Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.051996

ARTICLE

Network Security Enhanced with Deep Neural Network-Based Intrusion Detection System

Fatma S. Alrayes[1], **Mohammed Zakariah**[2], **Syed Umar Amin**[3,*], **Zafar Iqbal Khan**[3] **and Jehad Saad Alqurni**[4]

[1]Information Systems Department, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Riyadh, 11671, Saudi Arabia

[2]College of Computer and Information Sciences, King Saud University, Riyadh, 11362, Saudi Arabia

[3]College of Computer and Information Sciences, Prince Sultan University, Riyadh, 11586, Saudi Arabia

[4]Department of Educational Technologies, College of Education, Imam Abdulrahman Bin Faisal University, P.O. Box 1982, Dammam, 31441, Saudi Arabia

*Corresponding Author: Syed Umar Amin. Email: samin@psu.edu.sa

Received: 20 March 2024 Accepted: 05 June 2024 Published: 18 July 2024

ABSTRACT

This study describes improving network security by implementing and assessing an intrusion detection system (IDS) based on deep neural networks (DNNs). The paper investigates contemporary technical ways for enhancing intrusion detection performance, given the vital relevance of safeguarding computer networks against harmful activity. The DNN-based IDS is trained and validated by the model using the NSL-KDD dataset, a popular benchmark for IDS research. The model performs well in both the training and validation stages, with 91.30% training accuracy and 94.38% validation accuracy. Thus, the model shows good learning and generalization capabilities with minor losses of 0.22 in training and 0.1553 in validation. Furthermore, for both macro and micro averages across class 0 (normal) and class 1 (anomalous) data, the study evaluates the model using a variety of assessment measures, such as accuracy scores, precision, recall, and F1 scores. The macro-average recall is 0.9422, the macro-average precision is 0.9482, and the accuracy scores are 0.942. Furthermore, macro-averaged F1 scores of 0.9245 for class 1 and 0.9434 for class 0 demonstrate the model's ability to precisely identify anomalies precisely. The research also highlights how real-time threat monitoring and enhanced resistance against new online attacks may be achieved by DNN-based intrusion detection systems, which can significantly improve network security. The study underscores the critical function of DNN-based IDS in contemporary cybersecurity procedures by setting the foundation for further developments in this field. Upcoming research aims to enhance intrusion detection systems by examining cooperative learning techniques and integrating up-to-date threat knowledge.

KEYWORDS

Machine-Learning; Deep-Learning; intrusion detection system; security; privacy; deep neural network; NSL-KDD Dataset

1 Introduction

In today's interconnected digital landscape, ensuring computer network security is paramount [1,2]. With the proliferation of sophisticated cyber threats, traditional intrusion detection methods have proven inadequate in safeguarding sensitive information and critical infrastructure [3,4]. In response to this evolving threat landscape, integrating advanced technologies such as Deep Neural Networks (DNNs) into intrusion detection systems has emerged as a promising approach to fortify network security [5,6].

This study explores the efficacy of employing Deep Neural Network-based Intrusion Detection Systems (IDS) to enhance network security. Leveraging the NSL-KDD dataset [7–9], a comprehensive evaluation of network assaults categorized into distinct classes, including denial-of-service (DoS) [10], user-to-root (U2R) [11], remote-to-local (R2L) [12], and probing, forms the cornerstone of this investigation. Furthermore, the escalating sophistication of cyber threats necessitates a paradigm shift in intrusion detection methodologies [13,14]. While effective against known attack patterns, conventional signature-based intrusion detection systems falter in identifying novel and stealthy attacks. The inadequacy of signature-based approaches underscores the need for adaptive, intelligent systems capable of discerning anomalous patterns indicative of malicious activity [15–18]. Deep Neural Networks, owing to their inherent ability to learn intricate patterns and relationships within data, present a compelling solution to this pressing challenge.

The utilization of the NSL-KDD dataset, a variant of the widely utilized KDD Cup 99 dataset, offers a rich and diverse repository of network traffic data, encompassing various attack scenarios and benign activities [19,20]. By meticulously categorizing network intrusions into distinct classes, the dataset facilitates a granular analysis of the performance of intrusion detection systems across different attack vectors [21–23]. This segmentation enables a nuanced evaluation of the effectiveness of DNN-based IDS in detecting and mitigating specific types of cyber threats. Moreover, categorizing network assaults into DoS, U2R, R2L, and probing provides invaluable insights into the diverse tactics employed by adversaries to compromise network integrity [24,25]. Denial-of-service attacks, characterized by a relentless barrage of malicious traffic aimed at overwhelming network resources, pose a significant threat to service availability and operational continuity [26,27]. User-to-root attacks seek to exploit vulnerabilities to escalate user privileges and gain unauthorized access to critical system resources [28–30]. Moreover, in a real-time intrusion detection system based on DNN, remote-to-local attacks refer to unwanted external access, whereas probing activities involve surveillance to uncover potential vulnerabilities for exploitation [31–33].

Furthermore, given this, a careful examination of DNN-based IDS's capacity to identify odd patterns that indicate these various attack methods is warranted [34–36]. By harnessing the latent capabilities of deep learning algorithms, IDS can adaptively learn and discern subtle deviations from normal network behavior, thereby enhancing the robustness and resilience of intrusion detection mechanisms [37–40].

A thorough framework for network security using deep neural networks (DNNs) is shown in Fig. 1. The procedure includes feature selection, model selection, training data preparation, and data preparation, starting with the NSL-KDD dataset. After that, the DNN methodology is applied to the data for analysis. A confusion matrix evaluates performance, producing important metrics like accuracy, precision, sensitivity, specificity, and F1 score. In the end, the framework makes it possible to identify intrusions, greatly enhancing network security protocols.

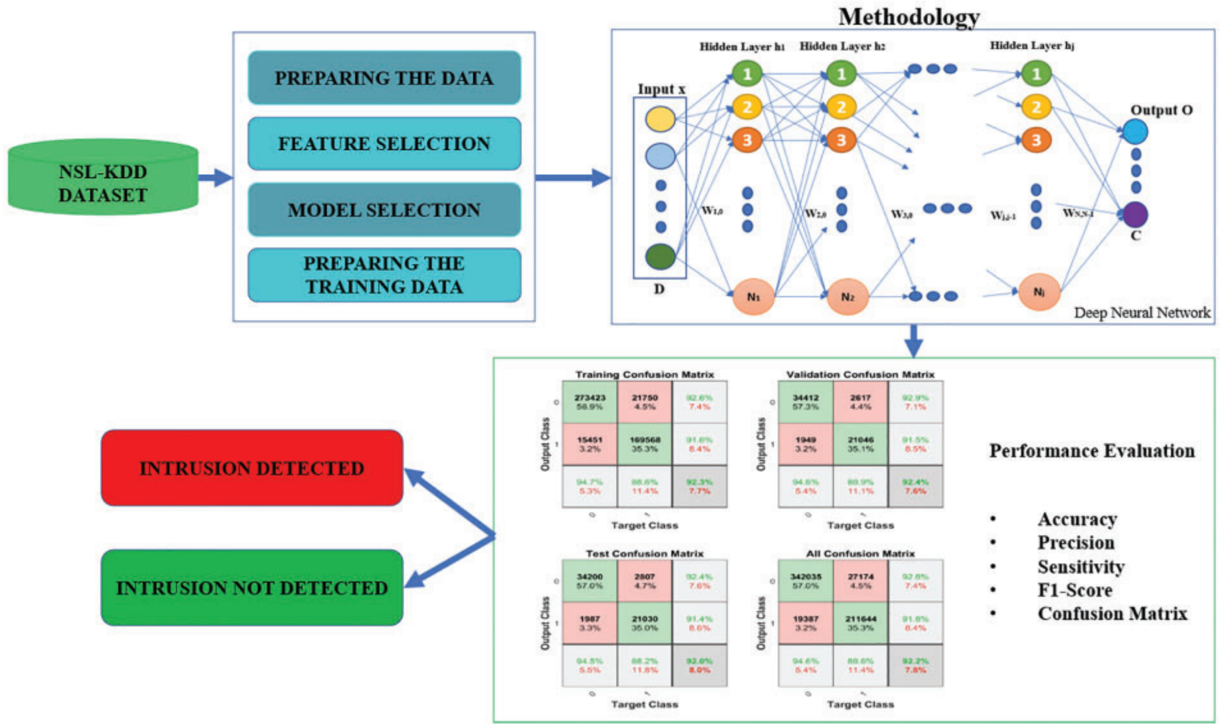

Figure 1: Framework for network security with deep neural network

The following is a summary of the current paper's significant contributions:

- This paper presents a novel deep neural network (DNN) ensemble learning system for intrusion detection. The system utilizes varied viewpoints from multiple models to improve the accuracy and resilience of the detection process.
- The intrusion detection model exhibits improved generalization and robustness by utilizing ensemble approaches, outperforming both traditional methodologies and contemporary algorithms.
- The proposed system is validated using the authentic NSL-KDD dataset, which allows for a realistic assessment of its performance in real-world circumstances.
- Demonstrates the recommended DNN ensemble's higher performance compared to conventional approaches, highlighting its potential for practical use in intrusion detection systems.
- The suggested approach can be applied in Intrusion Detection Systems (IDS), providing a strong solution for improving network security.

Our research paper is based on the following structure. The current section, Section 1, constitutes an introduction. In Section 2, some relevant earlier work is described. Data processing and visualizing data are covered in Section 3, whereas the suggested model is discussed in Section 4. Furthermore, Section 5 explains the results of the project. Section 6 evaluates the accomplishment of the project, and the research is concluded under Section 7.

2 Literature Review

The current body of research on DNN-based intrusion detection techniques faces some difficulties that demand careful consideration and investigation. The intricacy of network environments and the ever-changing nature of cyber threats present a significant obstacle. It can be challenging for DNN-based IDSs to adjust to changing attack patterns and changes in network traffic, which could result in false positives or negatives. Furthermore, there are several obstacles to overcome in feature extraction, data preprocessing, and model training due to the enormous amount and diversity of network data. It is imperative to guarantee the scalability and efficacy of DNN-based intrusion detection systems (IDSs) in large-scale networks, particularly concerning the computational resources needed for training and inference. Furthermore, there are significant obstacles to the interpretability and explainability of DNN-based intrusion detection systems, especially in critical infrastructures where decision-makers need to know the logic behind threats that have been detected. Building trust and easing the adoption of DNN models in real-world security operations requires understanding how these models make judgments and the ability to spot potential biases or vulnerabilities in their learning processes. Moreover, additional research is necessary to improve the resilience of DNN-based intrusion detection systems in hostile environments due to their resilience against adversarial attacks and evasion strategies. Notwithstanding these difficulties, the reasons for conducting research in this area are still evident: in the constantly changing world of cyber threats, DNN-based IDSs can provide better detection accuracy, greater threat intelligence, and proactive defense mechanisms.

The NSL-KDD dataset, widely considered a benchmark dataset in intrusion detection, has been utilized in some research projects. Layered denoising auto-encoders are used to get an accuracy range between 88% and 92% [2]. The combination of DNN and decision trees (DT) contributes to the achievement of an accuracy rate of 86.8% in [4]. In addition, reference [6] uses the CICIDS 2019 dataset and employs ensemble management strategies to reach an accuracy of 92%. A new strategy is presented in [9], which operates a stacked attention autoencoder (SAAE) in conjunction with a DNN, resulting in an accuracy of 77.57%.

To achieve higher levels of accuracy, researchers have studied a variety of different ways. For the research referred to as [11], multiclass support vector machines (SVM) are integrated with chi-square feature selection and gated recurrent unit (GRU) long- short-term memory (LSTM) networks. An accuracy of 88.81% is achieved by combining these two factors. Similarly, deep learning (DL) techniques are utilized in [15], more especially spatio-temporal learning (STL), which leads to binary classification accuracies of 88.39% and multiclass classification accuracies of 79.10%. Additionally, Reference [17] proposes an IDS based on a recurrent neural network (RNN) and achieves an accuracy of 82.38% or higher. In addition, reference [18] uses the NSL-KDD and LITNET-2020 datasets in conjunction with an enhanced LSTM and PSO technique, which successfully achieves an accuracy of 93.09%.

In addition, research has been conducted to assess the effectiveness of traditional machine learning algorithms compared to deep learning techniques. To identify intrusions and classify different types of attacks, an artificial neural network (ANN) was utilized in the research that was mentioned as a [24]. The accuracy that was achieved for the classification of attack types was 79.9%, while the accuracy that was gained for the detection of intrusions was 81.2%. In contrast, reference [25] proves the excellent performance of deep learning-based network intrusion detection (DLNID) systems. These systems achieved a fantastic F1 score of 90.73% and an accuracy of 89.65%, demonstrating their remarkable effectiveness.

In addition, researchers have investigated self-taught learning models for intrusion detection. An analysis of the idea of self-taught learning (STL) is presented in [26], where it is stated that the average f-score for this type of learning is 75.76%. Furthermore, research has taken advantage of convolutional neural networks (CNN) to improve accuracy rates. Through the utilization of a CNN architecture, the research that is referenced in [36] accomplishes an impressively high level of accuracy, which is 91.27%. In addition, reference [38] discusses the utilization of ensemble learning in conjunction with long short-term memory (LSTM) networks, which ultimately led to an astounding accuracy of 93.4% [39].

Additionally, advancements in network security have been made in particular sectors, such as software-defined networking (SDN), which has also seen growth. This intrusion detection system (IDS) is explicitly intended for SDN systems and uses deep learning techniques. A precision rate of approximately 76.86% is achieved by [40], demonstrating high efficiency. Moreover, a unique Dynamic Multi-scale Topological Representation (DMTR) technique designed to improve network intrusion detection is presented in this paper by [41]. DMTR efficiently captures multi-scale network topologies, guaranteeing resilience to data distribution changes and class imbalances. While its Group Shuffle Operation (GSO) allows dynamic topological representation without reprocessing all data upon new arrivals, it uses a variety of topology lenses to expose dimensional structures. Four datasets of experiments confirm that DMTR is effective at handling dynamic network traffic and class imbalances. This novel method addresses essential issues with static models in dynamic network environments, offering promising improvements in intrusion detection systems. Furthermore, by classifying research in graph-based intrusion detection into three categories—graph construction, network design, and GNN deployment—the review by [42] advances the field. It describes a generalized design approach that tackles obstacles at every turn and is followed by a systematic review of previous efforts. Prospective avenues for research and outstanding problems are thoroughly investigated. Using a focused survey and problem-oriented taxonomy, this study provides scholars with a clear, systematic framework to better understand and analyze the topic.

To provide a summary, the research highlights the significant advancements that have been made in network security through the utilization of intrusion detection systems that are based on deep neural networks. Investigations have been conducted into various approaches, including traditional machine learning algorithms and more complex deep learning architectures. In terms of accuracy and efficiency, each of these methods has its own set of advantages that are distinct from the others. Additionally, using benchmark datasets such as NSL-KDD has made it possible to assess and compare various approaches thoroughly. Despite the progress that has been accomplished, ongoing research efforts continue to focus on tackling emerging challenges and improving existing methods to increase network security systems' resilience against expanding threats.

Table 1 shows the list of references which includes dataset/parameters, methodology and results.

Table 1: List of references for past work with dataset/parameters, methodology, and results

Ref.	Dataset	Methodology	Accuracy
[2]	NSL-KDD	Stacked Denoising Auto-Encoders	Accuracy is between 88% to 92%.
[4]	NSL-KDD	DNN, DT	Having an accuracy of 86.8%.
[6]	CICIDS2019	Ensemble 1 technique	Accuracy of 92%.

(Continued)

Table 1 (continued)

Ref.	Dataset	Methodology	Accuracy
[9]	NSL-KDD	SAAE (Stacked Attention Autoencoder (SAAE))-DNN	Accuracy of 77.57%.
[11]	NSL-KDD	Multiclass SVM with Chi-square Feature Selection, GRU-LSTM	Accuracy is 88.81%.
[15]	NSL-KDD	Deep Learning (DL) with STL	Accuracy: 88.39% (2 classes), 79.10% (multiple classes).
[17]	NSL-KDD	RNN-based IDS	Accuracy: 82.38%.
[18]	NSL-KDD and LITNET-2020	Improved LSTM and PSO Method	ILSTM accuracy 93.09%.
[24]	NSL-KDD	Artificial Neural Network (ANN)	Attack type classification: 79.9%, Intrusion detection: 81.2%.
[25]	NSL-KDD	Deep Learning-based Network Intrusion Detection (DLNID)	F1 score: 90.73%, Accuracy: 89.65%.
[26]	NSL-KDD	Self-taught Learning (STL)	Average f-score: 75.76%
[38]	NSL-KDD	CNN	CNN achieved the highest accuracy of 91.27%.
[39]	NSL-KDD	Ensemble Learning and LSTM	LSTM achieved an accuracy of 93.4%.
[40]	NSL-KDD	Deep Learning-based IDS for SDN	Achieved efficiency with a precision of approximately 76.86%.

3 Dataset

The NSL-KDD dataset will be covered in this section as to how it was utilized in our suggested methodology. It was upgraded and redeveloped into the NSL-KDD dataset after being produced from the KDDCup99 dataset [9,15,17,18,24], which makes it considerably older than it was first provided and has various issues. A recursive dataset, the NSL-KDD dataset has fixed many of its predecessor's problems. The NSL-KDD dataset's main selling point is that it has an essentially constant size and a variety of useful features that support the creation of the best and most accurate classifiers. The NSL-KDD dataset is gathered via Kaggle. The data set required to build the ID model is contained in each of the two text files (named train and test) in the data folder. The features and classes in the train and test files are identical, but their sizes vary. A larger dataset is produced by combining these two training and testing files.

3.1 Data Description

These four sub-datasets—KDDTest+, KDDTest-21, KDDTrain+, and KDDTrain+ 20 Percent—combine to form the entire data collection. KDDTest-21 and KDDTrain+ 20 Percent are subsets of KDDTest+ and KDDTrain+, respectively. We now refer to KDDTrain+ and KDDTest+ as train and test, respectively. The KDDDTest-21 is a portion of the test that excludes the most challenging traffic records, and the KDDTrain+ 20 Percent is a portion of the train whose record count is 20% of the whole train dataset. Although they are present in both datasets, the traffic records in KDDDTest-21 and KDDTrain+ 20 percent already exist in the test and train datasets, respectively. The two fundamental subcategories for classes are security and normal. The subclass's security type is displayed in Table 2.

Table 2: Security type subclasses

Probe security	R2L security	U2R security	DoS security
saint	*guess_passwd*	*rootkit*	*Neptune*
satan	*warezmaster*	*buffer_overflow*	*mailbomb*
Upsweep	*warezclient*	*loadmodule*	*teardrop*
scan	*multihop*	*Perl*	*smurf*
nmap	*phf*	*xsnoop*	*httptunnel*
portsweep	*spy*		*apache2*
	ftp_write		*processtable*
			land
			back
			snmpguess
			xlock
			security
			udpstorm
			named
			sendmail
			xterm
			imap

The NSL-KDD dataset categorizes security types into four groups:

- Probe security: These attempts are intended to obtain data on a target system, such as its open ports and active services. The NSL-KDD dataset contains instances of satan, upsweep, and portsweep as probe security.
- R2L security: These assaults are carried out by a security who is not physically present on the target system but has gained access via a remote connection. The NSL-KDD dataset includes ftp_write, guess_passwd, and imap as examples of R2L security.
- U2R security: These assaults are carried out by a security who has gained access to a user account and utilizes that access to increase their privileges until they are the root user. In the NSL-KDD dataset, U2R security may be seen in a buffer overflow, load module, and Perl.
- DoS security: These attempts aim to deny access to a service or resource to authorized users. Numerous types of DoS assaults, including smurf, Neptune, and teardrop security, are present in the NSL-KDD dataset.

Below, Table 3 shows the security type 4 subclasses in the NSL-KDD dataset.

Table 3: Security type 4 subclasses example in NSL-KDD dataset

Probe security	R2L security	U2R security	DoS security
satan	ftp_write	Buffer overflow	smurf
upsweep	guess_password	Load module	Neptune
portsweep	imap	Perl	teardrop security

There are two critical classes for security types: normal and security. According to Table 2, there are nearly 38 subtypes of security. The following describes each class type:

- ftp_write: This class represents a security that attempts to write files to an FTP server.
- guess_passwd: This class represents a security that attempts to guess the user account password on a system.
- IMAP: This class represents a security that targets the IMAP, a protocol for accessing and managing email messages on a remote server.
- nmap: This class represents a security that uses the Nmap tool to scan a target system for open ports and running services.
- perl: This class represents a security that uses the Perl programming language to execute arbitrary code on the target system.
- satan: This class represents a security that uses the SATAN tool to scan a target system for vulnerabilities.
- smurf: This class depicts an instance of Smurf security, a sort of DoS security that disrupts the flow of a target system with many ICMP echo request (ping) packets.
- spy: This class exemplifies spy security, which entails trying to get into a system or network without authorization to obtain sensitive data.
- teardrop: This class represents Teardrop security, a type of DoS security that exploits a bug in the way some systems handle large IP packets.
- upsweep: This class represents IP Sweep security, which involves using a tool to scan a range of IP addresses in search of live hosts.
- land: This class illustrates LAND security, a kind of DoS security in which malicious security delivers several packets to a target host with the same source and destination IP addresses.
- load-module: This class represents a security that attempts to load a malicious module into the kernel of a target system.
- normal: This class represents normal, legitimate network traffic.
- Back: This class represents Backdoor security, which involves the use of a secret method to gain unauthorized access to a computer or network.
- buffer_overflow: This class represents Buffer Overflow security, which occurs when a program or process attempts to store more data in a buffer (a temporary storage area) than it can hold. This can cause the buffer to overflow and corrupt adjacent memory, potentially allowing security to execute arbitrary code.
- Multihop: This class depicts multi-hop security, which happens when security employs several compromised systems to reach a target system.
- Neptune: This class represents a Neune security, a type of DoS security that floods a target system with large ICMP packets.

- Warez-client: This class represents a security that involves using a "Warez" software client.
- phf: This class represents PHF security, which involves using a "PHP File Inclusion" vulnerability to execute arbitrary code on the target system.
- Pod: This class represents a Ping of Death security, which involves sending a malformed or oversized ping packet to a target system to crash or freeze it.
- portsweep: This class represents a Port Sweep security, which involves scanning a range of ports on a target system in search of open or vulnerable ports.
- Rootkit: This class represents Rootkit security, and it entails using malicious software intended to enter a system at the administrator level and conceal its presence from the system administrator.

There are many sub-classes for attack class. The effectiveness and performance of the deep learning model are complicated by having many classes during training. To improve performance and classification, we convert all the classes belonging to the attack category as labels, i.e., intrusion attacks. So, we have two classes: one class is normal, and the other is intrusion attack, which consists of all the features of different intrusion attack types. This way, the model will better classify the features based on two classes compared to classes around 30–40.

3.2 Data Visualization

Out of the 43 columns, 41 are features, and 2 are labels. There are 43 features in total; three are categories, and the rest are numerical. Fig. 2 illustrates the three groups of protocol features: UDP, TCP, and ICMP.

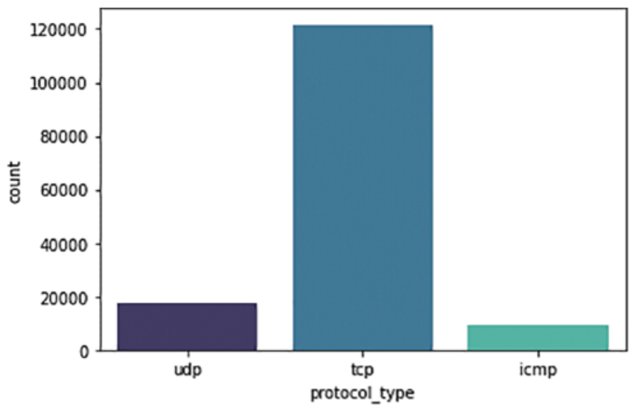

Figure 2: Protocol type feature classes

The NSL-KDD dataset's protocol type feature describes the transport protocol employed for network communication. The three possible values of the category variable used to describe the protocol type are:

- ICMP: The Internet Control Message Protocol, or ICMP, is represented by this value. This protocol transmits error messages and operational data regarding network conditions.
- UDP: The User Datagram Protocol (a transport protocol that offers connectionless, datagram-oriented service) is represented by this value.
- TCP: This value stands for the Transmission Control Protocol, which offers a dependable, stream-oriented connection for two networked devices.

3.3 Data Cleaning and Preparation

Data pre-processing plays a pivotal role in enhancing the outcomes of deep learning-based intrusion detection systems. Several critical techniques are employed to cleanse the NSL-KDD dataset before feeding it into the model. Firstly, duplicate removal eliminates redundant instances, reducing bias and ensuring each data point contributes uniquely to the learning process. The model becomes more robust and less susceptible to skewed results by mitigating the overemphasis on specific patterns or behaviors. Secondly, managing missing values is imperative for preserving data integrity. Whether through imputation or removal of rows with missing values, this step ensures a complete dataset, enhancing the accuracy and reliability of intrusion detection.

Furthermore, outlier detection is essential in identifying abnormal occurrences that deviate significantly from the norm. Proper management of outliers enhances the model's focus on significant patterns while minimizing their distorting influence. Normalization ensures uniform scale across features, facilitating fair comparison and aiding model convergence during training. Feature selection reduces dataset dimensionality by retaining the most informative features, thereby decreasing model complexity, mitigating overfitting, and enhancing computational efficiency. Additionally, encoding categorical variables enables the incorporation of categorical data into the learning process, capturing underlying patterns or connections. Lastly, removing unnecessary columns streamlines the dataset, reducing noise and enhancing the model's performance by focusing solely on the most relevant features. By meticulously implementing these data-cleansing strategies, deep learning-based intrusion detection systems achieve optimal performance, improved dataset quality, and more accurate recognition and mitigation results.

3.4 Model Development

Preparing a dataset for features and class sets comes before the model creation as the initial phase. The merged dataset has classes, such as assault or normal, in 42 columns after the first 41 columns of features. Features are retrieved and stored in X and classes in Y following the removal of any extraneous feature columns, data processing, and data dumping. The training and test split command from the sklearn library encode and converts the X and Y datastore.

Table 4 lists the following data sizes for testing and training.

Table 4: Features and labelling for training size and testing

Dataset	Size
Features train–Xtrain	(99505,41)
Features test–Xtest	(49010,41)
Labels train–Ytrain	(99505,1)
Labels test –Ytest	(49010,1)

The stages involved in the methodology for ID utilizing the NSL-KDD dataset and DL are as follows:

i. Gathering and Preparing Data

- Compile a varied dataset that includes network activity that is malicious as well as legitimate.

- To guarantee consistency, preprocess the data by eliminating noise, dealing with missing values, and normalizing characteristics.

ii. Feature Selection

- Determine pertinent characteristics that will help you distinguish between malicious and legitimate activity.
- Employ statistical methods and domain expertise to design or choose features that effectively capture the critical elements of network traffic.

iii. Model Selection

- Select appropriate neural network topologies, such as hybrid models or recurrent neural networks (RNNs), that are appropriate for intrusion detection.
- When choosing the architecture, consider performance requirements, computational resources, and model complexity.

iv. Preparing Training Data

- To train and assess the model, partition the dataset into test, validation, and training sets.
- Use strategies such as data augmentation to broaden the range of training cases and enhance the generalization capacity of the model.

v. Training Models

- Utilize the training dataset to instruct the selected neural network architecture.
- Optimize hyperparameters to improve model performance, including learning rate, batch size, and regularization strategies.
- Measurements such as accuracy and loss function value are used to track training progress to guarantee convergence and avoid overfitting.

vi. Model Evaluation

- Utilizing the validation dataset, assess the performance of the trained model.
- Examine necessary measures like recall, accuracy, precision, and F1 score to see how well the model detects intrusions while reducing false positives.

vii. Adjusting and Optimizing

- Based on the validation findings, fine-tune the model by experimenting with other network designs or changing the hyperparameters.
- Optimize the deployment model, considering the target environment's resource limitations, memory footprint, and inference performance.

It is imperative to iterate on every step of the model-building process to continuously improve its performance and adaptability to changing network threats. Furthermore, it is essential to implement thorough testing and validation protocols to guarantee the dependability and efficiency of the intrusion detection system in practical situations. The above details are shown in Fig. 3.

3.5 Feature Selection

There are 41 different feature kinds, and they are divided into security- and normal-type characteristics. Each feature is divided into three attribute value categories (numeric, binary, and nominal). The characteristics of a traffic record may be categorized into four groups: intrinsic, host-based, content-based, and time-based. These features give information on the interaction with the traffic input by the IDS. The many feature categories are described below:

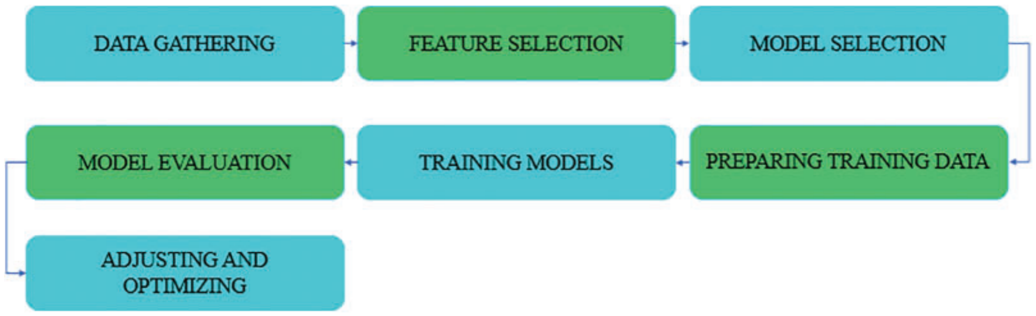

Figure 3: Steps of the DNN classification model for NSL-KDD dataset ID

i. Intrinsic Features

Source and destination addresses, protocol in use, type of service, and the connection's conclusion are only a few examples of the intrinsic qualities that these aspects describe the connection itself. The NSL-KDD dataset contains intrinsic features like *duration*, *service*, *flag*, *protocol_type*, *dst_bytes*, *land*, *wrong_fragment*, *src_bytes*, and *urgent*.

Impact of intrinsic features on results:

Incorporating intrinsic characteristics significantly influences the outcomes of deep learning-based intrusion detection. These features offer valuable details on the properties of network connections, like their length, service, flag, kind of protocol, byte counts, and others. By taking into account these characteristics, the model can capture many facets of both legitimate and criminal behavior, enabling more precise and reliable intrusion detection.

The model can recognize patterns linked to particular types of network traffic and assaults because of their inherent properties. The service feature, for instance, enables the model to distinguish between acceptable actions linked to various network services and potentially harmful ones.

While the flag feature enables the identification of odd flag combinations, which are frequently suggestive of assaults, the duration feature aids in capturing temporal patterns, by utilizing the distinctive qualities and behaviors of network connections, the intrusion detection model is improved in its ability to detect and mitigate intrusions by including these fundamental traits.

ii. Host-Based Features

These features describe the characteristics of the host or system that the connection originated from or targeted. Examples of host-based features in the NSL-KDD dataset include *logged_in*, *root_shell*, *su_attempted*, *num_file_creations*, and *num_shells*. The feature types in this dataset can be broken down into four types:

- **Categorical Features:** These features represent categorical data, a data type that can be divided into categories or groups. Categorical features in a dataset of NSL-KDD include *protocol_type*, *service*, *flag*, and *logged_in*.
- **Binary Features:** These features represent binary data, which is a type of data that can only take one of two possible values. Binary features in a dataset of NSL-KDD include *land*, *root_shell*, and *su_attempted*.

- **Discrete Features:** These characteristics show discrete data, a category of numerical data that can only have one or a few distinct values. The *src_bytes*, *dst_bytes*, *hot*, *num_file*, and *num_failed_logins* are examples of discrete features.
- **Continuous Features:** These features represent continuous data, numerical data that can take any value within a specific range. Continuous features in the NSL-KDD dataset include *duration*, *wrong_fragment*, and *urgent*, as depicted in Table 5.

Table 5: NSL-KDD datasheet

Index	Duration	Protocol type	Service type	Flag	Src-bytes	Dst bytes
0	0	UDP	Other	SF	146	0
1	0	TCP	Private	SO	0	0
2	0	TCP	HTTP	SF	232	8153
3	0	TCP	HTTP	SF	199	420
4	0	TCP	Private	REJ	0	0

iii. Influence of the above Features on Results

The results and accuracy of intrusion detection are greatly influenced by the host-based characteristics in the NSL-KDD dataset, such as logged_in, root_shell, su_attempted, num_file_creations, and num_shells. These traits and behaviors of the hosts or systems participating in network connections are revealed by these qualities, assisting in detecting possible intrusions.

The model can tell the difference between normal user activity and suspicious behavior by considering host-based information. For instance, the logged_in functionality lets you know if a user is currently signed in to the system. An unexpected or unauthorized login may indicate a possible infiltration or compromised account. This feature's incorporation enables the model to recognize and flag such behaviors accurately.

Similar features like root_shell and su_attempted show attempts to enter restricted areas or get elevated privileges. These characteristics can aid in the model's ability to recognize harmful activity and tell it apart from typical user behavior. These host-based attributes enable the intrusion detection model to accurately capture the distinctive traits and behaviors of the participating hosts, improving the accuracy of intrusion detection and intrusion mitigation.

Overall, host-based characteristics are essential for improving the outcomes and precision of intrusion detection. The model can distinguish between legitimate and malicious activity by considering the hosts' or systems' traits and behaviors, which enhances its capacity to successfully identify and stop intrusions.

iv. Class Datastore

All assault kinds are essentially divided into four categories, as follows:

a) DoS

These security measures restrict authorized users from using a particular service or resource. Numerous types of DoS assaults, including *smurf*, *Neptune*, and *teardrop* security, are present in the NSL-KDD dataset.

b) Probing

These assaults are intended to acquire details about a target system, including open ports and active services. In the NSL-KDD dataset, examples of probing security include *satan*, *upsweep*, and *poPortsweep*.

c) U2R

These securities are launched by a security who has managed to access the user account and then uses this access to escalate their privileges to become a root user. Examples of U2R security in the NSL-KDD dataset include *buffer_overflow*, *load module*, and *Perl*.

d) R2L

These securities are launched by a security who is not physically present at the target system but has managed to gain access through a remote connection. Examples of R2L security in the NSL-KDD dataset include *ftp_write*, *guess_passwd*, and *imap*.

Table 6 below shows the security-type classes where each dataset has more than 50% regular traffic records, and the distribution of U2R and R2L is incredibly low. Although this number is low, it accurately depicts how assaults on current internet traffic are distributed, with DoS security being the most frequent and U2R and R2L security being extremely rare.

The original dataset contains 42 different security classes, assembled into four major classes: DoS, Probe, R2L, and U2R.

Table 6: Security-type classes

0		Normal
1	Intrusion	Security
2		Normal
3		Normal
4	Intrusion	Security
22538	...	Normal
22539		Normal
22540	Intrusion	Security
22541		Normal
22542	Intrusion	Security

When the NSL-KDD classes are converted into two classes, the normal and intrusion security, the 42 different security classes are consolidated into just two classes: normal and intrusion. The normal class represents normal, non-malicious network traffic, while the intrusion class represents network traffic that is associated with security.

The process of converting the 42 classes into two classes involves merging some of the original classes. For example, all the different types of DoS security would be grouped into the intrusion class. At the same time, all the different types of Probe security would also be grouped into the intrusion class. The same would happen for U2R and R2L. Indeed, all the security is grouped into one class—the intrusion class.

This conversion process can simplify the classification task, as it reduces the number of classes between which the model must be able to distinguish. However, it also reduces the granularity of the security information available, making it more difficult to perform a detailed analysis of the types of security present in the data. It can also make the classification task more challenging, as the model needs to be able to accurately distinguish between normal traffic and a diverse range of intrusion traffic types.

4 Proposed Model

4.1 Model Design

With careful consideration of various architectural components, the suggested Deep Neural Network (DNN) model for improving network security through an Intrusion Detection System (IDS) is made in a way that can successfully detect and categorize probable intrusions [4,6,11,15,17,24]. The architecture of the model includes multiple hidden layers to make feature extraction and categorization easier. Now let's examine the suggested DNN model's design features:

i) Input Layer: Data samples that represent network traffic features are sent to the DNN's input layer. Packet headers, protocol types, source and destination addresses, and other characteristics important to network communication might be included in these elements.
ii) Batch Normalization Layer: The input data is normalized by using a batch normalization layer that comes after the input layer. Batch normalization guarantees that each layer's input has a mean of zero and a standard deviation of one, which helps to stabilize and speed up the training process.
iii) Dropout Layer: To lessen overfitting, a dropout layer is added following batch normalization. During training, dropout randomly deactivates a portion of neurons, pushing the network to acquire more resilient and universal characteristics.
iv) Dense Layer (Feature Extraction): The dense layer is the main component used for feature extraction and forms the basis of the model architecture. This dense layer converts the input data into a higher-level representation that captures pertinent patterns and traits suggestive of network incursions by experimenting with different activation functions and weights.
v) Batch Normalization Layer: To guarantee steady and effective training, a second batch normalization layer is introduced after the dense layer, just as the first batch normalization layer.
vi) Dropout Layer: To further regularize the model and avoid overfitting, a second dropout layer is introduced after the second batch normalization layer.
vii) Dense Layer (Output): The DNN's output layer is the last dense layer. This layer usually generates probability distributions over potential incursion classes using an appropriate activation function, like softmax for multi-class classification.

There are two batch normalization layers and two dropout layers among the five hidden levels in the design that was previously explained. Every layer is essential to the DNN model's overall operation because it helps with feature extraction, regularization, and classification.

The goal of the suggested DNN model is to efficiently identify and categorize network intrusions by utilizing the intrinsic properties of deep learning. The model learns complicated patterns and anomalies suggestive of harmful activity by processing network traffic data through numerous layers of interconnected neurons. This approach improves network security and reduces the likelihood of cyber threats.

As shown in Fig. 4, an input vector of dimension D is fed into the DNN, and it is modified by the hidden layers (consisting of hidden units) by a function g and the DNN's parameters (weights and bias). The output layer O offers the DNN output for the intended purpose of classification.

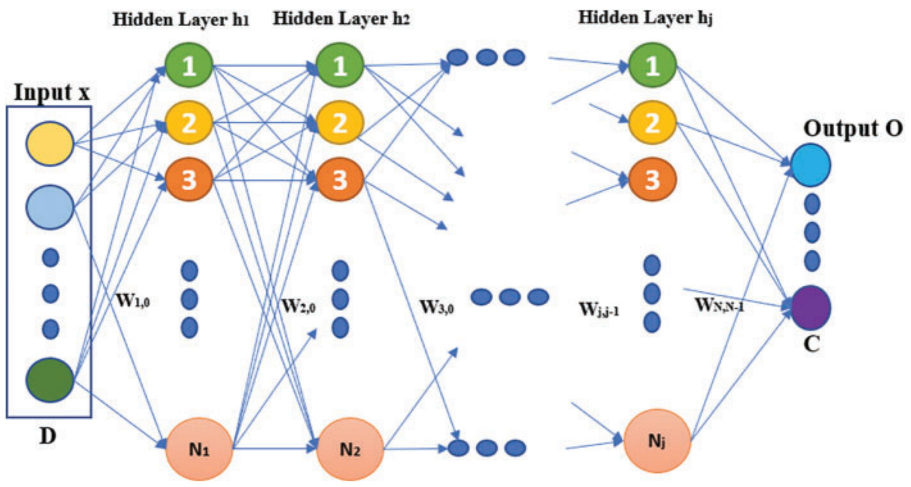

Figure 4: DNN general architecture

4.2 Model Architecture

In our experiment, we used a fully connected NN with three hidden layers, an output layer, and an input layer. The input layer is the layer used for batch normalization. The hidden layer is dense, along with the dropout and batch normalization layers. There is a thick output layer with a sigmoid activation function.

- Output Layer: DNN's output layer generates the final categorization predictions. In the classification task, there are exactly as many classes as there are neurons in the output layer. In this case, it would be the NSL-KDD dataset's security classes.
- Forward Propagation: The DNN may be used for classification by feeding input data into the layers, computing the output at each layer, and then utilizing the output of the final layer to provide the final classification predictions after training.
- Hidden Layers: The DNN has three hidden layers that are in charge of discovering the underlying correlations and patterns in the data. These layers are batch normalization, dense, and dropout layers. The hidden layers section has 1 batch normalization, 2 dropouts, and 1 dense layer. The dropout layer is being used twice with the same dropout ratio. To control the model's capacity, the number of neurons in every concealed layer, which composes each hidden layer, may be altered. For the model to understand complicated relationships, the hidden layers must introduce non-linearity via activation functions.

- Input Layer: The characteristics of the NSL-KDD dataset would be the input data for the input layer of the DNN, which is responsible for accepting input data. The majority of the time, the number of input characteristics and input layer neurons are identical.
- Training: To train the DNN, the weights of the connections between the neurons in the various levels are altered. The training procedure is finished by minimizing the output gap between expected and actual outcomes. An optimization algorithm like SGD or Adam is used for this.

The model architecture regulates the model's adaptability and depth. By utilizing hidden layers and non-linear activation functions, the model may record linkages in the data that are more complicated (like ReLU).

The model is typically more expensive to use and more challenging to train when additional parameters are included. A distinct classifier exists for each output and hidden layer node. The input layer receives inputs and sends the scores of those inputs to the subsequent hidden layer for additional activation. The repetition of this procedure results in the desired result. DNN architecture for intrusion detection is shown in Fig. 5. The dropout layer, batch normalization layer, dense layer, and dropout layer are used to form a hidden layer by keeping the batch normalization layer a part of the input layer. It then forms a thick layer towards the end of the output layer.

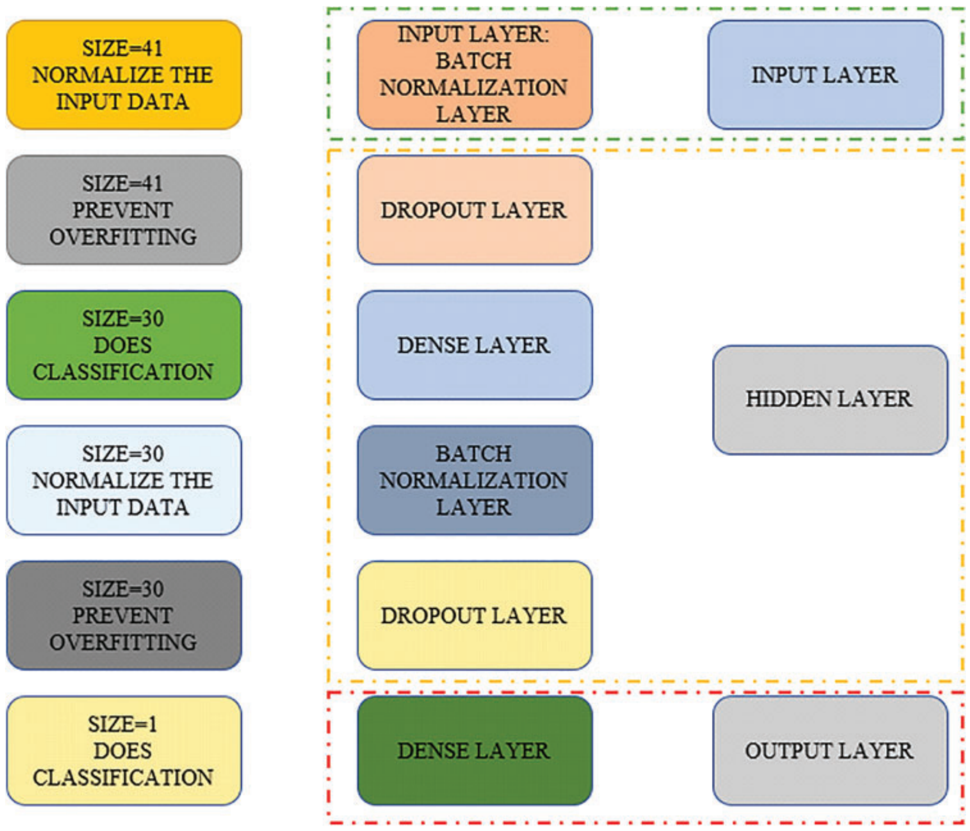
SIZE=41 NORMALIZE THE INPUT DATA
SIZE=41 PREVENT OVERFITTING
SIZE=30 DOES CLASSIFICATION
SIZE=30 NORMALIZE THE INPUT DATA
SIZE=30 PREVENT OVERFITTING
SIZE=1 DOES CLASSIFICATION
INPUT LAYER: BATCH NORMALIZATION LAYER
DROPOUT LAYER
DENSE LAYER
BATCH NORMALIZATION LAYER
DROPOUT LAYER
DENSE LAYER
INPUT LAYER
HIDDEN LAYER
OUTPUT LAYER

Figure 5: DNN architecture for intrusion detection

Table 7 provides an intricate analysis of the characteristics and variables of several layers in a deep neural network (DNN) model. Every layer has a unique function in the processing and conversion of incoming data to generate significant output. A batch normalization layer is applied after the input layer to normalize the activations of the preceding layer, hence enhancing the effectiveness and stability of the training process. This layer has 120 parameters and an input shape of (none, 41). In the next hidden layer, dropout regularization is used. This technique helps prevent overfitting by randomly excluding a portion of input units during training. Although the dropout layer plays a critical role, it does not introduce any more parameters.

Table 7: DNN layers properties

Layer	Properties	Parameters
Input layer-batch normalization layer	Input shape = (none, 41)	164
Hidden layer: dropout	Input shape = (none, 41)	0
Dense	Input shape = (none, 30)	1260
Batch normalization layer	Input shape = (none, 30)	120
Drop out layer	Input shape = (none, 30)	0
Dense layer	Input shape = (none, 1)	31

The dense layers form the central part of the network, where each neuron is connected to every neuron in the previous layer. The initial densely connected layer, with an input shape of (none, 30), consists of 1260 parameters that correspond to the weights of the connections between neurons. Subsequently, another layer of batch normalization is used to ensure the preservation of data consistency and normalizing throughout the entire network. The functionality of this layer is determined by its 120 parameters. The last dense layer, with an input form of (none, 1), contains 31 parameters that are essential for generating the network's output. Table 7 offers crucial information about the structure and configuration of the DNN model, which helps in comprehending and enhancing its performance.

4.2.1 Input Batch Normalization Layer

A technique known as batch normalization is frequently employed in DNN to standardize input data before it is sent using network layers. It is frequently utilized to enhance the performance of the model and maintain stability when training. During batch normalization, the input data are initially separated into tiny groups or "batches" of data, and the mean, and standard deviation of each batch are calculated. Additionally, input data are then normalized to have a mean of 0 and an SD of 1, respectively. This is achieved by excluding the mean and dividing the standard deviation. The input data are separated into batches of size 41 in the case of a batch normalization layer with a size of 41 used for classification, and the mean and SD of each batch are determined. The input data are then normalized again by excluding the mean and dividing it by the batch's standard deviation. By minimizing the internal covariate shift—a change in the distribution of input data to a layer brought on by a change in parameters during training—this normalization method contributes to the model's stability. Minimizing the internal covariate shift also speeds up the training process.

4.2.2 Hidden Drop-Out Layer

By randomly "dropping out" or setting to zero a specific proportion of the neurons during training, the dropout approach is frequently employed in deep neural networks to minimize overfitting. A hidden dropout layer used for classification with a size of 41 would consist of 41 neurons. During training, a certain proportion of these neurons, say 20%, would be randomly eliminated, rendering their outputs zero.

The dropout rate hyperparameter, which can be changed during model tuning, typically determines the proportion of neurons that fell out. A higher dropout rate will result in more neurons being lost, which will increase the model's resistance to overfitting but also lower its capacity.

4.2.3 Hidden Dense Layer

DNNs frequently employ a layer type called a dense layer, which is also referred to as a fully connected layer, for classification tasks. All of the neurons in a layer with a density of 30 neurons are linked to the neurons both in the layer below and the layer above it. The dense layer would include 30 neurons if it were a hidden layer with a size of 30 that was used for classification. Each neuron in the dense layer receives information from all the neurons in the layer before it, transforms that input linearly, and then activates nonlinearly. The following layer then receives each neuron's output. The capacity of the model can be managed by varying the size of the dense layer, which is in charge of learning intricate input data representation. The density of the neuron depends upon a hyperparameter which is altered throughout the model-tuning procedure.

4.2.4 Hidden Batch Normalization Layer

The input data are divided into 30 batches in the case of a hidden batch normalization layer with a size of 30 used for classification, and the mean and standard deviation of each batch are determined. Input data are then normalized by removing the mean and dividing it by the batch's standard deviation. By minimizing internal covariate shift—a change in the input data distribution to a layer brought on by a change in parameters during training—this normalization method contributes to the model's stability.

4.2.5 Output Dense Layer

A DNN's final layer, the output dense layer, is utilized to generate the final predictions or classifications. Each neuron in a layer that has a density of 1 neuron has a connection with every other neuron in the layer above it, transforming the input linearly. When used for classification, an output dense layer with a size of 1 would have a single neuron. This neuron's output, which can either be a probability score or a binary number depending on the problem type, represents the model's final prediction. The sigmoid function is an activation function for binary classification or the softmax function for multi-class classification and is frequently placed after the output dense layer. The activation function maps the dense layer's output to a set of values relevant to the current issue

Hyperparameters are crucial for increasing the accuracy of the training model. More data points will be trained for one epoch, the lower the batch size, and vice versa. As indicated in Table 8, a suitable batch size is produced by splitting input data points by steps per epoch. The training accuracy will increase as the epochs increase, and vice versa. The most used optimizer for both classification and regression is Adam. The loss function varies depending on the machine learning model type. In our situation, the binary cross entropy loss function is utilized because the classification is binary and there are two classes of labels. Hyperparameters are variables that are chosen in advance of the training process rather than ones that are learned from the data during training. They are in charge of how the model and learning algorithm behave, and they can significantly affect how long a machine learning model takes to train. Following are a few instances of how hyperparameters can interfere with the testing phase.

Table 8: Hyperparameters for DNN intrusion detection model

Hyperparameters	Properties
Epochs	40
Batch size	32
Callbacks	Early stop at min validation loss
Optimizer	Adam
Loss	Binary cross entropy

4.2.6 Hidden Layers and Neurons

A NN's hidden layers and neurons can be expanded, which can significantly increase the number of parameters that must be learned, and, in turn, the amount of computing needed during training.

The neural network model's hidden layers are shown in Table 9. With defined input and output dimensions, it consists of the Dropout_28, Dense_26, Batch_normalization_13, and Dropout_29 layers. In addition to adding to the model's architecture, these layers are essential to its functionality during inference and training.

Table 9: Hiden layers

Dropout_28 (Dropout)	(None, 41)
Dense_26 (Dense)	(None, 30)
Batch_normalization_13 (Bat Ch Normalization)	(None, 30)
Dropout_29 (Dropout)	(None, 30)

In a deep learning model, the hidden layer comes in between the input layer and the output layer. The input layer of our model is a batch normalization layer after the conventional sequential layer. The output layer is a dense layer that identifies the number of classes to be classified. The above figure indicates the sizes of hidden layers.

4.2.7 Learning Rate

The model may converge quickly as a result of a high learning rate, but it may also overshoot the ideal outcome and converge to a less-than-ideal outcome. The model could converge slowly and take more time to train if the learning rate is low. Increasing the batch size when utilizing the SGD technique might speed up training while simultaneously using more memory. Finding a fair balance between training time and model performance while tuning these hyperparameters can be a time-consuming effort. To determine the best hyperparameters, methods like grid search or random search are frequently used as solutions. Additionally, the procedure can be speed up by tweaking the hyperparameters using a smaller fraction of the data. The model is trained for epoch 40, but due to a callback for early stopping at the lowest validation loss, the model is terminated early at epoch 37. Table 10 shows the training outcomes for the final three epochs.

Table 10: Final epochs loss and accuracy performance

Epoch 38/40	
3110/3110 [—]-9 s 3 ms/step-loss	: 0.2218 accuracy: 0.9132 val_loss: 0.1559 val_accuracy: 0.9462
Epoch 39/40	
3110/3110 [= =]-9 s 3 ms/step-loss	: 0.2191 accuracy: 0.9159 val_loss: 0.1783 val accuracy: 0.9447
Epoch 40/40	
3110/3110 [=]-8 s 3 ms/step-loss	: 0.2216 accuracy: 0.9133 val_loss: 0.1820 val_accuracy: 0.9447

5 Results

The metrics of F1 score, precision, recall, and accuracy are used to present the algorithm results for each algorithm in our methodology. In comparison to the other classifiers employed in the metric, the DNN classifier fared better. By metrics, the CNN classifier trailed the DNN classifier but produced better results overall. The DNN model developed 41 characteristics for the normal and intrusion detected classes of intrusion detection. The model is trained using a sigmoid activation function over 100 epochs with a batch size 32. The model had an overfitting issue at the beginning of training, however, there was either an overfitting or an underfitting by the end of training. Although a callback is given to terminate training at the lowest validation loss value, the model was originally set for training epochs 100. Below Fig. 6 is performance graphs for training and validation accuracy, with training accuracy being 0.91 and validation accuracy being 0.94.

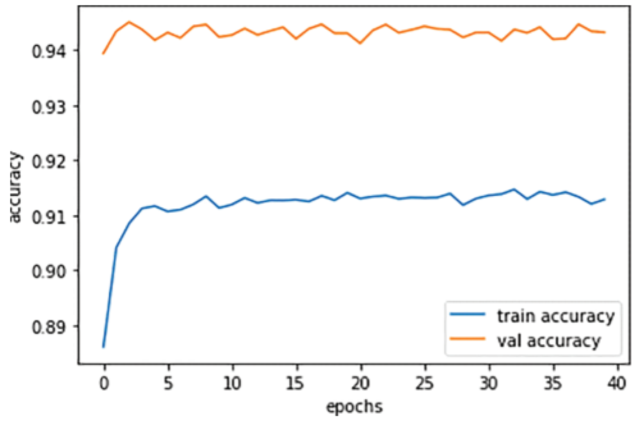

Figure 6: Training and validation accuracy performance plot

The training and validation precision of the performance plot is shown in Fig. 7 below, with training loss recorded at 0.22 performance value and validation loss at 0.1553 performance value.

Metrics like validation loss and training loss are frequently used to assess how well a deep learning model performs during training. Validation loss relates to the error or cost of the model on a different validation dataset, whereas training loss refers to the error or cost on the training dataset. Another often-used metric to assess a classification model's effectiveness is accuracy, which measures the proportion of accurate predictions the model makes. The training loss would probably reduce throughout the 40 training epochs for a classification model with 40 iterations and an accuracy of more than 90% since the model could understand the underlying correlations and patterns in the data and enhance its predictions. Over time, the validation loss would likewise shrink, though possibly not as quickly as the training loss. Indicating that the model is correctly predicting the bulk of data, the model's accuracy should be high, above 90%.

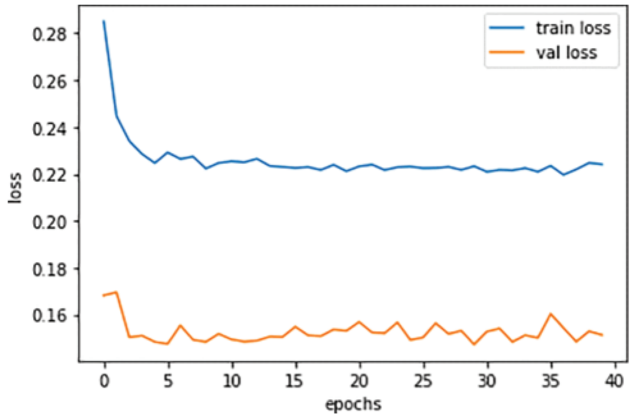

Figure 7: Training and validation loss performance plot

5.1 Model Prediction

The next step is to forecast the class labels using the omitted data, as the model has demonstrated highly satisfactory training and validation accuracy. The model appears to be correctly classifying and is free from overfitting and underfitting issues, as indicated by a substantial discrepancy between training accuracy and prediction accuracy. The provided figure illustrates the model's training and validation performance in correctly classifying data into two classes. It shows that there is no overfitting or underfitting, as there is no significant difference between the training and validation curves, indicating the excellent performance of the model. Additionally, the figure demonstrates that the training and validation performance does not exhibit sudden sharp rises and falls; rather, the model trains smoothly with around 90% accuracy, further signifying the accurate performance of our model.

Subsequently, the predicted labels are compared with the actual test labels using the *X_test* data, as shown in Eqs. (1) and (2).

$$ypred = model.predict\,(X\_test) \tag{1}$$

$$tired = (ypred > 0.5) \tag{2}$$

5.2 Model Evaluation

Comparing the actual and anticipated labels is the basis for model evaluation. The technique includes procedures for locating the true positives, true negatives, false positives, and false negatives. In determining F1 score, precision, recall and accuracy, these parameters are next employed in Eqs. (3)–(7). A higher value of these assessment measures demonstrates that the model can correctly predict unknown or known variables after being appropriately trained on the training dataset.

[1] **Precision:** When false positives are expensive, precision places a strong emphasis on the accuracy of positive predictions. It calculates the proportion of positively predicted observations that occurred (TP) to all positively anticipated observations ($TP + FP$).

$$precision = tp/(tp + fp); \tag{3}$$

[2] **Sensitivity:** Eq. (4) represents sensitivity, the proportion of true positives (tp) correctly identified by a model among all actual positive instances.

$$sensitivity = tp/(tp + fn); \tag{4}$$

[3] **Specificity:** Eq. (5) denotes specificity, the ratio of true negatives (TN) correctly identified among all actual negative instances.

$$specificity = tn/(tn + fp); \tag{5}$$

[4] **Accuracy:** Out of all the cases, it calculates the percentage of accurately anticipated instances. In terms of math, it is computed as:

$$accuracy = \frac{(tp + tp)}{tp + tn + fp + fn}; \tag{6}$$

- (TP): The number of instances where the positive class was accurately predicted.
- (TN): The number of instances where the negative class was accurately predicted.
- (FP): The number of instances where the positive class was incorrectly predicted (Type I error).
- (FN): The number of instances where the negative class was incorrectly predicted (Type II error).

[5] **F1 score:** It is often referred to as the F1 measure or F1 value, and it is a widely used statistic for assessing how well a classification model performs. It is very helpful when working with datasets with an uneven distribution, where one class considerably outnumbers the other. The F1 score balances recall and accuracy by combining the two into a single score.

$$F1 - score = \frac{2. * prec. * sens}{prec + sens} \tag{7}$$

The demonstration is done by evaluating the trained model on unseen test data for different evaluation metrics. As the model still performs better and gives good evaluation metrics results, it shows that the model can correctly predict. Another option for finding these metrics is the sklearn library, which contains pre-built functions for each assessment metric. Accuracy and other evaluation metrics are discovered in this evaluation using the learning library.

5.3 Confusion Matrix

The correctly categorized FP values, TP values that should be in a different class but are in the right class, FN values that should be in the right class but are in the wrong class, and correctly classified TN values that should be in the other class are all included in the confusion matrix. Following the categorization, the effectiveness of the methods was evaluated using the confusion matrix, as illustrated in Fig. 8, which displays the harvest result for the classifier in confusion matrix metrics.

The confusion matrix is a table that presents counts of several sorts of predictions to summarize the effectiveness of a classification model. False positives (FP), true positives (TP), false negatives (FN), and true negatives (TN) are among its four categories.

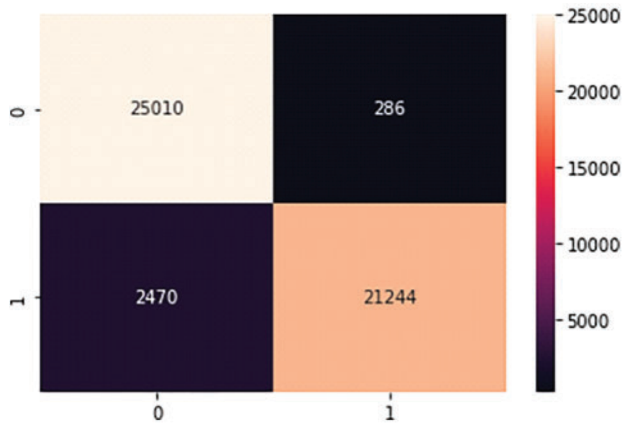

Figure 8: Confusion chart

False positives (FP) are situations that the model predicts as positive but fall into the negative category. In certain situations, the model wrongly classifies an instance as behaving as if it belongs to a particular class when it should not. Incorrect detections or false alarms may be represented as false positives.

Instances that are anticipated as positive and fall within the positive category are known as true positives (TP). These are situations where the model properly recognizes the instances' class membership.

False negatives (FN) are situations that are expected to be negative but are positive. In these situations, the model cannot recognize occurrences that ought to have been given a positive classification. False negatives are situations that were missed or incorrectly labeled as negative.

True negatives (TN) are both projected to be bad and fall within the category of negative events. These are situations when the model properly recognizes that they do not fall under a certain class.

The counts of these several categories are displayed in the confusion matrix, which offers a thorough evaluation of the model's performance and enables evaluation of several measures, including F1 score, recall, accuracy, and precision.

Analysis of values in the confusion matrix, which offers insights into how effectively the model can accurately categorize cases and discriminate between distinct classes, may be used to evaluate the success of the classification techniques.

Fig. 8 is of a confusion matrix which shows for class 0 the accuracy is such that 25,010 test features are correctly classified as class 0 and 286 are wrongly classified as class 1. Similarly, for class 1, it shows that 21,244 are correctly classified as class 1 and 2470 are wrongly classified as 0. The accuracy for class 0 is 98.86% (25,010/25,010 + 286). Similarly, for class 1 accuracy (21,244/2470 + 21,244) is 89.01%. From the confusion matrix accuracy of class 0 is more than class 1. The most often used performance indicators for classification based on these characteristics are precision (P), specificity (Sp), accuracy (ACC), F1 score, and sensitivity (Sn) values. Table 11 contains the classification report that the sklearn obtained:

Table 11: Classification report by learning

	Precision	Recall	F1 score	Support
False	0.91	0.99	0.95	25296
True	0.99	0.90	0.94	23714
Accuracy			0.94	
Accuracy class 0			0.98	
Accuracy class 1			0.89	49010
Macro avg	0.95	0.94	0.94	49010
Weighted avg	0.95	0.94	0.94	49010

All of the assessment metrics for the DNN classification model are presented in Table 12 below. Where several evaluation criteria such as Accuracy score, macro/micro average recall, macro/micro average precision, and macro/micro F1 score, performance values are reported as 0.94. 0.948, 0.9434, 0.9422, 0.9438, 0.9434, and 0.9437.

Table 12: Evaluation metrics performance

Evaluation metric	Performance value
Accuracy score	0.942
Accuracy score class 0	0.989
Accuracy score class 1	0.892
Macro average precision	0.9482
Macro average precision class 0	0.9987
Macro average precision class 1	0.9012
Micro averaged precision	0.9437
Micro averaged precision class 0	0.9811
Micro averaged precision class 1	0.9123
Macro average recall	0.9422
Macro average recall class 0	0.9832
Macro average recall class 1	0.8912
Micro average recall	0.9438
Micro average recall class 0	0.9871
Micro average recall class 1	0.8911
Macro averaged F1 score	0.9434
Macro averaged F1 score class 0	0.9945
Macro averaged F1 score class 1	0.9245
Micro averaged F1 score	0.9437
Micro averaged F1 score class 0	0.9847
Micro averaged F1 score class 1	0.9012

Table 13 indicated the values of loss performance and accuracy with a value of validation accuracy recorded at 0.9130 and 0.9438 and training and validation loss values recorded at 0.22 and 0.1533.

Table 13: Accuracy and loss performance

Evaluation metric	Performance value
Training accuracy	0.9130
Validation accuracy	0.9438
Training loss	0.22
Validation loss	0.1553

The above training accuracy and validation accuracy is for a single class but for the deep learning model trying to classify between two classes with an accuracy of 94.38%.

The ROC curve is the performance indicator for categorization problems at various threshold levels, as shown in Fig. 9. AUC stands for the level or measurement of separability, and ROC is a probability curve. It shows how the model can adapt to different classes. The greater the AUC, the better the model is at categorizing 0 classes as 0, and 1 classes as 1. The model, for instance, does a better job of distinguishing between those with the illness and those without.

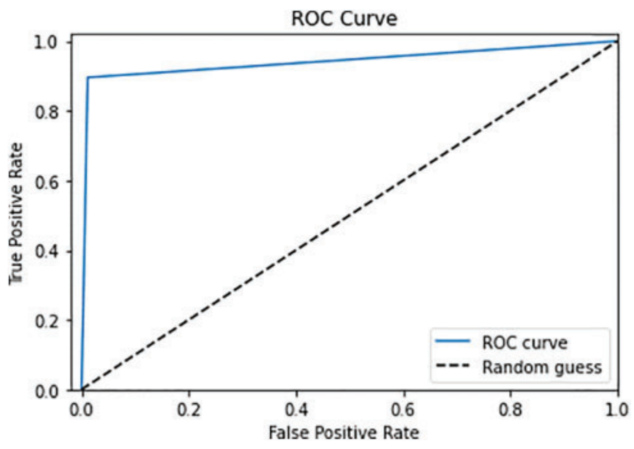

Figure 9: ROC curve performance

False positive is displayed *vs*. True positive on the ROC curve with True positive on the y-axis and False positive on the x-axis. The preceding picture makes it evident that all evaluation metrics produced good performance values, demonstrating that the model does not exhibit overfitting or underfitting in preference to correctly categorizing labels over a given set of data. The model may have performed better than 94% of the time for further epochs but training a model for more epochs causes the model to either overfit or underfit. The above results show the performance of our model for intrusion detection using the NSL-KDD dataset. it is important to note that current IOT systems are expanding from home applications to industrial products and public domain products. These IOT systems are prone to various kinds of attacks. The above results show that when any IOT system is subjected to any type of attack and if we have an intrusion detection system many IOT applications can be protected from various types of attacks. The above results are obtained after training our DNN model with the highest possible accuracy to detect normal traffic from intrusion traffic and when any IOT system is subjected to attack this DNN system when in real-time application can protect IOT system from many dangers.

5.4 Comparative Analysis

Table 14 below shows the comparison:

Table 14: Comparative analysis

References	Dataset	Methodology	Accuracy
Our Model	NSL-KDD	DNN based IDS	94.38%
[2]	NSL-KDD	Stacked Denoising Auto-encoders	88% to 92%
[4]	NSL-KDD	DNN, DT	86.8%
[6]	CICIDS2019	Ensemble 1	92%
[9]	NSL-KDD	SAAE (Stacked Attention Auto-Encoder (SAAE))-DNN	77.57%
[11]	NSL-KDD	Multiclass SVM with chi-square feature selection, GRU-LSTM	88.81%
[15]	NSL-KDD	Deep Learning (DL) with STL	88.39%
[17]	NSL-KDD	RNN-based IDS	82.38%
[18]	NSL-KDD	Improved LSTM	93.09%
[24]	NSL-KDD	Artificial Neural Network (ANN)	81.2%
[25]	NSL-KDD	Deep Learning-based Network Intrusion Detection (DLNID)	89.65%
[38]	NSL-KDD	CNN	91.27%
[39]	NSL-KDD	Ensemble Learning and LSTM	93.4%
[40]	NSL-KDD	Deep Learning-based IDS for SDN	76.86%

In the realm of network security, the efficacy of intrusion detection systems (IDS) is paramount in safeguarding digital assets against malicious activities. The integration of DNN methodologies has emerged as a promising approach to enhancing the accuracy and reliability of IDS. In Table 14, a comparative analysis of various IDS methodologies utilizing DNNs, as well as alternative techniques, is presented based on their performance metrics, primarily focusing on accuracy. Among these methodologies, our proposed model utilizing DNN showcases notable superiority in accuracy, achieving a commendable score of 94.38% on the NSL-KDD dataset.

Our model's remarkable performance underscores the effectiveness of employing DNNs in network security applications, particularly in the context of intrusion detection. By leveraging the inherent capabilities of deep learning, such as feature extraction and pattern recognition, our model demonstrates robustness in accurately identifying and mitigating potential security threats within network traffic. Comparatively, other studies have also explored the integration of DNNs and alternative methodologies for intrusion detection. For instance, reference [2] employs Stacked Denoising Auto-encoders on the NSL-KDD dataset, yielding accuracy ranging between 88% to 92%. While this approach demonstrates commendable performance, it falls short of our model's accuracy.

Similarly, reference [4] adopts a combination of DNN and Decision Trees (DT), achieving an accuracy of 86.8% on the NSL-KDD dataset. Despite its utilization of multiple techniques, the accuracy attained by this methodology is surpassed by our model, reaffirming the efficacy of a standalone DNN approach in intrusion detection. In addition to, reference [6] uses the CICIDS2019 dataset and applies ensemble techniques to achieve 92% accuracy. On the other hand, Reference [9] introduces the Stacked Attention Auto-Encoder (SAAE)-DNN model, achieving an accuracy of 77.57%. Although this methodology incorporates attention mechanisms for enhanced feature learning, its performance lags behind our model's accuracy, emphasizing the superiority of our proposed approach. Furthermore, studies such as [11] and [15] explore alternative techniques such as Multiclass SVM with feature selection and Deep Learning with Self-Taught Learning (STL), respectively. While these methodologies achieve respectable accuracies ranging from 88.39% to 88.81%, they are outperformed by our DNN-based model on the NSL-KDD dataset. Further, reference [17] leverages NSL-KDD and LITNET-2020 datasets with an improved LSTM and PSO method, reaching 93.09% accuracy. Additionally, references [18,24,25] investigate RNN-based IDS, Artificial Neural Networks (ANN), and Deep Learning-based Network Intrusion Detection (DLNID) systems, respectively. Although these methodologies exhibit accuracies ranging from 79.9% to 90.73%, none surpasses the accuracy achieved by our proposed DNN model, highlighting its efficacy in network security applications.

Moreover, approaches such as CNN-based IDS [38] and Ensemble Learning with LSTM demonstrate 91.27% and 93.4% competitive accuracies, respectively. While these methodologies exhibit promising results, our DNN-based model maintains its superiority with a higher accuracy score of 94.38% [39], consolidating its efficacy in intrusion detection on the NSL-KDD dataset. Furthermore, Reference [40] explores a Deep Learning-based IDS tailored for Software-Defined Networking (SDN), achieving efficiency with a precision rate of approximately 76.86%.

In summary, the comparative analysis underscores the effectiveness of DNN-based IDS methodologies in network security, particularly evidenced by our model's exemplary accuracy of 94% on the NSL-KDD dataset. Despite the existence of alternative approaches, including ensemble learning and specialized architectures for SDN environments, none surpasses the accuracy achieved by our proposed model. This reaffirms the prominence of DNNs as a viable solution for enhancing intrusion detection capabilities and fortifying network defenses against evolving cyber threats.

6 Discussion

The study focused on developing an effective Intrusion Detection System (IDS) leveraging Deep Neural Network (DNN) models to bolster network security. Intrusion detection is critical in cybersecurity, particularly as cyber threats evolve in complexity and frequency. Traditional intrusion detection methods often struggle to discern between benign and malicious network activities, necessitating more advanced approaches. The research addressed this challenge by utilizing DNNs, renowned for their ability to automatically extract intricate features and detect subtle correlations within data. By employing the NSL-KDD dataset, the study aimed to demonstrate the efficacy of DNN-based IDS in enhancing network security.

Dataset imbalance is a prevalent issue in intrusion detection studies, which can impede the learning process of machine learning models. To overcome this obstacle, the researchers balanced the dataset, ensuring equitable representation of each class within the NSL-KDD dataset. Balancing the dataset is crucial as it enables the DNN model to learn from a diverse range of network traffic instances, enhancing its ability to effectively distinguish between different classes. This approach lays a solid foundation for the DNN-based IDS, mitigating the risk of biased learning and improving the overall accuracy of intrusion detection.

Deep learning methodologies, particularly DNNs, offer several advantages over traditional machine learning approaches in the context of intrusion detection. DNNs excel at automatically learning intricate patterns and correlations within data, allowing them to discern even subtle anomalies indicative of malicious activities. The high-dimensional representation learned by DNNs enables them to capture complex relationships among network features, leading to enhanced detection accuracy. Moreover, the inherent scalability of DNNs makes them well-suited for handling large volumes of network traffic data, which is essential for real-time intrusion detection in modern network environments.

The evaluation metrics of the DNN-based IDS demonstrate promising performance in terms of accuracy and loss. With training accuracy reaching 91.30% and validation accuracy at 94.38%, the model exhibits a high degree of proficiency in distinguishing between benign and malicious network traffic. Furthermore, the relatively low training and validation losses of 0.22 and 0.1553, respectively, indicate the robust learning and generalization capabilities of the DNN model. These metrics underscore the effectiveness of the proposed approach in enhancing network security through intrusion detection.

In conclusion, the research underscores the potential of deep neural network-based intrusion detection systems in fortifying network security against evolving cyber threats. By leveraging the NSL-KDD dataset and employing advanced deep learning techniques, the study demonstrates the efficacy of DNNs in automatically learning intricate features and correlations within network traffic data. The balanced dataset, coupled with the inherent advantages of DNNs, contributes to improved accuracy and reliability in detecting malicious activities. Overall, the findings highlight the significance of adopting advanced machine learning techniques to address the challenges posed by modern cybersecurity threats and enhance network security infrastructure.

7 Conclusion

In this study, we looked into how to improve network security by using an intrusion detection system (IDS) based on a deep neural network (DNN). Using the NSL-KDD dataset, we have tackled the difficulties of precisely identifying and categorizing various kinds of network intrusions by concentrating on cutting-edge deep learning approaches. Our findings show that by attaining high accuracy and successfully resolving the dataset's intrinsic imbalance, DNN-based IDS can greatly enhance cybersecurity defenses.

We obtained a training accuracy of 91.30% and a validation accuracy of 94.38% through our studies, with corresponding training and validation losses of 0.1553 and 0.22, respectively. These findings verify the excellent performance of our model in both the learning and generalization phases. Additionally, our evaluation metrics—accuracy, precision, recall, and F1 score—across various averages and classes offer a thorough analysis of our model's performance. In particular, our DNN-based IDS proved to be reliable and effective, as evidenced by the macro average precision, recall, and F1 score values, which all showed good performance across classes.

Even though these results are encouraging, it's crucial to take into account additional performance measures to get a more comprehensive picture of our IDS's capabilities. Measures like recall, F1 score, and the Area Under the Receiver Operating Characteristic (AUROC) curve are essential for assessing how well an intrusion detection system (IDS) can identify and categorize different types of intrusions while reducing false positives and false negatives. These metrics provide information on how useful the model is in actual use.

The importance of model interpretability for deep learning-based IDS is further highlighted by our research. Understanding these models' decision-making procedures is crucial to fostering confidence and permitting human intervention as required as these models get more complex. Future studies should, therefore, investigate interpretability strategies designed especially for deep learning architectures so that interested parties may understand the reasoning behind the IDS's classifications and respond accordingly based on its predictions.

Future research must also focus on how well IDSs respond to new types of cyberattacks. Considering how constantly changing the threat landscape is, it is critical to fortify IDSs against new attacks. Techniques like transfer learning and ensemble approaches have the potential to improve the robustness and practicality of our suggested model. By combining several models to increase prediction accuracy, ensemble approaches can reduce the chance of overfitting and strengthen the resilience of the IDS against hostile attacks. Similar to this, transfer learning makes use of expertise from one domain to enhance performance in another, giving the IDS chances to adjust to novel intrusion scenarios and a variety of datasets.

Interdisciplinary cooperation and ongoing innovation will be essential in the future to keep up with the rapidly changing landscape of cyber threats. To progress cybersecurity and create comprehensive, adaptive intrusion detection systems (IDSs) that are capable of adapting to the ever-changing cyber threat scenario, researchers, practitioners, and policymakers must collaborate. Sensitive data can be protected from unwanted access, and network security can be further strengthened by integrating DNN-based IDSs with other security measures, such as firewalls and encryption.

To sum up, this study shows how deep learning may improve network security by creating a reliable and effective intrusion detection system based on DNNs. Through the resolution of issues like model interpretability and dataset imbalance, we have established a strong basis for the development of robust and efficient IDSs. Our model's excellent performance and high accuracy highlight the potential of deep learning techniques to strengthen cybersecurity defenses.

To safeguard digital ecosystems against new threats, we must continue to take a cooperative and creative approach to the long road toward complete cybersecurity. We can set the path for a more robust and safe digital future by working together across disciplines. Let us work to protect our networks and build a more secure, reliable digital environment by utilizing cutting-edge technology like deep learning.

Acknowledgement: The authors would like to thank Princess Nourah bint Abdulrahman University for funding this project through the Researchers Supporting Project (PNURSP2024R319) and Prince Sultan University for covering the article processing charges (APC) associated with this publication. Special acknowledgement to Automated Systems & Soft Computing Lab (ASSCL), Prince Sultan University, Riyadh, Saudi Arabia. Also, the authors wish to acknowledge the editor and anonymous reviewers for their insightful comments, which have improved the quality of this publication.

Funding Statement: The authors would like to thank Princess Nourah bint Abdulrahman University for funding this project through the Researchers Supporting Project (PNURSP2024R319) and this research was funded by the Prince Sultan University, Riyadh, Saudi Arabia.

Author Contributions: Study conception and design: Fatma S. Alrayes, Mohammed Zakariah, Syed Umar Amin; data collection: Mohammed Zakariah, Syed Umar Amin, Zafar Iqbal Khan; analysis and interpretation of results: Fatma S. Alrayes, Syed Umar Amin, Zafar Iqbal Khan; draft manuscript preparation: Mohammed Zakariah, Jehad Saad Alqurni, Syed Umar Amin. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Dataset is available on reasonable request.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] E. A. Aldakheel, M. Zakariah, G. A. Gashgari, F. A. Almarshad, and A. I. A. Alzahrani, "A deep learning-based innovative technique for phishing detection in modern security with uniform resource locators," *Sensors*, vol. 23, no. 9, pp. 4403, Apr. 2023. doi: 10.3390/s23094403.

[2] M. Hussien, "Enhancing network security with a deep learning-based intrusion detection system for 5G networks," *RS Open J. Innov. Commun. Technol.*, vol. 9, pp. 1–10, Oct. 2023. doi: 10.46470/03d8ffbd.bddb240c.

[3] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 20, Dec. 2019. doi: 10.1186/s42400-019-0038-7.

[4] Y. Tang, L. Gu, and L. Wang, "Deep stacking network for intrusion detection," *Sensors*, vol. 22, no. 1, pp. 25, Dec. 2021. doi: 10.3390/s22010025.

[5] E. A. Aldakheel and M. Zakariah, "Mobile devices and cybersecurity issues authentication techniques with machine learning," *Manag. Inf. Decis. Sci.*, vol. 24, pp. 1–15, 2021.

[6] D. Musleh, M. Alotaibi, F. Alhaidari, A. Rahman, and R. M. Mohammad, "Intrusion detection system using feature extraction with machine learning algorithms in IoT," *J. Sens. Actuator Netw.*, vol. 12, no. 2, pp. 29, Mar. 2023. doi: 10.3390/jsan12020029.

[7] A. T. Azar, E. Shehab, A. M. Mattar, I. A. Hameed, and S. A. Elsaid, "Deep learning-based hybrid intrusion detection systems to protect satellite networks," *J. Netw. Syst. Manage.*, vol. 31, no. 4, pp. 82, Oct. 2023. doi: 10.1007/s10922-023-09767-8.

[8] R. Qaddoura, A. M. Al-Zoubi, H. Faris, and I. Almomani, "A multi-layer classification approach for intrusion detection in IoT networks based on deep learning," *Sensors*, vol. 21, no. 9, pp. 2987, Apr. 2021. doi: 10.3390/s21092987.

[9] Z. Zhu and M. Zhu, "A novel method for anomaly detection in the internet of things using whale optimization algorithm," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 5, 2023. doi: 10.14569/issn.2156-5570.

[10] T. Saba, A. Rehman, T. Sadad, H. Kolivand, and S. A. Bahaj, "Anomaly-based intrusion detection system for IoT networks through deep learning model," *Comput. Electr. Eng.*, vol. 99, no. 5, pp. 107810, Feb. 2022. doi: 10.1016/j.compeleceng.2022.107810.

[11] C. Tang, N. Luktarhan, and Y. Zhao, "SAAE-DNN: Deep learning method on intrusion detection," *Symmetry*, vol. 12, no. 10, pp. 1695, Oct. 2020. doi: 10.3390/sym12101695.

[12] H. Shapoorifard and P. Shamsinejad, "Intrusion detection using a novel hybrid method incorporating an improved KNN," *Int. J. Comput. Appl.*, vol. 173, no. 1, pp. 5–9, Sep. 2017. doi: 10.5120/ijca2017914340.

[13] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *IEEE Trans. Syst., Man, Cybernetics, C (Appl. Rev.)*, vol. 38, no. 5, pp. 649–659, Sep. 2008. doi: 10.1109/TSMCC.2008.923876.

[14] S. Choudhary and N. Kesswani, "Analysis of KDD-Cup'99, NSL-KDD and UNSW-NB15 datasets using deep learning in IoT," *Proc. Comput. Sci.*, vol. 167, pp. 1561–1573, 2020. doi: 10.1016/j.procs.2020.03.367.

[15] A. El-Ghamry, A. Darwish, and A. E. Hassanien, "An optimized CNN-based intrusion detection system for reducing risks in smart farming," *Internet Things*, vol. 22, no. 4, pp. 100709, Jul. 2023. doi: 10.1016/j.iot.2023.100709.

[16] A. A. E. B. Donkol, A. G. Hafez, A. I. Hussein, and M. M. Mabrook, “Optimization of intrusion detection using likely point PSO and enhanced LSTM-RNN hybrid technique in communication networks,” *IEEE Access*, vol. 11, pp. 9469–9482, 2023. doi: 10.1109/ACCESS.2023.3240109.

[17] Z. M. Al-Khuzaie, S. A. K. Albermany, and M. A. AbdlNibe, “Intrusion detection in the IoT-Fog adopting the GRU and CNN: A deep learning-based approach,” *Micro-Electron. Telecommun. Eng.*, pp. 379–389, 2023. doi: 10.1007/978-981-19-9512-5.

[18] A. Thangasamy, B. Sundan, and L. Govindaraj, “A novel framework for DDoS attacks detection using hybrid LSTM techniques,” *Comput. Syst. Sci. Eng.*, vol. 45, no. 3, pp. 2553–2567, 2023. doi: 10.32604/csse.2023.032078.

[19] T. Sowmya and E. A. Mary Anita, “A comprehensive review of AI based intrusion detection system,” *Measur.: Sens.*, vol. 28, no. 4, pp. 100827, Aug. 2023. doi: 10.1016/j.measen.2023.100827.

[20] M. Bakro *et al.*, “Efficient intrusion detection system in the cloud using fusion feature selection approaches and an ensemble classifier,” *Electronics*, vol. 12, no. 11, pp. 2427, May 2023. doi: 10.3390/electronics12112427.

[21] M. Radhi Hadi and A. Saher Mohammed, “An efficient deep learning approach for network intrusion detection system on software defined network,” *Int. J. Netw. Secur. Appl.*, vol. 14, no. 4, pp. 1–14, Jul. 2022. doi: 10.5121/ijnsa.2022.14401.

[22] A. Prabhakaran, V. Kumar Chaurasiya, S. Singh, and S. Yadav, “An optimized deep learning framework for network intrusion detection system (NIDS),” in *2020 Int. Conf. Eng. Telecommun. (En&T)*, Dolgoprudny, Russia, IEEE, Nov. 2020, pp. 1–6. doi: 10.1109/EnT50437.2020.9431266.

[23] A. H. Janabi, T. Kanakis, and M. Johnson, “Convolutional neural network based algorithm for early warning proactive system security in software defined networks,” *IEEE Access*, vol. 10, pp. 14301–14310, 2022. doi: 10.1109/ACCESS.2022.3148134.

[24] S. K. Dey and M. M. Rahman, “Effects of machine learning approach in flow-based anomaly detection on software-defined networking,” *Symmetry*, vol. 12, no. 1, pp. 7, Dec. 2019. doi: 10.3390/sym12010007.

[25] K. Dushyant, G. Muskan, Annu, A. Gupta, and S. Pramanik, “Utilizing machine learning and deep learning in cybesecurity: An innovative approach,” *Cyber Secur. Digi. Forensics*, pp. 271–293, 2022. doi: 10.1002/9781119795667.ch12.

[26] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat and S. Venkatraman, “Deep learning approach for intelligent intrusion detection system,” *IEEE Access*, vol. 7, pp. 41525–41550, 2019. doi: 10.1109/ACCESS.2019.2895334.

[27] T. Su, H. Sun, J. Zhu, S. Wang, and Y. Li, “Deep learning methods on network intrusion detection using NSL-KDD dataset,” *IEEE Access*, vol. 8, pp. 29575–29585, 2020. doi: 10.1109/ACCESS.2020.2972627.

[28] A. A. Awad, A. F. Ali, and T. Gaber, “An improved long short term memory network for intrusion detection,” *PLoS One*, vol. 18, no. 8, pp. e0284795, Aug. 2023. doi: 10.1371/journal.pone.0284795.

[29] B. Ingre, A. Yadav, and A. K. Soni, “Decision tree based intrusion detection system for NSL-KDD dataset,” vol. 84, pp. 207–218, 2018. doi: 10.1007/978-3-319-63645-0.

[30] Y. Fu, Y. Du, Z. Cao, Q. Li, and W. Xiang, “A deep learning model for network intrusion detection with imbalanced data,” *Electronics*, vol. 11, no. 6, pp. 898, Mar. 2022. doi: 10.3390/electronics11060898.

[31] M. Ramaiah, V. Chandrasekaran, V. Ravi, and N. Kumar, “An intrusion detection system using optimized deep neural network architecture,” *Trans. Emerg. Telecomm. Technol.*, vol. 32, no. 4, pp. 86, Apr. 2021. doi: 10.1002/ett.4221.

[32] S. P. Thirimanne, L. Jayawardana, L. Yasakethu, P. Liyanaarachchi, and C. Hewage, “Deep neural network based real-time intrusion detection system,” *SN Comput. Sci.*, vol. 3, no. 2, pp. 145, Mar. 2022. doi: 10.1007/s42979-022-01031-1.

[33] K. Farhana, M. Rahman, and M. T. Ahmed, “An intrusion detection system for packet and flow based networks using deep neural network approach,” *Int. J. Electr. Comput. Eng.*, vol. 10, no. 5, pp. 5514, Oct. 2020. doi: 10.11591/ijece.v10i5.pp5514-5525.

[34] P. Toupas, D. Chamou, K. M. Giannoutakis, A. Drosou, and D. Tzovaras, "An intrusion detection system for multi-class classification based on deep neural networks," in *2019 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, IEEE, Dec. 2019, pp. 1253–1258. doi: 10.1109/ICMLA.2019.00206.

[35] R. Zarai, M. Kachout, M. A. G. Hazber, and M. A. Mahdi, "Recurrent neural networks and deep neural networks based on intrusion detection system," *OAlib*, vol. 7, no. 3, pp. 1–11, 2020. doi: 10.4236/oalib.1106151.

[36] S. D. Alotaibi *et al.*, "Deep neural network-based intrusion detection system through PCA," *Math. Probl. Eng.*, vol. 2022, no. 1, pp. 1–9, May 2022. doi: 10.1155/2022/6488571.

[37] N. Gao, L. Gao, Q. Gao, and H. Wang, "An intrusion detection model based on deep belief networks," in *2014 Second Int. Conf. Adv. Cloud Big Data*, IEEE, Nov. 2014, pp. 247–252. doi: 10.1109/CBD.2014.41.

[38] B. Susilo and R. F. Sari, "Intrusion detection in IoT networks using deep learning algorithm," *Information*, vol. 11, no. 5, pp. 279, May 2020. doi: 10.3390/info11050279.

[39] M. Farsi, "Application of ensemble RNN deep neural network to the fall detection through IoT environment," *Alex. Eng. J.*, vol. 60, no. 1, pp. 199–211, Feb. 2021. doi: 10.1016/j.aej.2020.06.056.

[40] W. Elmasry, A. Akbulut, and A. H. Zaim, "Evolving deep learning architectures for network intrusion detection using a double PSO metaheuristic," *Comput. Netw.*, vol. 168, no. 10, pp. 107042, Feb. 2020. doi: 10.1016/j.comnet.2019.107042.

[41] M. Zhong, M. Lin, and Z. He, "Dynamic multi-scale topological representation for enhancing network intrusion detection," *Comput. Secur.*, vol. 135, pp. 103516, Dec. 2023. doi: 10.1016/j.cose.2023.103516.

[42] M. Zhong, M. Lin, C. Zhang, and Z. Xu, "A survey on graph neural networks for intrusion detection systems: Methods, trends and challenges," *Comput. Secur.*, vol. 141, no. 3, pp. 103821, Jun. 2024. doi: 10.1016/j.cose.2024.103821.

Appendix A

Terms and abbreviation

Abbreviation	Meaning	Abbreviation	Meaning	Abbreviation	Meaning
IDS	Intrusion Detection System	NSL-KDD	Network Security Laboratory -Knowledge Discovery in Databases	R2L	Root to Local Security
ML	Machine Learning	ANN	Artificial Neural Network	U2R	User to Root Security
DL	Deep Learning	SDN	Software-Defined Networks	RNN	Recurrent Neural Networks
DNN	Deep Neural Network	DoS	Denial of Service	LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network	SVM	Support Vector Machine	STL	Self-Taught Learning
FS	Feature Selection	PSO	Particle Swarm Optimization	NIDS	Network IDS
LR	Logistic Regression	MLP	Multiple-Layer Perceptron	FAR	False Alarm Rate
ADAM	Adaptive Moment	ICMP	Internet Control Message Protocol	TCP	Transmission Control Protocol
SGD	Stochastic Gradient Descent	FINN	Feed-Forward Neural Networks	SD	Standard Deviation
SATAN	Security Administrator Tool for Analyzing Networks	NLP			