abstraction, enabling the model to discern subtle differences between normal and anomalous network behavior. Ultimately, the goal of this project is to contribute to the advancement of cybersecurity practices by developing a scalable and adaptable intrusion detection system. The outcomes of this research have the potential to enhance network security posture, empower organizations to proactively respond to threats, and ultimately mitigate risks associated with cyber attacks in an increasingly interconnected world. This project's motivation lies in its potential to contribute to a safer and more resilient digital infrastructure, benefiting individuals, businesses, and society at large.

## Existing systems

The existing systems for intrusion detection encompass a range of approaches and technologies designed to identify and respond to unauthorized access or malicious activities within computer networks. Here are some common types of existing systems and technologies used for intrusion detection:

### Signature-Based Systems:

These systems rely on predefined patterns or signatures of known attacks to detect intrusions. They compare incoming traffic or system activity against a database of signatures to identify malicious behavior. Examples include antivirus software and intrusion detection systems (IDS) that use signature databases to match against network packets or system logs.

### Anomaly-Based Systems:

Anomaly detection systems monitor system behavior and traffic patterns to identify deviations from normal behavior. They establish a baseline of "normal" activity and raise alerts when anomalies or outliers are detected. Machine learning algorithms, statistical methods, and heuristics are commonly used for anomaly detection.

### Network Intrusion Detection Systems (NIDS):

NIDS are dedicated systems that monitor network traffic for suspicious activity and potential attacks. They analyze packet headers and payloads to detect patterns indicative of known attack types. NIDS can operate in real-time and generate alerts based on detected intrusions.

### Host Intrusion Detection Systems (HIDS):

HIDS are installed on individual hosts (servers, workstations) to monitor and analyze activity within the host's operating system and applications. They detect unauthorized changes to files, system configurations, or user behavior that may indicate an intrusion. HIDS complement network-based detection by providing insights into host-level activities.

### Machine Learning-Based Systems:

Modern intrusion detection systems increasingly leverage machine learning and deep learning techniques. Supervised learning models (e.g., SVM, Random Forests) and deep neural networks (e.g., CNNs, RNNs) are used to analyze network traffic and identify complex patterns associated with intrusions. These systems can adapt to evolving threats and detect novel attack strategies not covered by signature-based approaches.

### Hybrid Systems:

Many organizations deploy hybrid intrusion detection systems that combine multiple detection techniques (e.g., signature-based, anomaly-based) for comprehensive coverage. Hybrid systems leverage the strengths of different approaches to improve detection accuracy and reduce false positives.

### Cloud-Based Intrusion Detection:

With the increasing adoption of cloud computing, intrusion detection systems are also being deployed in cloud environments. Cloud-based IDS leverage scalable resources and real-time monitoring to protect cloud-hosted applications and data.

In summary, the existing systems for intrusion detection encompass a diverse set of technologies ranging from signature-based to machine learning-based approaches. These systems play a crucial role in safeguarding networks, systems, and data against a wide range of cyber threats, contributing to overall cybersecurity posture and resilience. Ongoing research and development efforts continue to advance the capabilities and effectiveness of intrusion detection systems in response to evolving cybersecurity challenges.

## Objectives

The objectives of the project on the convergence of Intrusion Detection and Machine Learning (ML) techniques in computer networks are outlined as follows:

### Enhance Cybersecurity Resilience:

- Strengthen the overall cybersecurity posture of computer networks by developing an advanced Intrusion Detection System (IDS) that leverages machine learning capabilities.

- Improve the ability to detect and respond to both known and emerging cyber threats, thereby enhancing the resilience of the network.

### **Adaptability to Dynamic Threats:**

- Address the limitations of conventional rule-based approaches by implementing machine learning methodologies that can adapt to the dynamic nature of cyber threats.

- Enable the IDS to evolve and update its detection capabilities in response to new attack patterns and techniques.

### Proactive Threat Mitigation:

- Develop a proactive intrusion detection system capable of identifying potential security breaches before they escalate, thereby minimizing the impact of cyber incidents.

- Implement real-time monitoring and response mechanisms to swiftly mitigate identified threats and prevent unauthorized access.

### Optimize False Positive/Negative Rates:

- Utilize machine learning algorithms to optimize the balance between minimizing false positives (incorrectly identifying benign activity as malicious) and false negatives (failing to detect actual malicious activity).

- Fine-tune the IDS to achieve high accuracy in distinguishing between normal and abnormal network behavior.

### Integration of Diverse Data Sources:

- Explore and incorporate a variety of data sources, including network traffic, system logs, and potentially other contextual information, to provide a comprehensive view for intrusion detection.

- Ensure compatibility with different types of data to enhance the system's ability to detect sophisticated and multi-vector attacks.

### Evaluate and Benchmark Performance:

- Establish rigorous evaluation criteria and benchmarking metrics to assess the performance of the ML-based IDS.

- Conduct thorough testing using diverse datasets to measure the system's accuracy, efficiency, and effectiveness in comparison to traditional intrusion detection methods.

### Knowledge Contribution to Cybersecurity Field:

- Contribute valuable insights to the cybersecurity community by documenting the methodologies, challenges, and successes encountered during the integration of ML techniques into intrusion detection systems.

- Publish research findings and share knowledge to foster a deeper understanding of the potential applications and advancements achievable through this convergence.

### User-Friendly Implementation:

- Design the system with user-friendly interfaces and integration mechanisms, allowing cybersecurity professionals to easily deploy, configure, and manage the ML-based intrusion detection system within their network environments.

### Scalability and Resource Efficiency:

- Ensure that the developed system is scalable to accommodate varying network sizes and configurations.

- Optimize resource utilization to minimize the impact on network performance while maintaining robust security measures.

### Continuous Improvement and Updates:

- Establish a framework for continuous improvement, updates, and adaptation to evolving cyber threats, ensuring the long-term relevance and effectiveness of the ML-based intrusion detection system.

## Problem Statement

The project aims to develop an advanced intrusion detection system leveraging a Convolutional Neural Network (CNN) followed by Multilayer Perceptron (MLP) layers to enhance cybersecurity measures. The primary objective is to address the evolving nature of network threats by designing a deep learning-based model capable of accurately classifying network traffic and detecting anomalous activities indicative of intrusions.

By integrating CNN layers, the system will extract hierarchical features from raw network data, capturing intricate patterns that may signify malicious behavior. The subsequent MLP layers will enable further processing and classification of these extracted features, facilitating effective intrusion detection and response. The project will involve extensive dataset preparation, including preprocessing techniques such as normalization and feature engineering, to optimize the model's performance.

The expected outcomes include a robust intrusion detection system capable of identifying diverse types of network intrusions with high accuracy. This system aims to outperform traditional signature-based approaches by adapting to novel attack strategies and mitigating false positives. Insights gained from evaluating the CNN-MLP model will contribute to advancing cybersecurity practices and strengthening network defenses against sophisticated cyber threats.

Overall, the project's significance lies in its potential to bolster network security measures through innovative deep learning techniques, ultimately safeguarding critical assets and data from malicious activities in modern network environments. The developed intrusion detection system has broader implications for enhancing cybersecurity resilience and protecting organizations from evolving cyber threats

## Proposed Methadology

### Data Collection and Preprocessing:

Dataset Selection: Identify and acquire labeled datasets containing network traffic data with annotated intrusion labels (e.g., normal vs. attack traffic).

Data Preprocessing:

Normalize and standardize input features (e.g., packet headers, traffic statistics) to ensure uniform data representation.

Apply techniques such as one-hot encoding for categorical features and scaling for numerical features.

Handle class imbalance using oversampling (e.g., SMOTE) to address the skewed distribution of attack vs. normal instances.

### **Model Architecture Design:**

Convolutional Neural Network (CNN):

Design a CNN architecture suitable for processing sequential network data (e.g., packet sequences).

Use convolutional layers with appropriate filter sizes and activation functions (e.g., ReLU) to extract hierarchical features from raw data.

Apply pooling layers (e.g., MaxPooling) to downsample feature maps and capture essential patterns.

Multilayer Perceptron (MLP):

Incorporate fully connected MLP layers after the CNN layers to perform classification based on extracted features.

Utilize dense layers with dropout regularization to prevent overfitting and enhance generalization.

### Model Training and Optimization:

Split Dataset: Divide the preprocessed dataset into training and validation sets for model training and hyperparameter tuning.

Compile Model:

Configure the CNN-MLP model with appropriate loss function (e.g., categorical cross-entropy) and optimizer (e.g., Adam) for training.

Define evaluation metrics (e.g., accuracy, precision, recall) to monitor model performance during training.

Training Process:

Train the CNN-MLP model using the training dataset, adjusting model parameters iteratively to minimize the training loss.

Monitor validation performance to prevent overfitting and ensure generalizability.

### Model Evaluation and Validation:

Evaluate Performance:

Assess the trained model's performance on the validation set using predefined metrics.

Analyze confusion matrices and ROC curves to understand model behavior and fine-tune thresholds for classification.

Hyperparameter Tuning:

Conduct hyperparameter tuning (e.g., learning rate, batch size) using techniques like grid search or random search to optimize model performance.

### Model Deployment and Testing:

Testing Phase:

Test the finalized CNN-MLP model on unseen test data to evaluate real-world performance and validate intrusion detection capabilities.

Generate performance reports and visualize results to communicate the effectiveness of the developed system.

### Documentation and Reporting:

Document Findings:

Document the entire methodology, including dataset preparation, model architecture, training process, and evaluation outcomes.

Prepare comprehensive reports summarizing project objectives, methodology, results, and recommendations for future enhancements.

**CHAPTER 2**

**LITERATURE REVIEW**

# LITERATURE REVIEW

## An Improved CNN Approach for Network Intrusion Detection System

**Author:** Hu, J., Liu, C. and Cui, Y., 2021. An improved CNN approach for network intrusion detection system. International Journal of Network Security, 23(4), pp.569-575.

**Year**: 2021

**Methods Used:**

**CNN-based Intrusion Detection System:**

A Convolutional Neural Network (CNN) is designed to identify and classify network intrusions. The input data is converted into 8x8 grayscale images to leverage CNN's strong image classification capabilities.

**Data Equalization with Fruit Fly Optimization Algorithm (FOA):**

The Fruit Fly Optimization Algorithm is applied to address the class imbalance issue in the training data. FOA is used to find optimal resampling weights for each class to ensure balanced training.

**Training Process:**

The NSL-KDD dataset is used for training and testing.The dataset is pre-processed by normalizing continuous features and converting them into binary vectors. The training dataset is resampled according to weights obtained from FOA to balance the classes. Cross-entropy is used as the loss function, and stochastic gradient descent is used for optimization.

**Evaluation Metrics:**

Recall, precision, and F1 scores are calculated to evaluate the performance of the intrusion detection system.

**Strengths:**

**Addressing Class Imbalance**:

The use of FOA to find optimal resampling weights effectively addresses the class imbalance problem, leading to better identification rates for minority classes like U2R.

**Use of CNN**:

Leveraging CNN's strength in image classification provides a novel approach to intrusion detection, potentially improving accuracy and detection capabilities.

**Comprehensive Evaluation**:

The paper evaluates the model using multiple metrics (recall, precision, and F1 score), providing a well-rounded assessment of the model’s performance.

**Innovative Pre-processing**:

The conversion of network data into grayscale images is an innovative approach that aligns well with CNN’s capabilities.

**Gaps:**

**Limited Dataset**:

The evaluation is limited to the NSL-KDD dataset, which may not fully represent real-world network traffic and attacks.

**Handling of Unknown Attacks**:

The method filters out unknown attack types in the test dataset, which may not reflect the system's performance in real-world scenarios where new types of attacks are common.

**Overfitting Concerns**:

While the paper mentions the use of dropout to prevent overfitting, it does not provide details on its implementation or effectiveness in this context.

**Generalization to Other Datasets**:

The applicability and performance of the proposed method on other datasets or in real-world scenarios are not explored.

**GAN Integration Mentioned but Not Implemented**:

The paper suggests the potential integration of Generative Adversarial Networks (GAN) to increase detection difficulty, but this is not implemented or tested in the current study.

**Precision Score Impact**:

The impact on the precision score due to class imbalance in the test set is acknowledged but not thoroughly addressed or mitigated.

**Computational Cost**:

The paper does not discuss the computational cost or efficiency of the proposed method, especially considering the use of FOA for optimization.

## Host Based Intrusion Detection System with Combined CNN/RNN Model

**Author:** Ashima Chawla(B), Brian Lee, Sheila Fallon, and Paul Jacob

**Year**: 2019

**Methods Used:**

**Recurrent Neural Networks (RNNs):**

* Utilized RNNs, specifically Gated Recurrent Units (GRUs), for sequence modeling. GRUs are chosen for their reduced computational complexity compared to LSTMs, while still handling sequential data effectively.
* RNNs are used to capture time dependencies in sequences, aiding in anomaly detection.

**Convolutional Neural Networks (CNNs):**

* Integrated 1D CNNs to preprocess input sequences, extracting local features. CNNs help reduce the sequence length and computational cost, making the model more efficient.
* Stacked CNN layers are employed to capture higher-level local features before passing them to the GRU layer.

**Language Modeling:**

* A language model for system call sequences is trained to predict the probability distribution of the next call in a sequence.
* Anomaly detection is performed by calculating the probability of an entire sequence and comparing it against a threshold.

**Dataset:**

* Used the Australian Defence Force Academy Linux Dataset (ADFA-LD) which consists of normal and attack sequences for training and evaluation.

**Model Architecture:**

* The architecture includes an embedding layer, multiple CNN layers, GRU layers, and a TimeDistributed layer with a fully connected softmax layer for final output.
* Models tested include combinations of CNN and GRU layers with different hyperparameters to evaluate performance.

**Strengths:**

**Efficient Training:**

* The CNN-GRU model shows a significant reduction in training time compared to LSTM models, converging faster due to the parallel processing capability of CNNs.

**High Detection Accuracy:**

* The model achieved a 100% True Detection Rate (TDR) and a reasonable Area Under the ROC Curve (AUC) of up to 0.81, indicating strong performance in identifying anomalies.

**Scalability:**

* The architecture’s design, particularly the use of CNNs for preprocessing, makes it scalable to larger datasets and more extensive sequences.

**Reduced Computational Complexity:**

* Using GRUs instead of LSTMs reduces the model's complexity and computational load while maintaining comparable performance.

**Gaps:**

**High False Alarm Rate:**

* Despite high detection accuracy, the model has a high False Alarm Rate (FAR) of 60%, which can be problematic in practical applications.

**Limited Comparison:**

* The study primarily compares its model with LSTM-based models. It would be beneficial to see comparisons with other contemporary models, including non-deep learning approaches.

**Model Generalization:**

* The model’s generalization to other datasets or real-world scenarios is not extensively tested. Additional validation on different types of data would strengthen the findings.

**Ensemble Techniques:**

* While the paper mentions potential future work involving ensemble methods, it does not incorporate these techniques in the current study. Ensemble approaches could potentially improve detection performance and reduce false alarms.

**Feature Engineering:**

* The study relies heavily on the architecture's ability to learn features. Incorporating domain-specific feature engineering could potentially enhance model performance further.

## Deep Learning Approach for Intelligent Intrusion Detection System

**Author:** Vinayakumar, R., Alazab, M., Soman, K.P., Poornachandran, P., Al-Nemrat, A. and Venkatraman, S.

**Year**: 2019

**Methods Used:**

The implementation environment for conducting experiments in intrusion detection systems (IDS) involved Ubuntu 14.0.4 LTS using Python, with the use of Scikit-learn for classical machine learning algorithms and TensorFlow with Keras for deep neural networks (DNNs), taking advantage of GPU (Nvidia Tesla K40) and CPU (Intel Xeon E3-1220 v3 @ 3.10GHz) processing capabilities. The experimental setup included evaluation on both Network Intrusion Detection Systems (NIDS) and Host Intrusion Detection Systems (HIDS) datasets, covering tasks such as classifying network connections as benign or attacks using all available features, categorizing attacks into specific types, and exploring minimal feature sets for classification. Optimal parameters for DNNs were determined through experiments with the KDDCup 99 dataset, tuning hidden units, learning rates, and activation functions, alongside normalization techniques like L2 normalization. Different DNN architectures were explored with varying layers (1 to 5 layers) to assess learning patterns and attack recognition, integrating techniques like batch normalization and dropout to optimize training speed and prevent overfitting. A specific DNN architecture tailored for IDS was developed, featuring input, hidden, and output layers with fully connected nodes, ReLU activation, and batch normalization, using appropriate activation and loss functions (e.g., sigmoid for binary, softmax for multi-class classification). Additionally, a distributed framework called Scale-Hybrid-IDS-AlertNet (SHIA) was developed, leveraging distributed computing and machine learning to enable efficient network-level and host-level intrusion detection, incorporating modules for packet and system call processing, and DNN-based classification.

**Strengths:**

* **Comprehensive Parameter Tuning:** Extensive experimentation to determine optimal DNN parameters, leading to improved performance in attack detection.
* **Detailed Network Topology Exploration:** Evaluated various DNN architectures to identify effective structures for intrusion detection.
* **Proposed DNN Architecture:** Customized DNN architecture tailored for IDS tasks, leveraging deep learning for complex feature extraction.
* **Distributed Framework (SHIA):** Introduced a scalable and distributed approach for intrusion detection, enhancing real-time monitoring capabilities.

**Gaps:**

* **Limited Dataset Diversity:** Primarily focused on a few datasets like KDDCup 99 and NSL-KDD, potentially limiting generalizability.
* **Evaluation Metrics:** While performance metrics like accuracy and ROC curves were used, additional metrics like precision, recall, and F1-score could provide a more nuanced evaluation.
* **Real-World Deployment:** Lack of discussion on real-world deployment challenges and considerations for the proposed SHIA framework.
* **Model Interpretability:** Limited discussion on feature importance and interpretability of DNN models for IDS applications.

## A Deep Learning Approach for Network Intrusion Detection System

**Author:** Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, Mansoor Alam

**Year**: 2016

**Methods Used:**

* **Self-taught Learning (STL):** The authors employed a deep learning technique known as self-taught learning (STL) using sparse autoencoder and soft-max regression. STL involves two stages: unsupervised feature learning on unlabelled data and supervised classification on labelled data. The feature learning phase utilized a sparse autoencoder neural network to extract optimal feature representations from the NSL-KDD dataset. Subsequently, these learned features were used in a soft-max regression classifier for network intrusion detection.

**Strengths:**

* **Flexibility:** Deep learning methods like STL can adapt well to changing and evolving attack scenarios without requiring predefined attack signatures.
* **Feature Learning:** STL effectively learns meaningful representations from large amounts of unlabelled data, enhancing the model's ability to generalize.
* **Performance:** The proposed approach achieved high accuracy rates in detecting network intrusions, outperforming traditional machine learning techniques.

**Gaps:**

* **Data Requirements:** Deep learning methods typically require substantial amounts of labelled data for supervised training, which might be challenging to obtain for real-world network traffic.
* **Complexity:** Implementing and fine-tuning deep learning models like STL can be computationally intensive and may require specialized expertise.
* **Interpretability:** Deep learning models, including STL, can be less interpretable compared to traditional machine learning algorithms, making it challenging to understand the inner workings of the model.

## Intrusion Detection System (IDS): Anomaly Detection using Outlier Detection Approach

**Author:** Jabez, J., & Muthukumar,B

**Year**: 2015

**Methods used**

1. **Outlier Detection Approach**:
   * The paper introduces an anomaly detection method based on outlier detection using the Neighborhood Outlier Factor (NOF). NOF computes the outlier degree for each data example based on its neighborhood density, identifying data points that significantly differ from normal behavior.
2. **System Architecture**:
   * Describes the proposed IDS architecture involving data packet reception, feature extraction, and anomaly detection using the trained model with big datasets. Utilizes distributed storage for the trained model to enhance performance.
3. **Experimental Setup**:
   * Conducts experiments using a dataset comprising training and testing records to evaluate the IDS performance. Measures key metrics such as execution time, anomaly detection rate, and CPU utilization for comparison with existing approaches.

**Strengths:**

1. **Novel Approach**:
   * Introduces a unique method (Outlier Detection Approach) for intrusion detection using NOF, which shows promising results for anomaly detection. Enhances detection precision and stability compared to traditional statistical and rule-based systems.
2. **Performance Improvement**:
   * Utilizes big datasets and distributed storage to improve the performance of the IDS. Achieves efficient anomaly detection with reduced execution time and CPU utilization compared to existing machine learning techniques.
3. **Empirical Validation**:
   * Employs real-world datasets (e.g., KDD datasets) to validate the proposed IDS approach. Provides experimental results demonstrating the effectiveness of the NOF-based anomaly detection method.

**Gaps:**

1. **Limited Evaluation Scenarios**:
   * The paper may benefit from a more extensive evaluation across diverse datasets and network environments to assess the generalizability and robustness of the proposed approach.
2. **Scalability Considerations**:
   * While distributed storage is employed for model training, scalability aspects (e.g., handling larger datasets or dynamic network environments) could be further explored.
3. **Comparison with State-of-the-Art**:
   * While the paper compares the proposed approach with existing methods, a more comprehensive comparison with state-of-the-art anomaly detection techniques could enhance the depth of evaluation.

## Survey on Intrusion Detection System using Machine Learning Techniques

**Author:** Wagh, S.K., Pachghare, V.K. and Kolhe, S.R.

**Year**: 2013

**Methods Used:**

**Review of Machine Learning Approaches:** The paper reviews different machine learning approaches used in Intrusion Detection Systems (IDS), focusing on techniques to reduce false alarm rates and improve intrusion detection accuracy.

**Taxonomy of Anomaly Detection:** It presents a taxonomy based on various criteria to classify IDSs, distinguishing between signature-based (misuse detection) and anomaly-based detection methods. Anomaly detection techniques are explored in depth, including statistical, knowledge-based, and machine learning-based methods.

**In-Depth Analysis of Machine Learning Techniques:** The paper discusses several machine learning techniques applied to IDS, such as Bayesian Networks, Markov Models, Neural Networks, Fuzzy Logic, Genetic Algorithms, Clustering, and Data Mining.

**System Design Overview:** Provides a detailed overview of the system design for IDS, covering preprocessing, classification, post-processing, and methods to reduce false alarms.

**Strengths:**

**Comprehensive Survey:** The paper offers a comprehensive survey of machine learning techniques in IDS, covering a wide range of approaches and methodologies.

**Clear Taxonomy and Classification:** The taxonomy presented helps readers understand the different types of IDS and the classification criteria used to categorize them.

**In-Depth Analysis of Techniques:** It provides detailed insights into various machine learning techniques applied to IDS, including their principles, advantages, and applications.

**Practical System Design Overview:** The overview of the system design for IDS offers a practical understanding of how machine learning techniques are integrated into real-world intrusion detection systems.

**Gaps:**

**Limited Evaluation of Techniques:** The paper focuses more on describing various techniques rather than providing detailed evaluations or comparisons of their effectiveness in real-world scenarios.

**Lack of Empirical Results:** There is a lack of empirical results or case studies demonstrating the performance or efficacy of the discussed techniques in actual IDS implementations.

**Insufficient Discussion on Challenges:** While future directions are mentioned, there could be more discussion on the current challenges and limitations of applying machine learning to IDS.

**Absence of Updated References:** The paper was published in 2013, and there may be newer developments or techniques in the field of intrusion detection using machine learning that are not covered.

## Modeling intrusion detection system using hybrid intelligent systems

**Author:** Peddabachigari, S., Abraham, A., Grosan, C. and Thomas, J.

**Year**: 2007

**Methods Used:**

1. **Decision Trees (DT)**:

Used for classification based on attribute values. Utilizes attribute selection based on information gain to build the tree. Binary DTs were employed for distinguishing between 'Normal' and 'Attack' patterns.

1. **Support Vector Machines (SVM)**:

Utilizes kernel functions to transform data into higher-dimensional space. Employs different SVMs for binary classification of various attack types.

1. **Hybrid Decision Tree-SVM Approach**:

Integrates information from DT (node information) into SVM to enhance classification performance. Uses DT to preprocess data before feeding into SVM.

1. **Ensemble Approach**:

Combines outputs from DT, SVM, and hybrid DT-SVM models to make final decisions. Assigns weights to classifiers based on their performance on the training data.

**Strengths:**

**Comprehensive Comparison**: The paper extensively compares the performance of different classifiers (DT, SVM, hybrid DT-SVM, and ensemble methods) on a real-world intrusion detection dataset (KDD Cup 99).

**Methodological Diversity**: It explores a range of techniques including traditional classifiers (DT, SVM), hybrid models (DT-SVM), and ensemble methods to detect different types of intrusions, providing a comprehensive analysis of their strengths and weaknesses.

**Performance Evaluation**: The research conducts a thorough performance evaluation by analyzing training times, testing times, and accuracy for each classifier and class of attacks.

**Experimental Setup**: The study uses a well-known benchmark dataset (KDD Cup 99) and performs experiments using a consistent data split for training and testing, ensuring the validity and reliability of the results.

**Gaps or Limitations:**

1. **Limited Generalizability**: The study mainly focuses on a single benchmark dataset (KDD Cup 99), which may not fully represent all real-world intrusion scenarios. Generalizability to other datasets or network environments may be limited.
2. **Complexity and Interpretability**: Some of the models used (like SVM with kernels) might be complex and less interpretable, which could impact understanding the reasons behind classification decisions.
3. **Handling Imbalanced Data**: The dataset used likely contains imbalanced classes (e.g., more 'Normal' instances than rare attack types), which could affect model performance and bias towards majority classes.
4. **Evaluation Metrics**: The paper primarily uses accuracy as the evaluation metric, which might not be sufficient for imbalanced datasets. Metrics like precision, recall, and F1-score could provide a more nuanced evaluation, especially for detecting rare attacks.
5. **Scalability**: The computational efficiency of the proposed models (especially ensemble methods) on larger datasets or in real-time scenarios might not have been fully explored.

**CHAPTER 3**

**ANALYSIS**

# ANALYSIS

This project leveraging a Convolutional Neural Network (CNN) followed by Multilayer Perceptron (MLP) layers for intrusion detection exhibits strengths in advanced pattern recognition, adaptability to evolving threats, and scalability. Challenges include acquiring quality labeled datasets, optimizing model complexity, and addressing class imbalance. Opportunities lie in research innovation, industry adoption, and real-world deployment, with future directions focusing on enhanced model architectures, real-time detection, integration with threat intelligence, and ethical considerations for deploying AI-based intrusion detection systems. This project has the potential to significantly advance cybersecurity practices and contribute to the development of effective intrusion detection solutions using deep learning technologies.

## Strengths of the Project:

The project's strengths lie in its innovative approach to intrusion detection using a Convolutional Neural Network (CNN) followed by Multilayer Perceptron (MLP) layers. By leveraging deep learning techniques, the project aims to achieve the following strengths:

Advanced Pattern Recognition: The CNN-MLP architecture enables effective extraction and recognition of complex patterns from raw network traffic data, enhancing the model's ability to detect subtle anomalies indicative of intrusions.

Adaptability to Evolving Threats: Machine learning models can adapt to new and evolving attack strategies, unlike traditional signature-based systems, thereby improving detection accuracy and robustness.

Potential for Real-time Detection: With optimization and deployment, the developed system may offer real-time intrusion detection capabilities, enabling prompt response to security threats.

Scalability and Generalization: Deep learning models have the potential to scale with large datasets and generalize well across different types of intrusions, providing comprehensive coverage in diverse network environments.

## Challenges:

Despite its strengths, the project faces several challenges that need to be addressed:

Data Quality and Quantity: Acquiring labeled datasets of sufficient size and quality for training a CNN-MLP model can be challenging, especially given the diversity and complexity of network intrusion data.

Model Complexity and Optimization: Designing and optimizing the CNN-MLP architecture requires careful consideration of model complexity, hyperparameter tuning, and computational efficiency to achieve optimal performance.

Class Imbalance: Dealing with imbalanced datasets where certain types of intrusions are rare compared to normal activities may lead to biased model predictions and require specialized techniques for handling class imbalance.

Interpretability and Explainability: Deep learning models are often considered "black boxes," making it challenging to interpret and explain their decisions, which is crucial for building trust and understanding in security applications.

## Opportunities:

The project presents several opportunities for advancement and impact in the field of cybersecurity:

Research and Innovation: Exploration of novel deep learning architectures and techniques for intrusion detection can contribute to advancing the state-of-the-art in cybersecurity research.

Industry Adoption: Successful implementation of the CNN-MLP model may lead to adoption by cybersecurity companies and organizations seeking cutting-edge intrusion detection solutions.

Collaboration and Knowledge Sharing: Collaboration with industry experts and researchers can facilitate knowledge sharing and contribute to building a community around deep learning applications in cybersecurity.

Real-world Deployment: Opportunities to deploy the developed system in real-world environments, such as enterprise networks or critical infrastructure, to enhance network security and protect against cyber threats.

## Future Directions:

Looking ahead, the project opens up several avenues for future research and development:

Enhanced Model Architectures: Exploration of more sophisticated deep learning architectures, including recurrent neural networks (RNNs) or attention mechanisms, for sequential analysis of network data.

Real-time Detection and Response: Investigation of techniques for enabling real-time intrusion detection and automated response mechanisms based on model predictions.

Integration with Threat Intelligence: Incorporation of threat intelligence feeds and context-aware features to enhance the model's capability to detect and respond to emerging threats.

Ethical Considerations: Addressing ethical implications related to privacy, bias, and transparency in deploying AI-based intrusion detection systems in sensitive environments.

## UNSW NB-15 Dataset

The UNSW-NB15 dataset is a widely used benchmark dataset for network intrusion detection research and machine learning applications. It was created by the University of New South Wales (UNSW) to support the development and evaluation of intrusion detection systems using machine learning techniques. Here's an overview of the UNSW-NB15 dataset:

Overview of UNSW-NB15 Dataset:

### Dataset Composition:

The UNSW-NB15 dataset consists of network traffic data captured in a controlled environment, simulating a realistic network scenario.

It contains a diverse range of network-based attack scenarios and normal activities, making it suitable for training and testing intrusion detection systems.

### Data Collection:

The dataset was collected using the IXIA PerfectStorm traffic generator in a lab environment, simulating a real network with a mix of normal and malicious activities.

Various attack types and intrusion scenarios were intentionally injected into the network traffic to create a representative dataset.

### Dataset Characteristics:

Features: The dataset includes a rich set of features derived from network traffic, such as source and destination IP addresses, port numbers, protocol types, packet sizes, and timestamps.

Labels: Each instance in the dataset is labeled with the corresponding attack category or normal activity type.

### Types of Attacks:

The dataset covers a wide range of network intrusion categories, including but not limited to:

Denial of Service (DoS) attacks

Distributed Denial of Service (DDoS) attacks

Reconnaissance attacks

Exploitation attacks (e.g., buffer overflow)

Fuzzers, backdoors, and analysis attacks

### Dataset Size and Availability:

The UNSW-NB15 dataset is publicly available and widely used in academic research and cybersecurity studies.

It consists of approximately 2.5 million instances, making it a large-scale dataset suitable for training complex machine learning models.

### Use in Research:

Researchers and practitioners use the UNSW-NB15 dataset to develop and evaluate intrusion detection systems, anomaly detection algorithms, and other cybersecurity solutions.

The dataset facilitates comparative studies and benchmarking of different approaches to network intrusion detection and security analytics.

### Importance in Machine Learning:

The UNSW-NB15 dataset plays a crucial role in advancing the field of machine learning for cybersecurity applications.

It enables researchers to train and test models to detect and classify various types of network attacks, contributing to the development of more robust and effective intrusion detection systems.

Overall, the UNSW-NB15 dataset serves as a valuable resource for studying network security, evaluating machine learning algorithms, and fostering innovation in the field of cybersecurity research and development. Its comprehensive coverage of network attack scenarios and realistic traffic patterns makes it an essential dataset for addressing contemporary cybersecurity challenges.

## Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep neural networks commonly used for processing and analyzing visual data, such as images. They are designed to automatically learn hierarchical patterns and representations directly from raw data, making them particularly effective for tasks like image classification, object detection, and image segmentation. Here are key concepts and components of CNNs:

### Convolutional Layers:

Feature Extraction: CNNs use convolutional layers to extract features from input data. Each layer applies a set of learnable filters (kernels) to the input, performing convolution operations to produce feature maps.

Local Receptive Fields: Convolutional layers use small, localized receptive fields to capture spatial patterns within the input data, enabling translation-invariant feature learning.

### Pooling Layers:

Downsampling: Pooling layers (e.g., max pooling, average pooling) are used to downsample feature maps, reducing spatial dimensions while retaining important information.

Feature Invariance: Pooling helps create spatial invariance to small transformations in the input data, improving the model's robustness.

### Activation Functions:

Non-linearity: CNNs use activation functions like ReLU (Rectified Linear Unit) to introduce non-linearities into the network, enabling complex mapping of input data to output predictions.

### Fully Connected Layers:

Classification and Decision Making: The output from convolutional and pooling layers is typically flattened and passed through fully connected layers (MLP) for final classification or regression tasks.

### Training with Backpropagation:

Optimization: CNNs are trained using backpropagation and optimization algorithms (e.g., gradient descent) to minimize a loss function (e.g., cross-entropy) between predicted and actual outputs.

Learnable Parameters: The learnable parameters (weights and biases) of the CNN are adjusted during training to improve performance on the training data.

### Hierarchical Feature Learning:

Layered Representations: CNNs learn hierarchical representations of features, where lower layers capture basic features (e.g., edges, textures) and higher layers capture more abstract and complex features (e.g., object parts, semantic concepts).

### Transfer Learning:

Reuse of Pretrained Models: CNNs can leverage transfer learning by using pre-trained models (e.g., from ImageNet) as feature extractors for new tasks, accelerating training and improving generalization.

CNNs have revolutionized the field of computer vision and have applications beyond image processing, including speech recognition, natural language processing (NLP), and even time-series data analysis. Their ability to automatically learn meaningful features from raw data and their hierarchical structure make them powerful tools for solving complex pattern recognition problems in various domains.

## Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) is a class of feedforward artificial neural networks consisting of multiple layers of nodes (neurons), organized in a sequence of interconnected layers. MLPs are widely used for supervised learning tasks such as classification and regression. Here are key concepts and components of MLPs:

### Feedforward Architecture:

Input Layer: The first layer receives input features and passes them forward to subsequent layers.

Hidden Layers: Intermediate layers (hidden layers) between the input and output layers. Each neuron in a hidden layer is connected to all neurons in the previous layer.

Output Layer: The final layer produces the model's output, which could be a single value (for regression) or a probability distribution across multiple classes (for classification).

### Neurons and Activation Functions:

Neurons: Neurons in each layer perform a weighted sum of inputs followed by an activation function.

Activation Functions: Common activation functions include ReLU (Rectified Linear Unit) for hidden layers and softmax (for classification) or linear activation (for regression) for the output layer.

### Training with Backpropagation:

Forward Pass: During training, input data propagate through the network in a forward pass, computing predictions.

Loss Calculation: A loss function (e.g., mean squared error for regression, cross-entropy for classification) measures the difference between predicted and actual outputs.

Backpropagation: Errors are propagated backward through the network using backpropagation, adjusting weights and biases to minimize the loss.

### Weight Initialization and Optimization:

Initialization: Weights and biases of MLPs are initialized randomly, and optimization algorithms like gradient descent are used to update these parameters iteratively during training.

Regularization: Techniques like dropout or L2 regularization are applied to prevent overfitting and improve generalization.

### Non-linear Mapping:

Expressive Power: MLPs are capable of learning complex non-linear mappings between input and output data, allowing them to model intricate relationships in the data.

### Universal Approximation Theorem:

Approximation Capability: MLPs, under certain conditions, are proven to be universal approximators, meaning they can approximate any continuous function given enough hidden units and appropriate activation functions.

MLPs are foundational models in the field of neural networks and machine learning, serving as building blocks for more complex architectures such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). They are versatile and can be applied to a wide range of tasks across different domains, making them essential tools in modern data science and artificial intelligence applications.

## Flask integration

### Web-based User Interface (UI):

Input Forms: Flask can render HTML templates to create input forms where users can submit network traffic data or upload CSV files containing data for intrusion detection.

User Interaction: Users can interact with the UI to input data parameters, initiate intrusion detection tasks, and receive model predictions.

### Model Deployment and Prediction:

Backend Processing: Flask serves as the backend framework to handle incoming data requests and invoke the trained CNN-MLP model for intrusion detection tasks.

Prediction Results: Once the model processes the input data, Flask can display the predicted intrusion types or probabilities back to the user through the UI.

### File Upload and Processing:

CSV File Handling: Flask allows users to upload CSV files containing network traffic data, which can be parsed and processed by the backend to generate intrusion predictions.

Data Validation: Flask can validate uploaded files to ensure they meet specified criteria (e.g., file format, size) before processing.

### API Endpoints for Model Interaction:

RESTful API: Flask can expose RESTful endpoints that frontend components (e.g., JavaScript scripts) can call to interact with the intrusion detection model.

Asynchronous Processing: Flask can handle asynchronous requests, allowing users to submit multiple intrusion detection tasks concurrently.

### Error Handling and Feedback:

Feedback Messages: Flask can provide informative feedback messages to users based on model predictions or error conditions encountered during data processing.

Error Handling: Implement error handling mechanisms in Flask to gracefully manage exceptions and unexpected behaviors during user interactions.

### Scalability and Deployment:

Deployment Flexibility: Flask applications can be deployed on various platforms (e.g., local server, cloud services) to accommodate different deployment requirements.

Scaling Options: Flask applications can be scaled horizontally or vertically based on usage demands and system requirements.

### Security Considerations:

Input Sanitization: Implement input sanitization techniques in Flask to mitigate potential security risks associated with user-provided data.

Authentication and Authorization: Integrate authentication and authorization mechanisms into Flask to restrict access to sensitive features and endpoints.

In summary, Flask provides a lightweight and flexible framework to build a web-based interface for your intrusion detection system project. It enables seamless integration between the backend model components and frontend user interactions, facilitating efficient deployment, user engagement, and data-driven decision-making in cybersecurity applications.

**CHAPTER 4**

**DESIGN**

# DESIGN

 **Dataset**: This section shows the raw data, which comes from a UNSW NB-15 dataset [UNSW NB 15 dataset] and a user.

 **Data Preprocessing**: This section includes several steps to clean and prepare the data for training the model.

Removing unique attributes: This step removes any attributes from the data that only have one unique value. These attributes would not be useful for training the model.

Removing highly correlated attributes: This step removes attributes from the data that are highly correlated with other attributes. Including these attributes would make the model redundant.

Standardizing numerical attributes: This step scales the numerical attributes in the data to a common range. This can improve the performance of some machine learning algorithms.

Saving standardizer in pickle file: This step saves the scaler used for standardization in a file so it can be used later when making predictions on new data.

Encoding categorical attributes: This step converts categorical attributes in the data into numerical attributes that the machine learning model can understand.

Saving encoder in pickle file: This step saves the encoder used for categorical encoding in a file so it can be used later when making predictions on new data.

 **Splitting data for testing and training**: This step splits the data into two sets: a training set and a testing set. The training set is used to train the model, and the testing set is used to evaluate the performance of the model.

 **Training data to CNN-MLP architecture and testing**: This step trains a machine learning model on the training data. The model appears to be a convolutional neural network (CNN) and multi-layer perceptron (MLP) architecture. After training, the model is evaluated on the testing data.

 **Saving model**: This step saves the trained model to a file.

 **Bottom Right**: This section shows how the trained model can be used to make predictions on new data.

Applying SMOTE ENN for oversampling: This step applies a technique called SMOTE ENN to the data. SMOTE ENN is a way to address class imbalance in machine learning datasets. Class imbalance occurs when there are significantly more examples of some classes than others. SMOTE ENN creates synthetic examples of the minority class to help improve the performance of the model.

Predicted classes: This cell shows the predicted classes for the new data.

UI: This cell likely refers to a user interface that would allow users to interact with the model and see the predicted classes.

## System architecture

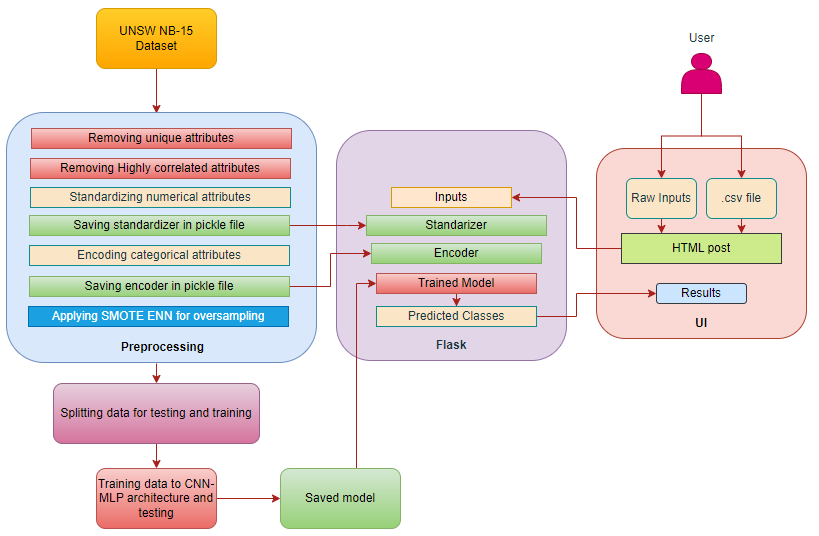


Figure 4.1: System architecture

## UML

The Unified Modeling Language (UML) is a standard language for writing software blueprints. The UML is a language for Visualizing, Specifying, Constructing, the artifacts of a software intensive system.

The UML is a language which provides vocabulary and the rules for combining words in that vocabulary for the purpose of communication. A modeling language is a language whose vocabulary and the rules focus on the conceptual and physical representation of a system. Modeling yields an understanding of a system.

* **Use Case diagrams**: Use case is a description of set of sequence of actions that a system performs that yields an observable result of value to actor. Actors are the entities that interact with a system. Although in most cases, actors used to represent the users of system, actors can be anything that needs to exchange information with the system. So, an actor may be people, computer hardware, other systems, etc.
* **Class diagrams**: In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram. It describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

### Use case diagram

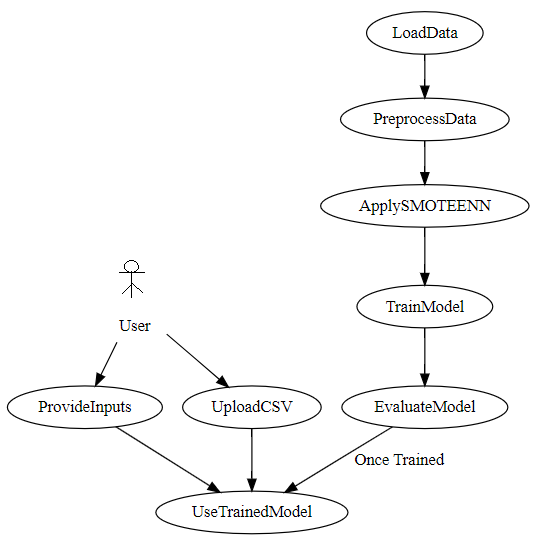


Figure 4.2: Use case diagram

### Class Diagram

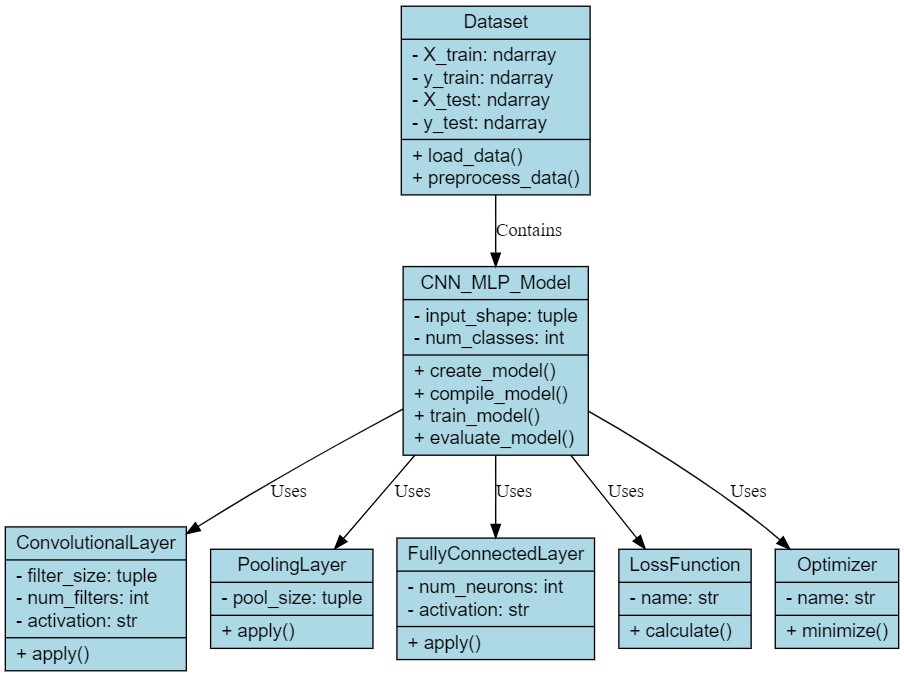


Figure 4.3: Class diagram

**CHAPTER 5**

**IMPLEMENTATION**

# IMPLEMENTATION

The implementation of an intrusion detection system using a Convolutional Neural Network (CNN) followed by Multilayer Perceptron (MLP) layers can be further enhanced by integrating it with Flask to create a user interface (UI) for inputting data and displaying model predictions includes below steps.

## Data Preparation and Preprocessing

Raw Data Acquisition: This involves gathering network traffic data that will be used to train the intrusion detection system. There are two main sources for this data:

### UNSW NB-15 dataset:

This is a publicly available dataset of labeled network traffic data that is commonly used for intrusion detection research. The system can also be designed to accept network traffic data directly from users. This could be useful for real-time intrusion detection scenarios.

### Data Cleaning and Preprocessing:

Once the data is acquired, it needs to be cleaned and prepared for training the machine learning model. This typically involves several steps:

### Removing unique and highly correlated attributes:

Data cleaning removes attributes that don't provide useful information for training the model. This includes attributes that have only one unique value (e.g., an identifier for each data point) and attributes that are highly correlated with other attributes (e.g., two different ways of measuring the same network traffic characteristic). Including these attributes would make the model redundant and potentially less accurate.

### Standardizing numerical features:

This step scales numerical attributes in the data to a common range. This is important because some machine learning algorithms perform better when all features are on a similar scale.

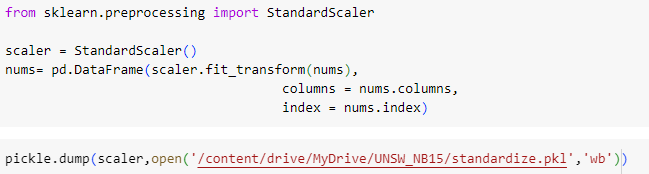


Figure 5.1: Standardization

### Encoding categorical attributes:

Categorical attributes, like text labels, need to be converted into numerical values that the machine learning model can understand. This is typically done using a technique called one-hot encoding.

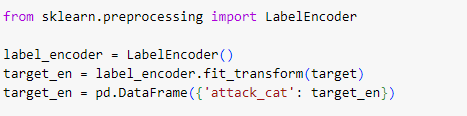


Figure 5.2: Encoding

### Splitting Data:

After cleaning and preprocessing, the data is divided into two sets: a training set and a testing set. The training set is used to train the machine learning model, and the testing set is used to evaluate the performance of the model on unseen data. This helps to ensure that the model generalizes well and can accurately detect intrusions on new network traffic data.



Figure 5.3: Splitting data

## Model Development and Training

### CNN-MLP Architecture Design:

The system utilizes a deep learning architecture that combines two powerful neural network techniques: Convolutional Neural Networks (CNNs) and Multi-layer Perceptrons (MLPs). Convolutional Neural Networks (CNNs): CNNs are well-suited for extracting patterns from sequential data, like network traffic data. They do this by using filters that slide over the data and learn to identify important features. Multi-layer Perceptrons (MLPs): MLPs are artificial neural networks with multiple layers of interconnected nodes. They are used for classification tasks, where the goal is to learn a mapping between the input data (network traffic features) and the output classes (normal traffic or intrusion type).

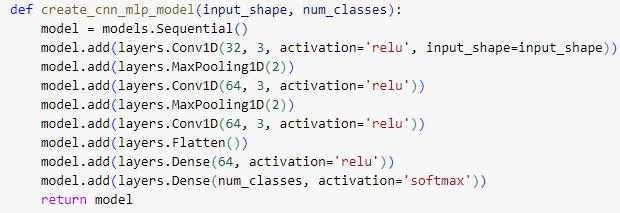


Figure 5.4: model architecture

### Model Training:

The CNN-MLP model is trained on the training dataset. This involves iteratively feeding the training data through the model, calculating the difference between the model's predictions and the true labels (supervised learning), and adjusting the model's internal parameters (weights and biases) to minimize this difference. The training process continues until the model reaches a desired level of performance.

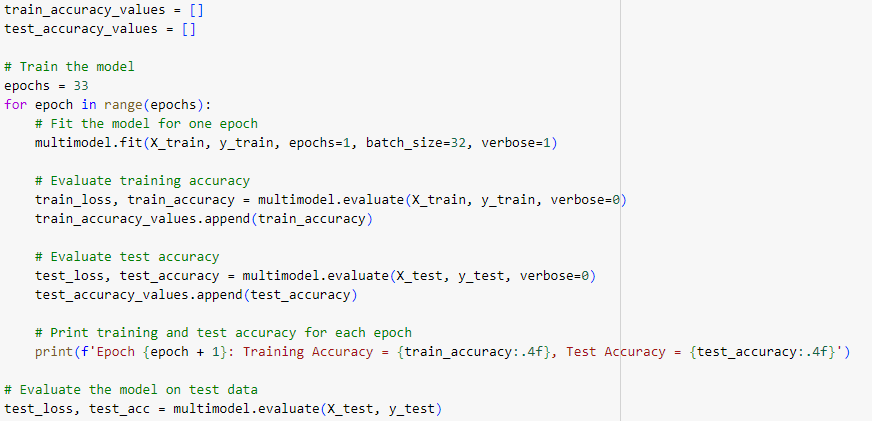


Figure 5.5: Training model

**Algorithm**: CNN-MLP Model Training

Input:

- Training data (X\_train, y\_train) and test data (X\_test, y\_test)

Output:

- Attack category prediction

1: Initialize weight parameters 𝑊 for each layer 𝑙

2: for each epoch 𝑒 ∈ [1, 𝐸] do

3: Shuffle the training set (X\_train, y\_train)

4: for each training batch (X\_batch, y\_batch) do

5: 𝑎[0] = X\_batch

6: 𝑧[1] = 𝑊[1] \* 𝑎[0] + 𝑏[1]

7: 𝑎[1] = ReLU(𝑧[1])

8: 𝑎[1] = MaxPooling(𝑎[1])

9: 𝑧[2] = 𝑊[2] \* 𝑎[1] + 𝑏[2]

10: 𝑎[2] = ReLU(𝑧[2])

11: 𝑎[2] = MaxPooling(𝑎[2])

12: 𝑧[3] = 𝑊[3] \* 𝑎[2] + 𝑏[3]

13: 𝑎[3] = ReLU(𝑧[3])

14: 𝑎[3] = Flatten(𝑎[3])

15: 𝑧[4] = 𝑊[4] \* 𝑎[3] + 𝑏[4]

16: 𝑎[4] = ReLU(𝑧[4])

17: 𝑧[5] = 𝑊[5] \* 𝑎[4] + 𝑏[5]

18: 𝑎[5] = Softmax(𝑧[5])

19: Loss = CrossEntropyLoss(𝑎[5], y\_batch)

20: Update 𝑊 and 𝑏 using gradient descent with respect to Loss

21: end for

22: TrainAccuracy = CalculateAccuracy(X\_train, y\_train)

23: TestAccuracy = CalculateAccuracy(X\_test, y\_test)

24: Print("Epoch", 𝑒, ": Training Accuracy =", TrainAccuracy, ", Test Accuracy =", TestAccuracy)

25: end for

26: FinalTestAccuracy = CalculateAccuracy(X\_test, y\_test)

27: Print("Final Test Accuracy:", FinalTestAccuracy)

## Flask Integration for User Interface (UI)

Flask Setup: Flask is a lightweight web framework written in Python that is used for creating web applications. Here, Flask is used to develop a user interface (UI) that allows users to interact with the intrusion detection system.Input Options: The UI provides users with two ways to submit data for intrusion detection:

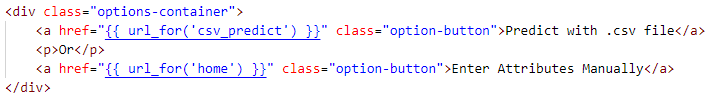


Figure 5.6: Inputs

### Direct Input:

Users can enter network traffic features directly into a web form on the UI. This is useful for quick checks or real-time intrusion detection.

### CSV File Upload:

Users can upload a CSV file containing network traffic data for batch processing. This is beneficial for analysing large amounts of data or scheduling regular scans. Prediction Output, Once the user submits data (either through direct input or CSV upload), the system processes it using the trained CNN-MLP model. The processed data is then fed into the model, and the predicted intrusion classes or probabilities are displayed back to the user on the UI.

## Post-Training Steps and Deployment

### Save Model and Preprocessing Tools:

After training, it's essential to save the trained CNN-MLP model, along with the scaler and encoder used during data preprocessing. This allows the system to be used later for real-time intrusion detection without retraining the model from scratch.



Figure 5.7: Saving model

### Handle Class Imbalance:

Real-world network traffic data often exhibits class imbalance, where there are significantly more examples of normal traffic than intrusion attempts. This can make it difficult for the model to learn to detect intrusions accurately. To address this, the system can employ a technique called SMOTE ENN during real-time predictions. SMOTE ENN artificially generates synthetic examples of the minority class (intrusion attempts) to balance them

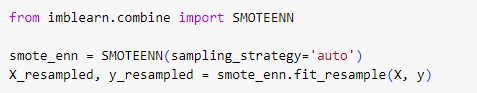


Figure 5.8: oversampling

**CHAPTER 6**

**TESTING**

# TESTING



Figure 6.1: Accuracy over epochs

 **Training Accuracy:** The training accuracy starts at around 0.2 and increases steadily to nearly 1.0 by epoch 30. This suggests that the model is effectively learning the patterns in the training data. However, it’s important to consider the test accuracy to avoid overfitting.

 **Test Accuracy:** The test accuracy starts at around 0.4 and increases to around 0.8 by epoch 30. While the test accuracy is also increasing, it is consistently lower than the training accuracy which indicates some overfitting.

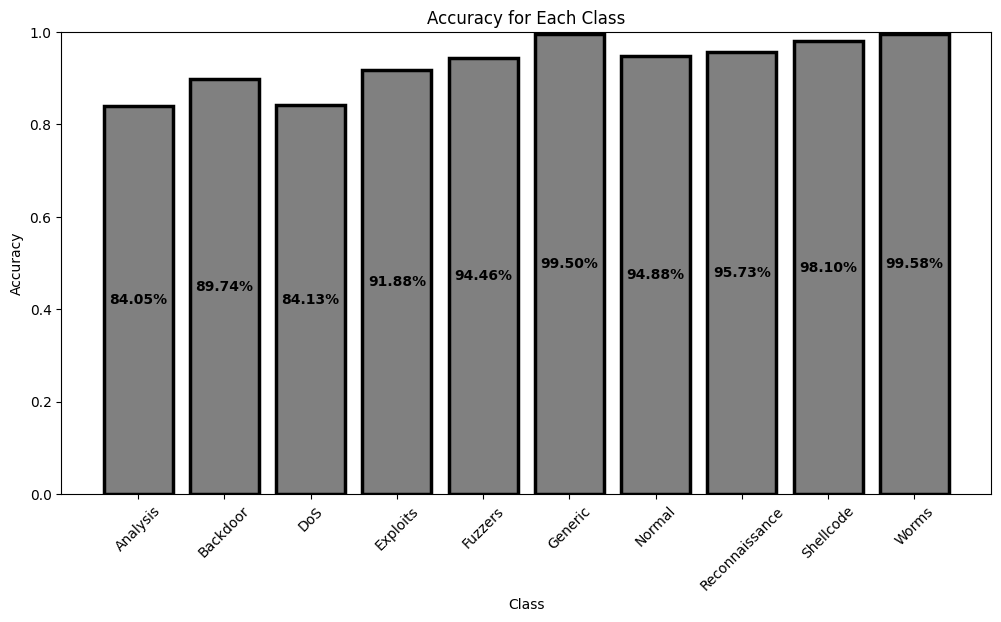


Figure 6.2: Class wise detection rate

 The accuracy for most of the classes is quite high, which suggests that the model is overall good at classifying different types of traffic.

 The accuracy for some classes is lower than for others. This could be because there is less data available for these classes, or because they are more difficult to distinguish from other types of traffic.

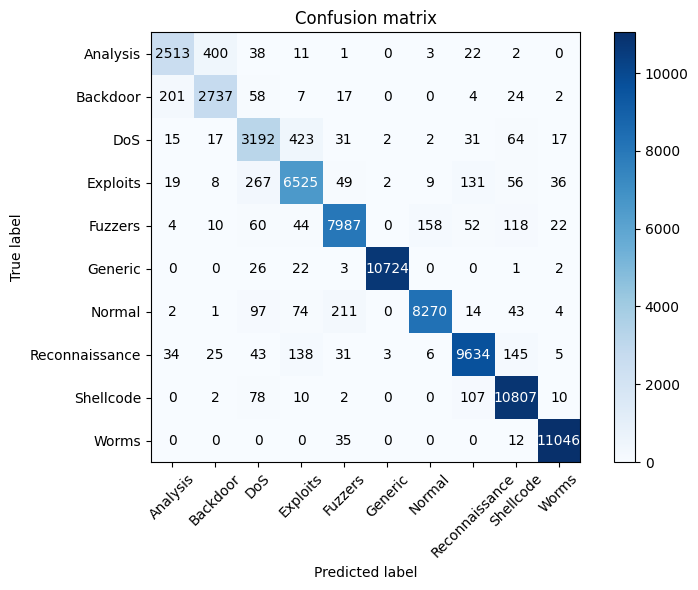


Figure 6.3: Confusion matrix

 **Correct Classifications:** The diagonal cells of the matrix (highlighted in bold) show the number of correct classifications for each traffic type. For instance, the model correctly classified 8270 normal traffic instances, 10724 generic traffic instances, and 11046 worms traffic instances.

 **Misclassifications:** The off-diagonal cells show the number of misclassifications for each traffic type. There were the most misclassifications for DoS traffic (3192) and Backdoor traffic (2737). In these cases, the model may have difficulty distinguishing between these types of traffic and other types, such as Exploits or Analysis traffic.

 **False Positives:** A false positive occurs when the model predicts a particular traffic type, but the actual traffic type is different. For example, the model predicted 423 DoS instances as Exploits traffic, and 58 Backdoor instances as DoS traffic.

 **False Negatives:** A false negative occurs when the model fails to predict the correct traffic type. There were very few false negatives in this confusion matrix, indicating the model did well at identifying the correct traffic type most of the time.

**CHAPTER 7**

**RESULTS**

# RESULTS

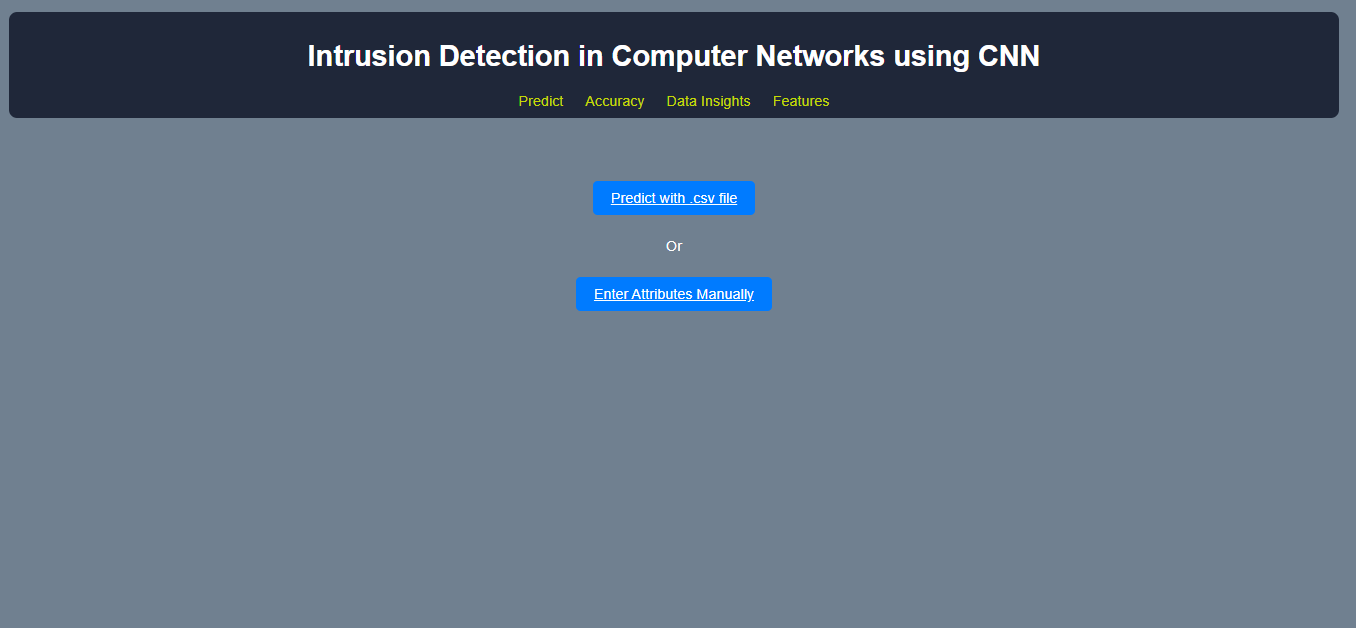


Figure 7.1: home.html

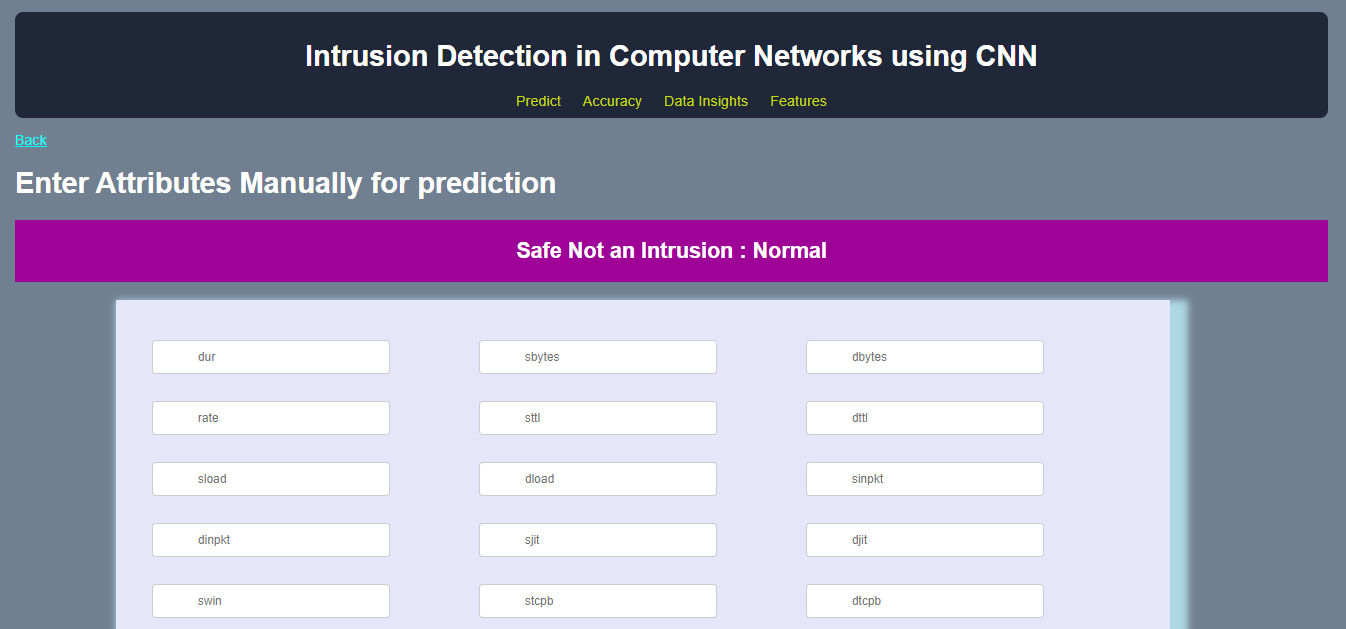


Figure 7.2: Predict using input data

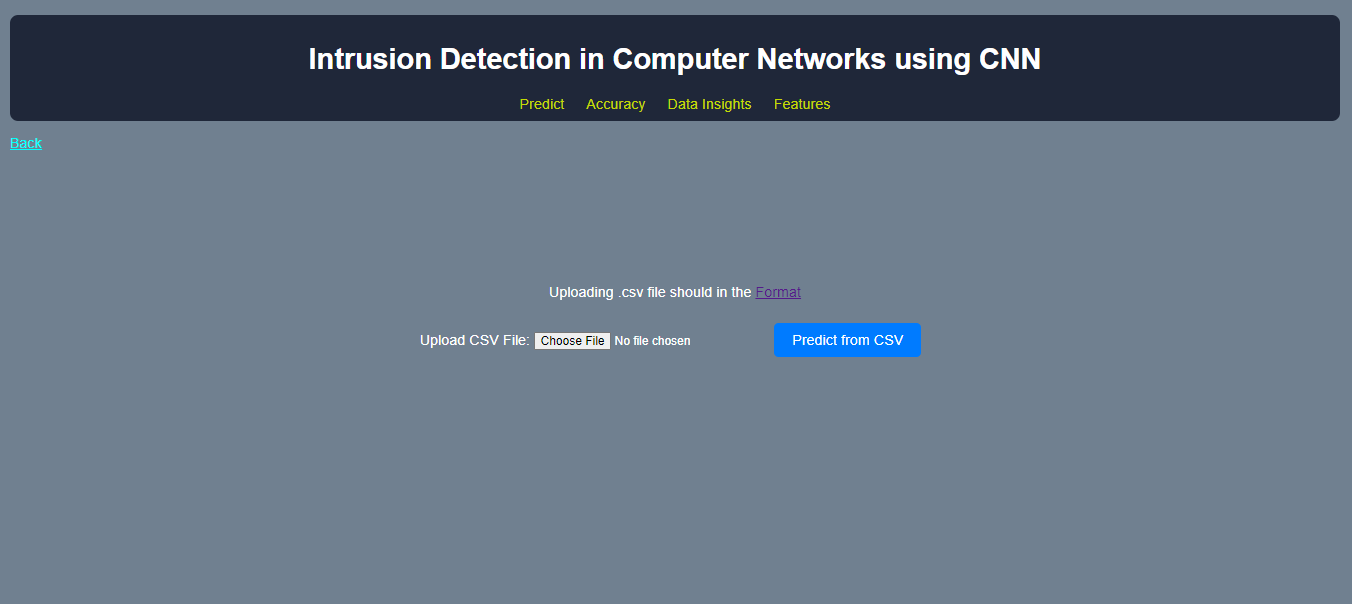


Figure 7.3: Prediction using .csv file

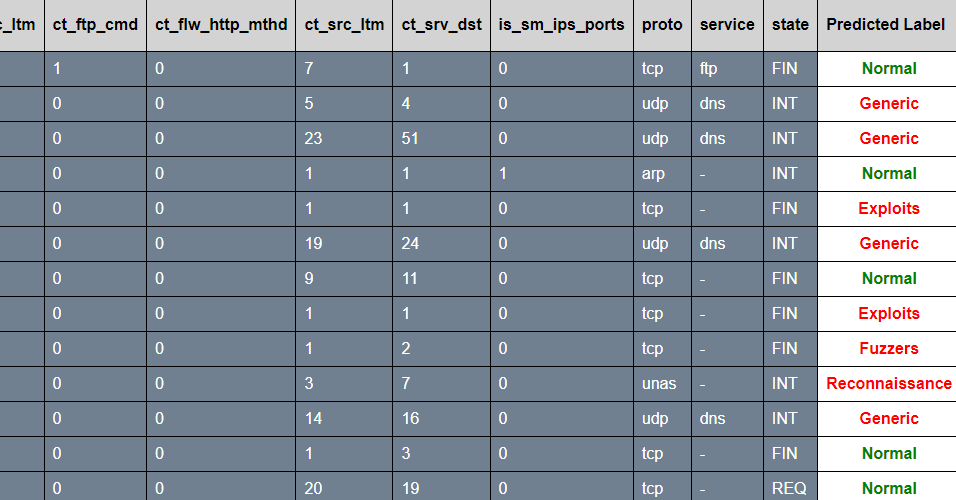


Figure 7.4: Results of .csv file prediction

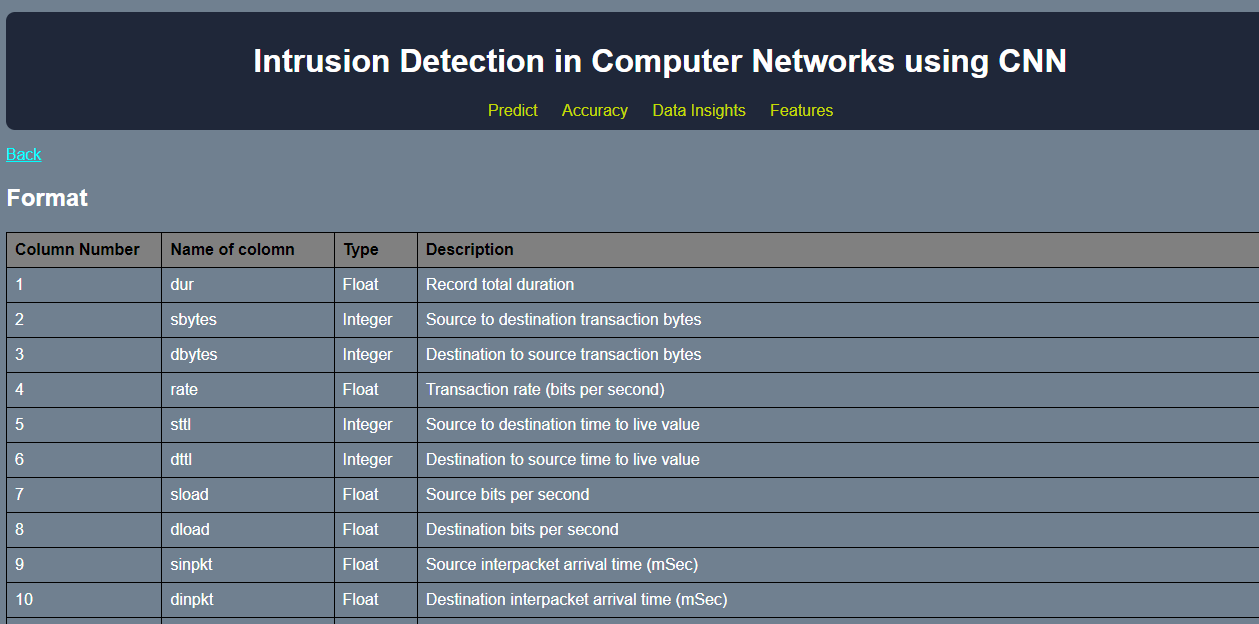


Figure 7.5: Features of model

**CHAPTER 8**

**CONCLUSION AND FUTURE SCOPE**

# CONCLUSION AND FUTURE SCOPE

## Conclusion

In conclusion, the project focused on developing an effective intrusion detection system using a hybrid CNN-MLP architecture trained on the UNSW-NB15 dataset, integrated with Flask to provide a user-friendly interface for interaction and model deployment. The key highlights and outcomes of the project include:

Data Preprocessing and Model Training: Rigorous preprocessing steps were applied to clean and prepare the UNSW-NB15 dataset, including feature selection, standardization, and encoding. The CNN-MLP architecture was designed and trained using this preprocessed data to detect various types of network intrusions.

Integration with Flask for UI: Flask was employed to create a web-based user interface where users can input network traffic data or upload CSV files for intrusion detection. The integration allowed seamless interaction with the trained model, enabling real-time predictions and feedback.

SMOTE ENN for Class Imbalance: Addressing class imbalance in the dataset, Synthetic Minority Over-sampling Technique combined with Edited Nearest Neighbors (SMOTE ENN) was applied to enhance the model's performance on underrepresented intrusion types.

User-Friendly Deployment: The deployment of the CNN-MLP model within a Flask application provided a user-friendly platform for cybersecurity analysts and stakeholders to leverage the intrusion detection system. The interface facilitated intuitive data input and retrieval of intrusion predictions.

## Future scope

The project on developing an intrusion detection system using a CNN-MLP architecture integrated with Flask presents several promising avenues for future exploration and expansion. Here are some potential future scopes and enhancements for the project:

### Advanced Model Architectures:

Exploration of Deep Learning Models: Investigate more complex deep learning architectures beyond CNN-MLP, such as recurrent neural networks (RNNs) or attention-based models, to capture temporal dependencies and improve detection accuracy.

Ensemble Learning Approaches: Explore ensemble learning techniques to combine predictions from multiple models or model variants, enhancing overall performance and robustness.

### Feature Engineering and Selection:

Feature Importance Analysis: Conduct in-depth feature importance analysis to identify and focus on critical network traffic features, potentially reducing model complexity and training time.

Automatic Feature Engineering: Explore automated feature engineering techniques (e.g., autoencoders, genetic algorithms) to generate meaningful representations directly from raw network data.

### Class Imbalance Handling:

Advanced Sampling Techniques: Investigate more sophisticated techniques for handling class imbalance, such as adaptive sampling strategies or novel oversampling and undersampling methods tailored to network intrusion detection.

### Real-time Monitoring and Deployment:

Integration with Streaming Data Sources: Extend the system to handle streaming data sources for real-time intrusion detection and monitoring, leveraging technologies like Apache Kafka or MQTT.

Deployment in Cloud Environment: Deploy the intrusion detection system on cloud platforms (e.g., AWS, Google Cloud) for scalable and cost-effective deployment, allowing for on-demand processing and resource utilization.

### Continuous Model Improvement:

Active Learning Strategies: Implement active learning techniques to dynamically update and refine the intrusion detection model using user feedback and new labeled data.

Model Interpretability: Enhance model interpretability by incorporating techniques such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations), enabling better understanding of model decisions and predictions.

### Integration with Security Operations:

Automated Response Systems: Integrate the intrusion detection system with automated response systems (e.g., Security Information and Event Management - SIEM) to enable immediate actions in response to detected threats.

Incident Analysis and Reporting: Develop capabilities for detailed incident analysis, reporting, and visualization to aid cybersecurity analysts in understanding and mitigating security incidents.

### Cross-Domain Applications:

Adaptation to IoT and Industrial Networks: Extend the intrusion detection system to address security challenges in Internet of Things (IoT) networks or industrial control systems (ICS), which have unique communication protocols and constraints.

**CHAPTER 9**

**REFERENCES**

# REFERENCES

1. Vinayakumar, R., Alazab, M., Soman, K.P., Poornachandran, P., Al-Nemrat, A. and Venkatraman, S., 2019. Deep learning approach for intelligent intrusion detection system. Ieee Access, 7, pp.41525-41550.
2. Javaid, A., Niyaz, Q., Sun, W. and Alam, M., 2016, May. A deep learning approach for network intrusion detection system. In Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS) (pp. 21-26).
3. Jabez, J. and Muthukumar, B.J.P.C.S., 2015. Intrusion Detection System (IDS): Anomaly detection using outlier detection approach. Procedia Computer Science, 48, pp.338-346.
4. Wagh, S.K., Pachghare, V.K. and Kolhe, S.R., 2013. Survey on intrusion detection system using machine learning techniques. International Journal of Computer Applications, 78(16), pp.30-37.
5. Peddabachigari, S., Abraham, A., Grosan, C. and Thomas, J., 2007. Modeling intrusion detection system using hybrid intelligent systems. Journal of network and computer applications, 30(1), pp.114-132.
6. Hu, J., Liu, C. and Cui, Y., 2021. An improved CNN approach for network intrusion detection system. International Journal of Network Security, 23(4), pp.569-575.
7. Chawla, A., Lee, B., Fallon, S. and Jacob, P., 2019. Host based intrusion detection system with combined CNN/RNN model. In *ECML PKDD 2018 Workshops: Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings 18* (pp. 149-158). Springer International Publishing.
8. ElSayed, M.S., Le-Khac, N.A., Albahar, M.A. and Jurcut, A., 2021. A novel hybrid model for intrusion detection systems in SDNs based on CNN and a new regularization technique. *Journal of Network and Computer Applications*, *191*, p.103160.
9. Halbouni, A., Gunawan, T.S., Habaebi, M.H., Halbouni, M., Kartiwi, M. and Ahmad, R., 2022. CNN-LSTM: hybrid deep neural network for network intrusion detection system. *IEEE Access*, *10*, pp.99837-99849.
10. Riyaz, B. and Ganapathy, S., 2020. A deep learning approach for effective intrusion detection in wireless networks using CNN. *Soft Computing*, *24*(22), pp.17265-17278.
11. Nguyen, S.N., Nguyen, V.Q., Choi, J. and Kim, K., 2018, February. Design and implementation of intrusion detection system using convolutional neural network for DoS detection. In *Proceedings of the 2nd international conference on machine learning and soft computing* (pp. 34-38).
12. Ding, Y. and Zhai, Y., 2018, December. Intrusion detection system for NSL-KDD dataset using convolutional neural networks. In *Proceedings of the 2018 2nd International conference on computer science and artificial intelligence* (pp. 81-85).