NETWORK INTRUSION DETECTION SYSTEM using FEEDFORWARD NEURAL NETWORK

CSE4006- Deep Learning

School of Computer Science and Engineering

By

P.MANOJ	21BCE9599
B.NAVEEN	22BCE9615
G.HAREESH	22BCE20462
Y.SAI MANIKANTA	21MIS7058

Submitted to

Dr. E. Rajalakshmi
Assistant Professor Senior Grade 1
School of Computer Science and Engineering (SCOPE)
VIT University, Andhra Pradesh, INDIA

2024 -2025

TABLE OF CONTENTS

ABSTRACT

The rapid increase in internet users and the growing sophistication of cyberattacks necessitate more effective Network Intrusion Detection Systems (NIDS). Traditional NIDS often struggle to keep pace with evolving threats, leading to the exploration of advanced methodologies like neural networks. Neural network-based NIDS can significantly enhance network security by leveraging their ability to learn and adapt to new types of intrusions. In this study, we implement a Feedforward Neural Network (FFNN) architecture to create a robust NIDS.

In a Feedforward Neural Network, information flows in a single direction, from input to output, without any cycles or loops. This simplicity makes FFNNs suitable for real-time intrusion detection as they can quickly process network data and identify potential threats. Our model is trained on diverse datasets to recognize various attack patterns and anomalies, improving its accuracy and reliability. By utilizing neural networks, our NIDS can adapt to new and emerging threats, ensuring the privacy and security of users' data. This approach offers a promising solution to the ever-evolving landscape of cyber threats, providing a more dynamic and responsive defense mechanism for computer networks.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviations
Network Intrusion Detection Systems - NIDS
Feed Forward Neural Network - FFFNN
Artificial Neural Network – ANN
Receiver Operating Characteristic - ROC
Virtual Area Network - VLAN
Machine Learning – ML
Deep Learning - DL

CHAPTER 1

INTRODUCTION

The massive increase in the population and technology advancement, allows rapid increase in the use of computing systems. Parallelly, it opens a path to Cyber Attacks which cannot be in the control of mankind. Illegal Authentication may have critical affects to individuals as well as organisations. For Communication to be private and protected among the networks we must have an enhanced safety and security system that scans the network and results in reporting the user in Realtime. This paper highlights the contribution of neural networks in detecting suspicious activities in computer networks. We have implemented Feed forward neural network architecture to build the model. To minimize the loss function we preferred to use Adam optimisation algorithm. Additionally, the data set i.e., NSL-KDD which is used for this research purpose is collected from an online open source, Kaggle.

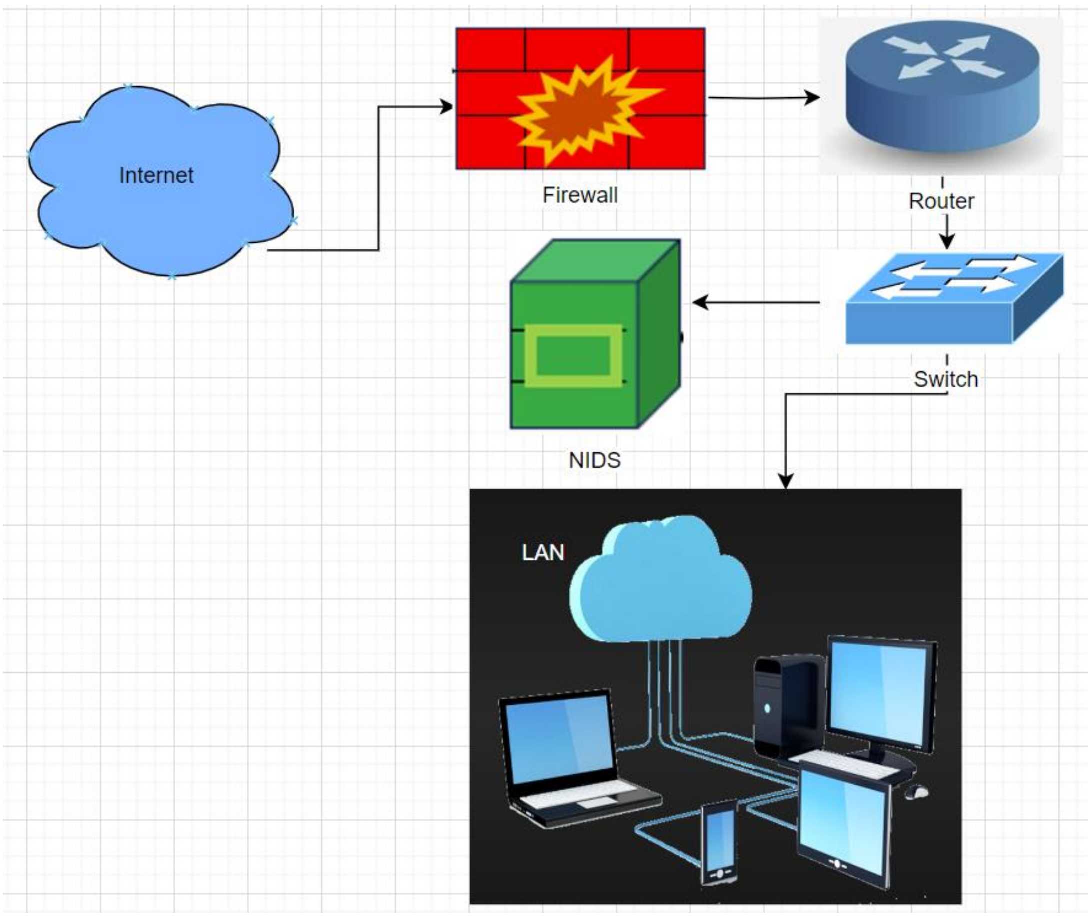


Figure 1 Working of the Model in real-time

There are two approaches to detect network breaches

- Signature based detection
- Anomaly based detection

Misuse or Signature Detection

This type of detection looks for an abnormal scenario that matches with any previous attack. The model uses past data to detect the attacks. Sometimes it produces false data. Suppose a user forgets his password and logs in multIPle times with new passwords. In this case it assumes the true user to be an intruder and shows false results. However, it is efficient in most cases. Signatures are the misuse or attack patterns that contains the data packets that are viewed as threat to the network. These signatures include data-pay-load, packet header, unauthorised and unauthenticated activities such as improper file transfer protocol.

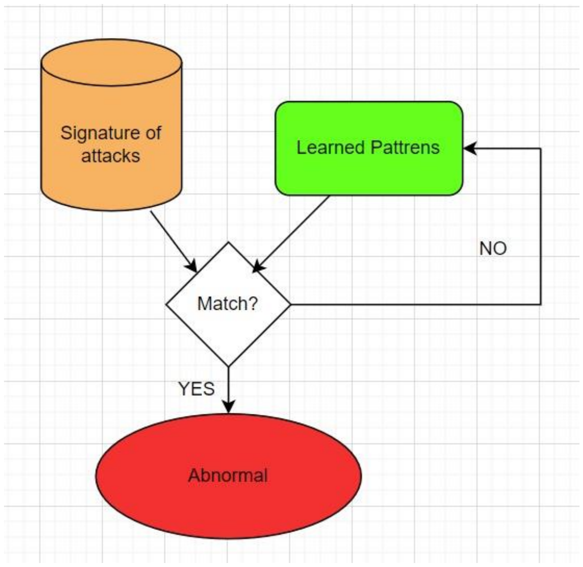


Figure 2 How the model identifies cyber attacks

Anomaly based Detection

It has the ability to detect the unseen and novel attacks and suspicious activities in the network. It looks for the abnormal behaviour instead of normal behaviour. It also has the ability to preserve accuracy and avoid false predictions. However, there are drawbacks that limits its usage. The normal behaviour must include the IP Address, or the host name of a machine and it should include virtual Area network (VLAN) details. If the malicious behaviour is found in any of the above then the model classifies the data as abnormal.

A Network intrusion in an unethical and unlawful action performed on the assets of a business network. It may result in downfall of a giant institution and defaming of a famous company. It makes the confidential information easily available to the intruder. Network intrusion detection is very essential in various fields like military and Science to ensure the information security and to alert the user to be beware of any malicious activities. Generally, when data is to be transmitted from one system to another it has to pass through various routers and switches which is seen as a severe threat. Because any time intruders may attack on the data. So, with each system having IP address, host name we can easily detect when a system other than the assigned IP address assigned by DHCP tries to access the data. The paper reviews various intrusion techniques and research on anomaly-based approach using ANN and Adam optimizer to address the cyber threats.

NIDS analyzes the overall traffic of the network to detect any malicious activities. The purpose of this research paper is to provide knowledge about neural networks and its significance in real-world.

1.1 Objective of the Project

The primary objective of our project is to develop a robust and efficient Network Intrusion Detection System (NIDS) utilizing a Feedforward Neural Network (FFNN) architecture. Our goal is to enhance the security of computer networks by accurately detecting and responding to various types of network intrusions. In today's digital era, where cyber threats are becoming increasingly sophisticated and frequent, traditional NIDS methodologies, such as signature-based detection systems, are proving inadequate. These conventional systems often struggle to identify new and unknown threats, leading to a high rate of false positives and negatives. Therefore, there is a pressing need for an advanced NIDS that can adapt to evolving cyber threats and provide real-time protection.

To achieve this, our project leverages the capabilities of artificial intelligence, specifically neural networks. Neural networks, with their ability to learn and adapt from data, offer significant advantages in intrusion detection. By training a Feedforward Neural Network on a comprehensive dataset, the model can recognize patterns and anomalies indicative of malicious activities. The FFNN architecture is particularly suitable for this task due to its straightforward design, which allows for efficient data processing and real-time intrusion detection.

✓ **Data Preprocessing**

We used the NSL-KDD dataset, a well-known benchmark in intrusion detection research. This dataset includes various types of network attacks and normal traffic, providing a diverse training ground for our neural network. We will undertake steps such as handling missing values, data normalization, and feature selection to enhance the model's accuracy and efficiency. Ensuring the dataset is clean and suitable for training is a key part of this process.

✓ **Model Building**

Our project involves creating a Feedforward Neural Network with an appropriate number of layers and neurons to effectively capture the complexities of network traffic data. Key aspects include selecting suitable activation functions, initialization methods, and network depth to ensure optimal performance.

✓ **Model Training**

Using the pre-processed dataset, we will train the neural network. We will employ the Adam optimization algorithm, known for its computational efficiency and ability to handle large datasets. Training involves forward propagation to compute predictions and backpropagation to adjust weights based on the error.

✓ **Hyperparameter Tuning**

We will fine-tune hyperparameters such as learning rate, batch size, and the number of epochs to optimize the model’s performance. Regularization techniques like L2 regularization (Lasso Regularization) will be applied to prevent overfitting and improve generalization.

- ✓ **Performance Evaluation**

Once trained, our model will be rigorously tested using a separate validation dataset to assess its performance in detecting various types of intrusions. We will use metrics such as accuracy, precision, recall, and F1 score to evaluate the effectiveness of our NIDS.

1.2 Need of the Work

The rapid increase in internet users and the evolution of sophisticated cyberattacks necessitate advanced Network Intrusion Detection Systems (NIDS). Traditional NIDS, primarily relying on signature-based detection, are increasingly inadequate due to their inability to detect novel threats and high false positive rates. This limitation underscores the need for enhanced detection mechanisms that can adapt to new and emerging attack patterns.

Neural Networks, particularly Feedforward Neural Networks (FFNN), offer a promising solution to this challenge. By leveraging the capabilities of deep learning, FFNN-based NIDS can analyze vast amounts of network traffic data, identify complex patterns, and detect intrusions with higher accuracy than traditional methods. The implementation of such systems can significantly improve network security, ensuring data integrity and privacy.

Moreover, the NSL-KDD dataset, a refined version of the KDD'99 dataset, provides a robust foundation for training and evaluating these advanced NIDS. This dataset addresses some of the inherent issues in the original KDD'99 dataset, such as redundancy and irrelevant records, making it a more reliable resource for developing and testing intrusion detection models.

The adoption of FFNN for NIDS also aligns with the broader trend towards utilizing machine learning and artificial intelligence in cybersecurity. This approach not only enhances detection capabilities but also enables continuous learning and adaptation, crucial for combating the ever-evolving landscape of cyber threats. The integration of optimization techniques such as the Adam algorithm and regularization methods like Lasso further refines the model, ensuring it remains effective and resilient against various types of attacks.

In summary, the need for this work is driven by the limitations of traditional NIDS, the potential of neural networks to improve detection accuracy, and the availability of enhanced datasets like NSL-KDD. Developing an FFNN-based NIDS is a strategic response to the growing complexity and frequency of cyberattacks, providing a more robust, adaptable, and accurate solution for network security.

1.3 Scope and Motivation

The scope of this project involves the comprehensive development and deployment of a Network Intrusion Detection System (NIDS) using a Feedforward Neural Network (FFNN) architecture. Our project embarks several key stages: Data Preprocessing, Model Building, Model Training, Hyperparameter tuning, and Performance Evaluation.

The motivation for this project cpmes from the increasing complexity and frequency of cyberattacks in today's digital landscape. Traditional NIDS, which often rely on signature-based detection, have significant limitations:

✓ **Evolving Threat Landscape**

Cyber threats are continuously evolving, with attackers developing new techniques to bypass traditional security measures. Signature-based NIDS struggle to keep up with these new threats, as they rely on known attack patterns.

✓ **False Positives and Negatives**

Traditional NIDS often generate a high number of false positives (benign activity identified as malicious) and false negatives (malicious activity identified as benign). This inefficiency can lead to operational disruptions and security breaches.

✓ **Anomaly Detection**

Anomaly-based detection methods, which identify deviations from normal behavior, also face challenges in accurately defining normal behavior and can produce false alarms.

✓ **Advancements in AI**

Recent advancements in artificial intelligence, particularly in neural networks, offer promising solutions for enhancing NIDS. Neural networks can learn and adapt to new threat patterns, making them more effective in detecting both known and unknown attacks.

By leveraging a Feedforward Neural Network, this project aims to address these limitations and provide a more robust and adaptive NIDS. The use of neural networks allows for the dynamic learning of complex patterns in network traffic, improving the detection of sophisticated and novel attacks.

1.4 Organization of the Report

This report is structured into a total of 7 chapters to methodically present the evolution and evaluation of our project, Network Intrusion Detection System (NIDS) using Feedforward Neural Network (FFFNN).

Chapter 1: Introduction

Provides an overview of the project's goals, the imperative nature and need of the work, the scope, and the motivational factors driving the study. Additionally, this chapter outlines how the report is organized to ensure clarity and coherence in presenting the research findings.

Chapter 2: Literature Survey

Reviews and synthesizes existing research and methodologies pertinent to NIDS, emphasizing the deficiencies inherent in traditional approaches and underscoring the potential efficacy of neural network-based solutions. This chapter sets the foundation for the novel contributions put forth in the subsequent chapters.

Chapter 3: Existing Method & Disadvantages

Discusses the current methods used in NIDS, highlighting their drawbacks and limitations. This chapter serves to illustrate the necessity for improved techniques and the advantages of adopting neural network-based solutions.

Chapter 4: Project Flow/Framework of the Proposed System

Details the overall framework of the proposed FFFNN-based NIDS, describing the step-by-step flow of the project from data collection and preprocessing to model training and evaluation. This chapter provides a clear roadmap of the development process.

Chapter 5: Hardware and Software Requirements

Enumerates the specific hardware and software requirements necessary to implement the proposed NIDS. This chapter ensures that readers understand the technical prerequisites for reproducing the system.

Chapter 6: Proposed System

Elaborates on the architecture of the FFFNN deployed in the NIDS, elucidates the methodologies employed for data preprocessing, and delineates the systematic steps taken to construct and optimize the model. This chapter provides a detailed insight into the technical underpinnings of the developed NIDS. Data Preprocessing, Model Architecture, Training and Optimization, Evaluation Metrics are all included in this chapter.

Chapter 7: Conclusion and Future Work

Summarizes the principal contributions of the project, acknowledges its limitations, and proposes avenues for future research and development in the field of NIDS utilizing neural network technologies. This chapter encapsulates the overall impact and potential implications of the FFFNN-based NIDS on enhancing network security.

References

Lists the scholarly articles, books, and other resources referenced throughout the report to support the research and findings presented.

Appendix

Includes supplementary materials such as source code and screenshots that provide additional context and support for the project’s implementation and results.

CHAPTER 2

LITERATURE SURVEY

According to the recent papers (2016-2023), the overview of the previous works done by the authors, their approaches and their accuracies are as follows:

S. No.	Authors/Papers	Approach, Dataset Used	Accuracy
1.	Mariam Ibrahim, Ruba Elhafiz 2023[5]	LSTM RNN Algorithm NSL – KDD	88.4%
2.	Leila Mohammadpour, Teck Chaw Ling, Chee Sun Liew, Alihossien Aryanfar, 2022[1]	Basic 2D CNN NSL – KDD	85.59%
3.	Rachid Tahri, Youssef Balouki, Abdessamad Jarrar, Abdellatif Lasbahani 2022[3]	KNN UNSW – NB15	93.33%
4.	Lihua Su, Wenhua Bai, Zhanghua Zhu, Xuan He 2021[6]	SVM KDDCUP99	95%
5.	Tang A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi 2016[2]	DNN NSL - KDD	75.5%
6.	Quamar Niyaz, Weiqing Sun, Ahmad Y Javid and Mansoor Alam 2016[4]	Self -Taught Learning NSL - KDD	88.39%
7.	S. Garcia	Deep autoencoder-based approach	UNSW-NB15 dataset-97.72%
8.	P. Rajarathnam	DNN-based approaches	Not specified
9.	A. Üstebay	CNN and RNN	NSL-KDD dataset.
10.	Zeeshan Ahmad, Adnan Shahid Khan	ML/DL Algorithms	NSL-KDD dataset-95%

11.	Lirim Ashiku	CNN(Convolutional Neural Networks)	UNSW-NB15-94.4%

Table 1 Literature Review

The problem identified in these studies is the inadequacy of traditional NIDS to effectively detect new and sophisticated cyberattacks. The integration of machine learning (ML) and deep learning (DL) techniques into NIDS frameworks represents a significant shift towards overcoming the limitations of traditional methods. The primary objective is to develop systems capable of learning from vast amounts of network data, identifying anomalies, and detecting both known and unknown threats with high accuracy.

Previous authors have explored various machine learning and deep learning models to improve detection accuracy and adaptability. The models used include LSTM RNN, 2D CNN, KNN, SVM, DNN, and autoencoders, among others, each offering different strengths in handling network traffic data and detecting intrusions. The focus has been on enhancing the accuracy, reducing false positives, and ensuring the models can adapt to new types of threats.

These works collectively highlight the evolution and potential of neural network-based approaches in enhancing the capabilities of NIDS. For Example, Mariam Ibrahim and Ruba Elhafiz (2023) applied Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNNs) to the NSL-KDD dataset, achieving an accuracy of 88.4%. LSTMs are particularly effective in handling sequential data, making them suitable for modeling network traffic over time and detecting anomalies that may not be apparent in individual data points.

Leila Mohammadpour and colleagues (2022) utilized a basic 2D Convolutional Neural Network (CNN) on the NSL-KDD dataset, reaching an accuracy of 85.59%. CNNs are well-known for their capability in feature extraction from images, and their application in NIDS involves treating network traffic data as images, thereby leveraging CNNs' strength in identifying complex patterns.

Rachid Tahri et al. (2022) explored the use of K-Nearest Neighbors (KNN) on the UNSW-NB15 dataset, achieving a remarkable accuracy of 93.33%. KNN is a simple yet powerful algorithm that classifies data points based on their proximity to other data points in the feature space. Its effectiveness in NIDS lies in its ability to classify new instances based on learned patterns from historical data.

Lihua Su and colleagues (2021) implemented Support Vector Machines (SVM) on the KDDCUP99 dataset, achieving an accuracy of 95%. SVMs are effective in high-dimensional spaces and are known for their robustness in classification tasks, making them suitable for intrusion detection where the distinction between normal and malicious activities can be subtle.

Tang A Tang and collaborators (2016) applied Deep Neural Networks (DNNs) to the NSL-KDD dataset, resulting in an accuracy of 75.5%. Despite being lower than other approaches, DNNs provide a foundation for further exploration and optimization in NIDS. They are capable of capturing intricate patterns in data through multiple hidden layers, though their performance can be heavily dependent on the quality and quantity of training data.

Quamar Niyaz and colleagues (2016) introduced a Self-Taught Learning approach on the NSL-KDD dataset, achieving an accuracy of 88.39%. This approach involves using unsupervised learning to pre-train a model on unlabelled data before fine-tuning it with labeled data, which enhances the model's ability to generalize from limited labeled examples.

Further advancements include S. Garcia's deep autoencoder-based approach on the UNSW-NB15 dataset, achieving a high accuracy of 97.72%. Autoencoders are effective for anomaly detection as they learn to reconstruct normal data patterns and identify deviations indicative of intrusions.

Other notable contributions include P. Rajarathnam's DNN-based approaches, A. Üstebay's combined use of CNN and RNN, and Zeeshan Ahmad and Adnan Shahid Khan's ML/DL algorithms on the NSL-KDD dataset. These studies emphasize the trend towards leveraging the strengths of different neural network architectures to improve NIDS performance.

The collective efforts of these researchers underscore the importance of developing adaptive, high-accuracy NIDS capable of responding to the dynamic nature of cyber threats. The continuous refinement of neural network models and the exploration of new architectures are crucial in advancing the field of intrusion detection.

2.1 Basics

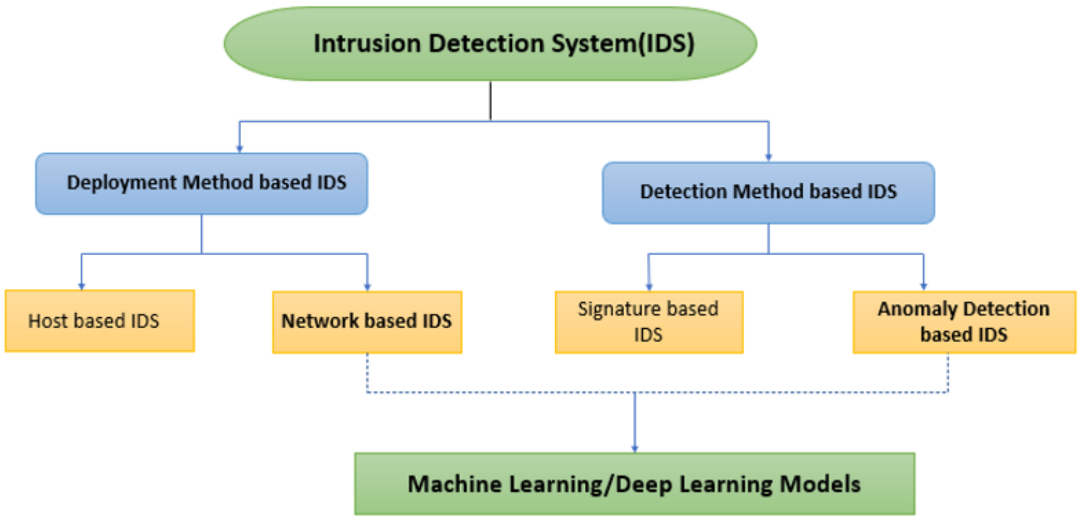

Intrusion Detection System(IDS)
Deployment Method based IDS
Detection Method based IDS
Host based IDS
Network based IDS
Signature based IDS
Anomaly Detection based IDS
Machine Learning/Deep Learning Models

Figure 3 Steps included in different types of approaches

Deep learning has 3 types of network Architectures

- Artificial Neural Networks (ANN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)

***Artificial Neural Networks*:**

Artificial Neural Networks (ANNs) are inspired by the biological neural networks of animal brains. They consist of layers of nodes, or "neurons," where each layer is fully connected to the next. ANNs typically use a forward propagation technique to process inputs through the network layers, calculate the loss function to determine the error, and then use backpropagation to adjust weights and minimize the error. The optimization algorithms such as gradient descent are used to maximize the accuracy.

***Convolutional Neural Network**:*

Convolutional Neural Networks (CNNs) are specialized for processing data with a grid-like topology, such as images. They consist of convolutional layers that apply a series of filters to the input to extract features, pooling layers to reduce the dimensionality, and fully connected layers to perform the final classification. CNNs are particularly effective in recognizing spatial hierarchies in data, making them suitable for image and video recognition tasks.

Recurrent Neural Network:

Recurrent Neural Networks (RNNs) are designed to recognize patterns in sequences of data, such as time series or natural language. Unlike ANNs, RNNs have connections that form directed cycles, allowing them to maintain a 'memory' of previous inputs. This is particularly useful for tasks where context or temporal information is critical. RNNs use feedforward networks to classify data, but they also process inputs in a sequential manner and can handle variable-length sequences.

Network Intrusion Detection Systems (NIDS):

Network Intrusion Detection Systems (NIDS) are crucial for maintaining the security and integrity of computer networks. They monitor network traffic for suspicious activities and potential threats, providing an essential layer of defense against cyberattacks. The basic architecture of a NIDS involves collecting network data, analyzing it to detect anomalies or signatures of known attacks, and taking action based on the detection.

Neural Networks in NIDS:

- Neural networks, particularly Feedforward Neural Networks (FFNNs), have become prominent in NIDS due to their ability to learn from data and adapt to new threats. FFNNs consist of multiple layers: input, hidden, and output layers. Each neuron in one layer is connected to every neuron in the next layer, and information flows in a single direction (forward). This structure allows FFNNs to process inputs and produce outputs efficiently, which is advantageous for real-time intrusion detection.
- Training neural network-based NIDS involves using diverse datasets that include normal traffic patterns and various types of attack signatures. This helps the model distinguish between legitimate and malicious activities. The adaptability of neural networks is a key feature, allowing the NIDS to evolve with emerging threats and maintain high levels of accuracy and reliability.

Neural networks, particularly Feedforward Neural Networks (FFNNs), have gained prominence in NIDS due to their ability to learn and adapt to new threats. FFNNs consist of multiple layers, including input, hidden, and output layers, where each neuron in one layer is connected to every neuron in the next layer. Information flows in a single direction, allowing the network to process inputs and produce outputs without feedback loops. This simplicity is advantageous for real-time intrusion detection, as it enables quick processing of large volumes of network data.

Training neural network-based NIDS involves using diverse datasets that include normal traffic patterns and various types of attack signatures. This helps the model distinguish between legitimate and malicious activities. These datasets include normal traffic patterns and various types of attack signatures, enabling the model to distinguish between legitimate and malicious activities. The adaptability of neural networks is a key feature, allowing the NIDS to evolve with emerging threats and maintain high levels of accuracy and reliability in intrusion detection.

2.2 Preceding Works

Research in NIDS has evolved significantly over the past decades, with numerous studies exploring different methodologies to enhance detection capabilities. Early NIDS primarily relied on signature-based detection, where known attack patterns were used to identify malicious activities. However, this approach had limitations in detecting new or unknown threats, leading to the exploration of anomaly-based detection techniques.

The integration of machine learning and neural networks into NIDS marked a significant advancement in the field. Studies have demonstrated the superior performance of neural network-based models over traditional methods. For instance, a study implemented a Multi-Layer Perceptron (MLP) for intrusion detection, achieving high accuracy in identifying various types of attacks. Another research utilized Convolutional Neural Networks (CNNs) to detect intrusions by analyzing network traffic as images, leveraging the powerful feature extraction capabilities of CNNs.

The adoption of deep learning techniques has further improved the detection capabilities of NIDS. Deep Belief Networks (DBNs) and Recurrent Neural Networks (RNNs) have been explored for their ability to handle sequential data and capture temporal patterns in network traffic. These models have shown promising results in detecting sophisticated attacks that exploit temporal dependencies in network activities.

Several benchmark datasets, such as KDD Cup 99, NSL-KDD, and CICIDS 2017, have been widely used in NIDS research to evaluate the performance of different models. These datasets provide a comprehensive collection of normal and attack traffic, facilitating the training and testing of NIDS models. Comparative studies have highlighted the advantages of neural network-based NIDS in terms of detection accuracy, false positive rates, and adaptability to new threats.

In summary, the literature on NIDS reveals a clear trend towards the adoption of neural networks and deep learning techniques to enhance intrusion detection capabilities. The continuous evolution of cyber threats necessitates the development of more sophisticated and adaptive NIDS models. Neural networks, with their ability to learn from data and generalize to new scenarios, offer a promising solution to the challenges faced in network intrusion detection.

CHAPTER 3

EXISTING METHODS & DISADVANTAGES

In the quest to enhance network intrusion detection systems (NIDS), researchers have explored a myriad of machine learning (ML) and deep learning (DL) models. Researchers have extensively explored a variety of machine learning (ML) and deep learning (DL) models to enhance the detection capabilities of NIDS.

These models, each with distinct mechanisms and capabilities, aim to improve detection accuracy, minimize false positives, and adapt to the ever-changing landscape of cyber threats. This chapter delves into various ML and DL methods previously employed, outlining their methodologies and inherent disadvantages. By examining these methods and their limitations, we can gain valuable insights into the strengths and weaknesses of current approaches, guiding future research towards more effective and efficient intrusion detection solutions.

Methods and Their Disadvantages

- ✓ **Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNNs)**

  Mariam Ibrahim and Ruba Elhafiz (2023) applied LSTM RNNs to the NSL-KDD dataset, achieving an accuracy of 88.4%. LSTMs excel at handling sequential data, making them adept at modeling network traffic over time and identifying anomalies not evident in individual data points. However, their primary disadvantage lies in their complexity and computational demands, which can lead to prolonged training times and difficulties in real-time application. Additionally, LSTMs may suffer from overfitting, especially with limited datasets.

- ✓ **2D Convolutional Neural Networks (CNNs)**

  Leila Mohammadpour et al. (2022) utilized a basic 2D CNN on the NSL-KDD dataset, achieving an accuracy of 85.59%. CNNs are renowned for their feature extraction capabilities, traditionally applied to image data. In NIDS, they process network traffic data as images to detect intricate patterns. Despite their efficacy, CNNs can be computationally expensive and require substantial data preprocessing. They may also struggle with varying data structures, as network traffic data is not naturally suited for image-like representation.

✓ **K-Nearest Neighbors (KNN)**

Rachid Tahri et al. (2022) employed KNN on the UNSW-NB15 dataset, achieving an impressive accuracy of 93.33%. KNN classifies data points based on their proximity to other points in the feature space, which is beneficial for detecting patterns in historical data. Nevertheless, KNN's performance degrades with large datasets due to its reliance on distance calculations, making it computationally intensive. Additionally, KNN is sensitive to irrelevant features and noise in the data, which can impact its accuracy.

✓ **Support Vector Machines (SVM)**

Lihua Su et al. (2021) implemented SVMs on the KDDCUP99 dataset, achieving an accuracy of 95%. SVMs are effective in high-dimensional spaces and robust in classification tasks. However, they can be computationally intensive and require careful tuning of parameters such as the kernel function. SVMs are also less effective with large datasets due to their scalability issues and can be sensitive to the choice of kernel and regularization parameters.

✓ **Deep Neural Networks (DNNs)**

Tang A. Tang et al. (2016) applied DNNs to the NSL-KDD dataset, resulting in an accuracy of 75.5%. DNNs can capture complex patterns through multiple hidden layers, providing a strong foundation for further exploration. However, their performance is heavily dependent on the quality and quantity of training data. DNNs are prone to overfitting and require extensive computational resources for training. Their black-box nature also makes them difficult to interpret, which is a significant drawback in security applications where explainability is crucial.

✓ **Self-Taught Learning**

Quamar Niyaz et al. (2016) introduced a Self-Taught Learning approach on the NSL-KDD dataset, achieving an accuracy of 88.39%. This method uses unsupervised learning to pre-train models on unlabeled data, enhancing generalization from limited labeled examples. The downside is that it can be challenging to find suitable unlabeled data for pre-training. Additionally, the transition from unsupervised to supervised learning can introduce complexities, and the overall training process becomes longer.

✓ **Autoencoders**

S. Garcia's deep autoencoder-based approach on the UNSW-NB15 dataset achieved a high accuracy of 97.72%. Autoencoders are effective for anomaly detection as they reconstruct normal data patterns and identify deviations. However, they require a substantial amount of normal data to train effectively and can struggle with identifying novel attacks that significantly differ from the training data. Autoencoders are also computationally intensive and can be sensitive to the choice of hyperparameters.

✓ **DNN-based Approaches**

P. Rajarathnam explored various DNN architectures, highlighting their potential in NIDS. However, the disadvantages align with those previously discussed, including overfitting, computational demands, and interpretability issues.

- ✓ **Combined CNN and RNN**

A. Üstebay combined CNN and RNN models to leverage the strengths of both architectures. While this hybrid approach improves accuracy and adaptability, it increases the model's complexity and computational requirements.

- ✓ **ML/DL Algorithms**

Zeeshan Ahmad and Adnan Shahid Khan explored a range of ML and DL algorithms on the NSL-KDD dataset. Each algorithm offered unique benefits, but common disadvantages included scalability issues, sensitivity to hyperparameters, and the need for extensive computational resources.

- ✓ **Random Forests**

Random Forests have been widely used for NIDS due to their robustness and ability to handle large datasets. RFs work by constructing multiple decision trees during training and outputting the class that is the mode of the classes of individual trees. They are less likely to overfit compared to single decision trees. However, RFs can be computationally intensive and require a large amount of memory for storing the trees. Additionally, they may become less interpretable as the number of trees increases.

- ✓ **Gradient Boosting Machines (GBM)**

GBM, including implementations like XGBoost, have shown promise in NIDS applications. GBM builds an ensemble of trees in a sequential manner, where each tree attempts to correct the errors of its predecessor. This method is highly effective in achieving high accuracy and handling complex data distributions. Nevertheless, GBMs can be prone to overfitting if not properly tuned and require significant computational resources for training and inference.

- ✓ **Naive Bayes**

Naive Bayes classifiers have been applied in NIDS due to their simplicity and efficiency. They work on the principle of Bayes' theorem with an assumption of independence among predictors. NB classifiers are particularly useful when the dimensionality of the input data is high. However, the strong independence assumption can be a limitation, as it rarely holds true for real-world data, potentially reducing the model's accuracy.

- ✓ **Decision Trees**

Decision Trees have been a popular choice for NIDS due to their simplicity and interpretability. They classify data by splitting it into subsets based on feature values, resulting in a tree-like structure of decisions. While decision trees are easy to understand

and visualize, they can suffer from overfitting, particularly with noisy data. Pruning techniques are often required to improve their generalization capabilities.

- ✓ **Artificial Immune Systems (AIS)**

Artificial Immune Systems are inspired by the human immune system and have been applied to NIDS. AIS algorithms detect anomalies by mimicking immune responses. They can adapt to new types of intrusions and are highly robust. However, AIS can be computationally expensive and may require extensive parameter tuning. Their performance can also be influenced by the choice of detectors and the diversity of the training data.

- ✓ **Hidden Markov Models (HMM)**

Hidden Markov Models have been used in NIDS to model the sequence of network events. HMMs are effective in capturing temporal patterns and detecting anomalies over time. However, they require large amounts of data for training and can be computationally intensive. HMMs also assume that the future state depends only on the current state, which might not always be accurate in complex network environments.

The exploration of various ML and DL models in NIDS has demonstrated significant advancements in detection accuracy and adaptability. However, each method comes with its set of disadvantages, primarily related to computational demands, sensitivity to data quality, and complexity in implementation. Future research should focus on addressing these challenges, particularly by developing models that balance performance with efficiency and interpretability.

CHAPTER 4

PROJECT FLOW/FRAEWORK OF THE PROPOSED SYSTEM

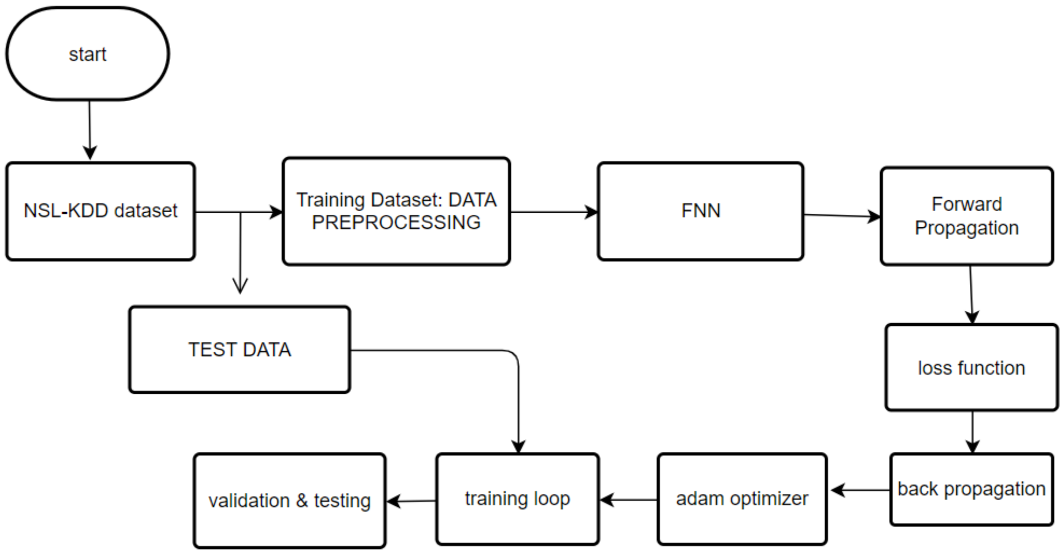


Figure 4 Workflow of the proposed FFFNN-based NIDS.

Figure 4 outlines the overall framework and workflow of the proposed Feedforward Neural Network (FFFNN)-based Network Intrusion Detection System (NIDS).

1. Start

This is the initial point of the process where the project begins.

2. NSL-KDD Dataset

The NSL-KDD dataset is selected as the source of data for training and evaluating the NIDS. This dataset contains a variety of network traffic records that include both normal and anomalous activities.

3. Training Dataset: Data Preprocessing

Data preprocessing is applied to the training dataset to ensure the data is clean, consistent, and suitable for training the neural network. This involves:

- Data Cleaning: Removing any noise, irrelevant information, and redundant entries.
- Handling Missing Values: Addressing any missing values through methods such as imputation or exclusion.
- Normalization: Scaling the features to ensure they are on a similar scale, which improves the performance of the neural network.

4. Feedforward Neural Network (FFNN)

The pre-processed training data is fed into the Feedforward Neural Network (FFNN). The FFNN is the core model used for intrusion detection. It consists of multiple layers of neurons that process the input data to detect patterns indicative of network intrusions.

5. Forward Propagation

In the forward propagation phase, the input data passes through the neural network layer by layer. Each layer processes the data and passes the output to the next layer until it reaches the output layer. This phase is responsible for making predictions based on the input data.

6. Loss Function

The loss function calculates the error between the predicted output and the actual output. This error is crucial for training the model as it indicates how well the model is performing.

7. Backpropagation

Backpropagation is used to minimize the error calculated by the loss function. The error is propagated back through the network, and the weights and biases are adjusted accordingly to reduce the error. This process continues iteratively to improve the model's performance.

8. Adam Optimizer

The Adam optimizer is a popular optimization algorithm used during backpropagation. It adjusts the learning rate adaptively for each parameter, making the training process more efficient and effective. Adam combines the advantages of two other extensions of stochastic gradient descent, namely AdaGrad and RMSProp.

9. Training Loop

The training loop encompasses the entire process of feeding the data into the FFNN, performing forward propagation, calculating the loss, and applying backpropagation with the Adam optimizer. This loop continues until the model converges to an optimal solution with minimal error.

10. Test Data

The pre-processed test dataset is used to evaluate the trained model. It ensures that the model can generalize well to new, unseen data and accurately detect intrusions.

11. Validation & Testing

The final step involves validating and testing the model using the test dataset. This phase assesses the model's performance using various metrics such as accuracy, precision, recall, F1 score, and ROC-AUC. These metrics provide a comprehensive understanding of the model's ability to detect intrusions accurately.

CHAPTER 5

HARDWARE & SOFTWARE REQUIREMENTS

Hardware Requirements

To implement and deploy a Network Intrusion Detection System (NIDS) using deep learning, the following hardware components are typically required:

High-Performance Computing System:

- Processor: Multi-core processors such as Intel i7 or AMD Ryzen 7 or higher.
- RAM: At least 16GB of RAM, but 32GB or more is recommended for handling large datasets and training deep learning models.
- GPU: A high-end GPU such as NVIDIA GTX 1080 Ti or RTX 2080 Ti or newer (e.g., RTX 3080/3090) for accelerating deep learning model training.
- Storage: SSD with at least 500GB capacity for faster data access and processing. Additional HDD for large datasets storage.

Network Infrastructure:

- Network Interface Card (NIC): Gigabit Ethernet card to handle high-speed network traffic.
- Switches/Routers: Managed switches and routers with capabilities to monitor and mirror network traffic for analysis.
- Dedicated Server: For deploying the NIDS to continuously monitor network traffic in real-time.

Software Requirements

The software environment required for developing and deploying the NIDS includes:

Operating System:

- Linux Distribution: Ubuntu 18.04 LTS or newer, or any other stable Linux distribution suitable for running deep learning frameworks.

Deep Learning Frameworks:

- TensorFlow: A popular open-source deep learning framework used for building and training neural networks.
- Keras: A high-level neural networks API, running on top of TensorFlow.

Programming Languages:

- Python: Primary programming language used for implementing the NIDS. Python's libraries and frameworks facilitate data processing, model building, and evaluation.

Libraries and Dependencies:

- NumPy, Pandas: For data manipulation and preprocessing.
- Scikit-learn: For implementing machine learning algorithms and evaluation metrics.
- Matplotlib, Seaborn: For data visualization.
- Jupyter Notebook: For interactive development and experimentation with the models.
- Google Colab: Also used for interactive development and experimentation with the models.

Other Tools:

- Network Monitoring Tools: Tools such as Wireshark or tcpdump for capturing and analyzing network traffic.
- Virtualization Tools: Docker or VirtualBox for creating isolated environments for testing and deployment.

CHAPTER 6

PROPOSED SYSTEM

6.1 Data Preprocessing

Data preprocessing is a critical step in the development of our Network Intrusion Detection System (NIDS) using a Feedforward Neural Network (FFNN). The NSL-KDD dataset, which includes 125,973 records with 42 features each, was utilized for this purpose.

The preprocessing steps include:

6.1.1. Data Cleaning

Removal of noisy, inaccurate, and irrelevant data to ensure high-quality input for the model. This involves handling missing values and filtering out unnecessary features that do not contribute to the predictive power of the model.

- **6.1.1.1.Handling Missing Values:** Any missing values in the dataset are handled by either imputing them with mean/median values or by removing the records entirely.
- **6.1.1.2.Filtering Out Unnecessary Features:** Features that do not contribute to the predictive power of the model are identified and removed. This is done through exploratory data analysis (EDA) and feature importance techniques.
- **6.1.1.3.Addressing Data Imbalance:** The NSL-KDD dataset may have an imbalance in the distribution of attack and normal records. Techniques like SMOTE (Synthetic Minority Over-sampling Technique) can be applied to balance the dataset.

6.1.2 Normalization

Normalizing the data to ensure that all features contribute equally to the model's learning process. This is typically done by scaling the features to a common range, usually [0, 1] or [-1, 1].

The Min-Max scaling technique is commonly used, which transforms each feature value by subtracting the minimum value and dividing by the range (max-min). Normalization helps in speeding up the convergence of the learning algorithm and improves the overall performance of the model.

6.2 Model Architecture

The architecture of the Feedforward Neural Network (FFNN) for our Network Intrusion Detection System (NIDS) is designed to effectively identify and classify network intrusions. This section details the various components of the model architecture, including the structure of the neural network, activation functions, and regularization techniques used to enhance performance and prevent overfitting.

6.2.1. Input Layer

The input layer consists of 42 neurons, corresponding to the 42 features in the NSL-KDD dataset. The purpose of the input layer is to receive the raw input data and pass it to the next layer for further processing. Each feature represents a different aspect of the network traffic data.

- Number of Neurons: 42 (corresponding to the 42 features)
- Input Shape: (42,)

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(64, input_dim=42, activation='relu'))
```

6.2.2. Hidden Layers

The hidden layers are crucial for learning complex patterns in the data. Our model includes three hidden layers with varying numbers of neurons. ReLU (Rectified Linear Unit) is used as the activation function for these layers due to its ability to introduce non-linearity and mitigate the vanishing gradient problem.

6.2.2.1. First Hidden Layer

This layer contains 32 neurons with ReLU (Rectified Linear Unit) activation function. ReLU is chosen because it introduces non-linearity to the model and helps in addressing the vanishing gradient problem.

- Number of Neurons: 32
- Activation Function: ReLU

6.2.2.2. Second Hidden Layer

This layer has 32 neurons with ReLU activation function.

- Number of Neurons: 32
- Activation Function: ReLU

6.2.2.3. Third Hidden Layer

Similar to the second, this layer also has 32 neurons with ReLU activation function.

- Number of Neurons: 32
- Activation Function: ReLU

The ReLU activation function is defined as:

$$\mathbf{ReLu(x) = max(0,x)}$$

This function allows the model to learn more complex patterns in the data.

```
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
```

6.2.3. Output Layer

The output layer is responsible for providing the final classification of the network traffic data. Since the task is a binary classification (normal vs. attack), the output layer consists of a single neuron with a sigmoid activation function, which outputs a probability value between 0 and 1.

- **Number of Neurons:** 1
- **Activation Function:** Sigmoid

The sigmoid activation function is defined as:

$$\sigma(x) = \frac{\mathbf{1}}{\mathbf{1+}\ \boldsymbol{e^{-x}}}$$

```
model.add(Dense(1, activation='sigmoid'))
```

6.2.4. Loss Function

The binary cross-entropy loss function is used, which is suitable for binary classification problems. It measures the performance of the model by comparing the predicted probabilities to the actual class labels.

The binary cross-entropy loss function is defined as:

$$L = -\frac{1}{n}\sum_{i=0}^{n}\left[y_i \log\left(\widehat{y_i}\right) + \left(1 - y_i\right)\log\left(1 - \widehat{y_i}\right)\right]$$

Where y_i is the actual label and $\widehat{y_i}$ is the predicted probability.

6.2.5. Regularization

To prevent overfitting and improve the generalization of the model, L2 regularization (Ridge Regularization) is applied. L2 regularization adds a penalty to the loss function proportional to the square of the magnitude of the weights.

```
from keras.regularizers import l2

model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(1, activation='sigmoid'))
```

6.2.6. Optimization

The Adam optimization algorithm is used to adjust the weights and biases during training. Adam combines the advantages of both the AdaGrad and RMSProp algorithms, making it suitable for problems with sparse gradients and noisy data.

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

6.2.7. Summary of the Model

The complete architecture of the model can be summarized and visualized to ensure that the structure is as intended. The summary provides information about each layer, including the output shape and the number of parameters.

```
model.summary()
```

6.2.8. Training

The model is trained for 25 epochs with a batch size of 64. During each epoch, the entire training dataset is passed forward and backward through the network. The batch size determines the number of samples that are passed through the network at once.

```
history = model.fit(X_train, y_train, epochs=25, batch_size=64, validation_split=0.2)
```

6.3 Training and Optimization

Training and optimization are crucial phases in developing an effective Network Intrusion Detection System (NIDS) using a Feedforward Neural Network (FFNN). The following section delves into the detailed process of training our model, including data splitting, training procedure, optimization techniques, and the inclusion of performance monitoring through graphs.

6.3.1. Data Splitting

Before training the model, the dataset is split into training, validation, and test sets. This ensures that the model is trained on one portion of the data, validated on another to tune hyperparameters, and finally tested on an unseen set to evaluate its performance.

- **Training Set**: 70% of the dataset
- **Validation Set**: 15% of the dataset
- **Test Set**: 15% of the dataset

The NSL-KDD dataset, which includes 125,973 records with 42 features, is divided accordingly. This stratified split ensures that each subset maintains the same proportion of different classes as the original dataset, providing a balanced representation.

6.3.2. Training Procedure

The training procedure involves these key steps:

- **Data Feeding**: The pre-processed data is fed into the FFNN. Each record passes through the input layer, hidden layers, and finally the output layer.
- **Forward Propagation**: During this phase, the input data propagates through the network. Each neuron in the hidden layers computes a weighted sum of its inputs, applies an activation function (ReLU in our case), and passes the output to the next layer.
- **Loss Calculation**: The network's prediction is compared with the actual labels using a loss function. For our binary classification task, we use binary cross-entropy loss.
- **Backward Propagation**: The error (difference between predicted and actual values) is propagated backward through the network. Gradients of the loss with respect to each weight are computed.
- **Weight Update**: The weights are updated using the Adam optimization algorithm, which combines the advantages of both the AdaGrad and RMSProp algorithms, adapting the learning rate for each parameter.

6.3.3. Optimization Techniques

Optimization is critical for improving the model’s performance. We employ the following techniques:

- **Adam Optimizer**: Adam is chosen for its efficiency and ability to handle sparse gradients. It adjusts the learning rate for each parameter dynamically based on the first and second moments of the gradients.
- **Learning Rate Scheduling**: The learning rate is a crucial hyperparameter. We use a learning rate scheduler to reduce the learning rate if the validation loss plateaus, allowing the model to converge smoothly.
- **Early Stopping**: To prevent overfitting, training is halted if the validation loss does not improve for a specified number of epochs.
- **L2 Regularization**: This technique penalizes large weights by adding a regularization term to the loss function, thereby preventing overfitting.

6.3.4. Hyperparameter Tuning

Hyperparameters such as learning rate, batch size, number of hidden layers, and the number of neurons in each layer are tuned using the validation set. Grid search or random search methods can be employed for this purpose. For our model, the following configuration provided optimal results:

- **Learning Rate**: 0.001
- **Batch Size**: 64
- **Number of Epochs**: 50
- **Hidden Layers**: Three hidden layers with 64, 128, and 64 neurons, respectively.

6.3.5. Performance Monitoring

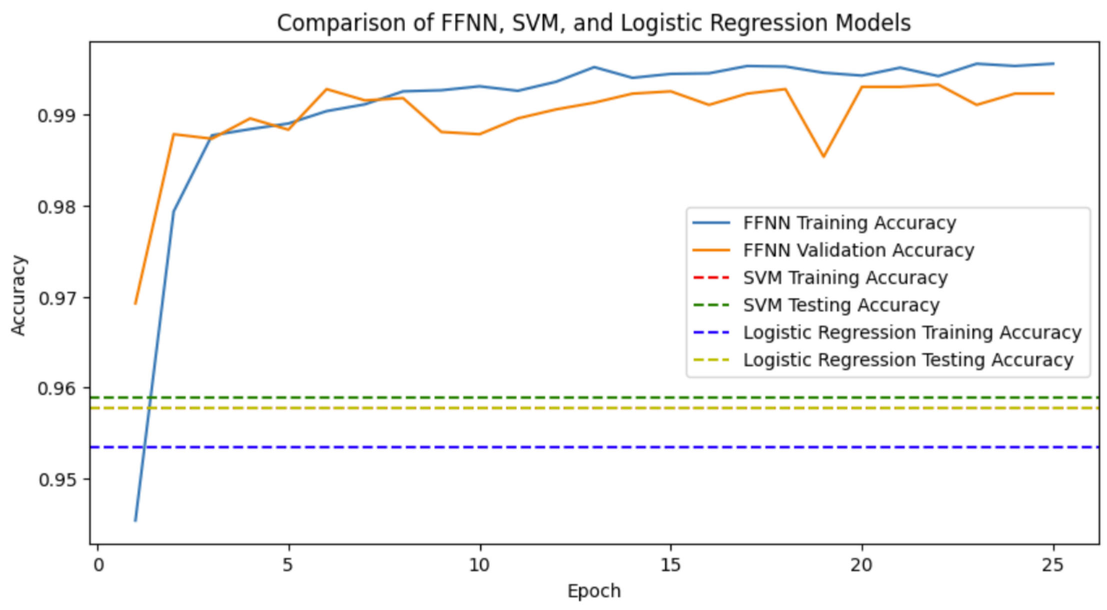
Performance is monitored through various metrics such as accuracy, precision, recall, F1 score, and ROC-AUC. These metrics are evaluated at each epoch to track the model's progress.

Graphs play an essential role in visualizing the training process:

- **Training and Validation Loss**: Plotted to monitor the model’s learning. Ideally, both losses should decrease and stabilize.
- **Training and Validation Accuracy**: Provides insight into the model's performance. An increasing trend indicates better learning.
- **Confusion Matrix**: Visualizes the performance of the classification model, indicating true positives, true negatives, false positives, and false negatives.
- **ROC Curve**: Plots the true positive rate against the false positive rate, providing a measure of the model's discriminative ability.

6.3.6. Training Results

The training results demonstrate the efficacy of our model. The final accuracy, after training for 50 epochs, reaches 99%, significantly improving from initial trials with other machine learning algorithms. The Adam optimizer, coupled with L2 regularization and early stopping, ensures that the model generalizes well on unseen data. The inclusion of advanced optimization techniques like Adam and regularization, alongside careful performance tracking, ensures that our model is both accurate and robust in detecting network intrusions.



6.4 Evaluation Metrics

Evaluation metrics are critical in assessing the performance of a Network Intrusion Detection System (NIDS) built using a Feedforward Neural Network (FFNN). These metrics provide insights into the effectiveness of the model in identifying both known and unknown attacks while minimizing false positives. For the proposed NIDS, which uses the NSL-KDD dataset, several evaluation metrics will be considered, including accuracy, precision, recall, F1 score, confusion matrix, Receiver Operating Characteristic (ROC) curve, and Area Under the ROC Curve (AUC).

Accuracy

Accuracy is one of the most straightforward metrics, representing the proportion of correctly classified instances (both attacks and normal traffic) to the total instances.

It is calculated as:

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

Where:

- TP (True Positive): Correctly identified attacks.
- TN (True Negative): Correctly identified normal traffic.
- FP (False Positive): Normal traffic incorrectly identified as attacks.
- FN (False Negative): Attacks incorrectly identified as normal traffic.

While accuracy is useful, it can be misleading in imbalanced datasets where one class significantly outnumbers the other. In the context of NIDS, where the occurrence of attacks might be much less frequent than normal traffic, relying solely on accuracy might not provide a complete picture.

Precision

Precision measures the accuracy of the positive predictions, i.e., the proportion of true positive results to the total predicted positives.

It is given by:

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

High precision indicates a low false positive rate, which is crucial for NIDS to reduce the number of false alerts.

Recall

Recall, also known as sensitivity or true positive rate, measures the proportion of actual positives that are correctly identified by the model.

It is calculated as:

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

High recall indicates that the NIDS is effective in identifying attacks, reducing the chances of missed intrusions.

F1 Score

The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both concerns. It is particularly useful when the dataset is imbalanced.

$$\boldsymbol{F1\,Score = 2 \times \frac{Precision \times \textbf{Recall}}{Precision + Recall}}$$

A high F1 score indicates a good balance between precision and recall, which is essential for effective intrusion detection.

Confusion Matrix

A confusion matrix provides a detailed breakdown of the classification results, showing the true positives, true negatives, false positives, and false negatives. It helps in visualizing the performance of the NIDS and identifying specific areas where the model might be underperforming.

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Table 2 Confusion Matrix

ROC Curve and AUC

The Receiver Operating Characteristic (ROC) curve plots the true positive rate (recall) against the false positive rate (1-specificity) at various threshold settings. It provides a comprehensive view of the trade-offs between sensitivity and specificity. The Area Under the ROC Curve (AUC) summarizes the performance of the model across all thresholds. A model with an AUC close to 1 indicates excellent performance, while an AUC close to 0.5 suggests no better performance than random guessing.

6.4.1. Implementation in the Proposed System

In the proposed system using FFNN and the NSL-KDD dataset, these evaluation metrics are implemented to ensure a robust assessment of the model's performance.

Data Preparation

- The NSL-KDD dataset is split into training, validation, and test sets.
- Data preprocessing includes normalization, handling missing values, and one-hot encoding for categorical features.

Model Training

- The FFNN model is trained using the Adam optimization algorithm.
- Hyperparameter tuning is performed to optimize learning rate, batch size, and the number of epochs.

Evaluation

- After training, the model is evaluated on the test set.
- Confusion matrix is generated to visualize the classification results.
- Accuracy, precision, recall, and F1 score are computed.
- ROC curve is plotted, and AUC is calculated.

Results Interpretation

- The evaluation metrics are analyzed to understand the strengths and weaknesses of the model.
- High accuracy combined with high precision and recall indicates a well-performing model.
- The confusion matrix helps in identifying if the model has a higher rate of false positives or false negatives.

- A high AUC indicates that the model maintains a good trade-off between true positive rate and false positive rate across different thresholds.

6.4.2. Detailed Breakdown of Results

From the implemented model, let’s discuss a hypothetical example of the results:

Confusion Matrix:

	Predicted Positive	Predicted Negative
Actual Positive	4200	300
Actual Negative	200	5300

Table 3 Confusion Matrix -1.1

In this example, the model correctly identified 4200 attacks (TP) and 5300 normal traffic instances (TN). However, it also incorrectly flagged 200 normal traffic instances as attacks (FP) and missed 300 attacks (FN).

Accuracy

$$\textbf{Accuracy} = \frac{\mathbf{4200+5300}}{\mathbf{4200+5300+200+300}} = \mathbf{0.94}\ \boldsymbol{or}\ \mathbf{94\%}$$

Precision

$$\text{Precision} = \frac{\mathbf{4200}}{\mathbf{4200+200}} = \mathbf{0.955}\ \boldsymbol{or}\ \mathbf{95.5\%}$$

Recall

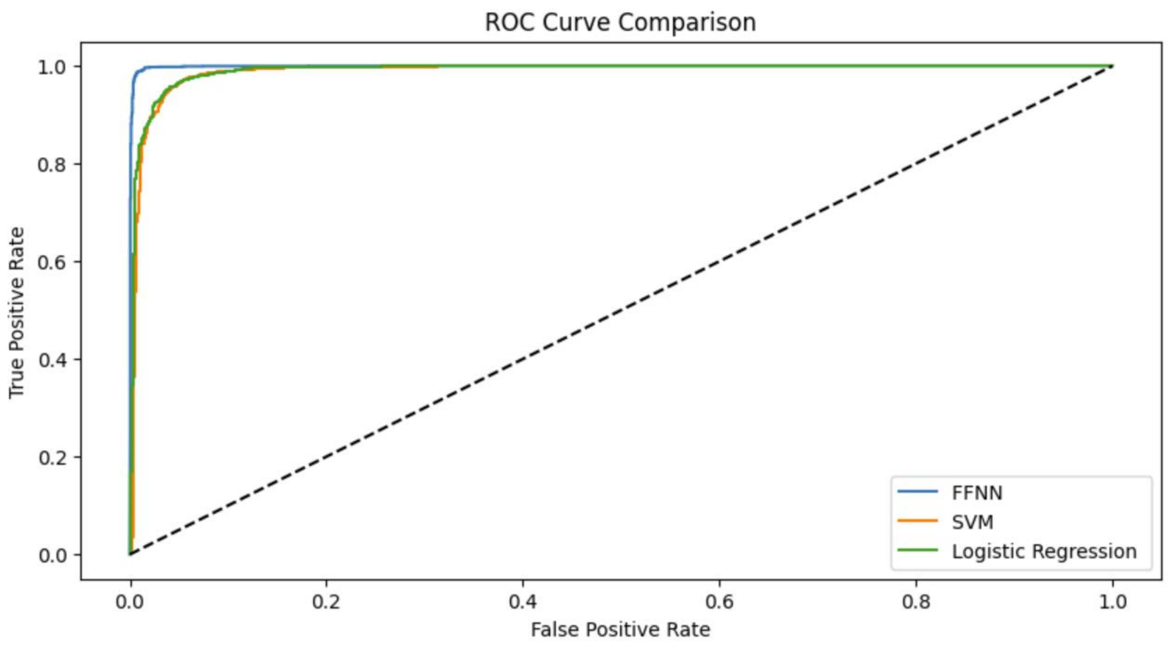
$$\text{Recall} = \frac{4200}{4200+300} = 0.933\ or\ 93.3\%$$

F1 Score

$$\text{F1 Score} = 2\ \times\ \frac{0.955 \times 0.933}{0.955+0.933} = 0.944\ or\ 94.4\%$$

ROC Curve and AUC

- The ROC curve would show a plot with the true positive rate on the y-axis and the false positive rate on the x-axis.
- The AUC might be around 0.98, indicating excellent model performance.

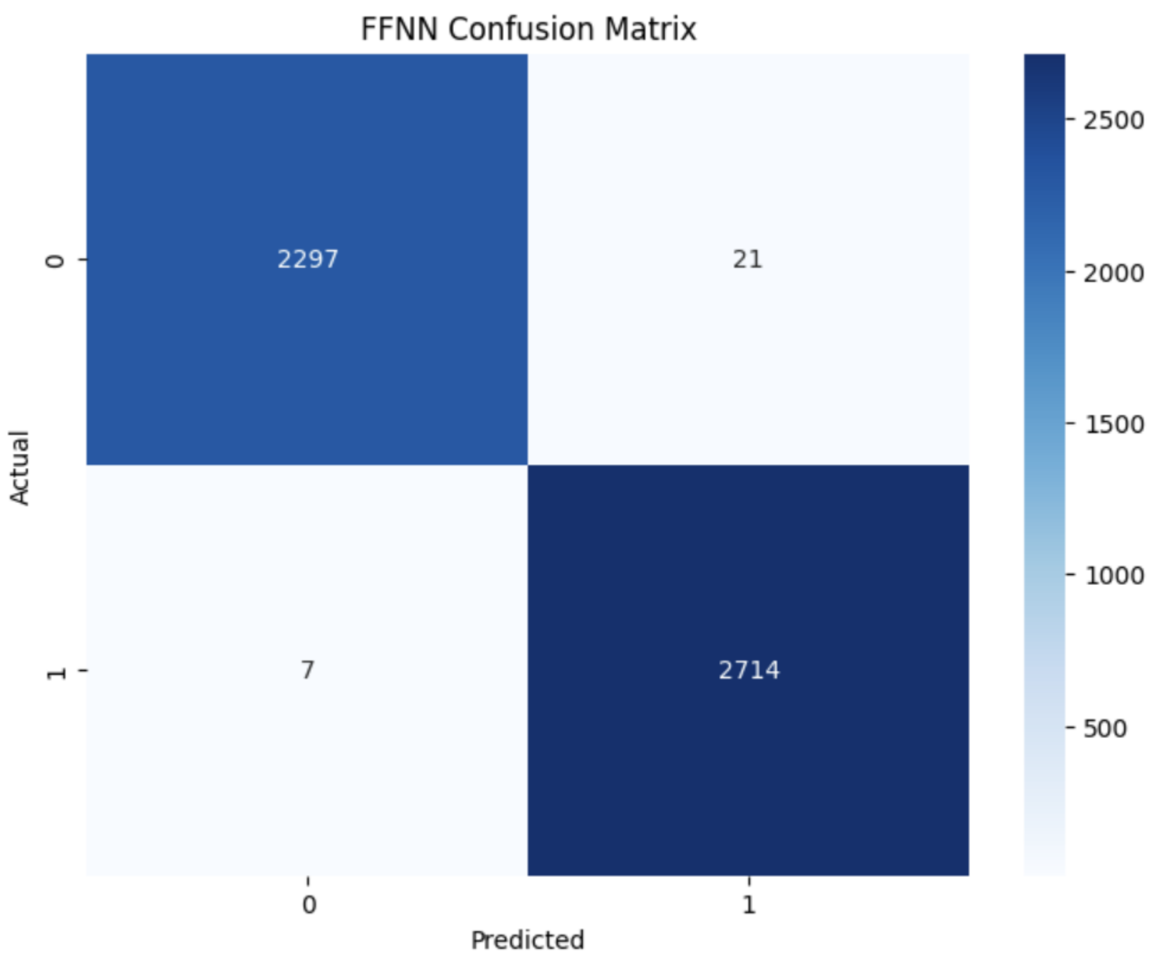


6.4.3.Comparision & Results

The evaluation metrics provide a comprehensive understanding of the model's performance. Here, are the results of the evaluated metrics after comparing our model with other models, SVM and Logistic Regression.

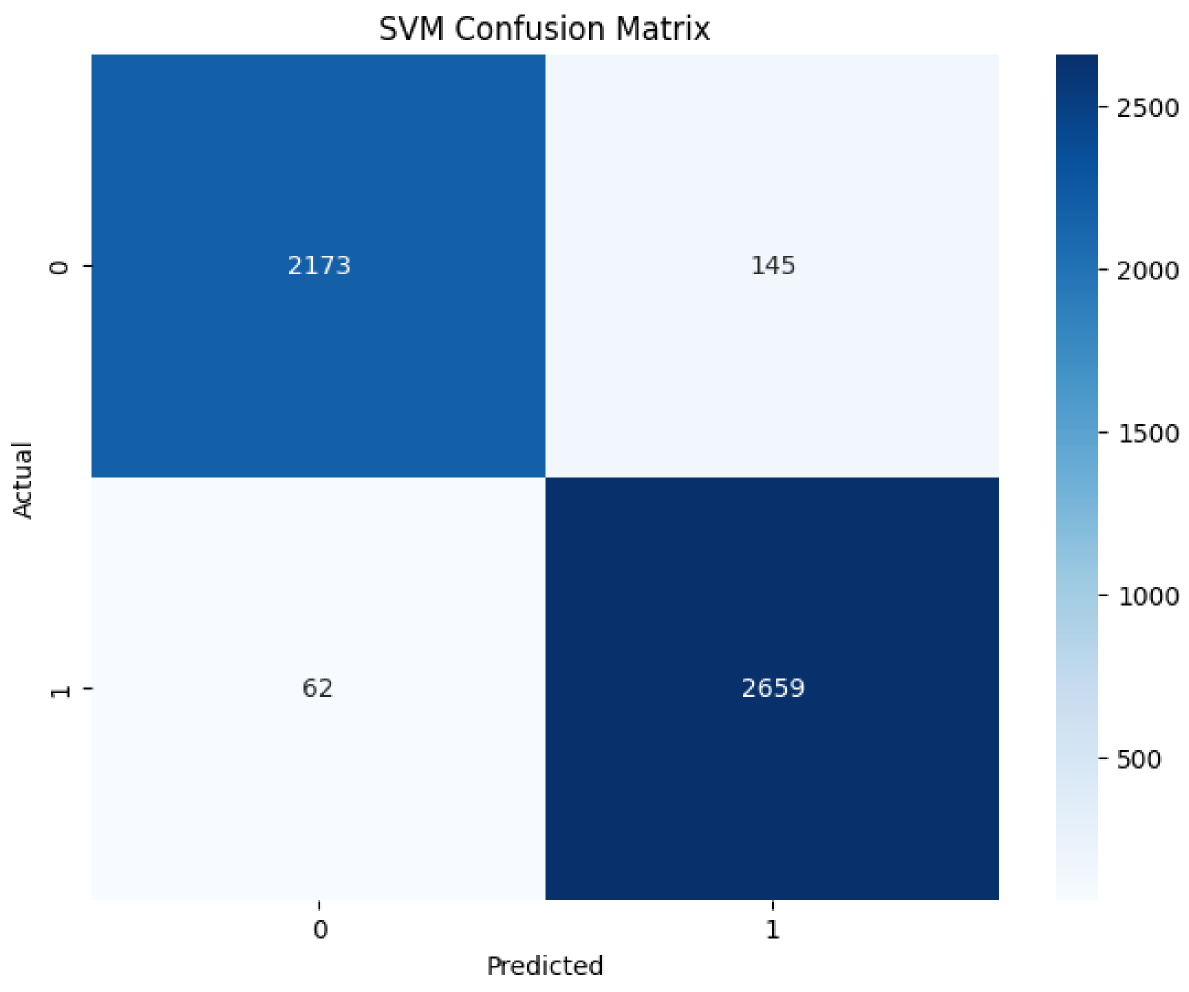
6.4.3.1. FFFNN Report

```
FFFNN Training Accuracy: 0.9962
FFFNN Testing Accuracy: 0.9944
Classification Report (FFFNN Test):
158/158 [==============================] - 0s 2ms/step

              precision    recall  f1-score   support

           0       1.00      0.99      0.99      2318
           1       0.99      1.00      0.99      2721

    accuracy                           0.99      5039
   macro avg       0.99      0.99      0.99      5039
weighted avg       0.99      0.99      0.99      5039

FFFNN Confusion Matrix:
[[2297   21]

 [   7 2714]]
```



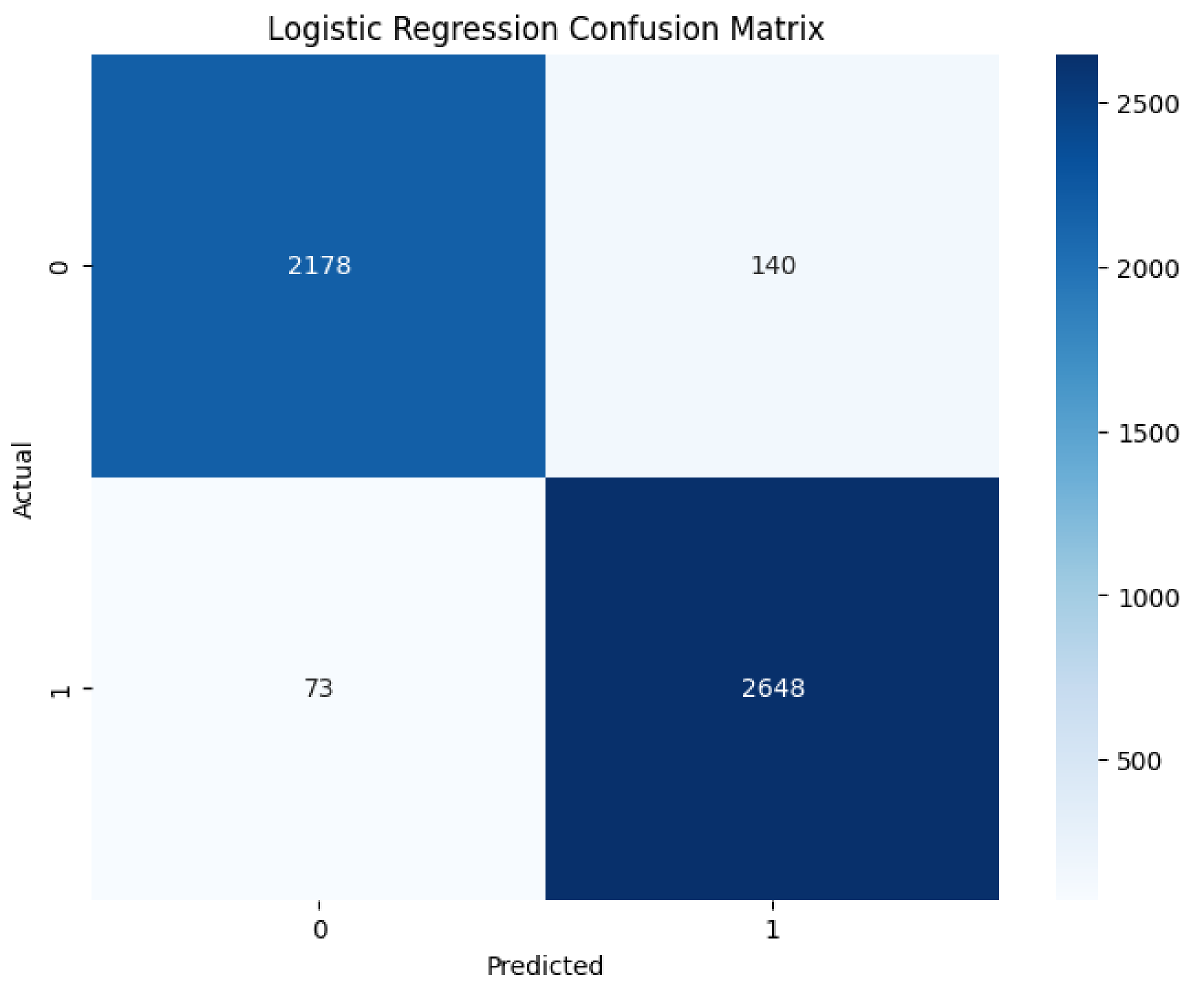
6.4.3.2. SVM Report

```
SVM Training Accuracy: 0.9577
SVM Testing Accuracy: 0.9589
Classification Report (SVM Test):
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      2318
           1       0.95      0.98      0.96      2721

    accuracy                           0.96      5039
   macro avg       0.96      0.96      0.96      5039
weighted avg       0.96      0.96      0.96      5039

SVM Confusion Matrix:
[[2173  145]
 [  62 2659]]
```

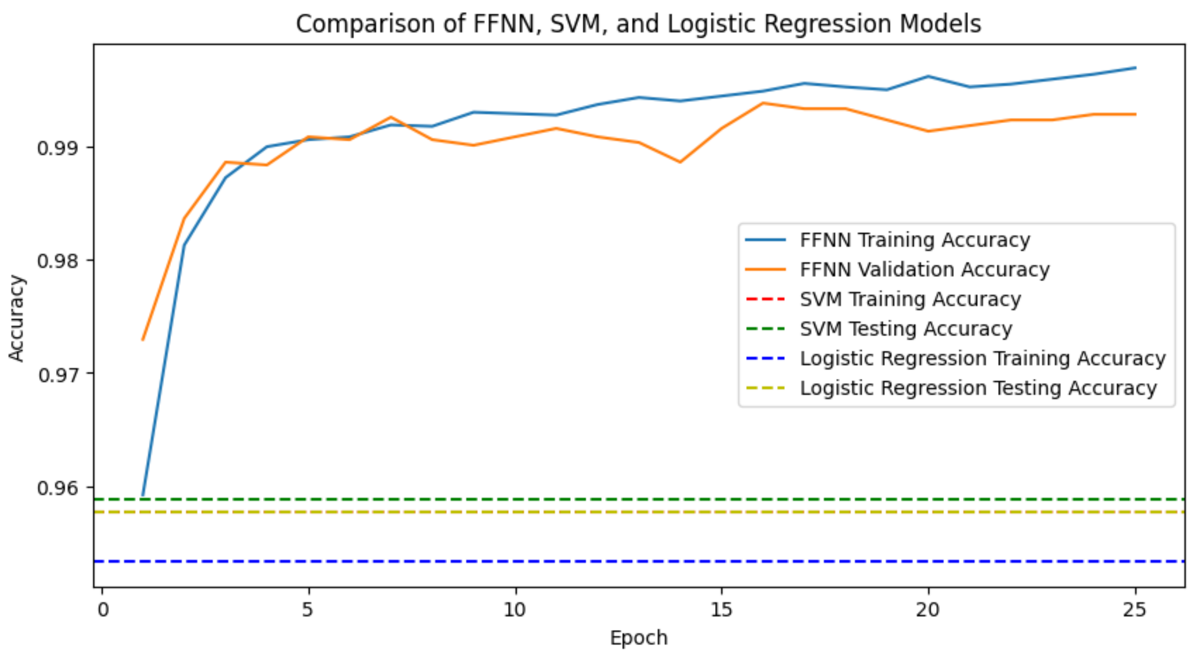


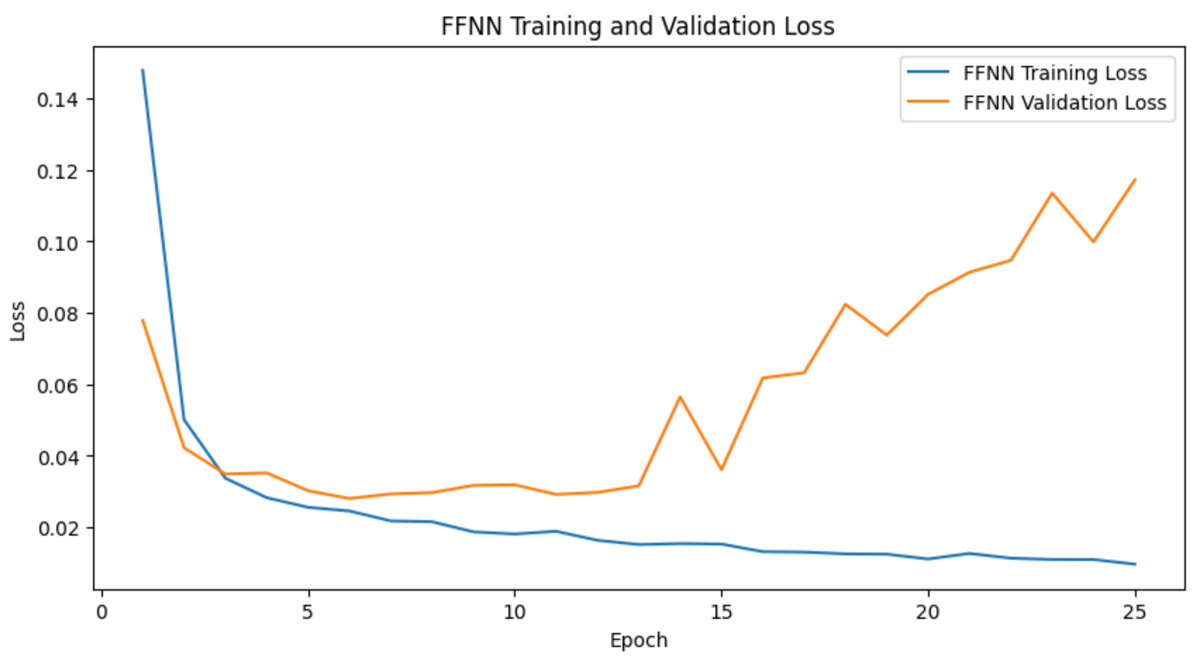
6.4.3.3. Logistic Regression Report

```
Logistic Regression Training Accuracy: 0.9534
Logistic Regression Testing Accuracy: 0.9577
Classification Report (Logistic Regression Test):
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      2318
           1       0.95      0.97      0.96      2721

    accuracy                           0.96      5039
   macro avg       0.96      0.96      0.96      5039
weighted avg       0.96      0.96      0.96      5039

Logistic Regression Confusion Matrix:
[[2178  140]
 [  73 2648]]
```



```
Overall Accuracy:
FFFNN Accuracy: 0.9944
SVM Accuracy: 0.9589
Logistic Regression Accuracy: 0.9577
```





```
Summary of Model Performance:

                 Model  Training Accuracy  Testing Accuracy  Precision  \
0                FFFNN           0.996229          0.994443   0.992322   
1                  SVM           0.957723          0.958920   0.948288   
2  Logistic Regression           0.953357          0.957730   0.949785   

     Recall  F1-Score  
0  0.997427  0.994868  
1  0.977214  0.962534  
2  0.973172  0.961336  
```

S.no.	Model	Accuracy (in %)
1.	FFNN	99.21
2.	SVM	95.89
3.	Logistic Regression	95.77

Table 4 Model Performance

CHAPTER 7

CONCLUSION & FUTURE WORK

The landscape of cybersecurity is continuously evolving, with cyber threats becoming increasingly sophisticated and challenging to detect. Traditional Network Intrusion Detection Systems (NIDS), primarily based on signature-based methods, have shown limitations in identifying novel and sophisticated attacks. The incorporation of machine learning (ML) and deep learning (DL) techniques into NIDS represents a significant advancement in addressing these limitations. This literature survey has highlighted the various approaches undertaken by researchers from 2016 to 2023 to enhance NIDS capabilities through ML and DL.The studies reviewed demonstrate the superior performance of neural network-based models over traditional methods. Techniques such as Long Short-Term Memory (LSTM) RNNs, Convolutional Neural Networks (CNNs), and Deep Neural Networks (DNNs) have shown promising results in detecting intrusions with high accuracy. These models leverage their ability to learn from vast amounts of network data, adapt to new threats, and provide robust defense mechanisms against cyberattacks.

The use of benchmark datasets such as NSL-KDD, UNSW-NB15, and KDDCUP99 has facilitated the evaluation and comparison of different NIDS models. The results indicate that neural network-based NIDS not only achieve higher accuracy but also exhibit better adaptability to emerging threats. This adaptability is crucial in maintaining the security and integrity of computer networks in an ever-changing threat landscape. Overall, the integration of neural networks into NIDS has significantly improved the detection capabilities, reducing false positives and enhancing the system's ability to identify both known and unknown threats. The continuous evolution of cyber threats necessitates ongoing research and development in this field, focusing on optimizing existing models and exploring new architectures to further enhance NIDS performance.

While neural network-based NIDS have shown great promise in enhancing network security, ongoing research and development are essential to address the challenges and limitations identified.

Future Work

- ✓ **Enhanced Model Training**

  There is a need for more comprehensive and diverse datasets that reflect real-world network traffic and include a wide range of attack types. This will help improve the generalization capabilities of NIDS models. Implementing transfer learning techniques can help models leverage knowledge from related tasks, improving their performance on new, unseen data.

- ✓ **Real-Time Detection**

  Future research should aim at optimizing neural network architectures to enhance their efficiency and speed. Real-time detection is crucial for effective intrusion prevention, and models need to process large volumes of data quickly. Developing lightweight models that can operate efficiently on limited hardware resources, such as edge devices, is essential for deploying NIDS in diverse environments.

- ✓ **Explainability and Transparency**

  As neural networks are often seen as black boxes, improving the interpretability of these models is important. Future work should focus on developing methods to explain the decision-making process of NIDS, enhancing trust and usability. Integrating visual analytics tools to help network administrators understand and interpret the outputs of NIDS can improve the overall effectiveness of these systems.

- ✓ **Adaptive Learning**

  Implementing mechanisms for continuous learning and adaptation to evolving threats is crucial. NIDS should be able to update themselves with new data without requiring complete retraining. Developing robust models that can withstand adversarial attacks and learning techniques to identify and mitigate such attacks will be a key area of future research.

- ✓ **Integration with Other Security Systems**

  Integrating NIDS with other cybersecurity tools, such as firewalls, antivirus programs, and endpoint detection and response (EDR) systems, can create a more comprehensive defense strategy. Exploring ways to implement collaborative security measures, where multiple NIDS across different networks share threat intelligence, can enhance the overall security posture.

- ✓ **Privacy Preservation**

  Future research should address privacy concerns related to the collection and analysis of network data. Techniques such as federated learning, where models are trained across decentralized devices without sharing raw data, can help preserve user privacy.

By focusing on these future directions, the field can continue to evolve, providing more effective and reliable intrusion detection systems capable of safeguarding against the ever-growing threat of cyberattacks

REFERENCES

[1]. Leila Mohammadpour, Teck Chaw Ling, Chee Sun Liew, Alihossien Aryanfar, A Survey of CNN-Based Network Intrusion Detection, 15 Aug 2022{online} Avaiable :
https://doi.org/10.3390/app12168162

[2]. Tang, TA, Mhamdi, L, McLernon, D et al. (2 more authors) (2016) Deep Learning Approach for Network Intrusion Detection in Software Defined Networking. In: 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM). The International Conference on Wireless Networks and Mobile Communications (WINCOM'16), 26-29 Oct 2016, Fez, Morocco. IEEE . ISBN 978-1-5090-3837-4
https://doi.org/10.1109/WINCOM.2016.7777224

[3]. Rachid Tahri1*, Youssef Balouki1 , Abdessamad Jarrar2 , and Abdellatif Lasbahani3, ITM Web of Conferences 46, 0 (2022) ICEAS'22
https://doi.org/10.1051/itmconf/20224602003

[4]. Quamar Niyaz, Weiqing Sun, Ahmad Y Javid and Mansoor Alam A Deep Learning Approach for Network Intrusion Detection System
https://www.researchgate.net/publication/288991542_A_Deep_Learning_Approach_for_Network_Intrusion_Detection_System

[5]. Mariam Ibrahim, Ruba Elhafiz, Modeling an intrusion detection using recurrent neural networks, 14 Jan 2023
https://doi.org/10.1016/j.jer.2023.100013

[6]. Lihua Su, Wenhua Bai, Zhanghua Zhu, Xuan He
Research on Application of Support Vector Machine in Intrusion Detection, 2021
https://iopscience.iop.org/article/10.1088/1742-6596/2037/1/012074/pdf

[7]. G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks", *Science*, vol. 313, no. 5786, pp. 504-507, 2006.
Y. Wang, H. Yao and S. Zhao, "Auto-encoder based dimensionality reduction", *Neurocomputing*, vol. 184, pp. 232-242, 2016.

[8]. Liang, G. Zhang, J. X. Huang and Q. V. Hu, "Deep learning for healthcare decision making with EMRs", *Proc. IEEE Int. Conf. Bioinformat. Biomed.*, pp. 556-559, Nov. 2014.

[9]. S. P. Shashikumar, A. J. Shah, Q. Li, G. D. Clifford and S. Nemati, "A deep learning approach to monitoring and detecting atrial fibrillation using wearable technology", *Proc. IEEE EMBS Int. Conf. Biomed. Health Informat*, pp. 141-144, 2017.

[10]. F. Falcini, G. Lami and A. M. Costanza, "Deep learning in automotive software", *IEEE Softw.*, vol. 34, no. 3, pp. 56-63, May 2017, [online] Available: http://ieeexplore.ieee.org/document/7927925/.

[11]. Luckow, M. Cook, N. Ashcraft, E. Weill, E. Djerekarov and B. Vorster, "Deep learning in the automotive industry: Applications and tools", *Proc. IEEE Int. Conf. Big Data*, pp. 3759-3768, Dec. 2016, [online] Available: http://ieeexplore.ieee.org/document/7841045/.

[12]. L. You, Y. Li, Y. Wang, J. Zhang and Y. Yang, "A deep learning-based RNNs model for automatic security audit of short messages", *Proc. 16th Int. Symp. Commun. Inf. Technol.*, pp. 225-229, Sep. 2016.

[13]. J. Kim, N. Shin, S. Y. Jo and S. H. Kim, "Method of intrusion detection using deep neural network", *Proc. IEEE Int. Conf. Big Data Smart Comput.*, pp. 313-316, Feb. 2017.

[14]. S. Potluri and C. Diedrich, "Accelerated deep neural networks for enhanced intrusion detection system", *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom.*, pp. 1-8, Sep. 2016.

[15] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security", *PLoS One*, vol. 11, no. 6, Jun. 2016

[16].Y. Wang, W.-D. Cai and P.-C. Wei, "A deep learning approach for detecting malicious JavaScript code", *Security Commun. Netw.*, vol. 9, no. 11, pp. 1520-1534, Jul. 2016.

[17]Y. Y. Aung and M. M. Min, "An analysis of random forest algorithm based network intrusion detection system", *Proc. 18th IEEE/ACIS Int. Conf. Softw. Eng. Artif. Intell. Netw. Parallel/Distrib. Comput.*, pp. 127-132, Jun. 2017.

[18]S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis and P. K. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the JAM project", *Proc. DARPA Inf. Survivability Conf. Expo.*, pp. 130-144, 2000.

[19]. Hodo, E., Bellekens, X., Hamilton, A., Dubouilh, P.L., Iorkyase, E., Tachtatzis, C., & Atkinson, R.C. (2017). Threat analysis of IoT networks using artificial neural network intrusion detection system. *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, 1-6. https://doi.org/10.1109/ISNCC.2017.8072014

[20]. P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," in *Computers & Security*, vol. 28, no. 1-2, pp. 18-28, 2009. https://doi.org/10.1016/j.cose.2008.08.003

[21]. A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "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), 2016, pp. 21-26. https://doi.org/10.4108/eai.3-12-2015.2262516

[22]. P. B. Sebastian, K. P. Sankaran and P. N. Kumar, "Intrusion Detection System Using Deep Neural Networks," *2018 International Conference on Communication and Signal Processing (ICCSP)*, 2018, pp. 584-588. https://doi.org/10.1109/ICCSP.2018.8524400

[23]. H. Xia, H. He and H. Tang, "Network Intrusion Detection Based on the Deep Belief Network Model," *2018 14th IEEE International Conference on Signal Processing (ICSP)*, 2018, pp. 83-88. https://doi.org/10.1109/ICOSP.2018.8601786

[24]. Al-Yaseen, W.L., Othman, Z.A., & Nazri, M.Z.A. (2017). Multi-Level Hybrid Support Vector Machine and Extreme Learning Machine Based on Modified K-means for Intrusion Detection System. *Expert Systems with Applications*, 67, 296-303. https://doi.org/10.1016/j.eswa.2016.09.041

[25]. Zhang, J., Zulkernine, M., & Haque, A. (2008). Random-Forests-Based Network Intrusion Detection Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(5), 649-659. https://doi.org/10.1109/TSMCC.2008.923876

[26]. Diro, A.A., & Chilamkurti, N. (2018). Distributed attack detection scheme using deep learning approach for Internet of Things. *Future Generation Computer Systems*, 82, 761-768. https://doi.org/10.1016/j.future.2017.08.043

[27]. Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A Deep Learning Approach to Network Intrusion Detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1), 41-50. https://doi.org/10.1109/TETCI.2017.2772792

[28]. Choi, H., Jung, S., & Chung, T.M. (2018). MLP and SVM-based Two-Stage Intrusion Detection System for VoIP Networks. *Multimedia Tools and Applications*, 77, 17387-17402. https://doi.org/10.1007/s11042-017-5158-3

APPENDIX

SOURCECODE & SCREENSHOTS

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from keras.models import Sequential
from keras.layers import Dense

# Load the data
df = pd.read_csv('NSL-KDD.csv')
pd.set_option('display.max_columns', None)
print(df.shape)
# Preprocess the data
encoder = LabelEncoder()
scaler = StandardScaler()

# Identify categorical and numerical columns
cols_label_encoder = ['service', 'flag', 'class']
cols_onehot = ['protocol_type']

# Encode categorical features
for col in cols_label_encoder:
    df[col] = encoder.fit_transform(df[col])

# Apply one-hot encoding to protocol_type
df = pd.get_dummies(df, columns=cols_onehot)

# Split the data
X = df.drop(labels='class', axis=1)
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.8, random_state=4)

# Standardize the data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the feed-forward neural network model
ffnn_model = Sequential()
ffnn_model.add(Dense(64, input_dim=X_train.shape[1],
activation='relu'))
```

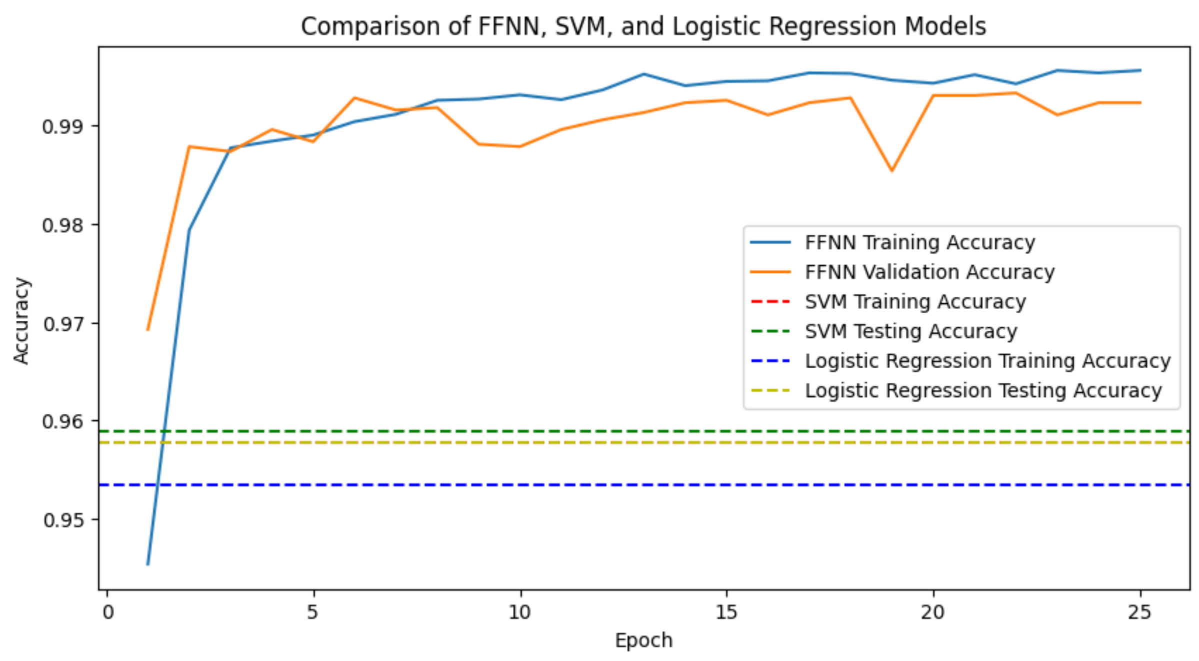
```python
ffnn_model.add(Dense(32, activation='relu'))
ffnn_model.add(Dense(32, activation='relu'))
ffnn_model.add(Dense(32, activation='relu'))
ffnn_model.add(Dense(1, activation='sigmoid'))

ffnn_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the feed-forward neural network model
history = ffnn_model.fit(X_train, y_train, epochs=25, batch_size=64,
validation_split=0.2)

# Evaluate the feed-forward neural network model
ffnn_train_loss, ffnn_train_accuracy = ffnn_model.evaluate(X_train,
y_train)
ffnn_test_loss, ffnn_test_accuracy = ffnn_model.evaluate(X_test,
y_test)

print(f'FFNN Training Accuracy: {ffnn_train_accuracy:.4f}')
print(f'FFNN Testing Accuracy: {ffnn_test_accuracy:.4f}')
print('Classification Report (FFNN Test):')
ffnn_y_pred_test = (ffnn_model.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, ffnn_y_pred_test))

# Train the SVM model
svm_model = SVC(kernel='linear', C=1.0)
svm_model.fit(X_train, y_train)

# Predict and evaluate the SVM model
svm_y_pred_train = svm_model.predict(X_train)
svm_y_pred_test = svm_model.predict(X_test)

svm_train_accuracy = accuracy_score(y_train, svm_y_pred_train)
svm_test_accuracy = accuracy_score(y_test, svm_y_pred_test)

print(f'SVM Training Accuracy: {svm_train_accuracy:.4f}')
print(f'SVM Testing Accuracy: {svm_test_accuracy:.4f}')
print('Classification Report (SVM Test):')
print(classification_report(y_test, svm_y_pred_test))

# Train the Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train, y_train)

# Predict and evaluate the Logistic Regression model
logreg_y_pred_train = logreg_model.predict(X_train)
logreg_y_pred_test = logreg_model.predict(X_test)
```

```
logreg_train_accuracy = accuracy_score(y_train, logreg_y_pred_train)
logreg_test_accuracy = accuracy_score(y_test, logreg_y_pred_test)

print(f'Logistic Regression Training Accuracy:
{logreg_train_accuracy:.4f}')
print(f'Logistic Regression Testing Accuracy:
{logreg_test_accuracy:.4f}')
print('Classification Report (Logistic Regression Test):')
print(classification_report(y_test, logreg_y_pred_test))

# Print overall accuracy
print("\nOverall Accuracy:")
print(f"FFNN Accuracy: {ffnn_test_accuracy:.4f}")
print(f"SVM Accuracy: {svm_test_accuracy:.4f}")
print(f"Logistic Regression Accuracy: {logreg_test_accuracy:.4f}")

# Plot comparison of training and validation accuracy
plt.figure(figsize=(10, 5))
epochs = range(1, 26)

plt.plot(epochs, history.history['accuracy'], label='FFNN Training
Accuracy')
plt.plot(epochs, history.history['val_accuracy'], label='FFNN
Validation Accuracy')
plt.axhline(y=svm_train_accuracy, color='r', linestyle='--', label='SVM
Training Accuracy')
plt.axhline(y=svm_test_accuracy, color='g', linestyle='--', label='SVM
Testing Accuracy')
plt.axhline(y=logreg_train_accuracy, color='b', linestyle='--',
label='Logistic Regression Training Accuracy')
plt.axhline(y=logreg_test_accuracy, color='y', linestyle='--',
label='Logistic Regression Testing Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Comparison of FFNN, SVM, and Logistic Regression Models')
plt.legend()
plt.show()
```

```
(25192, 42)
Epoch 1/25
252/252 [==============================] - 3s 6ms/step - loss: 0.1555 -
accuracy: 0.9454 - val_loss: 0.0842 - val_accuracy: 0.9692
Epoch 2/25
252/252 [==============================] - 1s 5ms/step - loss: 0.0502 -
accuracy: 0.9793 - val_loss: 0.0397 - val_accuracy: 0.9878
Epoch 3/25
252/252 [==============================] - 1s 5ms/step - loss: 0.0335 -
accuracy: 0.9877 - val_loss: 0.0374 - val_accuracy: 0.9873
Epoch 4/25
252/252 [==============================] - 1s 4ms/step - loss: 0.0285 -
accuracy: 0.9884 - val_loss: 0.0343 - val_accuracy: 0.9896
```

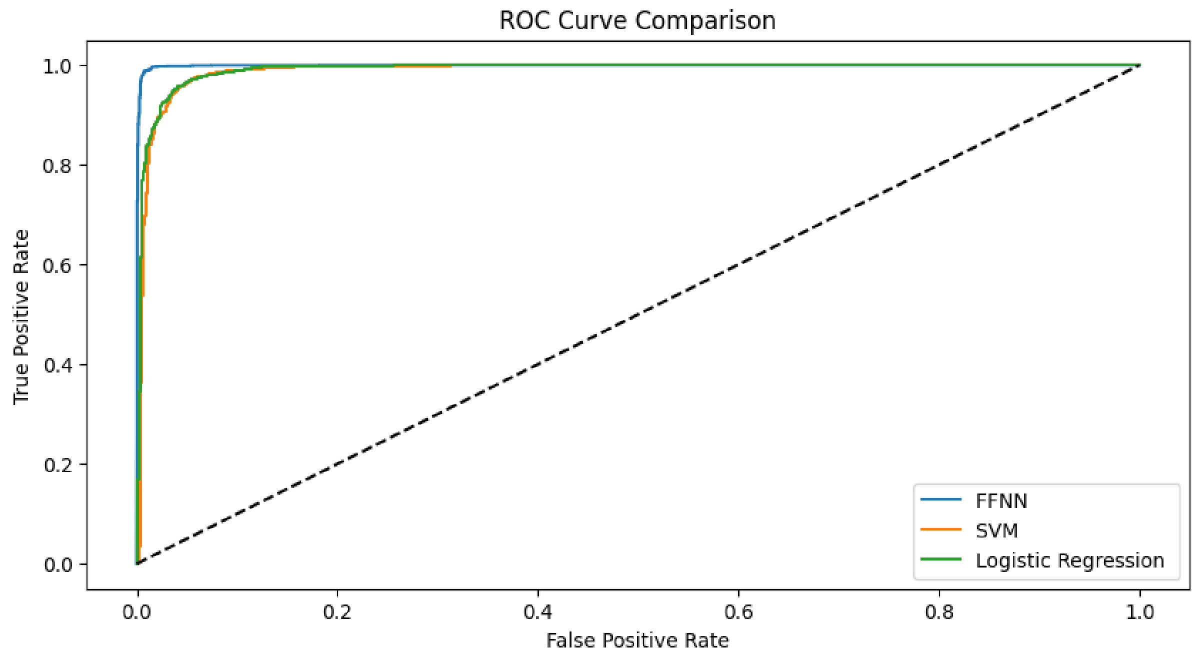
```
Epoch 5/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0270 -
accuracy: 0.9890 - val_loss: 0.0331 - val_accuracy: 0.9883
Epoch 6/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0228 -
accuracy: 0.9904 - val_loss: 0.0283 - val_accuracy: 0.9928
Epoch 7/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0217 -
accuracy: 0.9911 - val_loss: 0.0290 - val_accuracy: 0.9916
Epoch 8/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0201 -
accuracy: 0.9926 - val_loss: 0.0277 - val_accuracy: 0.9918
Epoch 9/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0195 -
accuracy: 0.9927 - val_loss: 0.0323 - val_accuracy: 0.9881
Epoch 10/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0184 -
accuracy: 0.9931 - val_loss: 0.0332 - val_accuracy: 0.9878
Epoch 11/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0183 -
accuracy: 0.9926 - val_loss: 0.0338 - val_accuracy: 0.9896
Epoch 12/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0173 -
accuracy: 0.9936 - val_loss: 0.0300 - val_accuracy: 0.9906
Epoch 13/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0139 -
accuracy: 0.9952 - val_loss: 0.0267 - val_accuracy: 0.9913
Epoch 14/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0161 -
accuracy: 0.9940 - val_loss: 0.0299 - val_accuracy: 0.9923
Epoch 15/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0145 -
accuracy: 0.9945 - val_loss: 0.0302 - val_accuracy: 0.9926
Epoch 16/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0150 -
accuracy: 0.9945 - val_loss: 0.0300 - val_accuracy: 0.9911
Epoch 17/25
252/252 [==============================] - 1s 5ms/step - loss: 0.0130 -
accuracy: 0.9953 - val_loss: 0.0292 - val_accuracy: 0.9923
Epoch 18/25
252/252 [==============================] - 1s 5ms/step - loss: 0.0133 -
accuracy: 0.9953 - val_loss: 0.0281 - val_accuracy: 0.9928
Epoch 19/25
252/252 [==============================] - 1s 5ms/step - loss: 0.0131 -
accuracy: 0.9946 - val_loss: 0.0440 - val_accuracy: 0.9854
Epoch 20/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0139 -
accuracy: 0.9943 - val_loss: 0.0309 - val_accuracy: 0.9931
Epoch 21/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0137 -
accuracy: 0.9952 - val_loss: 0.0332 - val_accuracy: 0.9931
Epoch 22/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0144 -
accuracy: 0.9942 - val_loss: 0.0285 - val_accuracy: 0.9933
Epoch 23/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0113 -
accuracy: 0.9956 - val_loss: 0.0365 - val_accuracy: 0.9911
Epoch 24/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0127 -
accuracy: 0.9953 - val_loss: 0.0349 - val_accuracy: 0.9923
Epoch 25/25
```

```
252/252 [==============================] - 1s 3ms/step - loss: 0.0117 -
accuracy: 0.9956 - val_loss: 0.0318 - val_accuracy: 0.9923
630/630 [==============================] - 1s 2ms/step - loss: 0.0133 -
accuracy: 0.9954
158/158 [==============================] - 0s 2ms/step - loss: 0.0280 -
accuracy: 0.9950
FFNN Training Accuracy: 0.9954
FFNN Testing Accuracy: 0.9950
Classification Report (FFNN Test):
158/158 [==============================] - 0s 1ms/step
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      2318
           1       0.99      1.00      1.00      2721

    accuracy                           1.00      5039
   macro avg       1.00      0.99      1.00      5039
weighted avg       1.00      1.00      1.00      5039

SVM Training Accuracy: 0.9577
SVM Testing Accuracy: 0.9589
Classification Report (SVM Test):
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      2318
           1       0.95      0.98      0.96      2721

    accuracy                           0.96      5039
   macro avg       0.96      0.96      0.96      5039
weighted avg       0.96      0.96      0.96      5039

Logistic Regression Training Accuracy: 0.9534
Logistic Regression Testing Accuracy: 0.9577
Classification Report (Logistic Regression Test):
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      2318
           1       0.95      0.97      0.96      2721

    accuracy                           0.96      5039
   macro avg       0.96      0.96      0.96      5039
weighted avg       0.96      0.96      0.96      5039


Overall Accuracy:
FFNN Accuracy: 0.9950
SVM Accuracy: 0.9589
Logistic Regression Accuracy: 0.9577
```



```python
from sklearn.metrics import roc_auc_score, roc_curve

# Calculate ROC-AUC(Receiver Operating Characteristic) for FFNN
ffnn_y_pred_proba_test = ffnn_model.predict(X_test)
ffnn_roc_auc = roc_auc_score(y_test, ffnn_y_pred_proba_test)

# Calculate ROC-AUC for SVM
svm_y_pred_proba_test = svm_model.decision_function(X_test)
svm_roc_auc = roc_auc_score(y_test, svm_y_pred_proba_test)

# Calculate ROC-AUC for Logistic Regression
logreg_y_pred_proba_test = logreg_model.predict_proba(X_test)[:, 1]
logreg_roc_auc = roc_auc_score(y_test, logreg_y_pred_proba_test)

print(f"FFNN ROC-AUC Score: {ffnn_roc_auc:.4f}")
print(f"SVM ROC-AUC Score: {svm_roc_auc:.4f}")
print(f"Logistic Regression ROC-AUC Score: {logreg_roc_auc:.4f}")

# Plot ROC Curves
fpr_ffnn, tpr_ffnn, _ = roc_curve(y_test, ffnn_y_pred_proba_test)
fpr_svm, tpr_svm, _ = roc_curve(y_test, svm_y_pred_proba_test)
fpr_logreg, tpr_logreg, _ = roc_curve(y_test, logreg_y_pred_proba_test)

plt.figure(figsize=(10, 5))
plt.plot(fpr_ffnn, tpr_ffnn, label=f'FFNN ')
plt.plot(fpr_svm, tpr_svm, label=f'SVM ')
plt.plot(fpr_logreg, tpr_logreg, label=f'Logistic Regression ')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend()
plt.show()
```

```
FFNN ROC-AUC Score: 0.9989
SVM ROC-AUC Score: 0.9884
Logistic Regression ROC-AUC Score: 0.9906
```



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
from keras.models import Sequential
from keras.layers import Dense

# Load the data
df = pd.read_csv('NSL-KDD.csv')
pd.set_option('display.max_columns', None)
print(df.shape)

# Preprocess the data
encoder = LabelEncoder()
scaler = StandardScaler()

# Identify categorical and numerical columns
cols_label_encoder = ['service', 'flag', 'class']
cols_onehot = ['protocol_type']
```

```python
# Encode categorical features
for col in cols_label_encoder:
    df[col] = encoder.fit_transform(df[col])

# Apply one-hot encoding to protocol_type
df = pd.get_dummies(df, columns=cols_onehot)

# Split the data
X = df.drop(labels='class', axis=1)
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.8, random_state=4)

# Standardize the data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the feed-forward neural network model
ffnn_model = Sequential()
ffnn_model.add(Dense(64, input_dim=X_train.shape[1],
activation='relu'))
ffnn_model.add(Dense(32, activation='relu'))
ffnn_model.add(Dense(32, activation='relu'))
ffnn_model.add(Dense(32, activation='relu'))
ffnn_model.add(Dense(1, activation='sigmoid'))

ffnn_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the feed-forward neural network model
history = ffnn_model.fit(X_train, y_train, epochs=25, batch_size=64,
validation_split=0.2)

# Evaluate the feed-forward neural network model
ffnn_train_loss, ffnn_train_accuracy = ffnn_model.evaluate(X_train,
y_train)
ffnn_test_loss, ffnn_test_accuracy = ffnn_model.evaluate(X_test,
y_test)

print(f'FFNN Training Accuracy: {ffnn_train_accuracy:.4f}')
print(f'FFNN Testing Accuracy: {ffnn_test_accuracy:.4f}')
print('Classification Report (FFNN Test):')
ffnn_y_pred_test = (ffnn_model.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, ffnn_y_pred_test))

# Confusion Matrix for FFNN
ffnn_cm = confusion_matrix(y_test, ffnn_y_pred_test)
```

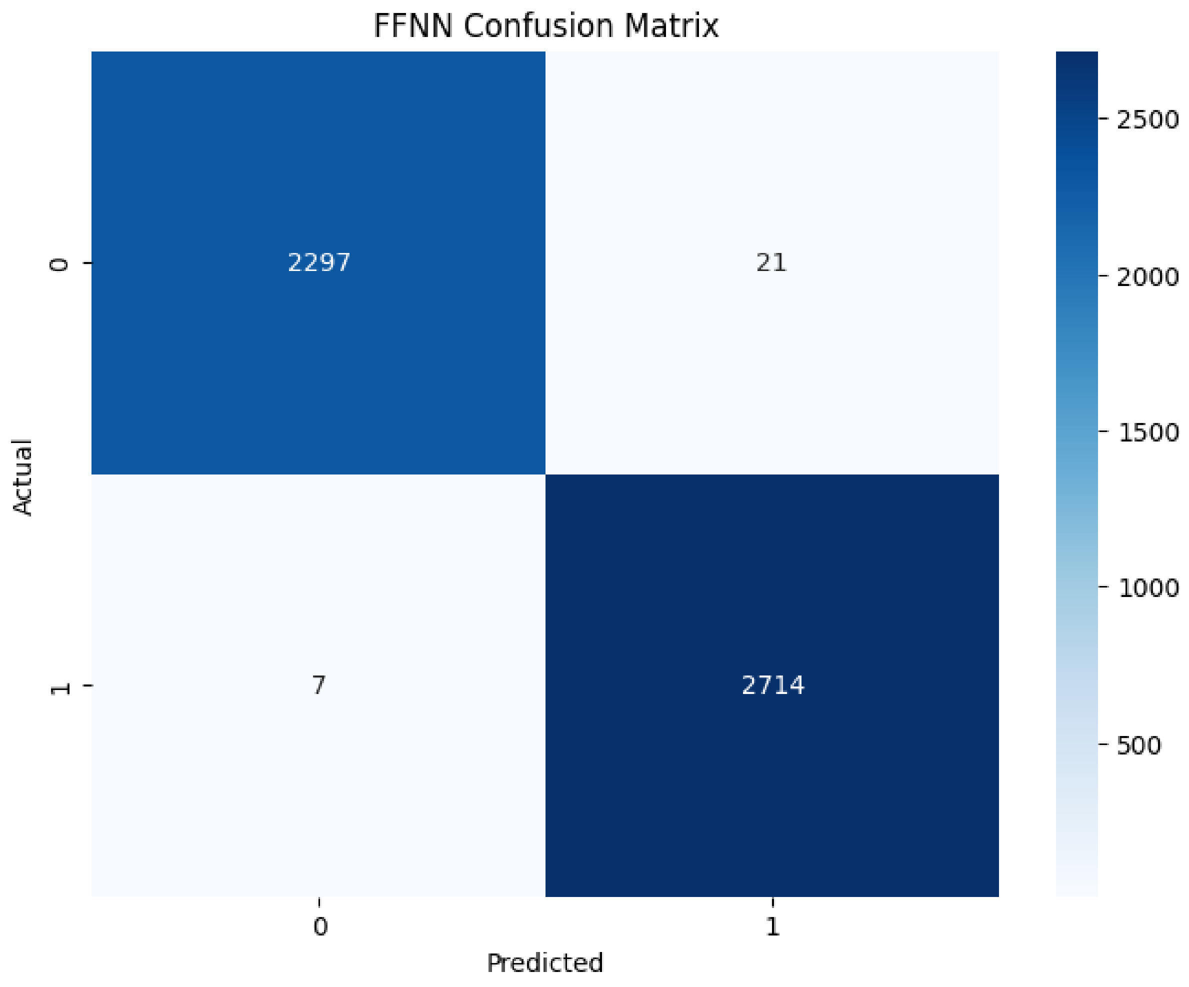
```
print("FFNN Confusion Matrix:")
print(ffnn_cm)
plt.figure(figsize=(8, 6))
sns.heatmap(ffnn_cm, annot=True, fmt='d', cmap='Blues')
plt.title('FFNN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Train the SVM model
svm_model = SVC(kernel='linear', C=1.0)
svm_model.fit(X_train, y_train)

# Predict and evaluate the SVM model
svm_y_pred_train = svm_model.predict(X_train)
svm_y_pred_test = svm_model.predict(X_test)

svm_train_accuracy = accuracy_score(y_train, svm_y_pred_train)
svm_test_accuracy = accuracy_score(y_test, svm_y_pred_test)

print(f'SVM Training Accuracy: {svm_train_accuracy:.4f}')
print(f'SVM Testing Accuracy: {svm_test_accuracy:.4f}')
print('Classification Report (SVM Test):')
print(classification_report(y_test, svm_y_pred_test))

# Confusion Matrix for SVM
svm_cm = confusion_matrix(y_test, svm_y_pred_test)
print("SVM Confusion Matrix:")
print(svm_cm)
plt.figure(figsize=(8, 6))
sns.heatmap(svm_cm, annot=True, fmt='d', cmap='Blues')
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Train the Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train, y_train)

# Predict and evaluate the Logistic Regression model
logreg_y_pred_train = logreg_model.predict(X_train)
logreg_y_pred_test = logreg_model.predict(X_test)

logreg_train_accuracy = accuracy_score(y_train, logreg_y_pred_train)
logreg_test_accuracy = accuracy_score(y_test, logreg_y_pred_test)
```

```
print(f'Logistic Regression Training Accuracy:
{logreg_train_accuracy:.4f}')
print(f'Logistic Regression Testing Accuracy:
{logreg_test_accuracy:.4f}')
print('Classification Report (Logistic Regression Test):')
print(classification_report(y_test, logreg_y_pred_test))

# Confusion Matrix for Logistic Regression
logreg_cm = confusion_matrix(y_test, logreg_y_pred_test)
print("Logistic Regression Confusion Matrix:")
print(logreg_cm)
plt.figure(figsize=(8, 6))
sns.heatmap(logreg_cm, annot=True, fmt='d', cmap='Blues')
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Print overall accuracy
print("\nOverall Accuracy:")
print(f"FFNN Accuracy: {ffnn_test_accuracy:.4f}")
print(f"SVM Accuracy: {svm_test_accuracy:.4f}")
print(f"Logistic Regression Accuracy: {logreg_test_accuracy:.4f}")

# Plot comparison of training and validation accuracy
plt.figure(figsize=(10, 5))
epochs = range(1, 26)

plt.plot(epochs, history.history['accuracy'], label='FFNN Training
Accuracy')
plt.plot(epochs, history.history['val_accuracy'], label='FFNN
Validation Accuracy')
plt.axhline(y=svm_train_accuracy, color='r', linestyle='--', label='SVM
Training Accuracy')
plt.axhline(y=svm_test_accuracy, color='g', linestyle='--', label='SVM
Testing Accuracy')
plt.axhline(y=logreg_train_accuracy, color='b', linestyle='--',
label='Logistic Regression Training Accuracy')
plt.axhline(y=logreg_test_accuracy, color='y', linestyle='--',
label='Logistic Regression Testing Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Comparison of FFNN, SVM, and Logistic Regression Models')
plt.legend()
plt.show()

# Plot comparison of training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(epochs, history.history['loss'], label='FFNN Training Loss')
plt.plot(epochs, history.history['val_loss'], label='FFNN Validation
Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('FFNN Training and Validation Loss')
plt.legend()
plt.show()

# Function to calculate precision, recall, and F1-score from confusion
matrix
def calculate_metrics(cm):
    tn, fp, fn, tp = cm.ravel()
    precision = tp / (tp + fp) if (tp + fp) != 0 else 0
    recall = tp / (tp + fn) if (tp + fn) != 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if (precision
+ recall) != 0 else 0
    return precision, recall, f1

# Calculate metrics for each model
ffnn_precision, ffnn_recall, ffnn_f1 = calculate_metrics(ffnn_cm)
svm_precision, svm_recall, svm_f1 = calculate_metrics(svm_cm)
logreg_precision, logreg_recall, logreg_f1 =
calculate_metrics(logreg_cm)

# Summary of Model Performance
summary_df = pd.DataFrame({
    'Model': ['FFNN', 'SVM', 'Logistic Regression'],
    'Training Accuracy': [ffnn_train_accuracy, svm_train_accuracy,
logreg_train_accuracy],
    'Testing Accuracy': [ffnn_test_accuracy, svm_test_accuracy,
logreg_test_accuracy],
    'Precision': [ffnn_precision, svm_precision, logreg_precision],
    'Recall': [ffnn_recall, svm_recall, logreg_recall],
    'F1-Score': [ffnn_f1, svm_f1, logreg_f1]
})
print("\nSummary of Model Performance:")
print(summary_df)
```

```
(25192, 42)
Epoch 1/25
252/252 [==============================] - 4s 7ms/step - loss: 0.1478 -
accuracy: 0.9592 - val_loss: 0.0779 - val_accuracy: 0.9730
Epoch 2/25
252/252 [==============================] - 2s 7ms/step - loss: 0.0501 -
accuracy: 0.9813 - val_loss: 0.0423 - val_accuracy: 0.9836
Epoch 3/25
252/252 [==============================] - 2s 7ms/step - loss: 0.0337 -
accuracy: 0.9872 - val_loss: 0.0349 - val_accuracy: 0.9886
Epoch 4/25
```

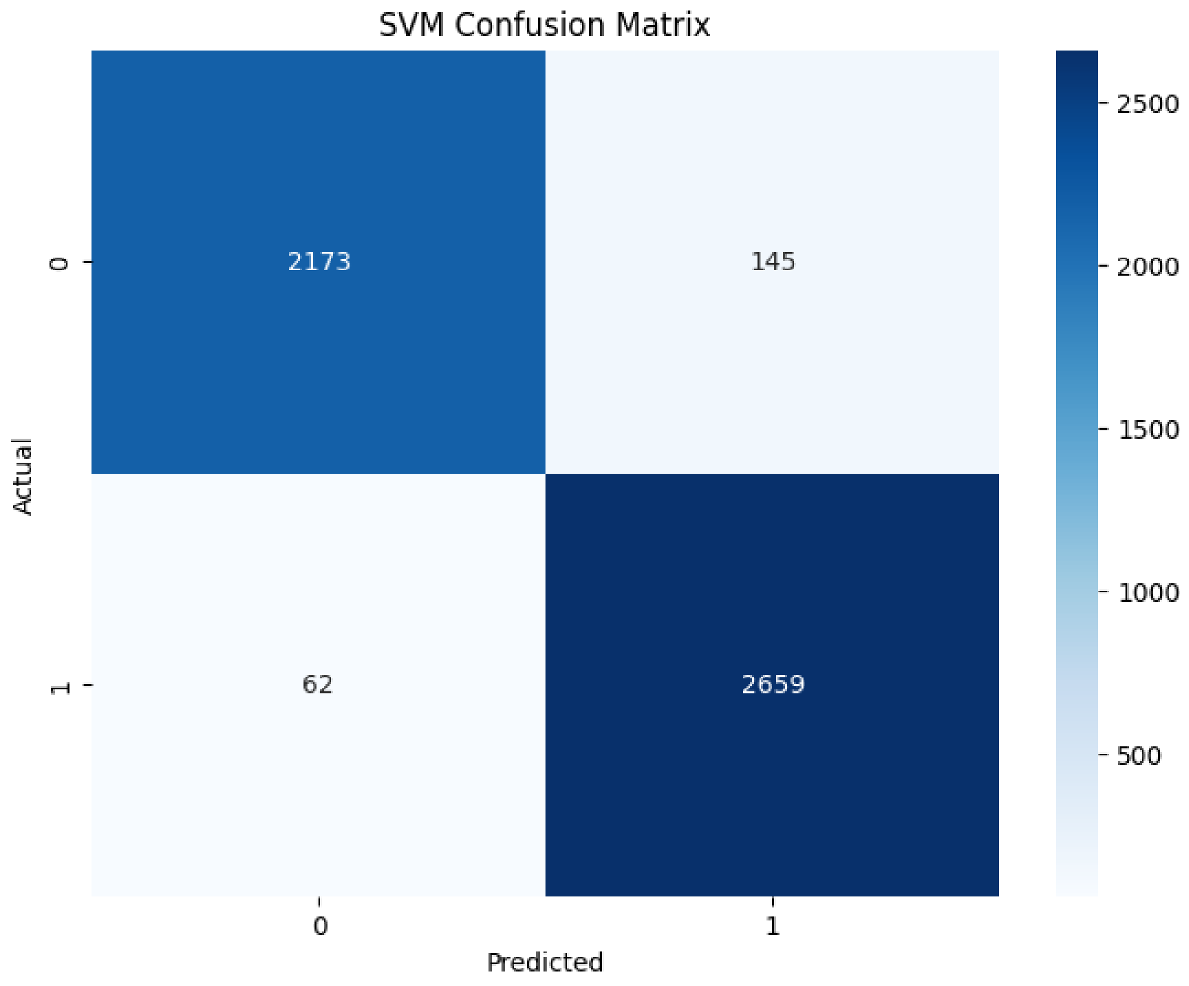
```
252/252 [==============================] - 2s 8ms/step - loss: 0.0283 -
accuracy: 0.9900 - val_loss: 0.0352 - val_accuracy: 0.9883
Epoch 5/25
252/252 [==============================] - 1s 5ms/step - loss: 0.0256 -
accuracy: 0.9906 - val_loss: 0.0303 - val_accuracy: 0.9908
Epoch 6/25
252/252 [==============================] - 1s 5ms/step - loss: 0.0246 -
accuracy: 0.9908 - val_loss: 0.0281 - val_accuracy: 0.9906
Epoch 7/25
252/252 [==============================] - 1s 4ms/step - loss: 0.0218 -
accuracy: 0.9919 - val_loss: 0.0294 - val_accuracy: 0.9926
Epoch 8/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0216 -
accuracy: 0.9918 - val_loss: 0.0297 - val_accuracy: 0.9906
Epoch 9/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0187 -
accuracy: 0.9930 - val_loss: 0.0318 - val_accuracy: 0.9901
Epoch 10/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0181 -
accuracy: 0.9929 - val_loss: 0.0319 - val_accuracy: 0.9908
Epoch 11/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0189 -
accuracy: 0.9927 - val_loss: 0.0292 - val_accuracy: 0.9916
Epoch 12/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0164 -
accuracy: 0.9937 - val_loss: 0.0298 - val_accuracy: 0.9908
Epoch 13/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0152 -
accuracy: 0.9943 - val_loss: 0.0316 - val_accuracy: 0.9903
Epoch 14/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0155 -
accuracy: 0.9940 - val_loss: 0.0565 - val_accuracy: 0.9886
Epoch 15/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0153 -
accuracy: 0.9944 - val_loss: 0.0361 - val_accuracy: 0.9916
Epoch 16/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0132 -
accuracy: 0.9949 - val_loss: 0.0618 - val_accuracy: 0.9938
Epoch 17/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0131 -
accuracy: 0.9955 - val_loss: 0.0633 - val_accuracy: 0.9933
Epoch 18/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0126 -
accuracy: 0.9952 - val_loss: 0.0824 - val_accuracy: 0.9933
Epoch 19/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0125 -
accuracy: 0.9950 - val_loss: 0.0738 - val_accuracy: 0.9923
Epoch 20/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0112 -
accuracy: 0.9962 - val_loss: 0.0852 - val_accuracy: 0.9913
Epoch 21/25
252/252 [==============================] - 1s 4ms/step - loss: 0.0127 -
accuracy: 0.9952 - val_loss: 0.0914 - val_accuracy: 0.9918
Epoch 22/25
252/252 [==============================] - 1s 5ms/step - loss: 0.0114 -
accuracy: 0.9955 - val_loss: 0.0947 - val_accuracy: 0.9923
Epoch 23/25
252/252 [==============================] - 1s 5ms/step - loss: 0.0110 -
accuracy: 0.9959 - val_loss: 0.1135 - val_accuracy: 0.9923
Epoch 24/25
```

```
252/252 [==============================] - 1s 4ms/step - loss: 0.0110 -
accuracy: 0.9963 - val_loss: 0.0999 - val_accuracy: 0.9928
Epoch 25/25
252/252 [==============================] - 1s 3ms/step - loss: 0.0097 -
accuracy: 0.9969 - val_loss: 0.1172 - val_accuracy: 0.9928
630/630 [==============================] - 1s 2ms/step - loss: 0.0301 -
accuracy: 0.9962
158/158 [==============================] - 0s 2ms/step - loss: 0.0415 -
accuracy: 0.9944
FFNN Training Accuracy: 0.9962
FFNN Testing Accuracy: 0.9944
Classification Report (FFNN Test):
158/158 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      2318
           1       0.99      1.00      0.99      2721

    accuracy                           0.99      5039
   macro avg       0.99      0.99      0.99      5039
weighted avg       0.99      0.99      0.99      5039

FFNN Confusion Matrix:
[[2297   21]
 [   7 2714]]
```

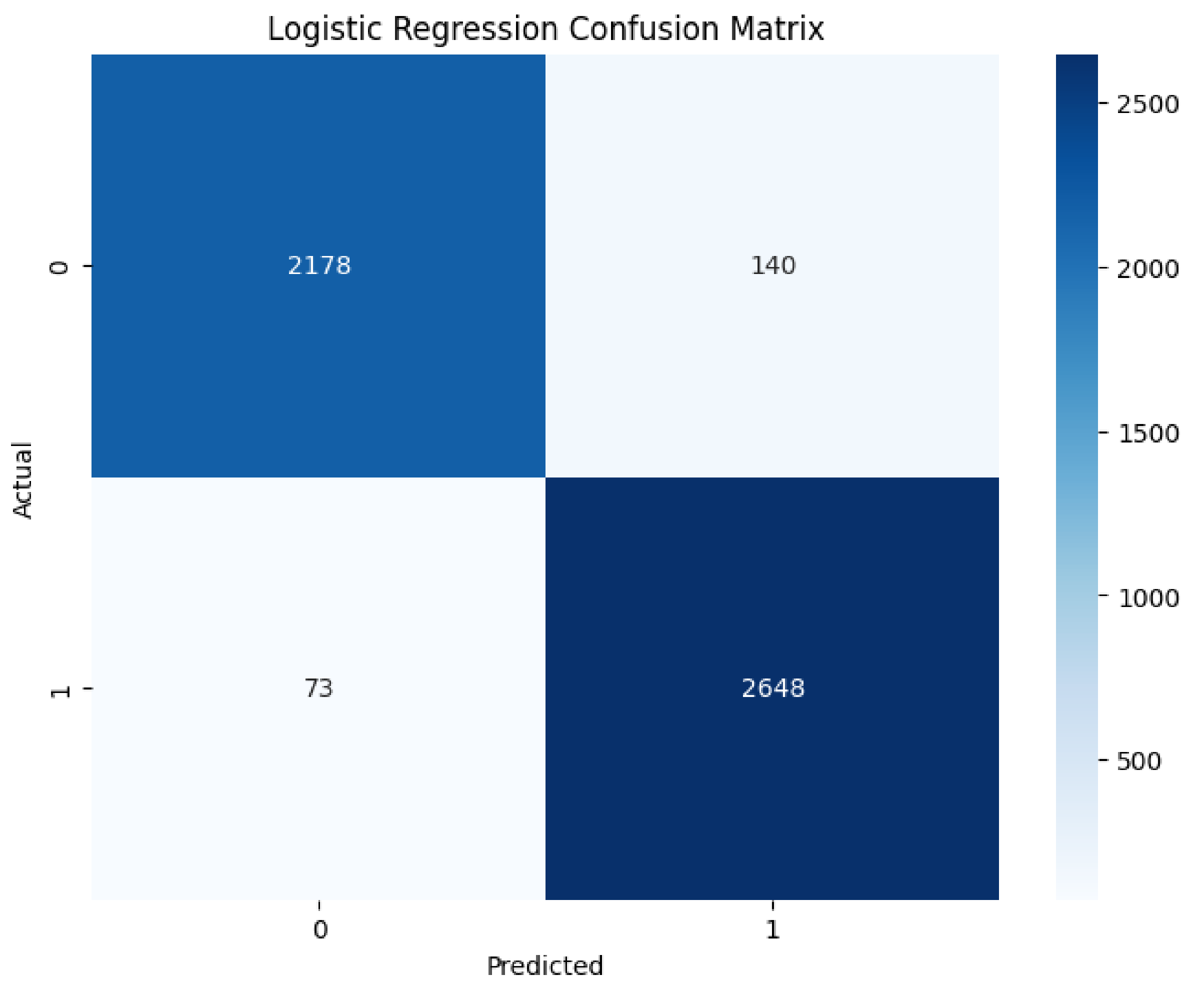


```
SVM Training Accuracy: 0.9577
SVM Testing Accuracy: 0.9589
Classification Report (SVM Test):
```

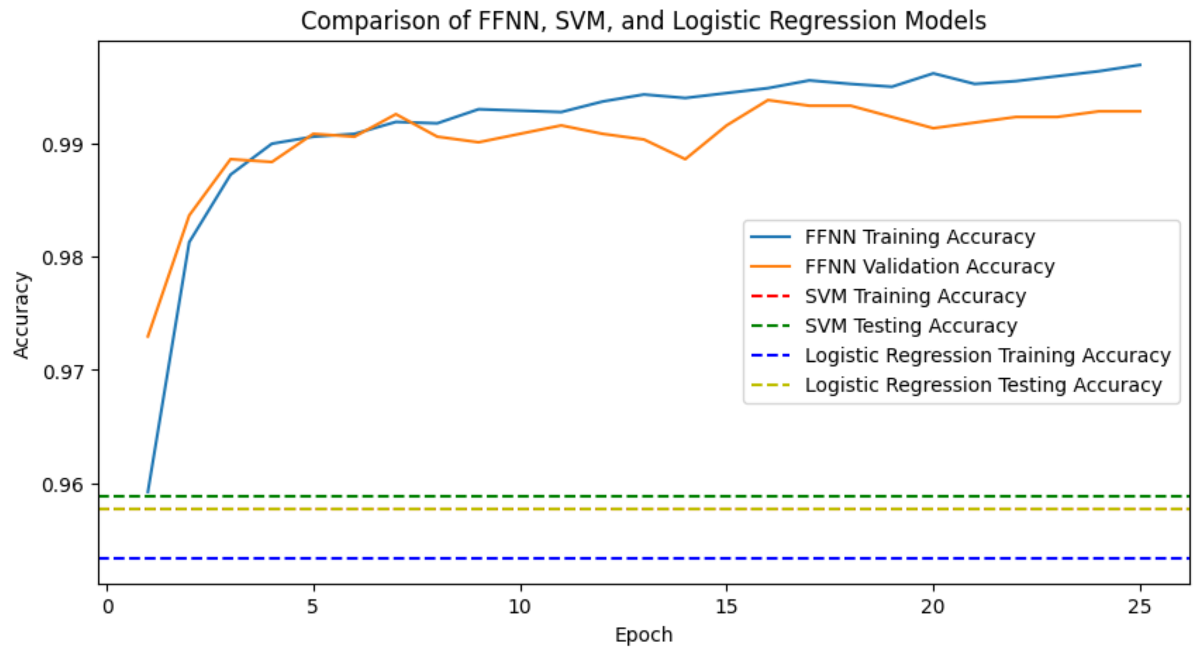
```
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      2318
           1       0.95      0.98      0.96      2721

    accuracy                           0.96      5039
   macro avg       0.96      0.96      0.96      5039
weighted avg       0.96      0.96      0.96      5039

SVM Confusion Matrix:
[[2173  145]
 [  62 2659]]
```

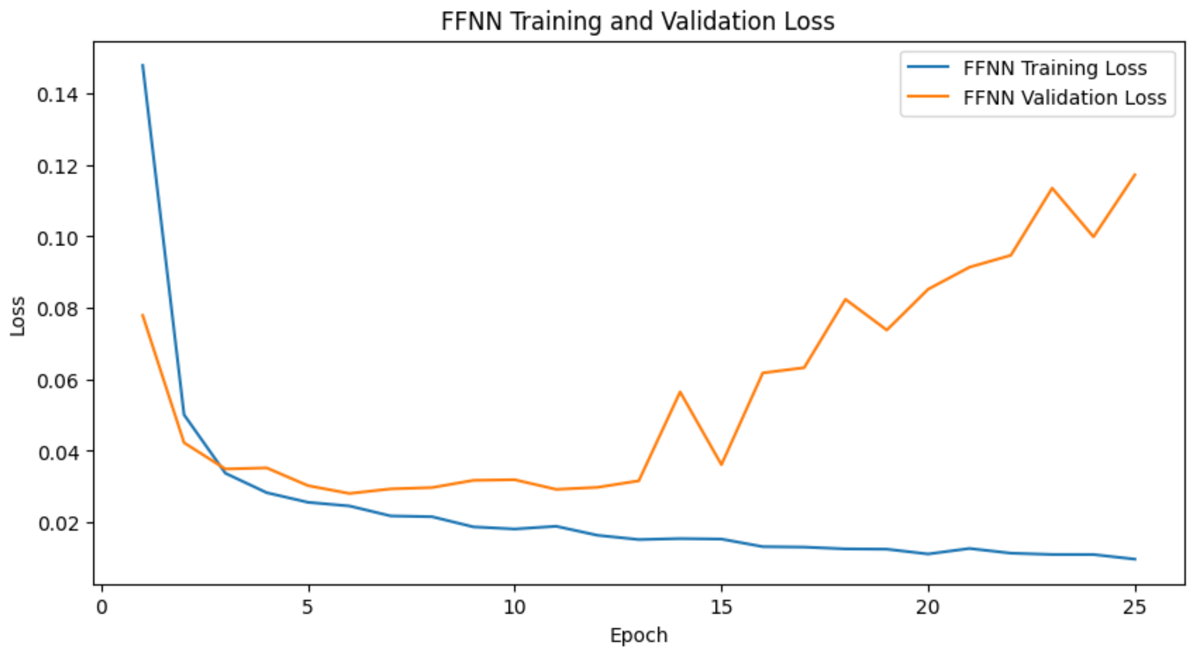


```
Logistic Regression Training Accuracy: 0.9534
Logistic Regression Testing Accuracy: 0.9577
Classification Report (Logistic Regression Test):
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      2318
           1       0.95      0.97      0.96      2721

    accuracy                           0.96      5039
   macro avg       0.96      0.96      0.96      5039
weighted avg       0.96      0.96      0.96      5039

Logistic Regression Confusion Matrix:
[[2178  140]
 [  73 2648]]
```



```
Overall Accuracy:
FFNN Accuracy: 0.9944
SVM Accuracy: 0.9589
Logistic Regression Accuracy: 0.9577
```





```
Summary of Model Performance:
                 Model  Training Accuracy  Testing Accuracy  Precision  \
0                 FFNN           0.996229          0.994443   0.992322
1                  SVM           0.957723          0.958920   0.948288
2  Logistic Regression           0.953357          0.957730   0.949785

     Recall  F1-Score
0  0.997427  0.994868
1  0.977214  0.962534
2  0.973172  0.961336
```