**Deep\_Ensemble\_Learning\_With\_Pruning\_for\_DDoS\_Attack**

**\_Detection\_in\_IoT\_Networks**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT** |  |
|  | **INTRODUCTION** |  |
|  | * 1. General Introduction |  |
|  | * 1. Project Objectives |  |
|  | **SYSTEM** **PROPOSAL** |  |
|  | * 1. Existing System |  |
|  | * + 1. Disadvantages |  |
|  | * 1. Proposed System |  |
|  | 2.2.1 Advantages |  |
|  | * 1. Literature Survey |  |
|  | **SYSTEM DIAGRAMS** |  |
|  | * 1. Architecture Diagram |  |
|  | * 1. Flow Diagram |  |
|  | * 1. UML Diagrams |  |
|  | **IMPLEMENTATION** |  |
|  | * 1. Modules |  |
|  | * 1. Modules Description |  |
|  | **SYSTEM** **REQUIREMENTS** |  |
|  | * 1. Hardware Requirements |  |
|  | * 1. Software Requirements |  |
|  | * 1. Software Description |  |
|  | * 1. Testing of Products |  |
|  | **CONCLUSION** |  |
| **7.** | **FUTURE** **ENHANCEMENT** |  |
| **8.** | **SAMPLE** **CODING** |  |
| **9.** | **SAMPLE** **SCREENSHOT** |  |
| **10.** | **REFERENCES** |  |

**ABSTRACT**

* Cyber Attack is one of the important security in today’s cyber world. Lot of techniques have been developed which are based on machine learning approaches.
* So for identifying the intrusion we have designed the different machine learning algorithms such as CNN, LSTM, Ensemble CNN and LSTM. The experimental results shows that the accuracy algorithms.
* A Network based Intrusion Detection System (NIDS) is usually placed at network points such as a gateway and routers to check for intrusions in the network traffic.
* In this process, CICIDS-2017 was taken from dataset repository. Then, we have to implement the pre-processing, data splitting and classification.

**CHAPTER 1**

**INTRODUCTION**

* 1. **General Introduction:**

Hacking incidents are increasing day by day as technology rolls out. A large number of hacking incidents are reported by companies each year. Distributed Denial of Service (DDoS) attack was launched against Estonian websites in 2007.On June 17, 2008, Amazon started receiving some authenticated request from multiple users in one of its location. The requests began to increase significantly causing the servers slow down. On Jan 2013, European Network and Information Security Agency (ENISA) reported that Dropbox was attacked by DDoS and suffered a substantial loss of service for more than 15 hours affecting all users across the globe. Facebook was hit by suspected distributed denial of service attack on Sept 28, 2014. Reported that some form of network scanning activity precedes 50% of the attacks against cyber systems. Attackers are not only launching flooding and probing attacks but also spreading malware files in the form of virus, worm, spams to exploit the vulnerabilities present in existing software, causing a threat to the sensitive information of users stored on machines. Cisco Annual Security report mentioned that spam related to the Boston Marathon bombing comprised 40% of all spam messages delivered worldwide on April 17, 2013. On a recent survey done by Cisco in 2017, Trojan was classified as one of the top five malware which is used to gain initial access to the user’s computers and organizational networks. Hence, security in such a complex technological environment is a big challenge and needs to be tackled intelligently. Researchers have considered a different category of attacks for intrusion detection. For example, Denial of Service (DoS) attacks (Bandwidth and Resource Depletion), Scanning attacks (Probe) and Remote to Local (R2L) attacks and User to Root (U2R) attacks which are based on KDD’99 dataset. A recent attack dataset (UNSW-NB), classifies attacks into nine categories: Fuzzer, Analysis, Reconnaissance, ShellCode, Worm, Generic, DoS, Exploit and Generic. All these attacks have been discussed in detail in Section III. Current security solutions include the use of middle-boxes such as Firewall, Antivirus and Intrusion Detection Systems (IDS). A firewall controls traffic that enters or leaves a network based on source or destination address. It alters the traffic according to the firewall rules. Firewalls are also limited to the amount of state available and their knowledge of the hosts receiving the content. An IDS is a type of security tool that monitors network traffic and scans the system for suspicious activities and alerts the system or network administrator. A Network based Intrusion Detection System (NIDS) is usually placed at network points such as a gateway and routers to check for intrusions in the network traffic. At high-level, the detection mechanism used by these IDSes are of three types: misuse detection, anomaly detection, and hybrid detection. In misuse detection approach, IDS maintains a set of the knowledge base (rules) for detecting the known attack types. Misuse detection techniques can be broadly classified into Knowledge based and machine learning based techniques. In the knowledge based technique, network traffic or host audit data (such as system call traces) are compared against predefined rules or attack patterns. Knowledge based techniques can be categorized into three types: (i) Signature matching (ii) State transition analysis and (iii) Rule based expert systems.

* 1. **Objectives:**

The main objective of our project is,

* To classify or predict the attacks effectively for the dataset.
* To implement the different machine learning algorithms for comparing the better performances.
* To enhance the overall performance for classification algorithms by means of the accuracy.

**CHAPTER 2**

**SYSTEM PROPOSAL**

**2.1 EXISTING SYSTEM:**

In existing system, our main focus is on Network Intrusion Detection Systems (NIDS); hence, this paper reviews existing NIDS implementation tools and datasets as well as free & open-source network sniffing software. Then, it surveys, analyses and compares state-of-the-art NIDS proposals in the IoT context in terms of architecture, detection methodologies, validation strategies, treated threats and algorithm deployments. The review deals with both traditional and machine learning (ML) NIDS techniques and discusses future directions. In this survey, our focus is on IoT NIDS deployed via Machine Learning since learning algorithms have a good success rate in security and privacy. The survey provides a comprehensive review of NIDSs deploying different aspects of learning techniques for Internet of Things, unlike other top surveys targeting the traditional systems. We believe that, the paper will be useful for academia and industry research, first, to identify IoT threats and challenges, second, to implement their own NIDS and finally to propose new smart techniques in IoT - context considering IoT limitations. Moreover, the survey will enable security individuals differentiate IoT NIDS from traditional ones.

**2.1.1 DISADVANTAGES:**

* The results is low when compared with proposed
* It doesn’t efficient for large volume of data’s
* It didn’t implement the deep learning algorithm.
* Theoretical limits.
  1. **PROPOSED SYSTEM:**
* In this system, CICIDS-2017 dataset was taken as input. The input data was taken from the dataset repository. Then, we have to implement the data pre-processing step.
* In this step, we have to handle the missing values for avoid wrong prediction, and to encode the label for input data. Then, we have to split the dataset into test and train.
* Training portion is used to evaluate the model and testing portion is used to predicting the model.
* The algorithms such as CNN, LSTM, Ensemble CNN and LSTM neighbour for predicting and classifying the attacks.
* Finally, the experimental results shows that the performance metrics such as accuracy, precision and recall.
* Finally, the experimental results shows that the performance metrics such as accuracy, precision and recall.

**2.2.1 ADVANTAGES:**

* It is efficient for large number of datasets.
* The experimental result is high when compared with existing system.
* Time consumption is low.

**2.3 LITERATURE SURVEY:**

**2.3.1 The impact of adversarial attacks on interpretable semantic segmentation in cyber–physical systems, 2023**

***Author****:* R. Gipiškis, D. Chiaro, M. Preziosi, E. Prezioso, and F. Piccialli

***Methodology*:**

Network attacks refer to malicious activities exploiting computer network vulnerabilities to compromise security, disrupt operations, or gain unauthorized access to sensitive information. Common network attacks include phishing, malware distribution, and brute-force attacks on network devices and user credentials. Such attacks can lead to financial losses due to downtime, recovery costs, and potential legal liabilities. To counter such threats, organizations use Intrusion Detection Systems (IDS) that leverage sophisticated algorithms and machine learning techniques to detect network attacks with enhanced accuracy and efficiency.

***Disadvantage*:**

* The main disadvantages of various feature learning systems is their complexity and are expensive to implement.

**2.3.2 Intelligent detection system for a distributed denial-of–service (DDoS) attack based on time series, 2023**

***Author***: M. S. I. Alsumaidaie, K. M. A. Alheeti, and A. K. Al-Aloosy

***Methodology*:**

Cyberattacks are a rising threat to companies and people. The Distributed Denial of Service (DDoS) attack is one of the destructive hacks that have swiftly acquired appeal among hackers. In this work, a security system is proposed to prevent DDoS. In other words, it has the ability to protect external and internal communication systems from attacks. The primary contribution of this work is to acquire the best accuracy based on time series. Multiple machine learning algorithms are applied and compared between them.

***Advantage:***

* The advantage of this system is malicious network events using IDS signatures and follows their development as consecutive events, finding matches in terms of IP address or port.

***Disadvantage:***

* An attacker does not need to follow a precise order for executing a multi-step attack, so the set of possible sequences of actions can be very complex.

**2.3.3 A novel optimization based deep learning with artificial intelligence approach to detect intrusion attack in network system, 2023**

***Author*:** S. Siva Shankar, B. T. Hung, P. Chakrabarti, T. Chakrabarti, and G. Parasa

***Methodology*:**

Networks are up against detecting dynamic and unknown threats. Anomaly-based neural network (NN) intrusion detection systems (IDSs) can manage this if trained and tested accordingly. This requires the IDS to be evaluated on how well it can detect these intrusions. Evaluating NN IDSs can be a complex and difficult task. One needs to be able to measure the convergence rate and performance (detection and failure) rate of the IDS. This paper explores the different methods used by researchers to train and test their IDS models. It also found that the data used can effect the results of training and testing the NN IDS models.. Finally, we conduct an in-depth analysis for Shodan scans and evaluate the impact of Shodan on industrial control systems in terms of scanning time, scanning frequency, scanning port, region preferences, ICS protocol preferences and ICS protocol function code proportion. Accordingly, we provide some defensive measures to mitigate Shodan threat

Modern life is increasingly inﬂuenced by networks, making cybersecurity a crucial

area of study. However, due to their few resources and varied makeup, they are more

vulnerable to a wide range of cyber-attacks. Such risks result in sensitive informa-

tion being stolen as well as ﬁnancial and reputational harm to ﬁrms. How far mali-

cious detection techniques have advanced in the intrusion detection system (IDS)

industry is diﬃcult to quantify. Therefore, a unique IDS known optimized Artiﬁ-

cial Intelligence approach is designed to eﬀectively identify the intrusions. The pre-

processing activity is initially carried out to improve the data quality for identifying

network intrusions utilizing normalization and standardization. The Corporate Hier-

archy optimization (CHO) technique is then used to choose the important features

from the databases. By accurately identifying intrusions, the suggested golden eagle

optimization-based Self-constructing Multi-layer Perceptron Interfaced Fuzzy sys-

tem (GEO-SMPIF) solution improves privacy and security within the professional

network infrastructure. The ensuing hyperparameters are adjusted optimally using

GEO methods during parameter identiﬁcation, and a backpropagation technique

tunes the precondition parameters

Modern life is increasingly inﬂuenced by networks, making cybersecurity a crucial

area of study. However, due to their few resources and varied makeup, they are more

vulnerable to a wide range of cyber-attacks. Such risks result in sensitive informa-

tion being stolen as well as ﬁnancial and reputational harm to ﬁrms. How far mali-

cious detection techniques have advanced in the intrusion detection system (IDS)

industry is diﬃcult to quantify. Therefore, a unique IDS known optimized Artiﬁ-

cial Intelligence approach is designed to eﬀectively identify the intrusions. The pre-

processing activity is initially carried out to improve the data quality for identifying

network intrusions utilizing normalization and standardization. The Corporate Hier-

archy optimization (CHO) technique is then used to choose the important features

from the databases. By accurately identifying intrusions, the suggested golden eagle

optimization-based Self-constructing Multi-layer Perceptron Interfaced Fuzzy sys-

tem (GEO-SMPIF) solution improves privacy and security within the professional

network infrastructure. The ensuing hyperparameters are adjusted optimally using

GEO methods during parameter identiﬁcation, and a backpropagation technique

tunes the precondition parameters

a unique IDS known optimized Artiﬁ-

cial Intelligence approach is designed to eﬀectively identify the intrusions. The pre-

processing activity is initially carried out to improve the data quality for identifying

network intrusions utilizing normalization and standardization. The Corporate Hier-

archy optimization (CHO) technique is then used to choose the important features

from the databases. By accurately identifying intrusions, the suggested golden eagle

optimization-based Self-constructing Multi-layer Perceptron Interfaced Fuzzy sys-

tem (GEO-SMPIF) solution improves privacy and security within the professional

network infrastructure. The ensuing hyperparameters are adjusted optimally using

GEO methods during parameter identiﬁcation, and a backpropagation technique

tunes the precondition parameters

a unique IDS known optimized Artiﬁ-

cial Intelligence approach is designed to eﬀectively identify the intrusions. The pre-

processing activity is initially carried out to improve the data quality for identifying

network intrusions utilizing normalization and standardization. The Corporate Hier-

archy optimization (CHO) technique is then used to choose the important features

from the databases. By accurately identifying intrusions, the suggested golden eagle

optimization-based Self-constructing Multi-layer Perceptron Interfaced Fuzzy sys-

tem (GEO-SMPIF) solution improves privacy and security within the professional

network infrastructure. The ensuing hyperparameters are adjusted optimally using

GEO methods during parameter identiﬁcation, and a backpropagation technique

tunes the precondition parameters

Modern life is increasingly inﬂuenced by networks, making cybersecurity a crucial

area of study. However, due to their few resources and varied makeup, they are more

vulnerable to a wide range of cyber-attacks. Such risks result in sensitive informa-

tion being stolen as well as ﬁnancial and reputational harm to ﬁrms. How far mali-

cious detection techniques have advanced in the intrusion detection system (IDS)

industry is diﬃcult to quantify. Therefore, a unique IDS known optimized Artiﬁ-

cial Intelligence approach is designed to eﬀectively identify the intrusions. The pre-

processing activity is initially carried out to improve the data quality for identifying

network intrusions utilizing normalization and standardization. The Corporate Hier-

archy optimization (CHO) technique is then used to choose the important features

from the databases. By accurately identifying intrusions, the suggested golden eagle

optimization-based Self-constructing Multi-layer Perceptron Interfaced Fuzzy sys-

tem (GEO-SMPIF) solution improves privacy and security within the professional

network infrastructure. The ensuing hyperparameters are adjusted optimally using

GEO methods during parameter identiﬁcation, and a backpropagation technique

tunes the precondition parameter

a unique IDS known optimized Artificial Intelligence approach is designed to effectively identify the intrusions. The preprocessing activity is initially carried out to improve the data quality for identifying network intrusions utilizing normalization and standardization. The Corporate Hierarchy optimization (CHO) technique is then used to choose the important features from the databases. By accurately identifying intrusions, the suggested golden eagle optimization-based Self-constructing Multi-layer Perceptron Interfaced Fuzzy system (GEO-SMPIF) solution improves privacy and security within the professional network infrastructure. The ensuing hyperparameters are adjusted optimally using GEO methods during parameter identification, and a backpropagation technique tunes the precondition parameter The exponential rise in internet technologies and allied

applications encompass a significantly large number of networked

devices have alarmed academia-industries to achieve more effective

and robust security solutions. Undeniably, digitization has led to

revolution globally; however, the security threats, breaches, and

subsequent losses indicate the need for a robust cybersecurity

solution. Unlike classical intrusion detection systems (IDS), network

IDS (NIDS) has been becoming more challenging due to continuous

changes in attack-patterns and anomaly behavior. As solution data-

driven machine learning methods have exhibited better by learning

over network traffic information and detecting anomalies; however,

its generalization over a network with both known and unknown

patterns remains questionable. Moreover, most of the classical

approaches fail to address the key issues of class-imbalance, level-of-

significance centric feature selection, normalization and over-fitting

problems resulting in different performance by varied machine

learning models. In this paper, a novel and robust heterogeneous

ensemble machine learning model is developed to detect anomalies

in NIDS. The proposed model first applies sub-sampling to alleviate

the class-imbalance problem of NIDS datasets. Subsequently,

performing normalization using the Min-Max algorithm, it mapped

the input data in the range of 0 to 1, thus alleviating overfitting and

convergence. The feature reduction is used to reduce the features; it

retained the most suitable features without imposing computational

overheads, often in meta-heuristic-based approaches. Finally, the

proposed NIDS solution designed a Heterogeneous ensemble

learning model with J48, k-NN

The exponential rise in internet technologies and allied

applications encompass a significantly large number of networked

devices have alarmed academia-industries to achieve more effective

and robust security solutions. Undeniably, digitization has led to

revolution globally; however, the security threats, breaches, and

subsequent losses indicate the need for a robust cybersecurity

solution. Unlike classical intrusion detection systems (IDS), network

IDS (NIDS) has been becoming more challenging due to continuous

changes in attack-patterns and anomaly behavior. As solution data-

driven machine learning methods have exhibited better by learning

over network traffic information and detecting anomalies

The exponential rise in internet technologies and allied

applications encompass a significantly large number of networked

devices have alarmed academia-industries to achieve more effective

and robust security solutions. Undeniably, digitization has led to

revolution globally; however, the security threats, breaches, and

subsequent losses indicate the need for a robust cybersecurity

solution. Unlike classical intrusion detection systems (IDS), network

IDS (NIDS) has been becoming more challenging due to continuous

changes in attack-patterns and anomaly behavior. As solution data-

driven machine learning methods have exhibited better by learning

over network traffic information and detecting anomalies

The exponential rise in internet technologies and allied

applications encompass a significantly large number of networked

devices have alarmed academia-industries to achieve more effective

and robust security solutions. Undeniably, digitization has led to

revolution globally; however, the security threats, breaches, and

subsequent losses indicate the need for a robust cybersecurity

solution. Unlike classical intrusion detection systems (IDS), network

IDS (NIDS) has been becoming more challenging due to continuous

changes in attack-patterns and anomaly behavior. As solution data-

driven machine learning methods have exhibited better by learning

over network traffic information and detecting anomalies; however,

its generalization over a network with both known and unknown

patterns remains questionable. Moreover, most of the classical

approaches fail to address the key issues of class-imbalance, level-of-

significance centric feature selection, normalization and over-fitting

problems resulting in different perform

The exponential rise in internet technologies and allied

applications encompass a significantly large number of networked

devices have alarmed academia-industries to achieve more effective

and robust security solutions. Undeniably, digitization has led to

revolution globally; however, the security threats, breaches, and

subsequent losses indicate the need for a robust cybersecurity

solution. Unlike classical intrusion detection systems (IDS), network

IDS (NIDS) has been becoming more challenging due to continuous

changes in attack-patterns and anomaly behavior. As solution data-

driven machine learning methods have exhibited better by learning

over network traffic information and detecting anomalies; however,

its generalization over a network with both known and unknown

patterns remains questionable. Moreover, most of the classical

approaches fail to address the key issues of class-imbalance, level-of-

significance centric feature selection, normalization and over-fitting

problems resulting in different performance by varied machine

learning models. In this paper,

***Advantage:***

* The spark iterative computation architectural enables large-scale machine learning algorithms to achieve high level efficiency in results, and spark.ml API for pipeline offers developers with extensive range of new module to integrate with their architecture.

**2.3.4 Building Machine Learning and Deep Learning Models on Google Cloud Platform**, **2019**

***Author*:** E. Bisong and E. Bisong

***Methodology*:**

Deep learning extends the machine learning algorithm of neural networks to learn complex tasks that are difficult for computers to perform, such as recognizing faces and understanding languages. And you will know how to leverage cloud computing to accelerate data science and machine learning deployments. Building Machine Learning and Deep Learning Models on Google Cloud Platform is divided into eight parts that cover the fundamentals of machine learning and deep learning, the concept of data science and cloud services, programming for data science using the Python stack, Google Cloud Platform (GCP).

***Disadvantage***:

* The main Disadvantage of this problem of signature based method is that the database signature needs to be updated as the new signatures become available and therefore it is not suitable for the real-time network anomaly detection.
* It is somewhat difficult to compare the results of the seven algorithms discussed here using multiple performance metrics.

**2.3.5 On hyperparameter optimization of machine learning methods using a Bayesian optimization algorithm to predict work travel mode choice 2023**

***Author*:** M. Aghaabbasi, M. Ali, M. Jasinski, Z. Leonowicz, and T. Novák

***Methodology***:

Networks are up against detecting dynamic and unknown threats. Anomaly-based neural network (NN) intrusion detection systems (IDSs) can manage this if trained and tested accordingly. This requires the IDS to be evaluated on how well it can detect these intrusions. Evaluating NN IDSs can be a complex and difficult task. One needs to be able to measure the convergence rate and performance (detection and failure) rate of the IDS. This paper explores the different methods used by researchers to train and test their IDS models. It also found that the data used can effect the results of training and testing the NN IDS models.. Finally, we conduct an in-depth analysis for Shodan scans and evaluate the impact of Shodan on industrial control systems in terms of scanning time, scanning frequency, scanning port, region preferences, ICS protocol preferences and ICS protocol function code proportion. Accordingly, we provide some defensive measures to mitigate Shodan threat.

***Advantage***:

* The main advantage of SVM is a machine learning model with the advantages of high detection rate of small samples and strong generalization ability, which is suitable for handling high-dimensional and non-linear Shodan traffic from a small amount of Shodan scanners.

**2.3.6 A novel ensemble method for enhancing Internet of Things device security against botnet attacks, 2023**

***Author*:** A. Arshad, M. Jabeen, S. Ubaid, A. Raza, L. Abualigah, K. Aldiabat

***Methodology***:

The intricate neuroinflammatory diseases multiple sclerosis (MS) and neuromyelitis optica (NMO) often present similar clinical symptoms, creating challenges in their precise detection via magnetic resonance imaging (MRI). This challenge is further compounded when detecting the active and inactive states of MS. To address this diagnostic problem, we introduce an innovative framework that incorporates state-of-the-art machine learning algorithms applied to features culled from MRI scans by pre-trained deep learning models, VGG-NET and InceptionV3. To develop and test this methodology, we utilized a robust dataset obtained from the King Abdullah University Hospital in Jordan, encompassing cases diagnosed with both MS and NMO. We benchmarked thirteen distinct machine learning algorithms and discovered that support vector machine (SVM) and K-nearest neighbor (KNN) algorithms performed superiorly

***Advantage:***

* Statistical techniques do not require prior knowledge of network attacks.
* These can accurately detect the attacks which cause an abrupt and highly deviated changes in the network traffic e.g. DoS attacks.

***Disadvantage:***

* High dimensionality and variation in network traffic can affect the performance of statistical intrusion detection systems.
* It is difficult to compute the statistics of normal network traffic.

**2.3.7 Composition of hybrid deep learning model and feature optimization for intrusion detection system, 2023**

***Author*:** A. Henry, S. Gautam, S. Khanna, K. Rabie, T. Shongwe, P. Bhattacharya, B. Sharma, and S. Chowdhury

***Methodology***:

 In networks, Intrusion Detection Systems (IDSs) are employed to raise critical flags during network management. One aspect is malicious traffic identification, where zero-day attack detection is a critical problem of study. Current approaches are aligned towards deep learning (DL) methods for IDSs, but the success of the DL mechanism depends on the feature learning process, which is an open challenge. Thus, in this paper, the authors propose a technique which combines both CNN, and GRU, where different CNN–GRU combination sequences are presented to optimize the network parameters. In the simulation, the authors used the CICIDS-2017 benchmark dataset and used metrics such as precision, recall, False Positive Rate (FPR), True Positive Rate (TRP), and other aligned metrics.

***Advantage:***

* The main advantage is Heterogeneity among the actual Sensors, IDSs, Analyzers, or even SIEMs can be beneficial for Intrusion Detection where detection accuracy can be improved

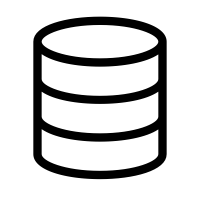
**CHAPTER 3**

**SYSTEM DIAGRAMS**

**3.1 SYSTEM ARCHITECTURE:**

Load the Dataset

Pre-processing



Prediction Phase

**Classification Algorithms**

Performance Metrics

Train Test Split

Trained Model

Test Data

CNN, LSTM Proposed Model

CICIDS-2017 Dataset

FIGURE 3.1: SYSTEM DIAGRAM

**3.2 FLOW DIAGRAM**

Input Data

Preprocessing

Data Splitting

Classification

Performance analysis

FIGURE 3.2: FLOW DIAGRAM

**3.3 UML DIAGRAMS:**

**3.3.1 USE CASE DIAGRAM:**

System

User

FIGURE 3.3.1: USE CASE DIAGRAM

**3.3.2 ACTIVITY DIAGRAM:**

Input Data

Preprocessing

Data splitting

Performance metrics

Classification

FIGURE 3.3.2: ACTIVITY DIAGRAM

**3.3.3 SEQUENCE DIAGRAM:**

Input Data

Preprocessing

Data splitting

Classification

Select data

Missing value

Test and Train

Load data

Data splitting

CNN , LSTM , Proposed

FIGURE 3.3.3: SEQUENCE DIAGRAM

**3.3.4 ER DIAGRAM:**

Data selection

Preprocessing

Data splitting

Classification

FIGURE 3.3.4: ER DIAGRAM

**3.3.5 CLASS DIAGRAM:**

Select data ()

Load data ()

View data ()

INPUT

Test ()

Data Splitting

Prediction ()

Performance analysis

Preprocessing

Missing values ()

Label encode ()

Normalize ()

Classification

CNN ()

LSTM ()

CNN-LSTM ()

Train ()

**3.3.6 Data Flow Diagram :**

**Level 0 :**

CICIDS-2017 Dataset

Intrusion Detection Classification

**Level 1 :**

CICIDS-2017 Dataset

Pre-processing

Train Test split

Classification Algorithms

Output

**Level 2 :**

CICIDS-2017 Dataset

Pre-processing

Train Test split

Classification Algorithms

Output

CNN , LSTM , CNN-LSTM

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 MODULES:**

* Data selection
* Preprocessing
* Data splitting
* Classification
* Result generation

**4.2 MODULES DESCRIPTION:**

**4.2.1: DATA SELECTION:**

* The input data was collected from dataset repository.
* In our process, the **CICIDS-2017**dataset is used.
* The data selection is the process of detecting the attacks.
* The input dataset was taken from dataset repository such as UCI repository.
* The dataset contains the information such protocol, duration, host error rate, label and so on.

**4.2.2: DATA PREPROCESSING:**

* Data pre-processing is the process of removing the unwanted data from the dataset.
* Pre-processing data transformation operations are used to transform the dataset into a structure suitable for machine learning.
* This step also includes cleaning the dataset by removing irrelevant or corrupted data that can affect the accuracy of the dataset, which makes it more efficient.
* Missing data removal
* Encoding Categorical data
* Missing data removal: In this process, the null values such as missing values and Nan values are replaced by 0.
* Missing and duplicate values were removed and data was cleaned of any abnormalities.
* Encoding Categorical data: That categorical data is defined as variables with a finite set of label values.
* That most machine learning algorithms require numerical input and output variables.

**4.2.3: DATA SPLITTING:**

* During the machine learning process, data are needed so that learning can take place.
* In addition to the data required for training, test data are needed to evaluate the performance of the algorithm in order to see how well it works.
* In our process, we considered 70% of the input dataset to be the training data and the remaining 30% to be the testing data.
* Data splitting is the act of partitioning available data into two portions, usually for cross-validator purposes.
* One Portion of the data is used to develop a predictive model and the other to evaluate the model's performance.
* Separating data into training and testing sets is an important part of evaluating data mining models.
* Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing.

**4.2.4: CLASSIFICATION:**

In our process, we have to implement the machine learning algorithm such as CNN, LSTM, and Integrate CNN AND LSTM.

* **CNN** – A Convolutional Neural Network (CNN) is a deep learning algorithm designed for image analysis. It uses convolutional layers to detect patterns, pooling layers to reduce dimensionality, and fully connected layers for classification. CNNs are highly effective for tasks like image recognition and object detection.
* **LSTM- Long** Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) designed to handle long-term dependencies. It uses memory cells, gates (input, forget, and output) to control information flow, making it effective for sequence prediction tasks like language modeling and time-series forecasting.
* **Proposed:** Integrating CNN and LSTM combines their strengths: CNNs extract spatial features from image data, while LSTMs capture temporal dependencies. This hybrid model processes input sequences, such as video frames, by first using CNN layers for feature extraction, followed by LSTM layers to analyze the sequential information, enhancing tasks like video analysis.

**4.2.5: RESULT GENERATION:**

The Final Result will get generated based on the overall classification and prediction. The performance of this proposed approach is evaluated using some measures like,

* **Accuracy**

Accuracy of classifier refers to the ability of classifier. It predicts the class label correctly and the accuracy of the predictor refers to how well a given predictor can guess the value of predicted attribute for a new data.

AC= (TP+TN)/ (TP+TN+FP+FN)

* **Precision**

Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives.

Precision=TP/ (TP+FP)

* **Recall**

Recall is the number of correct results divided by the number of results that should have been returned. In binary classification, recall is called sensitivity. It can be viewed as the probability that a relevant document is retrieved by the query.

Recall=TP/ (TP+FN)

**CHAPTER 5**

**SYSTEM REQUIREMENTS**

**5.1 HARDWARE REQUIREMENTS:**

* System : Pentium IV 2.4 GHz
* Hard Disk : 200 GB
* Mouse : Logitech.
* Keyboard : 110 keys enhanced
* Ram : 4GB

**5.2 SOFTWARE REQUIREMENTS:**

* O/S : Windows 7.
* Language : Python
* Front End : Colab

**5.3 SOFTWARE DESCRIPTION:**

**5.3.1 Python**

Python is one of those rare languages which can claim to be both *simple* and powerful. You will find yourself pleasantly surprised to see how easy it is to concentrate on the solution to the problem rather than the syntax and structure of the language you are programming in. The official introduction to Python is Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. I will discuss most of these features in more detail in the next section.

## **5.3.2 Features of Python**

### **Simple**

Python is a simple and minimalistic language. Reading a good Python program feels almost like reading English, although very strict English! This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the language itself.

### **Easy to Learn**

As you will see, Python is extremely easy to get started with. Python has an extraordinarily simple syntax, as already mentioned.

### **Free and Open Source**

Python is an example of a FLOSS (Free/Libré and Open Source Software). In simple terms, you can freely distribute copies of this software, read its source code, make changes to it, and use pieces of it in new free programs. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and is constantly improved by a community who just want to see a better Python.

### **High-level Language**

When you write programs in Python, you never need to bother about the low-level details such as managing the memory used by your program, etc.

### **Portable**

Due to its open-source nature, Python has been ported to (i.e. changed to make it work on) many platforms. All your Python programs can work on any of these platforms without requiring any changes at all if you are careful enough to avoid any system-dependent features.

You can use Python on GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC!

You can even use a platform like [Kivy](http://kivy.org) to create games for your computer and for iPhone, iPad, and Android.

### **Interpreted**

This requires a bit of explanation.

A program written in a compiled language like C or C++ is converted from the source language i.e. C or C++ into a language that is spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from hard disk to memory and starts running it.

Python, on the other hand, does not need compilation to binary. You just run the program directly from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your computer and then runs it. All this, actually, makes using Python much easier since you don't have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc. This also makes your Python programs much more portable, since you can just copy your Python program onto another computer and it just works!

### **Object Oriented**

Python supports procedure-oriented programming as well as object-oriented programming. In procedure-oriented languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In object-oriented languages, the program is built around objects which combine data and functionality. Python has a very powerful but simplistic way of doing OOP, especially when compared to big languages like C++ or Java.

### **Extensible**

If you need a critical piece of code to run very fast or want to have some piece of algorithm not to be open, you can code that part of your program in C or C++ and then use it from your Python program.

### **Embeddable**

You can embed Python within your C/C++ programs to give scripting capabilities for your program's users.

### **Extensive Libraries**

The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, FTP, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), and other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the Batteries Included philosophy of Python.

Besides the standard library, there are various other high-quality libraries which you can find at the Python Package Index.

**5.4 TESTING PRODUCTS:**

System testing is the stage of implementation, which aimed at ensuring that system works accurately and efficiently before the live operation commence. Testing is the process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an error. A successful test is one that answers a yet undiscovered error.

Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. . A series of tests are performed before the system is ready for the user acceptance testing. Any engineered product can be tested in one of the following ways. Knowing the specified function that a product has been designed to from, test can be conducted to demonstrate each function is fully operational. Knowing the internal working of a product, tests can be conducted to ensure that “al gears mesh”, that is the internal operation of the product performs according to the specification and all internal components have been adequately exercised.

**5.4.1 UNIT TESTING:**

Unit testing is the testing of each module and the integration of the overall system is done. Unit testing becomes verification efforts on the smallest unit of software design in the module. This is also known as ‘module testing’.

The modules of the system are tested separately. This testing is carried out during the programming itself. In this testing step, each model is found to be working satisfactorily as regard to the expected output from the module. There are some validation checks for the fields. For example, the validation check is done for verifying the data given by the user where both format and validity of the data entered is included. It is very easy to find error and debug the system.

**5.4.2 INTEGRATION TESTING:**

Data can be lost across an interface, one module can have an adverse effect on the other sub function, when combined, may not produce the desired major function. Integrated testing is systematic testing that can be done with sample data. The need for the integrated test is to find the overall system performance. There are two types of integration testing. They are:

i) Top-down integration testing. ii) Bottom-up integration testing.

**5.4.3 TESTING TECHNIQUES/STRATEGIES:**

* **WHITE BOX TESTING:**

White Box testing is a test case design method that uses the control structure of the procedural design to drive cases. Using the white box testing methods, we

Derived test cases that guarantee that all independent paths within a module have been exercised at least once.

* **BLACK BOX TESTING:**

1. Black box testing is done to find incorrect or missing function
2. Interface error
3. Errors in external database access
4. Performance errors.
5. Initialization and termination errors

In ‘functional testing’, is performed to validate an application conforms to its specifications of correctly performs all its required functions. So this testing is also called ‘black box testing’. It tests the external behaviour of the system. Here the engineered product can be tested knowing the specified function that a product has been designed to perform, tests can be conducted to demonstrate that each function is fully operational.

**5.4.4 SOFTWARE TESTING STRATEGIES**

**VALIDATION TESTING:**

After the culmination of black box testing, software is completed assembly as a package, interfacing errors have been uncovered and corrected and final series of software validation tests begin validation testing can be defined as many,

But a single definition is that validation succeeds when the software functions in a manner that can be reasonably expected by the customer

**USER ACCEPTANCE TESTING:**

User acceptance of the system is the key factor for the success of the system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system at the time of developing changes whenever required.

**OUTPUT TESTING**:

After performing the validation testing, the next step is output asking the user about the format required testing of the proposed system, since no system could be useful if it does not produce the required output in the specific format. The output displayed or generated by the system under consideration. Here the output format is considered in two ways. One is screen and the other is printed format. The output format on the screen is found to be correct as the format was designed in the system phase according to the user needs. For the hard copy also output comes out as the specified requirements by the user. Hence the output testing does not result in any connection in the system.

**CHAPTER 6**

**CONCLUSION**

We conclude that, the cicids2017 dataset was taken as input. The input dataset was mentioned in our research paper.We are implemented the classification algorithms (i.e) machine learning algorithms. Then, machine learning algorithms such as . Finally, the result shows that the accuracy for above mentioned algorithm and estimated the performances metrics such as accuracy, precision, recall and f1 score.

**CHAPTER 7**

**FUTURE ENHANCEMENT**

In the future, we should like to hybrid the two different machine learning or to hybrid two deep learning algorithms. In future, it is possible to provide extensions or modifications to the proposed clustering and classification algorithms to achieve further increased performance. Apart from the experimented combination of data mining techniques, further combinations and other clustering algorithms can be used to improve the detection accuracy. Finally, the sentiment analysis detection system can be extended as a prevention system to enhance the performance of the system.

**CHAPTER 8**

**SAMPLE CODING**

**#======================= IMPORT PACKAGES =============================**

**#======================================================================================**

**"Importing Libraries"**

**#======================================================================================**

**#=====================================================================================**

**"Importing Libraries"**

**#=====================================================================================**

**import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import LabelEncoder, MinMaxScaler**

**from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_curve, auc**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**import torch**

**import torch.nn as nn**

**import torch.nn.functional as F**

**from torch.utils.data import DataLoader, TensorDataset**

**# Load the dataset**

**df = pd.read\_csv('Friday-WorkingHours-Morning.pcap\_ISCX.csv')**

**# Encode labels**

**encoder = LabelEncoder()**

**df[' Label'] = encoder.fit\_transform(df[' Label'])**

**# Handle missing and infinite values**

**df = df.fillna(0)**

**df = df.replace([np.inf, -np.inf], 0)**

**# Convert dataframe to integer type**

**df = df.astype(int)**

**print(df)**

**# Check for missing values**

**print("Missing values in the dataset:", df.isnull().sum().sum())**

**# Prepare the data for training**

**x = df.drop([' Label'], axis=1).values**

**y = df[' Label'].values**

**# Split the data into training and testing sets**

**x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)**

**print("Shape of x\_train:", x\_train.shape)**

**print("Shape of x\_test:", x\_test.shape)**

**print("Shape of y\_train:", y\_train.shape)**

**print("Shape of y\_test:", y\_test.shape)**

**# Scale the features**

**scaler = MinMaxScaler()**

**x\_train = scaler.fit\_transform(x\_train)**

**x\_test = scaler.transform(x\_test)**

**# Convert to PyTorch tensors**

**x\_train\_tensor = torch.tensor(x\_train, dtype=torch.float32)**

**y\_train\_tensor = torch.tensor(y\_train, dtype=torch.long)**

**x\_test\_tensor = torch.tensor(x\_test, dtype=torch.float32)**

**y\_test\_tensor = torch.tensor(y\_test, dtype=torch.long)**

**# Create DataLoader**

**train\_dataset = TensorDataset(x\_train\_tensor, y\_train\_tensor)**

**test\_dataset = TensorDataset(x\_test\_tensor, y\_test\_tensor)**

**train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True)**

**test\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=False)**

**# Define CNN model**

**class CNN(nn.Module):**

**def \_\_init\_\_(self):**

**super(CNN, self).\_\_init\_\_()**

**self.conv1 = nn.Conv1d(in\_channels=1, out\_channels=16, kernel\_size=3, stride=1, padding=1)**

**self.conv2 = nn.Conv1d(in\_channels=16, out\_channels=32, kernel\_size=3, stride=1, padding=1)**

**self.pool = nn.MaxPool1d(kernel\_size=2, stride=2, padding=0)**

**self.fc1 = None**

**self.fc2 = nn.Linear(128, 64)**

**self.fc3 = nn.Linear(64, len(np.unique(y))) # Number of output classes**

**def forward(self, x):**

**x = self.pool(F.relu(self.conv1(x)))**

**x = self.pool(F.relu(self.conv2(x)))**

**x = x.view(x.size(0), -1) # Flatten the tensor**

**if self.fc1 is None:**

**self.fc1 = nn.Linear(x.size(1), 128).to(x.device)**

**x = F.relu(self.fc1(x))**

**x = F.relu(self.fc2(x))**

**x = self.fc3(x)**

**return x**

**# Instantiate the model, define the loss function and the optimizer**

**model = CNN()**

**criterion = nn.CrossEntropyLoss()**

**optimizer = torch.optim.Adam(model.parameters(), lr=0.001)**

**# Lists to store metrics**

**train\_losses = []**

**train\_accuracies = []**

**val\_accuracies = []**

**# Train the model**

**num\_epochs = 10**

**for epoch in range(num\_epochs):**

**model.train()**

**running\_loss = 0.0**

**correct\_train = 0**

**total\_train = 0**

**for inputs, labels in train\_loader:**

**inputs = inputs.unsqueeze(1) # Add channel dimension**

**optimizer.zero\_grad()**

**outputs = model(inputs)**

**loss = criterion(outputs, labels)**

**loss.backward()**

**optimizer.step()**

**running\_loss += loss.item() \* inputs.size(0)**

**\_, predicted = torch.max(outputs, 1)**

**total\_train += labels.size(0)**

**correct\_train += (predicted == labels).sum().item()**

**epoch\_loss = running\_loss / len(train\_loader.dataset)**

**train\_accuracy = correct\_train / total\_train**

**# Validate the model**

**model.eval()**

**correct\_val = 0**

**total\_val = 0**

**with torch.no\_grad():**

**for inputs, labels in test\_loader:**

**inputs = inputs.unsqueeze(1) # Add channel dimension**

**outputs = model(inputs)**

**\_, predicted = torch.max(outputs, 1)**

**total\_val += labels.size(0)**

**correct\_val += (predicted == labels).sum().item()**

**val\_accuracy = correct\_val / total\_val**

**# Append metrics to lists**

**train\_losses.append(epoch\_loss)**

**train\_accuracies.append(train\_accuracy)**

**val\_accuracies.append(val\_accuracy)**

**print(f"Epoch {epoch+1}/{num\_epochs}, Loss: {epoch\_loss:.4f}, Train Accuracy: {train\_accuracy:.4f}, Val Accuracy: {val\_accuracy:.4f}")**

**# Evaluate the model**

**model.eval()**

**y\_pred = []**

**y\_true = []**

**with torch.no\_grad():**

**for inputs, labels in test\_loader:**

**inputs = inputs.unsqueeze(1) # Add channel dimension**

**outputs = model(inputs)**

**\_, preds = torch.max(outputs, 1)**

**y\_pred.extend(preds.numpy())**

**y\_true.extend(labels.numpy())**

**# Calculate accuracy**

**accuracy = accuracy\_score(y\_true, y\_pred)**

**print(f"Accuracy: {accuracy:.4f}")**

**# Classification report**

**print("Classification Report:")**

**print(classification\_report(y\_true, y\_pred))**

**# Plot training loss**

**plt.figure()**

**plt.plot(range(1, num\_epochs+1), train\_losses, label="Training Loss")**

**plt.xlabel("Epoch")**

**plt.ylabel("Loss")**

**plt.title("Training Loss Over Epochs")**

**plt.legend()**

**plt.show()**

**# Plot training and validation accuracy**

**plt.figure()**

**plt.plot(range(1, num\_epochs+1), train\_accuracies, label="Training Accuracy")**

**plt.plot(range(1, num\_epochs+1), val\_accuracies, label="Validation Accuracy")**

**plt.xlabel("Epoch")**

**plt.ylabel("Accuracy")**

**plt.title("Training and Validation Accuracy Over Epochs")**

**plt.legend()**

**plt.show()**

**#==================================="LSTM"**

**# Define LSTM model**

**class LSTMModel(nn.Module):**

**def \_\_init\_\_(self, input\_size, hidden\_size, num\_layers, num\_classes):**

**super(LSTMModel, self).\_\_init\_\_()**

**self.hidden\_size = hidden\_size**

**self.num\_layers = num\_layers**

**self.lstm = nn.LSTM(input\_size, hidden\_size, num\_layers, batch\_first=True)**

**self.fc = nn.Linear(hidden\_size, num\_classes)**

**def forward(self, x):**

**h0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(x.device)**

**c0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(x.device)**

**out, \_ = self.lstm(x, (h0, c0))**

**out = out[:, -1, :]**

**out = self.fc(out)**

**return out**

**# Parameters**

**input\_size = x\_train.shape[1]**

**hidden\_size = 128**

**num\_layers = 2**

**num\_classes = len(np.unique(y))**

**# Instantiate the model, define the loss function and the optimizer**

**model = LSTMModel(input\_size, hidden\_size, num\_layers, num\_classes)**

**criterion = nn.CrossEntropyLoss()**

**optimizer = torch.optim.Adam(model.parameters(), lr=0.001)**

**# Lists to store metrics**

**train\_losses = []**

**train\_accuracies = []**

**val\_accuracies = []**

**# Train the model**

**num\_epochs = 10**

**for epoch in range(num\_epochs):**

**model.train()**

**running\_loss = 0.0**

**correct\_train = 0**

**total\_train = 0**

**for inputs, labels in train\_loader:**

**inputs = inputs.unsqueeze(1) # Add channel dimension**

**optimizer.zero\_grad()**

**outputs = model(inputs)**

**loss = criterion(outputs, labels)**

**loss.backward()**

**optimizer.step()**

**running\_loss += loss.item() \* inputs.size(0)**

**\_, predicted = torch.max(outputs, 1)**

**total\_train += labels.size(0)**

**correct\_train += (predicted == labels).sum().item()**

**epoch\_loss = running\_loss / len(train\_loader.dataset)**

**train\_accuracy = correct\_train / total\_train**

**# Validate the model**

**model.eval()**

**correct\_val = 0**

**total\_val = 0**

**with torch.no\_grad():**

**for inputs, labels in test\_loader:**

**inputs = inputs.unsqueeze(1) # Add channel dimension**

**outputs = model(inputs)**

**\_, predicted = torch.max(outputs, 1)**

**total\_val += labels.size(0)**

**correct\_val += (predicted == labels).sum().item()**

**val\_accuracy = correct\_val / total\_val**

**# Append metrics to lists**

**train\_losses.append(epoch\_loss)**

**train\_accuracies.append(train\_accuracy)**

**val\_accuracies.append(val\_accuracy)**

**print(f"Epoch {epoch+1}/{num\_epochs}, Loss: {epoch\_loss:.4f}, Train Accuracy: {train\_accuracy:.4f}, Val Accuracy: {val\_accuracy:.4f}")**

**# Evaluate the model**

**model.eval()**

**y\_pred = []**

**y\_true = []**

**with torch.no\_grad():**

**for inputs, labels in test\_loader:**

**inputs = inputs.unsqueeze(1) # Add channel dimension**

**outputs = model(inputs)**

**\_, preds = torch.max(outputs, 1)**

**y\_pred.extend(preds.numpy())**

**y\_true.extend(labels.numpy())**

**# Calculate accuracy**

**accuracy = accuracy\_score(y\_true, y\_pred)**

**print(f"Accuracy: {accuracy:.4f}")**

**# Classification report**

**print("Classification Report:")**

**print(classification\_report(y\_true, y\_pred))**

**# Plot training loss**

**plt.figure()**

**plt.plot(range(1, num\_epochs+1), train\_losses, label="Training Loss")**

**plt.xlabel("Epoch")**

**plt.ylabel("Loss")**

**plt.title("Training Loss Over Epochs LSTM Model")**

**plt.legend()**

**plt.show()**

**# Plot training and validation accuracy**

**plt.figure()**

**plt.plot(range(1, num\_epochs+1), train\_accuracies, label="Training Accuracy")**

**plt.plot(range(1, num\_epochs+1), val\_accuracies, label="Validation Accuracy")**

**plt.xlabel("Epoch")**

**plt.ylabel("Accuracy")**

**plt.title("Training and Validation Accuracy Over Epochs LSTM Model")**

**plt.legend()**

**plt.show()**

**#================== "Deep Ensemble Learning"**

**class CNN(nn.Module):**

**def \_\_init\_\_(self):**

**super(CNN, self).\_\_init\_\_()**

**self.conv1 = nn.Conv1d(in\_channels=1, out\_channels=16, kernel\_size=3, stride=1, padding=1)**

**self.conv2 = nn.Conv1d(in\_channels=16, out\_channels=32, kernel\_size=3, stride=1, padding=1)**

**self.pool = nn.MaxPool1d(kernel\_size=2, stride=2, padding=0)**

**self.fc1 = None**

**self.fc2 = nn.Linear(128, 64)**

**self.fc3 = nn.Linear(64, len(np.unique(y))) # Number of output classes**

**def forward(self, x):**

**x = self.pool(F.relu(self.conv1(x)))**

**x = self.pool(F.relu(self.conv2(x)))**

**x = x.view(x.size(0), -1) # Flatten the tensor**

**if self.fc1 is None:**

**self.fc1 = nn.Linear(x.size(1), 128).to(x.device)**

**x = F.relu(self.fc1(x))**

**x = F.relu(self.fc2(x))**

**x = self.fc3(x)**

**return x**

**class LSTMModel(nn.Module):**

**def \_\_init\_\_(self, input\_size, hidden\_size, num\_layers, num\_classes):**

**super(LSTMModel, self).\_\_init\_\_()**

**self.hidden\_size = hidden\_size**

**self.num\_layers = num\_layers**

**self.lstm = nn.LSTM(input\_size, hidden\_size, num\_layers, batch\_first=True)**

**self.fc = nn.Linear(hidden\_size, num\_classes)**

**def forward(self, x):**

**h0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(x.device)**

**c0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(x.device)**

**out, \_ = self.lstm(x, (h0, c0))**

**out = out[:, -1, :]**

**out = self.fc(out)**

**return out**

**# Parameters**

**input\_size = x\_train.shape[1]**

**hidden\_size = 128**

**num\_layers = 2**

**num\_classes = len(np.unique(y))**

**# Instantiate the model, define the loss function and the optimizer**

**model = LSTMModel(input\_size, hidden\_size, num\_layers, num\_classes)**

**criterion = nn.CrossEntropyLoss()**

**optimizer = torch.optim.Adam(model.parameters(), lr=0.001)**

**# Lists to store metrics**

**train\_losses = []**

**train\_accuracies = []**

**val\_accuracies = []**

**# Train the model**

**num\_epochs = 10**

**for epoch in range(num\_epochs):**

**model.train()**

**running\_loss = 0.0**

**correct\_train = 0**

**total\_train = 0**

**for inputs, labels in train\_loader:**

**inputs = inputs.unsqueeze(1) # Add channel dimension**

**optimizer.zero\_grad()**

**outputs = model(inputs)**

**loss = criterion(outputs, labels)**

**loss.backward()**

**optimizer.step()**

**running\_loss += loss.item() \* inputs.size(0)**

**\_, predicted = torch.max(outputs, 1)**

**total\_train += labels.size(0)**

**correct\_train += (predicted == labels).sum().item()**

**epoch\_loss = running\_loss / len(train\_loader.dataset)**

**train\_accuracy = correct\_train / total\_train**

**# Validate the model**

**model.eval()**

**correct\_val = 0**

**total\_val = 0**

**with torch.no\_grad():**

**for inputs, labels in test\_loader:**

**inputs = inputs.unsqueeze(1) # Add channel dimension**

**outputs = model(inputs)**

**\_, predicted = torch.max(outputs, 1)**

**total\_val += labels.size(0)**

**correct\_val += (predicted == labels).sum().item()**

**val\_accuracy = correct\_val / total\_val**

**# Append metrics to lists**

**train\_losses.append(epoch\_loss)**

**train\_accuracies.append(train\_accuracy)**

**val\_accuracies.append(val\_accuracy)**

**print(f"Epoch {epoch+1}/{num\_epochs}, Loss: {epoch\_loss:.4f}, Train Accuracy: {train\_accuracy:.4f}, Val Accuracy: {val\_accuracy:.4f}")**

**# Evaluate the model**

**model.eval()**

**y\_pred = []**

**y\_true = []**

**with torch.no\_grad():**

**for inputs, labels in test\_loader:**

**inputs = inputs.unsqueeze(1) # Add channel dimension**

**outputs = model(inputs)**

**\_, preds = torch.max(outputs, 1)**

**y\_pred.extend(preds.numpy())**

**y\_true.extend(labels.numpy())**

**# Calculate accuracy**

**accuracy = accuracy\_score(y\_true, y\_pred)**

**print(f"Accuracy: {accuracy:.4f}")**

**print("Classification Report:")**

**print(classification\_report(y\_true, y\_pred))**

**# Plot training loss**

**plt.figure()**

**plt.plot(range(1, num\_epochs+1), train\_losses, label="Training Loss")**

**plt.xlabel("Epoch")**

**plt.ylabel("Loss")**

**plt.title("Training Loss Over Epochs Proposed Model")**

**plt.legend()**

**plt.show()**

**# Plot training and validation accuracy**

**plt.figure()**

**plt.plot(range(1, num\_epochs+1), train\_accuracies, label="Training Accuracy")**

**plt.plot(range(1, num\_epochs+1), val\_accuracies, label="Validation Accuracy")**

**plt.xlabel("Epoch")**

**plt.ylabel("Accuracy")**

**plt.title("Training and Validation Accuracy Over Epochs Proposed Model")**

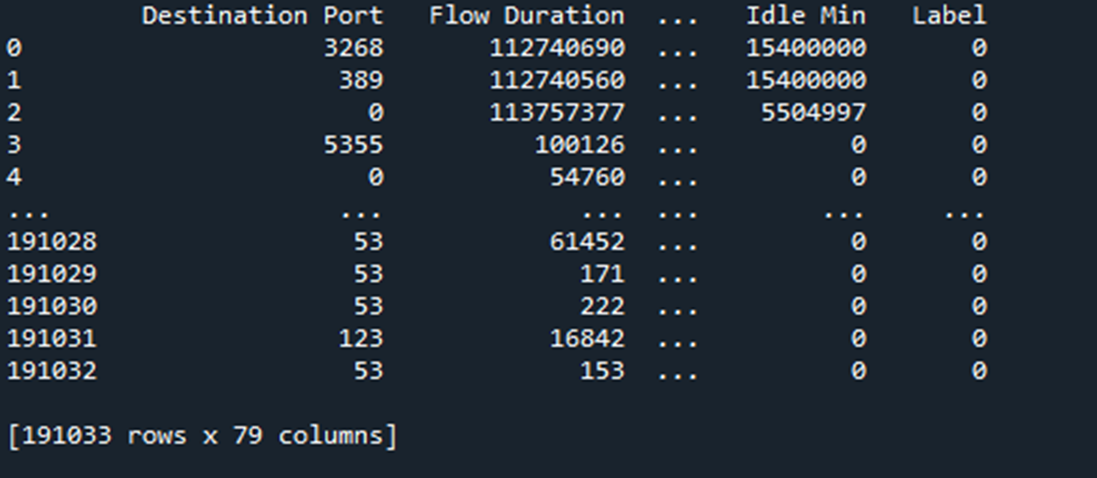
**plt.legend()**

**plt.show()**

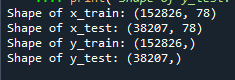
**CHAPTER 9**

**SAMPLE SCREENSHOTS**

**Data Selection:**

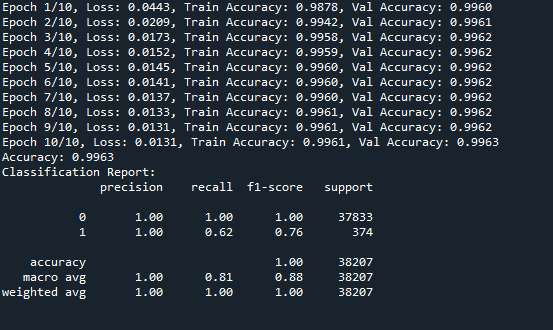


**Data Pre-processing:**

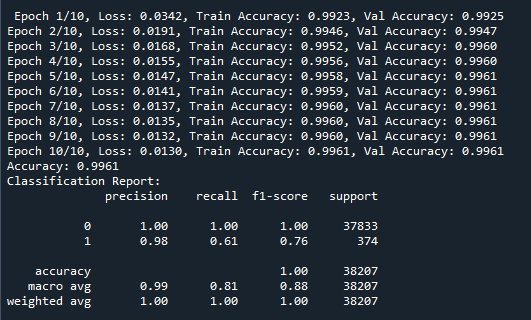


**Performance Metrics:**

**Proposed:**

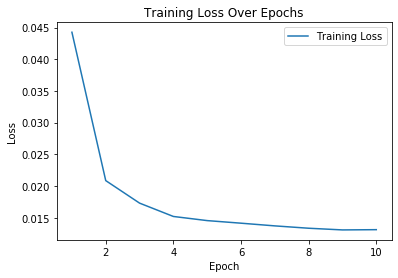


**CNN:**

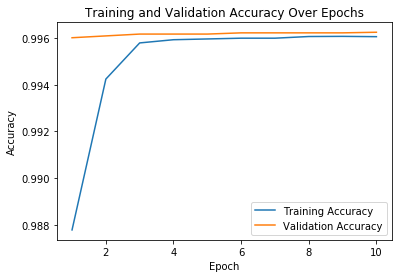


**Model Plot:**

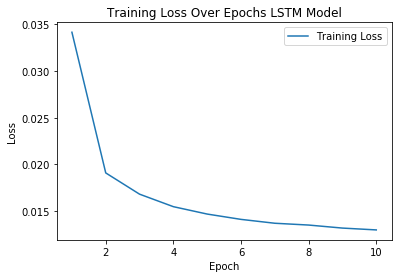
**CNN:**



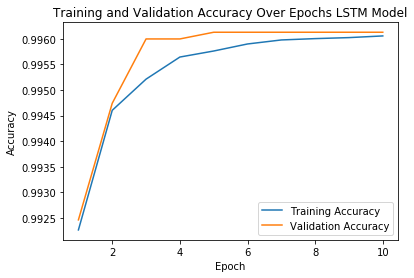
**Accuracy Plot:**



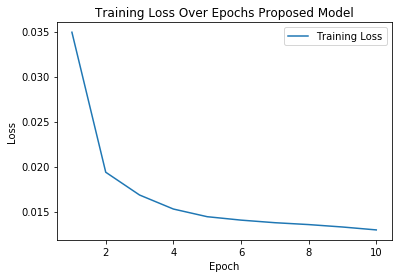
**LSTM:**



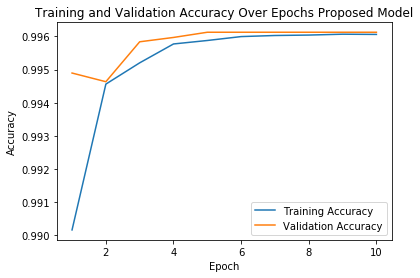
**Accuracy Plot LSTM:**



**Proposed Model:**

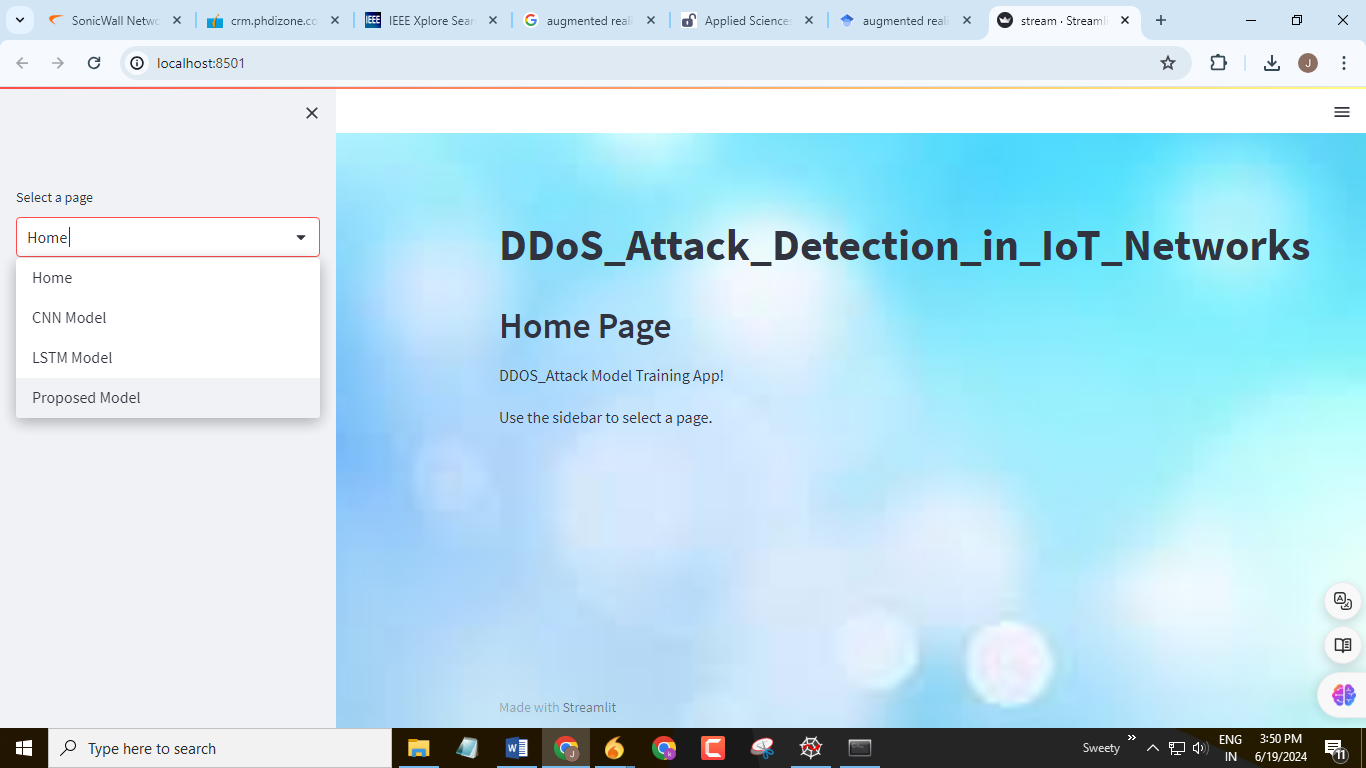


**Accuracy Plot:**

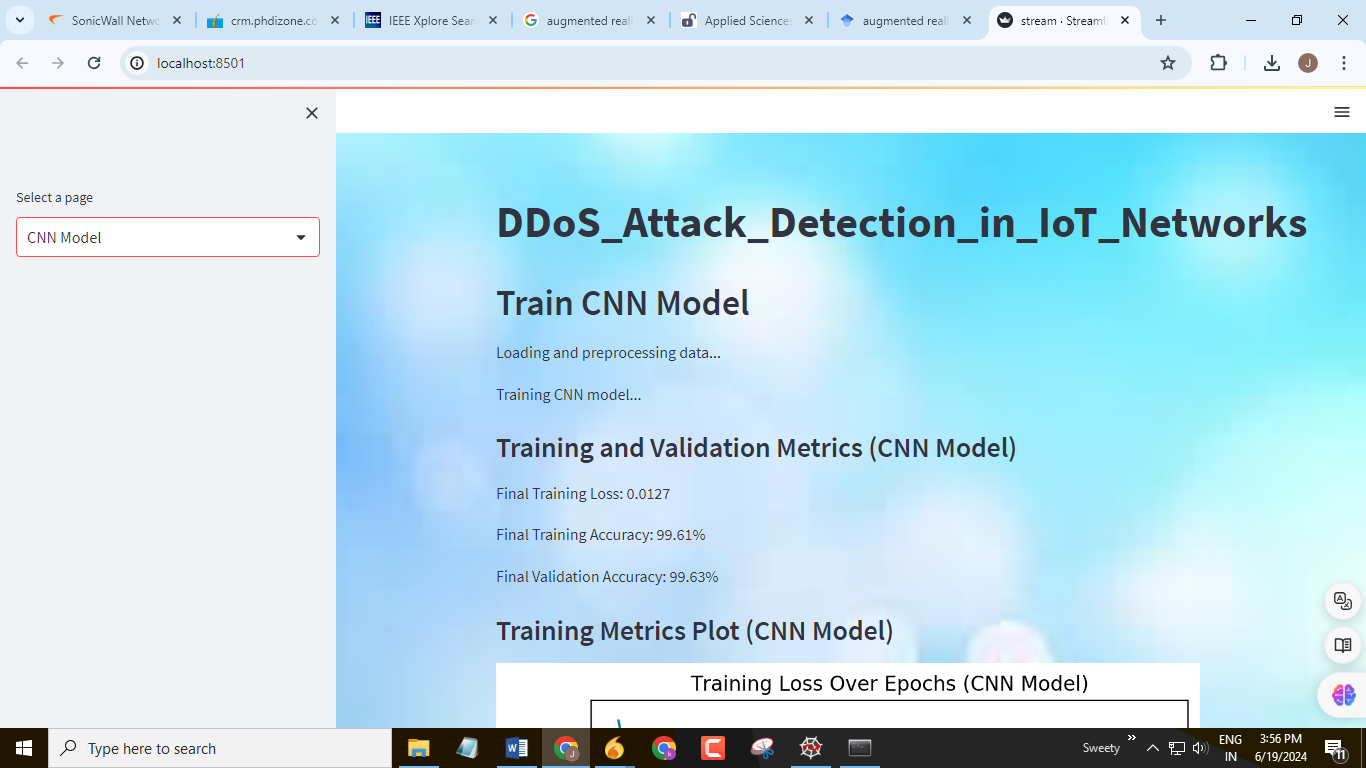


**Streamlit Framework:**

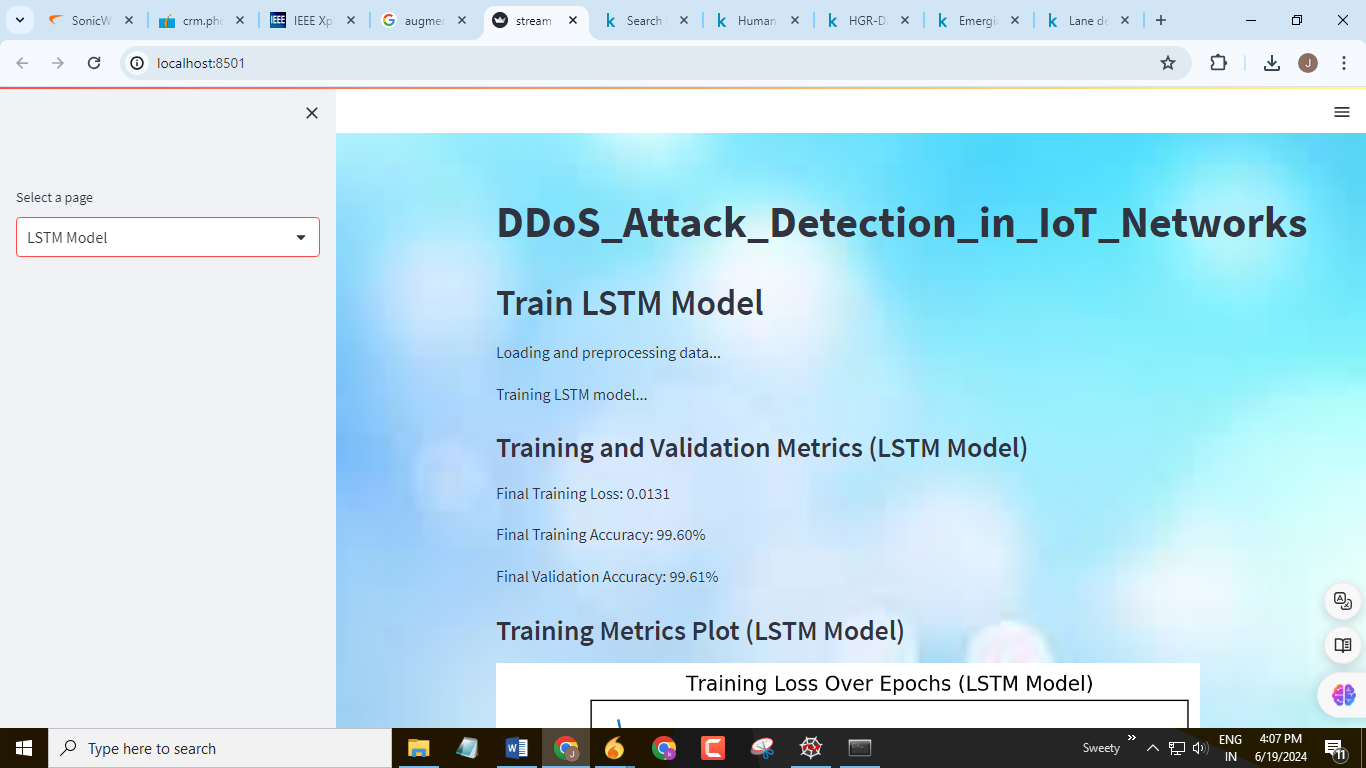
**Home Page:**



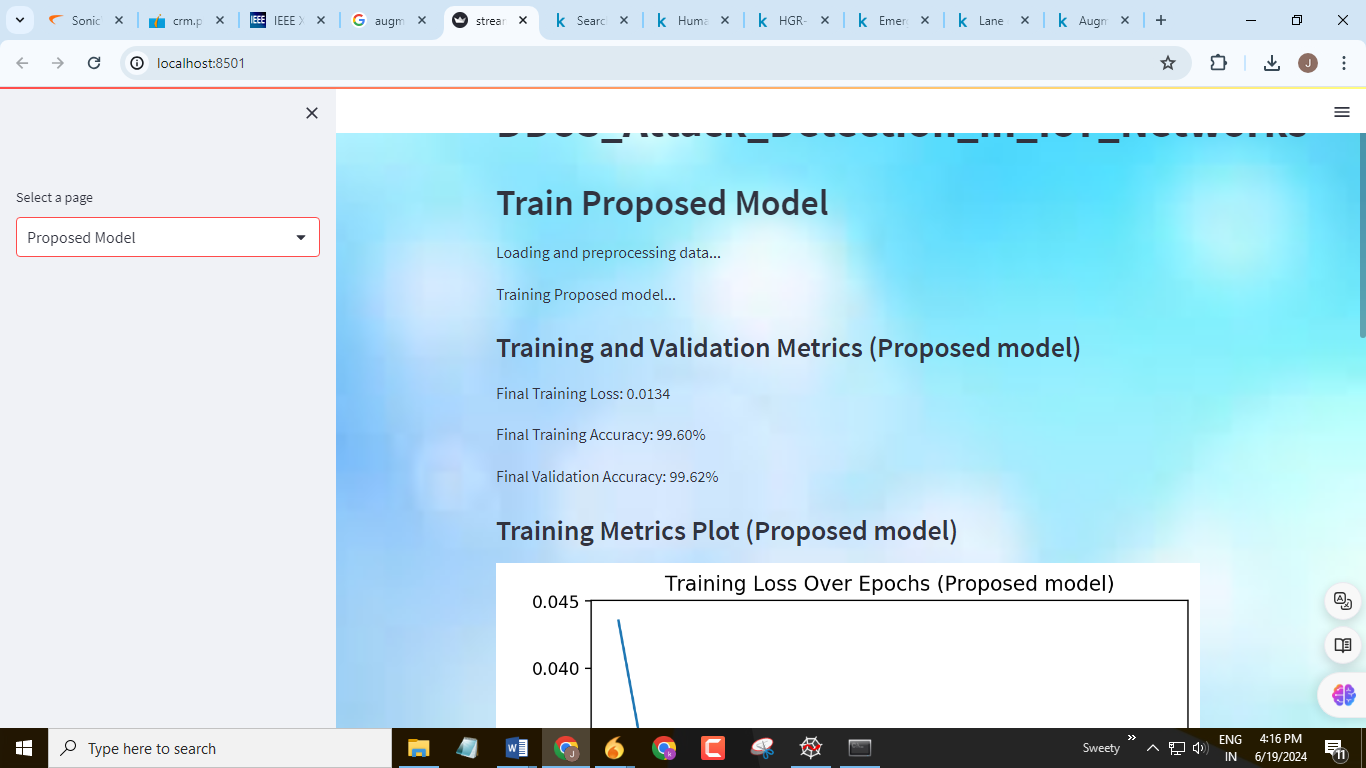
**CNN Model:**



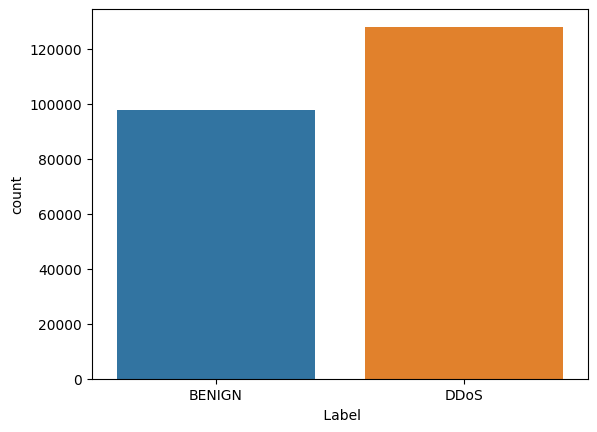
**LSTM Model:**



**Proposed Model :**



**Count Plot:**

****

**CHAPTER 10**

**REFERENCES**

1. F. Rustam, A. Raza, I. Ashraf, and A. D. Jurcut, ‘‘Deep ensemble-based efficient framework for network attack detection,’’ in Proc. 21st Medit. Commun. Comput. Netw. Conf. (MedComNet), Jun. 2023, pp. 1–10.
2. G. Shi, X. Shen, F. Xiao, and Y. He, ‘‘DANTD: A deep abnormal network traffic detection model for security of Industrial Internet of Things using high-order features,’’ IEEE Internet Things J., early access, Mar. 7, 2023, doi: 10.1109/JIOT.2023.3253777.
3. A. Papanikolaou, A. Alevizopoulos, C. Ilioudis, K. Demertzis, and K. Rantos, ‘‘An autoML network traffic analyzer for cyber threat detection,’’ Int. J. Inf. Secur., May 2023.
4. R. Gipiškis, D. Chiaro, M. Preziosi, E. Prezioso, and F. Piccialli, ‘‘The impact of adversarial attacks on interpretable semantic segmentation in cyber–physical systems,’’ IEEE Syst. J., early access, Jun. 12, 2023, doi: 10.1109/JSYST.2023.3281079.
5. M. K. Hasan, A. K. M. A. Habib, S. Islam, N. Safie, S. N. H. S. Abdullah, and B. Pandey, ‘‘DDoS: Distributed denial of service attack in communication standard vulnerabilities in smart grid applications and cyber security with recent developments,’’ Energy Rep., vol. 9, pp. 1318–1326, Oct. 2023.
6. M. S. I. Alsumaidaie, K. M. A. Alheeti, and A. K. Al-Aloosy, ‘‘Intelligent detection system for a distributed denial-of–service (DDoS) attack based on time series,’’ in Proc. 15th Int. Conf. Develop. eSyst. Eng. (DeSE), Jan. 2023, pp. 445–450.
7. S. Siva Shankar, B. T. Hung, P. Chakrabarti, T. Chakrabarti, and G. Parasa, ‘‘A novel optimization based deep learning with artificial intelligence approach to detect intrusion attack in network system,’’ Educ. Inf. Technol., Jun. 2023.
8. R. Salama, F. Al-Turjman, C. Altrjman, and D. Bordoloi, ‘‘The ways in which artificial intelligence improves several facets of cyber security—A survey,’’ in Proc. Int. Conf. Comput. Intell., Commun. Technol. Netw. (CICTN), Apr. 2023, pp. 825–829.
9. A. Raza, K. Munir, and M. Almutairi, ‘‘A novel deep learning approach for deepfake image detection,’’ Appl. Sci., vol. 12, no. 19, p. 9820, Sep. 2022.
10. Z. Pelletier and M. Abualkibash, ‘‘Evaluating the CIC IDS-2017 dataset using machine learning methods and creating multiple predictive models in the statistical computing language R,’’ Science, vol. 5, no. 2, pp. 187–191, 2020.
11. T. Elmasri, N. Samir, M. Mashaly, and Y. Atef, ‘‘Evaluation of CICIDS2017 with qualitative comparison of machine learning algorithm,’’ in Proc. IEEE Cloud Summit, Oct. 2020, pp. 46–51.
12. V. Gustavsson, ‘‘Machine learning for a network-based intrusion detection system: An application using Zeek and the CICIDS2017 dataset,’’ Tech. Rep., 2019.
13. A. Yulianto, P. Sukarno, and N. A. Suwastika, ‘‘Improving AdaBoostbased Intrusion Detection System (IDS) performance on CIC IDS 2017 dataset,’’ J. Phys., Conf. Ser., vol. 1192, Mar. 2019, Art. no. 012018.
14. K. Kostas, ‘‘Anomaly detection in networks using machine learning,’’ Res. Proposal, vol. 23, p. 343, Mar. 2018.
15. H. Chindove and D. Brown, ‘‘Adaptive machine learning based network intrusion detection,’’ in Proc. Int. Conf. Artif. Intell. Appl., Dec. 2021, pp. 1–6. [16] A. S. Jaradat, M. M. Barhoush, and R. B. Easa, ‘‘Network intrusion detection system: Machine learning approach,’’ Indonesian J. Electr. Eng. Comput. Sci., vol. 25, no. 2, pp. 1151–1158, 2022.
16. E. U. H. Qazi, A. Almorjan, and T. Zia, ‘‘A one-dimensional convolutional neural network (1D-CNN) based deep learning system for network intrusion detection,’’ Appl. Sci., vol. 12, no. 16, p. 7986, Aug. 2022.
17. A. Henry, S. Gautam, S. Khanna, K. Rabie, T. Shongwe, P. Bhattacharya, B. Sharma, and S. Chowdhury, ‘‘Composition of hybrid deep learning model and feature optimization for intrusion detection system,’’ Sensors, vol. 23, no. 2, p. 890, Jan. 2023.
18. A. S. Bangari, ‘‘A comparative evaluation of machine learning models and EDA through Tableau using CICIDS2017 dataset,’’ Ph.D thesis, National College of Ireland, Dublin, Ireland, 2023.
19. I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, ‘‘Toward generating a new intrusion detection dataset and intrusion traffic characterization,’’ in Proc. ICISSp, 2018, pp. 108–116.
20. A. Raza, H. U. R. Siddiqui, K. Munir, M. Almutairi, F. Rustam, and I. Ashraf, ‘‘Ensemble learning-based feature engineering to analyze maternal health during pregnancy and health risk prediction,’’ PLoS One, vol. 17, no. 11, Nov. 2022, Art. no. e0276525.
21. M. Imran, H. U. R. Siddiqui, A. Raza, M. A. Raza, F. Rustam, and I. Ashraf, ‘‘A performance overview of machine learning-based defense strategies for advanced persistent threats in industrial control systems,’’ Comput. Secur., vol. 134, Nov. 2023, Art. no. 103445.
22. A. Arshad, M. Jabeen, S. Ubaid, A. Raza, L. Abualigah, K. Aldiabat, and H. Jia, ‘‘A novel ensemble method for enhancing Internet of Things device security against botnet attacks,’’ Decis. Anal. J., vol. 8, Sep. 2023, Art. no. 100307.
23. A. Mukherjee and A. Ghosh, ‘‘Predictive framework for crime data analysis using a hybrid logistic regression—Support vector machine based ensemble classifier powered by CART (LR-SVMCART ),’’ Multimedia Tools Appl., Mar. 2023.
24. M. Wang, X. Xu, and Z. Yan, ‘‘Online fault diagnosis of PV array considering label errors based on distributionally robust logistic regression,’’ Renew. Energy, vol. 203, pp. 68–80, Feb. 2023.
25. C. Wang, Y. Sun, W. Wang, H. Liu, and B. Wang, ‘‘Hybrid intrusion detection system based on combination of random forest and autoencoder,’’ Symmetry, vol. 15, no. 3, p. 568, Feb. 2023.
26. P. Dey and D. Bhakta, ‘‘A new random forest and support vector machinebased intrusion detection model in networks,’’ Nat. Acad. Sci. Lett., Feb. 2023.
27. P. Mansourian, N. Zhang, A. Jaekel, M. Zamanirafe, and M. Kneppers, ‘‘Anomaly detection for connected autonomous vehicles using LSTM and Gaussian naïve Bayes,’’ in Proc. Int. Conf. Wireless Satell. Syst. Cham, Switzerland: Springer, 2023, pp. 31–43.
28. D. Chen, Q. Song, Y. Zhang, L. Li, and Z. Yang, ‘‘Identification of network traffic intrusion using decision tree,’’ J. Sensors, vol. 2023, pp. 1–9, Apr. 2023.
29. M. Douiba, S. Benkirane, A. Guezzaz, and M. Azrour, ‘‘An improved anomaly detection model for IoT security using decision tree and gradient boosting,’’ J. Supercomput., vol. 79, no. 3, pp. 3392–3411, Feb. 2023.
30. M. Aghaabbasi, M. Ali, M. Jasinski, Z. Leonowicz, and T. Novák, ‘‘On hyperparameter optimization of machine learning methods using a Bayesian optimization algorithm to predict work travel mode choice,’’ IEEE Access, vol. 11, pp. 19762–19774, 2023.
31. E. Bisong and E. Bisong, ‘‘Google colaboratory,’’ Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners. 2019, pp. 59–64