algorithms

MDPI

Article

Enhanced Hybrid Deep Learning Models-Based Anomaly Detection Method for Two-Stage Binary and Multi-Class Classification of Attacks in Intrusion Detection Systems

Hesham Kamal * and Maggie Mashaly *

Networks Department, Faculty of Information Engineering and Technology (IET), German University in Cairo (GUC), New Cairo 11835, Egypt

* Correspondence: hesham.khalil@student.guc.edu.eg (H.K.); maggie.ezzat@guc.edu.eg (M.M.)

Abstract: As security threats become more complex, the need for effective intrusion detection systems (IDSs) has grown. Traditional machine learning methods are limited by the need for extensive feature engineering and data preprocessing. To overcome this, we propose two enhanced hybrid deep learning models, an autoencoder–convolutional neural network (Autoencoder–CNN) and a transformer–deep neural network (Transformer–DNN). The Autoencoder reshapes network traffic data, addressing class imbalance, and the CNN performs precise classification. The transformer component extracts contextual features, which the DNN uses for accurate classification. Our approach utilizes an enhanced hybrid adaptive synthetic sampling–synthetic minority oversampling technique (ADASYN-SMOTE) for binary classification and enhanced SMOTE for multi-class classification, along with edited nearest neighbors (ENN) for further class imbalance handling. The models were designed to minimize false positives and negatives, improve real-time detection, and identify zero-day attacks. Evaluations based on the CICIDS2017 dataset showed 99.90% accuracy for Autoencoder–CNN and 99.92% for Transformer–DNN in binary classification, and 99.95% and 99.96% in multi-class classification, respectively. On the NF-BoT-IoT-v2 dataset, the Autoencoder–CNN achieved 99.98% in binary classification and 97.95% in multi-class classification, while the Transformer–DNN reached 99.98% and 97.90%, respectively. These results demonstrate the superior performance of the proposed models compared with traditional methods for handling diverse network attacks.

Keywords: Autoencoder–CNN; binary classification; data resampling; deep learning; IDS; multi-class classification; Transformer–DNN

Academic Editor: Grammati Pantziou

Received: 7 December 2024
Revised: 10 January 2025
Accepted: 17 January 2025
Published: 28 January 2025

Citation: Kamal, H.; Mashaly, M. Enhanced Hybrid Deep Learning Models-Based Anomaly Detection Method for Two-Stage Binary and Multi-Class Classification of Attacks in Intrusion Detection Systems. *Algorithms* **2025**, *18*, 69. https://doi.org/10.3390/a18020069

1. Introduction

With time, the internet has developed and grown, and it now provides a plethora of beneficial services to enhance people's lives. However, there are a number of security hazards connected to these services, including a rise in network infections, eavesdropping, and hostile attacks [1–3], all of which complicate detection and raise false alarms. A growing number of internet users, including important businesses like banks, enterprises, and governmental agencies, have network security as their top priority.

Cyber-attacks typically begin with reconnaissance to identify vulnerabilities, which are then exploited to initiate damaging actions [4]. Unauthorized access to computer systems poses threats to their confidentiality, integrity, and availability (CIA), leading to what we term an "intrusion" [5]. In recent years, numerous innovative cyber-attack techniques have emerged, such as brute force attacks, botnets, distributed denial of service

(DDoS), and cross-site scripting [6]. These attacks have raised significant concerns about cyber security. Cybercriminals have increasingly utilized various platforms and infrastructures as vectors for malware and botnets, including for Bitcoin Trojans. According to the internet security threat report (ISTR), malware is discovered every thirteen seconds on average during web searches. Incidences of ransom ware, email spam, and other online threats have seen substantial increases, as reported by CNBC [7]. Cybercrime continues to escalate at an alarming rate, with the global cost of cyber-attacks projected to reach an unprecedented $9.5 trillion annually by 2024 [8]. This sharp increase highlights the growing complexity and sophistication of cyber threats. As organizations encounter higher stakes in protecting sensitive data, the role of cyber security measures such as intrusion detection systems (IDSs) becomes more critical. These systems help identify and mitigate potential risks, offering proactive defense against the ever-evolving landscape of cyber threats. As we approach 2025, cybercrime is expected to remain a top global concern, further driving the need for advanced, real-time threat detection and mitigation strategies.

Real-time intrusion detection is critical for ensuring the security and integrity of network systems. Deep learning models have proven effective in real-time analysis of network traffic, enabling swift identification of intrusions [9]. Various machine learning approaches enhance the responsiveness of IDSs, particularly in adapting to emerging threats [10]. Additionally, integrating real-time capabilities within IDSs significantly improves network security by promptly detecting and mitigating attacks [11].

IDSs are among the most widely used security solutions, designed to identify unauthorized access and safeguard devices and network infrastructure from malicious activities. IDSs can be broadly categorized based on their detection approach into two main types. The first type, signature-based IDSs, compare network traffic or host activity against a database of known malicious patterns. While this method is effective for identifying known threats, it requires frequent updates to remain effective and may struggle with unknown or zero-day attacks as it relies on pre-existing signatures. In contrast, anomaly-based IDSs identify potential threats by detecting deviations from normal behavior. Unlike a signature-based IDS, this approach does not depend on known attack patterns, making it particularly effective for identifying zero-day attacks that exploit previously unknown vulnerabilities. To achieve this, anomaly-based IDSs often leverage machine learning and deep learning techniques to analyze large datasets, learn normal behavior patterns, and detect anomalies with high accuracy. This approach not only enhances the system's adaptability to new threats but also reduces false positives and negatives. In our study, we adopted the anomaly-based approach to effectively address these challenges.

This study introduces two advanced hybrid deep learning models for intrusion detection, a transformer–deep neural network (Transformer–DNN) and an autoencoder–convolutional neural network (Autoencoder–CNN). Both models effectively address class imbalance using advanced techniques like enhanced hybrid adaptive synthetic sampling–synthetic minority oversampling technique (ADASYN-SMOTE), enhanced SMOTE, and edited nearest neighbors (ENN). On the CICIDS2017 dataset [12], both models achieved high accuracy, with Autoencoder–CNN reaching 99.90% in binary and 99.95% in multi-class classification, and Transformer–DNN achieving 99.92% and 99.96%, respectively. With the NF-BoT-IoT-v2 dataset [13], Autoencoder–CNN attained 99.98% in binary and 97.95% in multi-class classification, while Transformer–DNN achieved 99.98% and 97.90%, respectively, demonstrating strong performance across both datasets. The following is an overview of the principal contributions:

- An effective intrusion detection system was developed using two enhanced hybrid deep learning models, Transformer–DNN and Autoencoder–CNN. The transformer extracts contextual features for pattern analysis, while the DNN performs final clas-

sification. The Autoencoder reshapes data, preparing it for precise classification by the CNN.

- Enhanced hybrid ADASYN-SMOTE resampling is leveraged for binary classification, while enhanced SMOTE resampling is applied for multi-class classification. These techniques are combined with ENN to effectively address class imbalance and enhance model performance.
- Integrating the enhanced local outlier factor (LOF) strengthens anomaly detection by detecting and removing outliers, significantly boosting the model's ability to identify minority class attacks and improving overall detection performance.
- Evaluation on the CICIDS2017 and NF-BoT-IoT-v2 datasets demonstrated the superior performance of the proposed models compared with state-of-the-art approaches.

This paper is structured as follows. Section 2 provides a comprehensive review of the related literature. Section 3 describes the methodology employed in this study. Section 4 presents the results obtained from the experiments, while Section 5 offers a detailed discussion of the findings. Section 6 addresses the limitations of the proposed approach. Section 7 concludes the study by summarizing its key contributions and insights. Finally, Section 8 outlines potential directions for future research.

2. Related Work

IDSs now provide essential protection for national, economic, and personal security, in the context of the exponential development of data collection and the growing interconnection of the global internet infrastructure. In an effort to reduce computer system vulnerabilities and improve surveillance capabilities, James P. Anderson invented the notion of intrusion detection in 1980 [14]. While security experts continue to work to improve the effectiveness and precision of IDSs, these systems have been widely implemented over time. This section reviews various machine learning and deep learning techniques used for intrusion detection described in the literature. Since DL has so many uses and performs so well in areas like image recognition and natural language processing, it has become an obvious choice for traffic anomaly detection in IDSs. Deep learning approaches for categorizing attack types in intrusion detection systems have been mostly described in academic publications.

2.1. Binary Classification

Using the Transformer–DNN and Autoencoder–CNN technique for IDS binary classification combines the strengths of advanced contextual feature extraction and spatial pattern recognition. The transformer component excels at capturing contextual relationships in network traffic data, enhancing the ability to discern intricate data dependencies essential for effective classification. The DNN leverages these extracted features to perform accurate final classifications, ensuring robust detection of attack types. Meanwhile, the Autoencoder compresses and reshapes network traffic data, addressing class imbalance and enhancing the representation of data characteristics. These optimized data are then passed to the CNN, which specializes in identifying intricate spatial patterns crucial for distinguishing malicious from legitimate activity. By integrating these methods, this hybrid approach significantly enhances the effectiveness of cyber security defenses against dynamic and evolving threats. The result is an IDS with improved accuracy, reduced false positives and negatives, and superior real-time threat detection capabilities.

In ref. [15], the authors proposed a DNN model that achieved a binary classification accuracy of 93.1%. This study explores the development of a versatile and efficient IDS capable of detecting and categorizing unexpected and evolving cyber-attacks. The dynamic nature of networks and the rapid evolution of attacks necessitate the evaluation of multiple

datasets over time, using both static and dynamic techniques. This approach aided in identifying the most effective methods for detecting emerging threats, and provided a thorough evaluation of DNN models alongside other traditional machine learning classifiers using several publicly available benchmark malware datasets. The authors in ref. [16] introduced a DNN-based intrusion detection model with a reported accuracy of 99%. The model was applied to a recently available dataset that included packet-based and flow-based data along with additional metadata. The dataset was labeled and imbalanced; it included 79 attributes, with some classes having significantly fewer training samples. The study highlights the challenges of working with imbalanced data and the importance of using deep learning models to address these issues. In ref. [17], the authors recommend the use of principal component analysis (PCA) along with classifiers such as random forest (RF), linear discriminant analysis (LDA), and quadratic discriminant analysis (QDA), to achieve 99.6% accuracy. PCA can be employed as a feature dimensionality reduction method, and the reduced features used to construct various classifiers for IDS development.

In ref. [18], the authors introduced a long short-term memory (LSTM) model that achieved 92.2% accuracy in binary classification. This approach incorporates attention mechanisms with LSTM networks to effectively capture both temporal and spatial patterns in network traffic data. The model was tested on the UNSW-NB15 dataset, containing diverse patterns and notable differences between training and testing data, creating a challenging evaluation setting. In ref. [19], a CNN–bi-directional long short-term memory (BiLSTM) model was proposed, attaining 97.90% accuracy in binary classification. This hybrid approach integrated bidirectional LSTM with a lightweight CNN, applying feature selection techniques to optimize model efficiency. The authors of ref. [20] proposed a hybrid model combining LSTM, CNN, and SVM to achieve 98.47% accuracy. Their hybrid semantic deep learning (HSDL) architecture incorporated word2vec embedding to capture semantic information in network traffic, and AES encryption to enhance cloud storage security. The crossover-based mine blast optimization algorithm (CMBA) was used to select the optimal AES key. Additionally, ref. [21] proposed a two-stage deep learning structure incorporating a gated recurrent unit (GRU) network in the first stage and a denoising auto-encoder (DAE) in the second stage, which achieved an accuracy of 90.21% on Test+ for intrusion detection. Furthermore, ref. [22] tackled class imbalance by integrating CNN–bidirectional long short-term memory (BiLSTM) with ADASYN, reaching an accuracy of 90.73% on Test+. In ref. [23], the authors proposed a hybrid Transformer–CNN deep learning model to overcome these challenges. The model utilized data resampling methods like ADASYN, SMOTE, ENN, and class weights to address class imbalance, achieving an accuracy of 99.71%.

In ref. [24], the authors proposed advancements for IDSs in cloud environments by developing and evaluating two innovative deep neural network models. The first model utilized a multi-layer perceptron (MLP) trained with backpropagation (BP), while the second combined particle swarm optimization (PSO) with MLP training. Both models achieved impressive accuracy of 98.97%, demonstrating significant improvements in IDS performance and efficiency for detecting and mitigating intrusions. In ref. [25], the authors demonstrated a notable reduction in the time required for traffic analysis and significant success with their proposed model. Tested on the CSE-CIC-IDS2018 dataset, the model's effectiveness was confirmed. Experimental results revealed that using the ExtraTree algorithm, the model achieved an impressive accuracy of 98.5% for binary classification. In ref. [26], the authors utilized PySpark with Apache Spark in the Google Colaboratory (Colab) environment, relying on the Keras and Scikit-Learn libraries. The training and testing datasets included 'CICIoT2023' and 'TON_IoT'. To enhance the feature set, the datasets were refined using the correlation method. The authors developed a hybrid deep

learning algorithm combining one-dimensional CNN and LSTM, obtaining an accuracy of 98.75% for optimal performance.

In ref. [27], the authors introduced a new classifier algorithm designed to detect malicious traffic in IoT environments using machine learning techniques. The approach utilized a real IoT dataset that simulates actual traffic conditions, assessing the performance of different classification algorithms to evaluate their effectiveness, achieving an accuracy of 99.2%. In ref. [28], the authors described three essential machine learning techniques used for binary classification. These methods were applied within an IDS designed to detect IoT-based attacks and classify them into two categories, benign and malicious. The study utilized the IoT-23 dataset, a recent and extensive dataset, to create an intelligent IDS capable of identifying and categorizing attack patterns in IoT environments for binary classification, achieving an accuracy of 99%. In ref. [29], the authors examined the factors influencing existing near-Earth remote sensing systems and introduced a spatio-temporal graph attention network (N-STGAT) incorporating node states for application in network intrusion detection in near-Earth remote sensing systems, obtaining an accuracy of 97.88%. In ref. [30], the authors introduced a self-supervised graph neural network for network intrusion detection systems, designed to effectively and thoroughly differentiate between normal and malicious network flow associated with various attack types. To the best of our knowledge, this represents the first graph neural network (GNN)-based method for classification tasks in NIDS using an unsupervised approach, resulting in an accuracy of 99.64%.

The Transformer–DNN and Autoencoder–CNN models outperformed existing methods, achieving 99.92% and 99.90% accuracy, respectively, in binary classification using CICIDS2017. With the NF-BoT-IoT-v2 dataset, both models reach 99.98% accuracy. These results represent significant advancements, with a comparison in Table 1.

Table 1. Related work for binary classification.

Author	Dataset	Year	Utilized Technique	Accuracy	Contribution	Limitations
R. Vinaya-Kumar et al. [15]	CICIDS2017	2019	DNN	93.1%	Investigated DNNs for a versatile IDS to detect and categorize novel cyber-attacks, comparing their effectiveness with traditional machine learning classifiers using benchmark datasets.	• Inadequate scalability and performance analysis for distributed systems and advanced DNNs.
Kaniz Farhana et al. [16]	CICIDS2017	2020	DNN	99%	Presents a deep neural network-based intrusion detection model developed using Keras within the Google TensorFlow environment. The model was applied to a recent, imbalanced dataset containing 79 attributes including packet-based, flow-based data, and metadata, with some classes having significantly fewer samples.	• The model's inability to classify 'Heartbleed', 'Infiltration', and 'Web Attack Sql Injection' highlights issues with class imbalance due to the low number of records for these attacks.
Razan Abdulhammed et al. [17]	CICIDS2017	2019	PCA, RF, LDA and QDA	99.6%	PCA was used for dimensionality reduction and the resulting features were applied to build classifiers such as RF, Bayesian networks, LDA, and QDA for IDS development.	• Does not address the need for updating training data in real-time or for developing IDS models that can adapt and train on the go for online network intrusion detection.

Table 1. *Cont.*

Author	Dataset	Year	Utilized Technique	Accuracy	Contribution	Limitations
Mohammad A. Alsharaiah et al. [18]	UNSW-NB15	2024	AT-LSTM	92.2%	Introduced a novel NIDS approach combining LSTM with attention mechanisms to analyze temporal and spatial characteristics of network traffic, employing the UNSW-NB15 dataset, utilizing different sizes for training and testing sets.	• Complicated architecture. • Although the AT-LSTM model achieved high accuracy on the UNSW-NB15 dataset, it lacks techniques for handling class imbalance and has not been tested on datasets like NSL-KDD, potentially limiting its adaptability and performance across various scenarios.
Mohammed Jouhari et al. [19]	UNSW-NB15	2024	CNN-BiLSTM	97.90%	Presents an efficient IDS model that combines BiLSTM with a lightweight CNN and applies feature selection to reduce complexity.	• Complicated architecture. • The research focused on refining the IDS model for computational efficiency, which may have constrained assessment of other key factors like generalization and resilience across different datasets.
Varun Prabhakaran and Ashokkumar Kulandasamy [20]	UNSW-NB15	2021	LSTM-CNN-SVM	98.47%.	Developed the HSDL architecture by combining SVM, CNN, and LSTM for intrusion detection, using word2vec for semantic analysis and AES encryption for cloud security when no intrusion is found. The CMBA selects the most secure encryption key.	• Complex architecture. • The study's focus on the NSL-KDD and UNSW-NB15 datasets limits the generalizability of the proposed hybrid semantic deep learning model to broader applications, such as IoT and Big Data environments.
Ming-Tsung Kao et al. [21]	NSL KDD	2022	Combining the models of GRU and DAE	90.21%	The proposed structure uses a GRU model to analyze network flow and generates a confidence score via softmax. If the score meets or exceeds a threshold, the GRU directly classifies the flow as positive. If not, the GRU classifies it as negative and forwards it to the DAE model, which then evaluates typical flow patterns and sets the reconstruction error threshold.	• Binary classification only. • Complex architecture. • The proposed anomaly detection structure showed improved accuracy but needs validation with additional datasets and real-world data.
Yanfang Fu et al. [22]	NSL KDD	2022	CNN and BiLSTMs	90.73%	Presents DLNID, a traffic anomaly detection model that combines an attention mechanism with Bi-LSTM to improve detection accuracy. A CNN extracts sequence features, then applies attention to adjust channel weights, and Bi-LSTM is finally used to learn the sequence features.	• Binary classification only. • Complex architecture. • The proposed DLNID model showed improved performance on the KDDTest+ test set but has not been tested in real-world environments or for online intrusion detection.

Table 1. *Cont.*

Author	Dataset	Year	Utilized Technique	Accuracy	Contribution	Limitations
Hesham Kamal and Maggie Mashaly [23]	NF-UNSW-NB15-v2	2024	Transformer-CNN	99.71%	Introduces a hybrid Transformer–CNN model that addresses these challenges by utilizing data resampling methods such as ADASYN, SMOTE, ENN, and class weights to manage class imbalance.	• The study's limitations include the need for broader dataset evaluation, improved data preprocessing, model adaptation, hyperparameter optimization, and enhanced scalability and efficiency for larger datasets and complex network traffic.
Saud Alzughaibi and Salim El Khediri [24]	CSE-CIC-IDS2018	2023	MLP-BP, MLP-PSO	98.97%	Improved IDS in cloud environments by developing and testing two deep neural network models: one using MLP with BP and the other with PSO. These models aim to enhance IDS performance and efficiency in detecting and responding to intrusions.	• Low accuracy. • Complex architecture. • The study achieved high accuracy with MLP-BP and MLP-PSO models but the method has yet to be tested in real time or cloud environments; could benefit from exploring different optimization algorithms.
Recep Sinan Arslan [25]	CSE-CIC-IDS2018	2021	ExtraTree algorithm	98.5%	Implementing the proposed model achieved a substantial reduction in traffic analysis time while attaining remarkable success rates. Extensive testing on the CSE-CIC-IDS2018 dataset has validated the model's advantages. Specifically, the ExtraTree algorithm demonstrated an impressive 99.0% detection rate in binary classification tasks.	• The study showed that tree-based classifiers worked well on the CSE-CICIDS2018 dataset but did not explore the use of fewer features, neural networks, or unsupervised learning methods for better speed, accuracy, and adaptability.
Sami Yaras and Murat Dener [26]	ToN-IoT	2024	CNN-LSTM	98.75%	Study conducted in the Colab environment using PySpark with Apache Spark and the Keras and Scikit-Learn libraries. The 'CICIoT2023' and 'TON_IoT' datasets were utilized for both training and testing the model. The correlation method was applied for feature reduction, ensuring that only the most pertinent features were included. A hybrid deep learning algorithm was designed by the researchers, combining one-dimensional CNN and LSTM to optimize the model's performance.	• A major limitation of this study is that, while high accuracy rates were achieved, the large data volumes used led to increased training and testing times, emphasizing the need for future optimization to strike a balance between accuracy, computational efficiency, and cost.
Mohamed ElKashlan et al. [27]	IoT-23	2023	Filtered classifier	99.2%	Presents a classifier algorithm for detecting malicious traffic in IoT environments using machine learning. The proposed system uses a real IoT dataset that reflects actual traffic, and several classification algorithms are evaluated for their performance.	• The limitation of the study lies in its use of only the IoT-23 dataset, which may not capture all attack scenarios in IoT EVCS environments. Future research should include more datasets and deep learning techniques for a more comprehensive evaluation.

Table 1. *Cont.*

Author	Dataset	Year	Utilized Technique	Accuracy	Contribution	Limitations
Trifa S. Othman and Saman M. Abdullah [28]	IoT-23	2023	ANN	99%	Presents three key machine learning techniques for both binary and multi-class classification, forming the foundation of an IDS designed to protect IoT environments. These techniques are used to detect various IoT-based cyber-attacks and accurately classify their types. Utilizing the advanced IoT-23 dataset, the research develops a sophisticated intelligent IDS that identifies malicious activities and classifies attack vectors in real-time, improving security for IoT networks.	• A major limitation of this study is the inability of the SMOTE method to enhance the accuracy of the proposed IIDS model on the IoT-23 dataset, even though it is usually effective for imbalanced datasets.
Yalu Wang et al. [29]	NF-BoT-IoT-v2	2023	N-STGAT	97.88%	Investigated the factors affecting existing near-Earth remote sensing systems, presenting an N-STGAT that integrates node states for use in network intrusion detection in near-Earth remote sensing systems.	• A limitation of this study is that it relies on a specific dataset for evaluation, which may limit the generalizability of the N-STGAT model to other network environments with different characteristics.
Renjie Xu et al. [30]	NF-BoT-IoT-v2	2024	GNN	99.64%	Presents a self-supervised graph neural network for network intrusion detection systems, aimed at efficiently distinguishing normal network flows from malicious ones across different attack types. To the best of our knowledge, this is the first GNN-based approach for multi-class classification in NIDS using an unsupervised method.	• A limitation of this study is that it relies on netflow-based datasets, which may not fully capture the complexities of real-world network traffic, and it does not address data imbalance in network intrusion detection systems.

2.2. Multi-Class Classification

For multi-class categorization in IDSs, the combination of Transformer–DNN and Autoencoder–CNN offers a powerful solution. The transformer effectively captures contextual relationships within network traffic data, enabling the extraction of critical patterns and features. These features are utilized by the DNN to perform precise classifications of various attack types. Meanwhile, the Autoencoder reduces the dimensionality of the data, enhancing its representation, and the CNN leverages these refined features to detect intricate spatial patterns and anomalies. This integrated approach significantly enhances the IDS's ability to distinguish between multiple attack categories, improving detection accuracy and fortifying the system's overall performance.

Given the dynamic nature of network environments and the rapid evolution of attacks, evaluating various datasets using both static and dynamic methods is crucial for identifying the most effective algorithms for detecting future threats. In ref. [15], the authors recommend a DNN model that achieved a multi-class classification accuracy of 95.6%. Their aim was to develop a flexible and effective IDS capable of identifying and categorizing unexpected and evolving cyber-attacks using a deep neural network, and the research provided a comprehensive analysis of DNN and other conventional machine learning classifiers using multiple publicly available benchmark malware datasets. The authors in ref. [16] proposed a DNN model that achieved 99% accuracy. This model, tested on the lat-

est publicly available dataset with packet-based, flow-based data, and additional metadata, addressed the challenges of imbalanced and labeled datasets and used 79 attributes. The research in ref. [17] reported that using PCA, RF, LDA, and QDA models achieved 99.6% accuracy for multi-class classification. PCA was used for dimensionality reduction, and the reduced features were utilized to construct several classifiers for IDS development.

In ref. [23], the authors introduced a hybrid Transformer–CNN deep learning model designed to tackle these challenges. The model incorporates data resampling techniques such as ADASYN, SMOTE, edited ENN, and class weights to mitigate class imbalance, achieving an accuracy of 99.02%. In ref. [31], the authors present a conditional generative adversarial network (CGAN) augmented by bidirectional encoder representation from transformers (BERT), a powerful pre-trained language model, designed to enhance multi-class intrusion detection. The approach uses CGAN to generate additional data for minority attack classes, effectively tackling class imbalance. Moreover, BERT is integrated into the CGAN discriminator, improving feature extraction and strengthening input–output dependencies, thereby boosting detection performance through adversarial training, resulting in an accuracy of 87.40%. In ref. [27], the authors propose a novel classifier algorithm for detecting malicious traffic in IoT environments using machine learning techniques. The method employs a genuine IoT dataset representing real-world traffic patterns and evaluates the effectiveness of various classification algorithms to assess their performance, achieving an accuracy of 99.2%. In ref. [32], an RNN and a DNN were utilized to achieve accuracy rates of 98.68% and 98.95%, respectively. In ref. [33], the authors proposed a multilayer CNN integrated with LSTM, which achieved remarkable accuracy of 99.5%. This approach utilized CNN layers to extract and select features, followed by a softmax classifier to categorize network intrusions. In ref. [34], the authors presented an RF model that achieved an accuracy rate of 98.3%. The study extended its attack detection method to the UNSW-NB15 dataset, reaching 98.3% accuracy in multi-class classification tasks.

A hybrid model proposed by ref. [20], combining LSTM, CNN, and SVM, achieved an accuracy of 98.47%. This approach created a HSDL architecture integrating SVM, CNN, and LSTM to analyze semantic information from network traffic. The study also included AES encryption for cloud storage security, optimized using the CMBA technique. In ref. [24], the authors introduced enhancements for IDS in cloud environments by designing and assessing two novel deep neural network models. The first model utilized MLP optimized through BP, while the second combined MLP with PSO. These models significantly enhanced IDS effectiveness and efficiency, achieving accuracy of 98.41%. In ref. [35], the authors underscore the critical importance of cyber security in monitoring and safeguarding network infrastructures against vulnerabilities and intrusions. They emphasize that advancements in machine learning, particularly deep learning, have significantly improved the early detection and prevention of attacks through advanced self-learning and feature extraction techniques. Their research utilized deep learning to analyze the CSE-CIC-IDS2018 dataset, which included both normal network behavior and various attacks. The evaluation of the LSTM model demonstrated a remarkable detection accuracy of 99%. In ref. [29], the authors analyzed the factors affecting existing near-Earth remote sensing systems and proposed an N-STGAT that integrates node states for use in network intrusion detection in near-Earth remote sensing systems, obtaining an accuracy of 93%. In ref. [36], the authors propose a collaborative federated learning approach that facilitates the sharing of cyber threat intelligence (CTI) between organizations, with the goal of developing a more efficient ML-based network intrusion detection system (NIDS). By implementing LSTM on the NF-BoT-IoT-v2 dataset, the model achieved an accuracy of 94.61%.

The Transformer–DNN and Autoencoder–CNN models achieved 99.96% and 99.95% accuracy, respectively, in multi-class classification with CICIDS2017. With NF-BoT-IoT-v2,

the Autoencoder–CNN reached 97.95% and the Transformer–DNN achieved 97.90%. These results represent significant improvements over prior research, as detailed in Table 2.

Table 2. Related works on multi-class classification.

Author	Dataset	Year	Utilized Technique	Accuracy	Contribution	Limitations
R. Vinaya-Kumar et al. [15]	CICIDS2017	2019	DNN	95.6%	This research aimed to develop a versatile IDS using deep neural networks to identify and classify emerging cyber-attacks. It evaluated various datasets and algorithms, comparing DNNs with traditional classifiers using benchmark malware datasets to determine the most effective approach for detecting new threats.	• Insufficient scalability and performance analysis for distributed systems and advanced DNNs.
Kaniz Farhana et al. [16]	CICIDS2017	2020	DNN	99%	This study introduces a deep neural network-based intrusion detection model tested on a recent, imbalanced dataset with 79 attributes. The model, developed using Keras and TensorFlow, handles packet-based, flow-based data, and metadata.	• The model's failure to classify 'Heartbleed', 'Infiltration', and 'Web Attack Sql Injection' underscores the challenge of class imbalance caused by the limited number of records for these attacks.
Razan Abdulhammed et al. [17]	CICIDS2017	2019	PCA, RF, LDA and QDA	99.6%	This study used PCA to reduce feature dimensionality and then employed the low-dimensional features to build several classifiers, including RF, LDA, and QDA, for an IDS.	• Fails to address the requirement for real-time training data updates or the development of IDS models capable of adapting and training on the go for online network intrusion detection.
Hesham Kamal and Maggie Mashaly [23]	NF-UNSW-NB15-v2	2024	Transformer-CNN	99.02%	In this study, a hybrid Transformer–CNN model is proposed to tackle these challenges, employing data resampling techniques like ADASYN, SMOTE, and ENN, and class weights to effectively handle class imbalance.	• The study's limitations involve the need for broader dataset evaluation, improved preprocessing, model adaptation, hyperparameter optimization, and enhanced scalability.
Fang Li et al. [31]	NF-UNSW-NB15-v2	2024	CGAN-BERT	87.40%	This study introduces a novel approach that merges a CGAN with BERT to address multi-class intrusion detection problems. The method aims to augment data for minority attack classes, thus tackling class imbalance. By incorporating BERT into the CGAN's discriminator, the framework strengthens input–output relationships and boosts detection performance through adversarial training, leading to enhanced feature extraction and a more resilient cybersecurity detection system.	• A major limitation of this study is the challenge of effectively differentiating between attacks with similar traits or high levels of concealment, such as Analysis, Backdoor, and DoS, within the NF-UNSW-NB15-v2 dataset.

Table 2. *Cont.*

Author	Dataset	Year	Utilized Technique	Accuracy	Contribution	Limitations
Mohamed ElKashlan et al. [27]	IoT-23	2023	Filtered classifier	99.2%	Introduces a new classifier algorithm designed to detect malicious traffic in IoT environments using machine learning techniques, utilizing a real IoT traffic dataset and evaluating the performance of several classification algorithms.	• The study is limited by its sole use of the IoT-23 dataset, which may not cover all possible attack scenarios in IoT EVCS environments. Future work should include a wider variety of datasets and employ advanced deep learning methods for a more thorough evaluation.
Arun Kumar Silivery et al. [32]	NSL KDD	2023	RNN and DNN	98.68% and 98.95% respectively.	Validated on the NSL-KDD dataset, the model showed superior detection rates, accuracy, and lower rates of false alarms compared with those using the Adamax optimizer. It was assessed against shallow and deep learning models incorporating RNNs, LSTM-RNNs, and DNNs.	• Multi-class classification only. • The proposed RNN, LSTM-RNN, and DNN models show good performance on KDD'99 and NSL-KDD datasets but struggle with UNSW-NB15, highlighting limitations in handling imbalanced datasets and the need for improved false positive rates.
Muhammad Basit Umair et al. [33]	NSL KDD	2022	Multilayer CNN-LSTM	99.5%	Due to the limitations of traditional methods, the paper proposes a statistical approach for intrusion detection, involving feature extraction, classification with a multilayer convolutional neural network using softmax, and additional classification with a multilayer deep neural network.	• Multi-class classification only. • Complex architecture. • The proposed IDS achieves high accuracy and performance metrics but lacks testing on diverse datasets and real-world scenarios, which could impact its generalizability.
Fuat Turk [34]	UNSW-NB15	2023	RF	98.3%	Advanced ML and DL techniques for attack detection utilizing the UNSW-NB15 and NSL-KDD datasets, achieving 98.3% in multi-class classification on UNSW-NB15.	• Attack classes are occasionally misclassified, requiring better dataset balancing and real-time model updates to enhance performance.
Varun Prabhakaran and Ashokkumar Kulandasamy [20]	UNSW-NB15	2021	LSTM-CNN-SVM	98.47%	Combines SVM, CNN, and LSTM into a HSDL architecture for intrusion detection, using word2vec for semantic analysis of network traffic. The model classifies attacks, encrypts standard text with AES if no intrusion is detected, and employs CMBA for key selection to enhance cloud storage security.	• Complex architecture. • The study's emphasis on the NSL-KDD and UNSW-NB15 datasets restricts the generalizability of the proposed hybrid semantic deep learning model to wider applications, including IoT and Big Data environments.
Saud Alzughaibi and Salim El Khediri [24]	CSE-CIC-IDS2018	2023	MLP-BP, MLP-PSO	98.41%	Development and evaluation of two deep neural network models: one with MLP and BP, and the other with MLP and PSO. These models are designed to improve IDS performance and efficiency in detecting and responding to intrusions, enhancing IDS in cloud environments.	• Complex architecture. • The study demonstrated high accuracy with MLP-BP and MLP-PSO models but the system has not yet been evaluated in real time or cloud environments and may benefit from investigating alternative optimization algorithms.

Table 2. *Cont.*

Author	Dataset	Year	Utilized Technique	Accuracy	Contribution	Limitations
Baraa Ismael Farhan and Ammar D. Jasim [35]	CSE-CIC-IDS2018	2022	LSTM	99%	The increasing demand for cyber security highlights the need for robust network monitoring. The study applied deep learning to the CSE-CIC-IDS2018 dataset, achieving 99% detection accuracy with an LSTM model for identifying network attacks.	• Multi-class classification only. • The proposed LSTM-based intrusion detection system achieves high accuracy but faces challenges relating to dataset imbalance and large size, which may impact accuracy and complicate model design.
Yalu Wang et al. [29]	NF-BoT-IoT-v2	2023	N-STGAT	93%	Examines the factors influencing existing near-Earth remote sensing systems and introduces an N-STGAT that incorporates node states for application in network intrusion detection in near-Earth remote sensing systems.	• A limitation of this study is its dependence on a single dataset for evaluation, which may restrict the N-STGAT model's ability to generalize to diverse network environments with varying characteristics.
Mohanad Sarhan et al. [36]	NF-BoT-IoT-v2	2023	LSTM	94.61%	A collaborative federated learning approach is proposed to enable the sharing of CTI between organizations, aiming to develop a more efficient ML-based NIDS. The implementation of LSTM on the NF-BoT-IoT-v2 dataset achieved an accuracy of 94.61%	• A limitation of this study is the trade-off between slightly reduced classification performance and maintaining privacy, which may affect the model's ability to fully match centralized learning methods in terms of accuracy.

2.3. Challenges

State-of-the-art IDSs leveraging deep learning models face several significant challenges. A key issue is achieving high accuracy, which is often hindered by the class imbalance prevalent in benchmark datasets. These datasets typically contain a disproportionate amount of normal traffic compared with attack traffic, making it difficult to detect rare but critical attack types. This imbalance leads to elevated rates of false alarm and reduces overall detection effectiveness. While deep learning holds promise for improving detection capabilities, it also introduces substantial computational complexity and resource demands, raising concerns about scalability and efficiency, particularly in large-scale, real-time operational settings. Another critical challenge is the limited generalizability of these models. They often struggle to adapt to diverse network conditions or to identify novel attack types not included in the training data, reducing their robustness in practical application. Additionally, many studies have prioritized theoretical and experimental advancements in deep learning, often neglecting practical deployment challenges such as data privacy, system latency, and integration with existing security frameworks. A further limitation is the tendency to focus narrowly on accuracy, potentially overlooking other essential performance metrics such as precision, recall, F1 score, and the implications of false positives and negatives. Addressing these interconnected challenges requires a holistic approach that emphasizes balanced data handling, scalability, adaptability to evolving network threats, and practical considerations for deployment in real-world environments.

The Transformer–DNN and Autoencoder–CNN models address key limitations in existing intrusion detection systems, demonstrating superior accuracy and other performance metrics compared with traditional methods. These models effectively tackle class imbalance issues through advanced resampling techniques, including enhanced hybrid

ADASYN-SMOTE for binary classification and enhanced SMOTE for multi-class classification, combined with ENN. The Autoencoder enhances the preprocessing of network traffic data by improving feature representation and balancing class distributions, which significantly boosts the CNN's ability to classify and detect rare attack types. Meanwhile, the transformer excels in capturing contextual relationships within data, enabling the analysis of complex patterns and dependencies, while the DNN leverages these insights for precise classification. Both models are optimized for scalability and performance, efficiently handling large-scale datasets while maintaining real-time processing capabilities. Their robustness has been validated through extensive testing on the CICIDS2017 and NF-BoT-IoT-v2 datasets, confirming their effectiveness across diverse network environments and attack scenarios. Designed with real-world deployment in mind, these models minimize false positives and negatives, ensuring their applicability in live network settings. Moreover, a focus on comprehensive evaluation metrics beyond accuracy provides a holistic assessment of performance, addressing potential challenges in detection reliability and practical application.

3. Methodology

The Transformer–DNN and Autoencoder–CNN architectures were selected after evaluating the limitations of traditional IDS approaches and through an iterative process of testing and evaluation. The Transformer–DNN leverages the transformer's self-attention mechanism to capture long-range contextual relationships in sequential data, and the DNN transforms these contextual features into accurate classifications. On the other hand, the Autoencoder–CNN preprocesses network traffic data by reshaping and denoising through the Autoencoder, enabling the CNN to classify enhanced features with precision. These models demonstrated superior performance in handling imbalanced datasets during preliminary evaluations. To further address class imbalance, the proposed models leverage an enhanced hybrid ADASYN-SMOTE for oversampling in binary classification and enhanced SMOTE for oversampling in multi-class classification. Additionally, ENN is applied for undersampling to ensure the model learns from more challenging examples by removing noisy or borderline instances, thereby maintaining a balanced class distribution.

This section provides a detailed overview of the model's steps, including the extensive preprocessing procedures applied to the CICIDS2017 dataset, followed by an evaluation of its performance on both the CICIDS2017 and NF-BoT-IoT-v2 datasets. Figure 1 provides a visual representation of the proposed architecture, showcasing its application for both binary and multi-class classification tasks on the CICIDS2017 dataset.

3.1. Dataset Description

Certain aspects, such data structure and labeling, are critical for intrusion detection in network-based datasets. Markus et al. [37] provide a comprehensive explanation of these aspects for both supervised and unsupervised intrusion detection approaches. The history of the CICIDS2017 dataset, which served as this study's intrusion detection dataset, is covered in this section. The Canadian institute for cybersecurity (CIC) provide a dataset that is freely accessible to all scholars [38]. One of the most recent datasets for network intrusion detection accessible in the literature includes 2,830,743 records with 79 network traffic variables and 15 classes [12]. The dataset consists of eight files with five-day benign and attack activities, each including a collection of real-world data [38]. Records with extra metadata are mostly in packet-based and bifacial flow-based formats [37]. The dataset is completely labelled. The focus of the dataset is on the classes listed in Table 3, because its generation is intended for use in network intrusion detection. All attack types are

categorized as '1' in binary classification, whereas benign assaults are classified as '0'. All assault types were taken into consideration for multi-classification as presented.

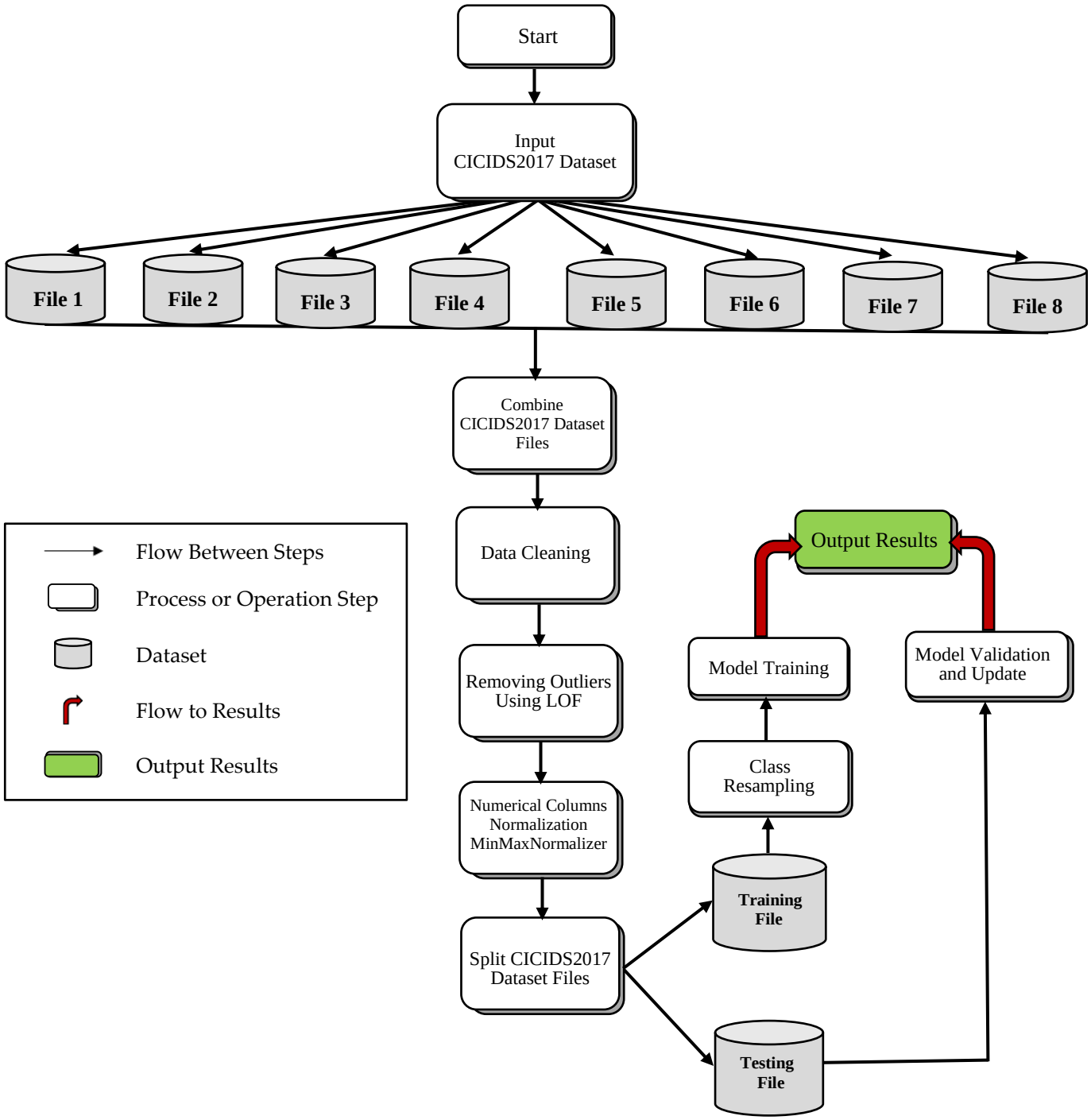

Figure 1. Designed architecture using the CICIDS2017 dataset for binary and multi-class classification.

The seven common attack families making up the assaults included the botnet, DoS, DDoS, brute force, web, infiltration, and Heartbleed attacks. Passwords can be cracked, secret information can be found, and attacks may be attempted via brute force attacks. Botnet assaults use internet-connected gadgets to launch spam attacks. DoS attacks overwhelm the system networks and cause the system to become unavailable for a while. DDoS attacks happen when several systems become targets. Web assaults are software programs designed to search for weaknesses in the user system and attack it. A backdoor can be built in an infiltration assault in order to launch attacks on the system once the user system has

been exploited. A Heartbleed attack operates by tricking servers into giving up their private encryption key, which then leaks their data. Figure 2 shows the CICIDS2017 dataset's sample distribution for each class. In the two-stage process, the first binary classification stage is followed by multi-class classification, where removing the normal traffic samples enabling the model to focus on distinguishing between different attack types.

Table 3. Dataset and Class Types [16].

File Names (.csv File Format) + Activity Day	Instances	Classes
Friday Working Hours Afternoon DDos	225,745	Benign, DDoS
Friday Working Hours Afternoon PortScan	286,467	Benign, Port Scan
Friday Working Hours Morning	191,033	Benign, Bot
Monday Working Hours	529,918	Benign
Thursday Working Hours Afternoon Infiltration	288,602	Benign, Infiltration
Thursday Working Hours Morning WebAttacks	170,366	Benign, Web Attack Brute Force, Web Attack Sql Injection, Web Attack XSS
Tuesday Working Hours	445,909	Benign, FTP Patator, SSH Patator
Wednesday Working Hours	692,703	Benign, DoS-GoldenEye, DoS-Hulk, DoS Slowhttptest, DoS-Slowloris, Heartbleed

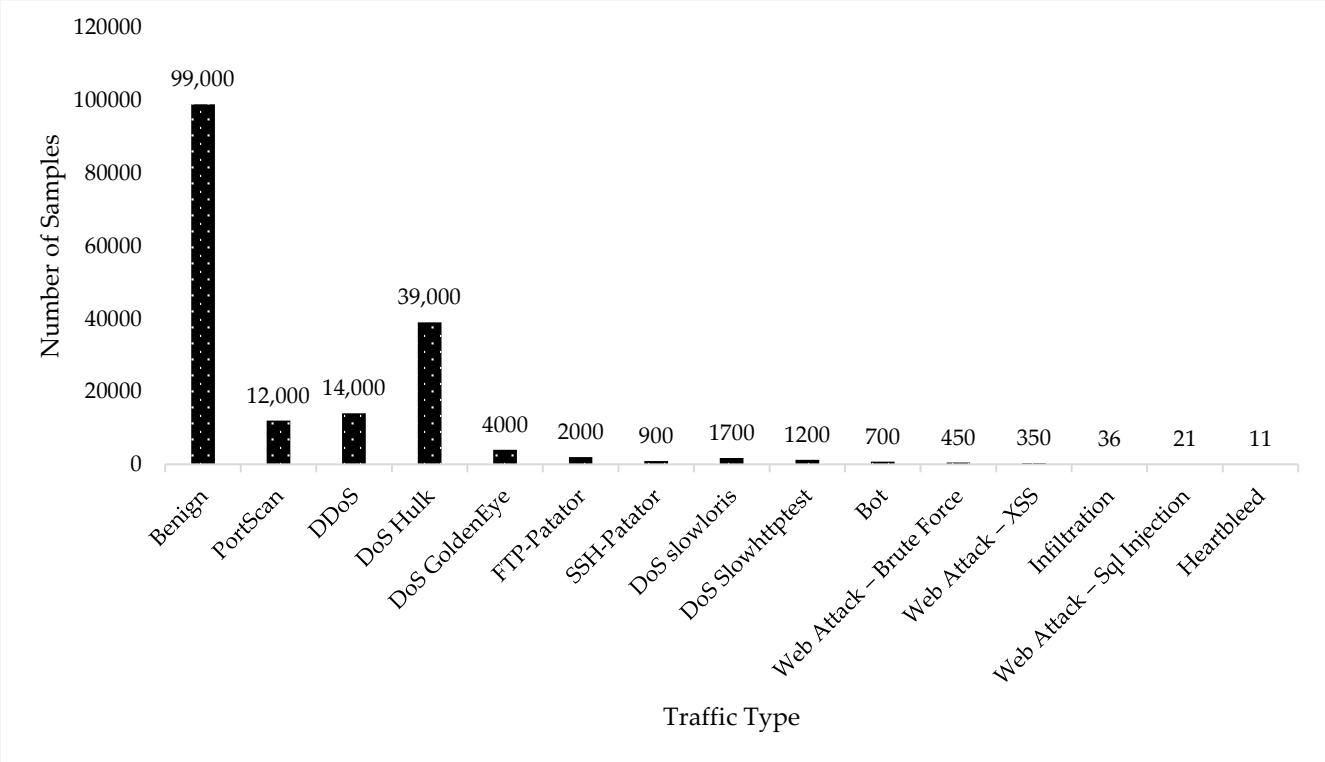

Figure 2. Sample distribution of the CICIDS2017 dataset for each class.

3.2. Data Preprocessing

The CICIDS2017 dataset, while comprehensive, contains missing or NaN (not a number) values that require careful handling during preprocessing to ensure high-quality data. The preprocessing began with merging all eight constituent files into a unified dataset before addressing any missing values. Next, duplicate entries were removed, and columns containing only a single unique value were discarded. The remaining NaN values were imputed and leading spaces stripped from feature names for consistency. After these cleaning steps, samples were taken from the dataset. In the case of multi-class classification, the Normal class was removed after sampling. Outliers were then removed using LOF to

ensure the dataset was free from extreme values that could have distorted model performance. Then the data were split into input and output variables. Numerical features were then normalized using MinMaxScaler 1.2.2 to ensure consistent scaling across the inputs. Following these preprocessing tasks, the dataset was split into training and testing sets. To address class imbalance, the enhanced hybrid ADASYN-SMOTE technique was applied for oversampling in binary classification and enhanced SMOTE was used for oversampling in multi-class classification to generate synthetic samples within the training set. Moreover, ENN was employed for undersampling the training data, ensuring a balanced class distribution for both binary and multi-class tasks. The preprocessing steps effectively cleaned, balanced, and optimized the dataset, ensuring it was well-prepared for model training and positioned to deliver high-quality and dependable results.

3.2.1. Data Cleaning

This section covers the critical preprocessing tasks of dropping duplicates, dropping columns with one unique value, filling NaN values, and removing leading space characters in the CICIDS2017 dataset. Dropping duplicates is an essential step for maintaining the integrity of the dataset. A total of 24 duplicate records were identified and removed for binary classification, and 12 duplicate records were removed for multi-class classification to ensure that each entry was unique, avoiding bias and overfitting, to enhance the reliability and performance of the machine learning models. Dropping columns with one unique value helped to reduce the dimensionality of the dataset by removing columns that provided no meaningful variance or discriminative power. Columns such as Bwd PSH Flags, Bwd URG Flags, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, and Bwd Avg Bulk Rate, all of which contained only a single unique value, were removed. This step ensured that the dataset included only features that contributed to the learning process, improving model efficiency and predictive accuracy. Filling the NaN values was another critical preprocessing step. A total of 1358 missing values were imputed with the column's mean value of 1358. This imputation ensured that the dataset was complete and improved the model's reliability by addressing gaps in the data. Lastly, removing leading space characters from feature names was necessary to ensure consistency in the dataset. A total of 59 feature names contained leading spaces for binary classification, and 46 feature names contained leading spaces for multi-class classification, including columns such as Destination Port, Flow Duration, and Total Fwd Packets. Removing these spaces improved the dataset's structure, making it easier to handle programmatically and reducing the likelihood of errors. These data cleaning steps collectively ensured that the CICIDS2017 dataset was robust, consistent, and ready for machine learning tasks.

3.2.2. Removing Outliers Using LOF

The LOF method was utilized to identify and remove outliers from the dataset, thereby improving the quality of the data used for model training. LOF works by detecting data points with significantly lower density than their surrounding neighbors, making it particularly effective for datasets with uneven density distributions. In this implementation, the LOF algorithm was set with n_neighbors at 20 and a contamination rate of 0.05, suggesting that approximately 5% of the data was expected to be outliers. Once the LOF model was fitted, each sample was categorized as either an outlier (labeled -1) or an inlier (labeled 1). Only the inliers were retained for further processing, ensuring the removal of extreme values that could have negatively impacted model performance. It is important to note that the LOF method is designed to remove data points that deviate significantly from the overall distribution of the dataset and does not target or remove cases of anomalous

behavior such as attacks, which exhibit identifiable characteristics distinct from normal traffic. Therefore, removing outliers does not interfere with the detection of anomalous behavior but ensures the dataset is free from extreme, irrelevant values that could distort model training. This outlier detection process was applied to the features of the dataset, excluding the labels. The resulting dataset, now free of outliers, was used in the subsequent stages of model training and evaluation, enhancing the performance and robustness of the classification models for both binary and multi-class tasks without compromising the detection of anomalous behavior.

The LOF score, as shown in Equation (1) [39], was used to identify outliers by comparing the local density of a data point to the densities of its neighbors. The score was calculated by measuring the local reachability density (lrd) of a point X and its neighbors. A point with a significantly lower LOF score than its neighbors is considered an outlier, indicating that it resides in a region of much lower density. This method helped to isolate data points that deviated from the overall distribution of the dataset, making it useful for outlier detection.

$$LOF(K.X) = \frac{lrd(X)}{\frac{1}{K}\sum_{i \epsilon N_K(X)} lrd(i)} \quad (1)$$

where LOF$(K.X)$ is the local outlier factor for the point, lrd(X) represents the local reachability density of the point X, and $N_K(X)$ denotes the set of k-nearest neighbors to point X. The LOF score compares the local density of point X to the density of its neighbors, helping to identify outliers by highlighting points that are less densely populated than their neighbors.

The sample distribution of the CICIDS2017 dataset for binary classification, before and after the application of the LOF method, is detailed in Table 4. The LOF technique significantly impacted the dataset, particularly by reducing the number of samples in both the Normal and Attack classes. Specifically, the Normal class, which initially contained 99,000 samples, was reduced to 93,053 samples following outlier removal. Similarly, the Attack class, which had 76,368 samples before applying LOF, saw a reduction to 73,546 samples after the outlier removal process. Overall, the application of LOF resulted in a cleaner dataset, removing extreme values and enhancing the balance and reliability of the data for the binary classification model.

Table 4. Sample distribution of CICIDS2017 dataset for binary classification using LOF.

Class Type	Number of Samples Before LOF	Number of Samples After LOF
Normal	99,000	93,053
Attack	76,368	73,546

The sample distribution of the CICIDS2017 dataset for multi-class classification before and after the application of the LOF method is shown in Table 5. The LOF method had a noticeable effect on the dataset, particularly in reducing the number of samples in various classes. For instance, the PortScan class, initially containing 12,000 samples, was reduced to 11,892 samples after outlier removal. Similarly, the DDoS class decreased from 14,000 to 13,544, while the number of samples in the DoS Hulk class dropped from 39,000 to 36,640. The DoS GoldenEye class saw a reduction from 4000 to 3554, and the FTP-Patator class decreased in number from 2000 to 1964. Classes such as SSH-Patator and DoS Slowloris also experienced reductions, from 900 to 868 and from 1700 to 1540, respectively. Proportionally, smaller classes like Infiltration and Web Attack—Sql Injection saw more significant reductions, from 36 to 15 and from 21 to 9, respectively. The Heartbleed class remained relatively unchanged, with a slight reduction from 11 to 10 samples. Overall, the

LOF method helped create a more balanced dataset by removing outliers, improving the reliability and performance of the classification models for multi-class tasks.

Table 5. Sample distribution of CICIDS2017 dataset for multi-class classification using LOF.

Class Type	Number of Samples Before LOF	Number of Samples After LOF
PortScan	12,000	11,892
DDoS	14,000	13,544
DoS Hulk	39,000	36,640
DoS GoldenEye	4000	3554
FTP-Patator	2000	1964
SSH-Patator	900	868
DoS Slowloris	1700	1540
DoS Slowhttptest	1200	1099
Bot	700	638
Web Attack—Brute Force	450	436
Web Attack—XSS	350	340
Infiltration	36	15
Web Attack—Sql Injection	21	9
Heartbleed	11	10

3.2.3. Normalization

Data scaling involves adjusting numerical values to fit within a specific range and is a critical preprocessing step in machine and deep learning. This process enhanced the efficiency and effectiveness of the training models by standardizing the data across all columns. One commonly used method for normalization is the MinMaxScaler, a popular tool available in the Scikit-learn library. Equation (2) [40] illustrates the procedure where each column is processed by subtracting its minimum value and then dividing by the range (maximum value minus minimum value) to normalize it. In Equation (2) [40], X represents the original value, $\min(X)$ is the minimum value in the column, and $\max(X)$ is the maximum value in the column. After evaluating several normalization techniques, MinMaxScaler was identified as the optimal choice for achieving the best results in our study. This normalization approach was applied to all features in the dataset to ensure consistent scaling across the entire dataset.

$$X(scaled) = \frac{X - min(x)}{max(x) - min(x)} \quad (2)$$

3.2.4. Train–Test Dataset Split

The dataset was split into training and testing sets, allowing the model to learn patterns from the training data and ensuring unbiased evaluation of the testing data for generalization in both binary and multi-class scenarios. The sample distribution of the CICIDS2017 dataset for binary classification is presented in Table 6, showcasing the allocation of samples between the training and testing sets. From the normal class, a total of 79,112 samples were designated for training, with 13,941 samples reserved for testing. The attack class consisted of 62,497 training samples and 11,049 test samples. This distribution ensured that both classes were adequately represented in both the training and testing phases, contributing to a balanced and effective evaluation of the model's performance.

Table 6. Sample distribution of the CICIDS2017 dataset for binary classification.

Class Type	Train	Test
Normal	79,112	13,941
Attack	62,497	11,049

The CICIDS2017 dataset for multi-class classification consisted of multiple attack classes, including PortScan, DDoS, DoS Hulk, DoS GoldenEye, FTP-Patator, SSH-Patator, DoS Slowloris, DoS Slowhttptest, Bot, Web Attack—Brute Force, Web Attack—XSS, Infiltration, Web Attack—Sql Injection, and Heartbleed. The largest class, DoS Hulk, included 31,172 training samples and 5468 test samples, followed by DDoS with 11,519 training samples and 2025 test samples. Other significant classes included DoS GoldenEye, with 3018 training samples and 536 test samples, and FTP-Patator, which contained 1658 training samples and 306 test samples. Smaller classes such as Infiltration and Heartbleed were represented by 13 and 9 training samples, respectively, with even fewer test samples, 2 and 1, respectively. This distribution highlights the varying sample sizes across different classes, with some attack types being more prevalent than others, while rarer attack types like Infiltration and Heartbleed are less frequent, making them critical for testing the model's ability to detect less common attacks. This variation in class size ensured a comprehensive evaluation of the model's performance across both major and minor attack types, as shown in Table 7.

Table 7. Sample distribution of the CICIDS2017 dataset for multi-class classification.

Class Type	Train	Test
PortScan	10,080	1812
DDoS	11,519	2025
DoS Hulk	31,172	5468
DoS GoldenEye	3018	536
FTP-Patator	1658	306
SSH-Patator	735	133
DoS Slowloris	1312	228
DoS Slowhttptest	943	156
Bot	539	99
Web Attack—Brute Force	373	63
Web Attack—XSS	288	52
Infiltration	13	2
Web Attack—Sql Injection	7	2
Heartbleed	9	1

3.2.5. Class Balancing

Class imbalance is a prominent issue in the CICIDS2017 dataset and can impact the effectiveness of machine learning models. To address this, a comprehensive class-balancing strategy was employed, combining both oversampling and undersampling techniques, a widely recognized solution to tackle class imbalance [41]. For binary classification, the enhanced hybrid ADASYN-SMOTE technique was utilized during training to oversample the minority class, while for multi-class classification, enhanced SMOTE was applied during training to generate synthetic samples and balance the class distribution. For both tasks, ENN was employed during training for undersampling, removing noisy or redundant instances from the majority class to improve data quality. This balanced approach enhanced the model's ability to accurately identify and classify minority classes, improving overall

performance and reliability. However, the accuracy paradox may be encountered, where a model achieves high accuracy but fails to effectively predict minority class instances [42]. To mitigate this, an enhanced method combining oversampling techniques such as hybrid ADASYN-SMOTE for binary classification and SMOTE for multi-class classification with undersampling using ENN during model training was implemented, based on previous research [43]. This integrated approach effectively addressed class imbalance, leading to more reliable predictions across all classes.

1. Hybrid ADASYN-SMOTE

In this work, hybrid ADASYN-SMOTE was applied for the binary classification task to address class imbalance effectively. This technique combined the strengths of both ADASYN and SMOTE in a two-stage process. First, ADASYN was used to generate synthetic samples for the minority class, focusing on difficult-to-learn instances. The output from ADASYN then served as the input for the second stage, where SMOTE further enhanced the minority class by creating additional synthetic samples along the line segments between existing instances. This hybrid approach not only improved the class balance but also strengthened the model's ability to learn from challenging examples, resulting in better performance in binary classification tasks.

Let X_i denote the samples from the minority class and $N(X_i, K)$ represent the k-nearest neighbors of X_i. The number of synthetic samples n_i to be generated for each minority sample is determined as outlined in Equation (3) [44].

$$n_i = \frac{N_{Maj} - N_{Min}}{N_{Min}} \times (1 - \frac{N_i}{k}) \tag{3}$$

where N_{Maj} and N_{Min} represent the sample sizes of the majority and minority classes, respectively, emphasizing the class imbalance. The term N_i refers to the number of minority class samples within the radius defined by the k-nearest neighbors, to assist in identifying minority instances located near decision boundaries where synthetic samples are typically generated, thereby enhancing the model's performance.

For each minority instance X_i, synthetic samples are generated using Equation (4) [44].

$$X_{syn} = X_i + \gamma \times (X_j - X_i) \tag{4}$$

where X_{syn} represents the synthetic sample generated to balance the class distribution, X_j is a randomly chosen neighbor from the k-nearest neighbors of X_i, the minority sample, and the term γ is a random value between 0 and 1, ensuring that the synthetic sample is created along the line segment connecting X_i and X_j.

Following the ADASYN step, SMOTE is applied. When a sample X_I from the minority class is provided, SMOTE generates a synthetic sample by selecting one of its k-nearest neighbors, X_{zi}. The new synthetic sample X_{new} is then created by interpolating between X_I and X_{zi}, as described in Equation (5) [45].

$$X_{new} = X_I + \sigma \times (X_{zi} - X_I) \tag{5}$$

where σ is a random number between 0 and 1. This formula ensures that the generated sample X_{new} lies on the line segment connecting X_I and X_{zi}, effectively creating a synthetic data point that better represents the minority class within the feature space. By emphasizing the feature relationships within the minority class, SMOTE helps balance the dataset, thereby improving model performance in binary classification tasks.

An initial imbalance existed in the binary classification of the CICIDS2017 dataset, with 79,112 samples in the Normal class and 62,497 in the Attack class. To address this, the hybrid

ADASYN-SMOTE technique was applied to generate synthetic samples for the minority Attack class, first increasing its count to 79,296 with ADASYN and then balancing both classes to 79,296 using SMOTE, as shown in Table 8. This balanced distribution ensured more reliable training and evaluation by providing equal representation for both classes.

Table 8. Sample distribution in each training class before/after resampling using hybrid ADASYN-SMOTE for binary classification, on the CICIDS2017 dataset.

Type of Class	Number of Samples Before Resampling (ADASYN-SMOTE)	Number of Samples After Resampling (ADASYN-SMOTE)
Normal	79,112	79,296
Attack	62,497	79,296

2. SMOTE

In this work, cascaded SMOTE is applied to multi-class classification tasks to address class imbalance effectively. This method involved applying SMOTE in two stages, where the output of the first SMOTE served as the input for the second. By progressively oversampling the minority classes in each stage, this technique helped to better balance the class distribution. The enhanced representation of the minority classes improved the model's ability to capture and learn patterns across all classes, resulting in improved accuracy and reliability in multi-class classification performance.

When given a sample X_I from the minority class, SMOTE creates a synthetic sample by selecting one of its k-nearest neighbors X_{zi}. The new synthetic sample X_{new} is then created by interpolating between X_I and X_{zi}, as shown in Equation (6) [45].

$$X_{new} = X_I + \sigma \times (X_{zi} - X_I) \quad (6)$$

where σ is a random number between 0 and 1. This formula ensures that the new sample X_{new} lies along the line segment between X_I and X_{zi}, effectively creating a synthetic data point that represents the minority class more accurately within the feature space. By focusing on the feature relationships within the minority class, SMOTE balances the dataset, enhancing model performance in multi-class classification tasks.

The sample distribution for the multi-class classification task in the CICIDS2017 dataset, before and after resampling with SMOTE, is illustrated in Table 9. Initially, the dataset exhibited a considerable imbalance, with certain classes such as Web Attack—Sql Injection and Heartbleed containing as few as 7 and 9 samples, respectively. This imbalance could have led to model bias, as the minority classes would have been underrepresented during training. After applying SMOTE, the sample sizes for these minority classes were significantly increased, with classes like Web Attack—Sql Injection and Heartbleed being resampled to 31,172 instances, ensuring a more balanced representation across all classes. This resampling approach mitigated the risks of class imbalance, allowing more equitable model training and improving overall classification performance by reducing bias towards the majority classes. The enhanced dataset facilitated more robust and reliable predictions for both the majority and minority classes in the multi-class classification tasks.

Table 9. Sample distribution in each training class before/after resampling using SMOTE for multi-class classification, on the CICIDS2017 dataset.

Type of Class	Number of Samples Before Resampling (SMOTE)	Number of Samples After Resampling (SMOTE)
PortScan	10,080	10,080
DDoS	11,519	11,519
DoS Hulk	31,172	31,172

Table 9. *Cont.*

Type of Class	Number of Samples Before Resampling (SMOTE)	Number of Samples After Resampling (SMOTE)
DoS GoldenEye	3018	3018
FTP-Patator	1658	1658
SSH-Patator	735	735
DoS Slowloris	1312	1312
DoS Slowhttptest	943	943
Bot	539	539
Web Attack—Brute Force	373	373
Web Attack—XSS	288	288
Infiltration	13	13
Web Attack—Sql Injection	7	31,172
Heartbleed	9	31,172

3. ENN

ENN is an advanced data preprocessing method designed to enhance the quality of training datasets by removing noisy and misclassified instances, ultimately refining class boundaries. This technique operates by evaluating the nearest neighbors of each data point and discarding those that are incorrectly classified, thereby improving the clarity and distinction between classes. Through this process, ENN effectively reduces class overlap, promoting a more balanced representation across the dataset. This is particularly beneficial for both binary and multi-class classification tasks, as it ensures that the training data are cleaner and more accurately reflect the underlying patterns. When integrated into the preprocessing pipeline, ENN aids in achieving better model generalization, leading to enhanced performance when applied to new, unseen data [46].

For each instance X_i in the dataset, its k-nearest neighbors are identified using the formula in Equation (7) [46].

$$N(X_i) = \left\{ X_{j1},\ X_{j2}, \ldots\ldots,\ X_{jk} \right\} \tag{7}$$

where X_{jk} are the nearest neighbors of X_i in terms of a distance metric (e.g., Euclidean distance).

To determine the majority class among the nearest neighbors, the approach outlined in Equation (8) [46] was used. This process involved examining the class labels of the nearest neighbors and identifying the class that appeared most often. By leveraging this method, the predicted class for an instance can be assigned based on the most frequent class within its neighborhood, thus improving the model's classification accuracy by ensuring it reflects the dominant pattern in the surrounding data points.

$$C(X_i) = argmax_c \left(\sum_{j=1}^{k} \prod (y_j = c) \right) \tag{8}$$

where $C(X_i)$ denotes the predicted class for instance X_i, with y_j representing the class label of its j-th neighbor. The indicator function $\prod$ outputs 1 if the condition holds true, otherwise returning 0.

An instance X_i is discarded if the predicted class $C(X_i)$ differs from its true class label y_j, as defined by the formula in Equation (9) [46]. This step helps in eliminating misclassified instances, thereby improving the quality of the dataset by retaining only those instances whose predicted and actual classes align.

$$\text{If } C(X_i) \neq y_j, \text{ then remove } X_i \tag{9}$$

The sample distribution for the binary classification task on the CICIDS2017 dataset, before and after applying ENN for resampling, is shown in Table 10. Before resampling using ENN, both the Normal and Attack classes contained an equal number of 79,296 samples. However, after applying ENN, a slight reduction in the Attack class was observed, with its sample count decreasing to 79,067. This reduction occurred as ENN removed misclassified instances from the Attack class, helping to eliminate noise and improve the quality of the training data. The resulting dataset maintained a nearly balanced distribution while enhancing the clarity of class boundaries and contributing to better model performance and generalization.

Table 10. Sample distribution in each training class before/after resampling using ENN for binary classification, on the CICIDS2017 dataset.

Type of Class	Number of Samples Before Resampling (ENN)	Number of Samples After Resampling (ENN)
Normal	79,296	79,296
Attack	79,296	79,067

The sample distribution for the multi-class classification task on the CICIDS2017 dataset, before and after applying ENN for resampling, is shown in Table 11. Prior to resampling, there was some variation in the numbers of samples across the different classes, with some classes having significantly fewer instances than others. After applying ENN, minor adjustments were made to the sample sizes of several classes. For example, the PortScan class underwent a reduction from 10,080 to 10,076 samples, and the DDoS class decreased from 11,519 to 11,474. Similarly, other classes such as DoS Hulk and DoS Slowhttptest experienced slight reductions in their sample sizes. These changes were the result of ENN removing misclassified instances, which helped to reduce noise and improve the quality of the training data. Despite these adjustments, the dataset remained largely balanced, with no significant loss of samples in the majority of classes. The application of ENN resulted in a cleaner, more accurate dataset, which was expected to improve the model's ability to distinguish between classes and enhance overall classification performance.

Table 11. Sample distribution in each training class before/after resampling using ENN for multi-class classification, on the CICIDS2017 dataset.

Type of Class	Number of Samples Before Resampling (ENN)	Number of Samples After Resampling (ENN)
PortScan	10,080	10,076
DDoS	11,519	11,474
DoS Hulk	31,172	31,147
DoS GoldenEye	3018	3017
FTP-Patator	1658	1651
SSH-Patator	735	733
DoS Slowloris	1312	1310
DoS Slowhttptest	943	927
Bot	539	537
Web Attack—Brute Force	373	373
Web Attack—XSS	288	286

Table 11. *Cont.*

Type of Class	Number of Samples Before Resampling (ENN)	Number of Samples After Resampling (ENN)
Infiltration	13	13
Web Attack—Sql Injection	31,172	31,172
Heartbleed	31,172	31,172

3.3. Architectures of Models

In this study, a diverse range of model architectures were employed, including CNN, Autoencoder, DNN, Autoencoder–CNN, and Transformer–DNN. These models were chosen based on their superior performance across various metrics [47–49].

3.3.1. Convolutional Neural Networks (CNNs)

The proposed model architecture combines CNN and MLP to tackle both binary and multi-class classification tasks. It begins with an input layer that accepts sequential data formatted as a one-dimensional array. The first CNN block performs convolution operations with ReLU activation, followed by batch normalization, max pooling, and dropout to extract and regularize key features in the data for binary classification. For multi-class classification, the initial CNN block applies convolution operations, followed by batch normalization, an activation function, max pooling, and dropout to extract and regularize important features in the data. This sequence is repeated in subsequent CNN layers with varying kernel sizes and pool sizes, enabling the model to capture diverse patterns and relationships within the input data. The different kernel sizes allow the model to detect features at multiple scales along the sequence, while the varying pool sizes help in downsampling and reducing the sequence's spatial dimensions, enhancing the model's ability to generalize across different input features. After passing through the CNN layers, the output is flattened and processed through an MLP block designed for binary classification. This block comprises a dense layer with an activation function, followed by batch normalization and dropout, to enhance the model's capability to learn intricate patterns while minimizing overfitting. Similarly, for multi-classification tasks, the output is flattened and passed through an MLP block that includes a dense layer, batch normalization, activation function, and dropout, effectively strengthening the model's ability to represent complex features and reducing the risk of overfitting. The features extracted from both the CNN and MLP components are concatenated to form a unified feature set. The final output layer utilizes a sigmoid activation function for binary classification or a softmax activation function for multi-class classification, paired with binary cross-entropy or categorical cross-entropy loss functions, respectively, to tailor the model's optimization process for the specific classification objective. The model is trained using the Adam optimizer, ensuring efficient learning and convergence throughout the training process.

The convolution operation within the CNN layers is fundamental for feature extraction, as it applies filters to the input data. This process involves sliding the convolution kernel over the input feature map and computing the dot product at each position, resulting in a new feature map. The mathematical expression for the convolution operation is represented in Equation (10) [50].

$$Z_{i,j} = (X * K)_{i,j} = \sum_{m} \sum_{n} Z_{i+m,j+n} k_{m,n} \quad (10)$$

where Z denotes the output feature map and X represents the input feature map. The convolution kernel, denoted by K, is applied during the convolution operation to convert the input features into the output features.

The ReLU activation function depicted in Equation (11) [51] is a straightforward yet effective non-linear transformation. It outputs zero for negative inputs, preserves positive values, helps address the vanishing gradient problem, and encourages sparse activations, all of which improve training efficiency and model performance.

$$ReLU(x) = max(0, x) \tag{11}$$

The max pooling operation simplifies the input feature map by reducing its spatial dimensions while preserving the most relevant features. This is achieved by selecting the maximum value within a defined pooling window, thus downsampling the data. The process is mathematically represented in Equation (12) [50].

$$P_{i,j} = max\left(X_{i:i+p,\ j:j+q}\right) \tag{12}$$

where P represents the output resulting from the pooling process, while p and q denote the dimensions of the pooling window applied to aggregate the input features into the pooled output.

The dropout layer introduces regularization by randomly disabling a proportion p of input units during training. This process reduces overfitting by preventing the model from becoming overly dependent on specific input features, thereby promoting better generalization. The effect of this regularization technique is mathematically represented in Equation (13) [52].

$$Dropout(x) = \begin{cases} x \text{ with probability } 1 - p \\ 0 \text{ with probability } p \end{cases} \tag{13}$$

For binary classification tasks, the output layer applies the sigmoid function to generate a probability score reflecting the likelihood that a given instance belongs to the positive class. This function outputs a value between 0 and 1, with values closer to 1 suggesting a higher probability of the instance being classified as positive. The mathematical expression for this operation is provided in Equation (14) [53].

$$\sigma(Z) = \frac{1}{1 + e^{-z}} \tag{14}$$

where Z represents the output generated by the final dense layer in the network.

The output layer utilizes the softmax function for multi-class classification, transforming the raw model outputs into a probability distribution over all possible classes. This function ensures that the sum of the probabilities across all classes equals one, providing a clear indication of the model's confidence in each class. The mathematical formulation of this process is presented in Equation (15) [53].

$$Softmax(Z_i) = \frac{e^{z_i}}{\sum_j e^{z_i}} \tag{15}$$

where Z_i represents the output generated by the final dense layer in the network for class i.

(i) Binary Classification

The CNN model architecture for binary classification applied to the CICIDS2017 and NF-BoT-IoT-v2 datasets, as outlined in Table 12, began with an input layer tailored to each dataset. For the CICIDS2017 dataset, the input layer consisted of 69 neurons, while the NF-BoT-IoT-v2 dataset used an input layer with 18 neurons. The model then progressed through the three hidden blocks. The first hidden block featured a 1D CNN layer with 256 filters and a ReLU activation function, followed by a 1D max pooling layer with a pool size of 2 and a dropout layer with a minimal rate of 0.00000001 to mitigate overfitting. The second hidden block mirrored the first, incorporating a 1D CNN layer with 256 filters and ReLU activation, followed by a 1D max pooling layer with a pool size of 4, and another

dropout layer with the same rate. The third hidden block included a dense layer with 256 neurons and ReLU activation, followed by a dropout layer with a rate of 0.00000001. Finally, the output block consisted of a single neuron in the output layer, activated by the sigmoid function, to generate the binary classification result. This architecture was optimized to process the distinct feature sets of both datasets effectively.

Table 12. CNN model layers for binary classification.

Dataset	Block	Layers	Layer Size	Activation
CICIDS2017	Input block	Input layer	69	-
NF-BoT-IoT-v2	Input block	Input layer	18	-
Shared structure	Hidden block 1	1D CNN layer	256	ReLU
		1D max pooling layer	2	-
		Dropout layer	0.00000001	-
	Hidden block 2	1D CNN layer	256	ReLU
		1D max pooling layer	4	-
		Dropout layer	0.00000001	-
	Hidden block 3	Dense layer	256	ReLU
		Dropout layer	0.00000001	-
	Output block	Output layer	1	Sigmoid

(ii) Multi-Class Classification

The CNN model architecture for multi-class classification applied to the CICIDS2017 and NF-BoT-IoT-v2 datasets, as detailed in Table 13, began with dataset-specific input layers, using 69 neurons for CICIDS2017 and 28 neurons for NF-BoT-IoT-v2. The model progressed through five shared hidden blocks. The first hidden block included a 1D CNN layer with 256 filters, followed by a ReLU activation, a 1D max pooling layer with a pool size of 2 and a dropout layer with a minimal rate of 0.0000001. The second hidden block mirrored this setup with another 1D CNN layer of 256 filters, ReLU activation, a max pooling layer of size 4, and a dropout layer with the same rate. The third hidden block followed a similar structure, featuring a 1D CNN layer with 256 filters, ReLU activation, a max pooling layer of size 8, and a dropout layer with a rate of 0.0000001. The fourth block included a dense layer with 1024 neurons, followed by a ReLU activation and a dropout layer with the same minimal rate. The fifth block consisted of a dense layer with 768 neurons using the softmax activation function, accompanied by a dropout layer with a rate of 0.0000001. The model concluded with an output layer of 14 neurons with softmax for the CICIDS2017 dataset and 4 neurons with softmax activation for the NF-BoT-IoT-v2 dataset, designed to handle multi-class classification tasks.

(iii) Hyperparameter Configuration for the CNN Model

The hyperparameters for the CNN models, as detailed in Table 14, were configured for both binary and multi-class classification tasks. For both types of classifiers, a batch size of 128 was used. The learning rate for both binary and multi-class classifiers was dynamically adjusted using the ReduceLROnPlateau schedule. If the validation loss did not improve for two consecutive epochs, the learning rate was reduced by a factor of 0.5. This approach ensured that the learning rate decreased gradually, improving convergence as training progresses. The minimum value for the learning rate was capped at 1×10^{-5}, preventing the learning rate from becoming too small to effect meaningful updates. This method allowed more efficient training and helped the model to converge more reliably. The Adam optimizer was employed across both classifiers. The loss function varied by task.

Binary cross-entropy was used for binary classification, while categorical cross-entropy was applied for multi-class classification. Accuracy served as the evaluation metric for both types of classifiers.

Table 13. CNN model layers for multi-class classification.

Dataset	Block	Layers	Layer Size	Activation
CICIDS2017	Input block	Input layer	69	-
NF-BoT-IoT-v2	Input block	Input layer	28	-
Shared structure	Hidden block 1	1D CNN layer	256	ReLU
		1D max pooling layer	2	-
		Dropout layer	0.0000001	-
	Hidden block 2	1D CNN layer	256	ReLU
		1D max pooling layer	4	-
		Dropout layer	0.0000001	-
	Hidden block 3	1D CNN layer	256	ReLU
		1D max pooling layer	8	-
		Dropout layer	0.0000001	-
	Hidden block 4	Dense layer	1024	ReLU
		Dropout layer	0.0000001	-
	Hidden block 5	Dense layer	768	Softmax
		Dropout layer	0.0000001	-
CICIDS2017	Output block	Output layer	14	Softmax
NF-BoT-IoT-v2	Output block	Output layer	4	Softmax

Table 14. CNN model hyperparameters.

Parameter	Binary Classifier	Multi-Class Classifier
Batch size	128	128
Learning rate	Scheduled: Initial = 0.001, Factor = 0.5, Min = 1×10^{-5} (ReduceLROnPlateau)	Scheduled: Initial = 0.001, Factor = 0.5, Min = 1×10^{-5} (ReduceLROnPlateau)
Optimizer	Adam	Adam
Loss function	Binary cross-entropy	Categorical cross-entropy
Metric	Accuracy	Accuracy

3.3.2. Autoencoder (AE)

The Autoencoder model is designed to handle both binary and multi-class classification tasks, starting with an input layer that accepts feature vectors. The architecture includes an encoder composed of multiple dense layers that progressively reduce the dimensionality of the input while utilizing the ReLU activation function to extract significant features. Following the encoder, a classification layer is employed to predict class probabilities. For binary classification, the sigmoid activation function is applied, whereas for multi-class classification, the softmax activation function is used. This allows the model to generate the appropriate output probabilities for each classification type. The model is optimized using the Adam optimizer and applies binary cross-entropy loss for binary classification and categorical cross-entropy loss for multi-class classification, ensuring accurate loss computation for each classification task.

In the encoder layers, the dimensionality of the input data is progressively reduced, allowing the model to extract essential features through dense layers. This process of reducing dimensions and extracting features can be mathematically represented as shown in Equation (16) [50].

$$h^{(l)} = f\left(W^{(l)}\, a^{(l-1)} + b^{(l)}\right) \tag{16}$$

where $h^{(l)}$ represents the output of encoder layer l, with $a^{(l-1)}$ denoting the output of the preceding layer, which acts as the input to the first layer. The weight matrix for layer l is denoted by $W^{(l)}$ and the bias vector for that layer is given by $b^{(l)}$. In this context, the activation function represented by f is specifically the ReLU function.

In a typical Autoencoder, the decoder layer reconstructs the original input based on the compressed representation produced by the encoder. This reconstruction process is mathematically expressed in Equation (17) [50].

$$\acute{\alpha} = g\left(W^{(d)}\, h^{(l)} + b^{(d)}\right) \tag{17}$$

where $\acute{\alpha}$ represents the reconstructed output, with $h^{(l)}$ being the output from the final encoder layer. The weight matrix for the decoder layer is represented by $W^{(d)}$, and $b^{(d)}$ denotes the bias vector for the decoder layer. The activation function g, generally selected to be linear, is applied to facilitate reconstruction.

The classification layer applies a designated activation function to produce the binary classification output, as outlined in Equation (18) [50].

$$\overline{y} = \sigma\left(W^{(out)}\, h^{(l)} + b^{(out)}\right) \tag{18}$$

where $\overline{y}$ represents the predicted probability of the positive class and the output layer's weight matrix is indicated by $W^{(out)}$, with $b^{(out)}$ representing the bias term for this layer. The sigmoid function, σ, is used to map the output to a probability score ranging from 0 to 1.

For multi-class classification, the classification layer uses the softmax activation function, allowing the model to produce a probability distribution over multiple classes. This process is mathematically represented in Equation (19) [53].

$$\overline{y} = softmax\left(W^{(out)}\, h^{(l)} + b^{(out)}\right) \tag{19}$$

where $\overline{y}$ denotes the vector of predicted probabilities for each class and the output layer is defined by the weight matrix $W^{(out)}$ and bias $b^{(out)}$. The softmax function is applied to transform the logits into probabilities, ensuring that the predicted values sum to one across all classes.

(i) Binary Classification

The design of the Autoencoder model for binary classification, as detailed in Table 15, begins with an input layer of 69 neurons for CICIDS2017 and 18 neurons for NF-BoT-IoT-v2. The encoder block consists of three dense layers, the first with 128 neurons, the second with 64 neurons, and the third with 32 neurons, all using the ReLU activation function. The model concludes with an output block containing a single neuron activated by the sigmoid function, providing the binary classification output.

(ii) Multi-Class Classification

The architecture of the Autoencoder model for multi-class classification is in Table 16, starting with an input layer consisting of 69 neurons for the CICIDS2017 dataset and 28 neurons for the NF-BoT-IoT-v2 dataset. In the encoder block, the first dense layer has 128 neurons, followed by two additional dense layers with 64 and 32 neurons, all activated

by the ReLU function, to progressively compress the input data. The model concludes with an output layer consisting of 14 neurons for CICIDS2017 and 4 neurons for NF-BoT-IoT-v2, both activated by the softmax function, to handle the multi-class classification task.

Table 15. Autoencoder model layers for binary classification.

Dataset	Block	Layers	Layer Size	Activation
CICIDS2017	Input block	Input layer	69	-
NF-BoT-IoT-v2	Input block	Input layer	18	-
Shared structure	Encoder	Dense layer	128	ReLU
	Encoder	Dense layer	64	ReLU
	Encoder	Dense layer	32	ReLU
	Output block	Output layer	1	Sigmoid

Table 16. Autoencoder model layers for multi-class classification.

Dataset	Block	Layers	Layer Size	Activation
CICIDS2017	Input block	Input layer	69	-
NF-BoT-IoT-v2	Input block	Input layer	28	-
Shared structure	Encoder	Dense layer	128	ReLU
	Encoder	Dense layer	64	ReLU
	Encoder	Dense layer	32	ReLU
CICIDS2017	Output block	Output layer	14	Softmax
NF-BoT-IoT-v2	Output block	Output layer	4	Softmax

(iii) Hyperparameter Configuration for the Autoencoder Model

The hyperparameters for the Autoencoder model detailed in Table 17 are specified for both the binary and multi-class classification tasks. For both types of classifiers, a batch size of 128 was used, and the Adam optimizer was employed. The learning rate for both the binary and multi-class classification tasks was controlled using the ReduceLROnPlateau scheduling technique. The learning rate started with an initial value of 0.001, reduced by a factor of 0.5 whenever the validation loss did not improve for two consecutive epochs, as specified by the patience parameter. The learning rate does not decrease below a minimum value of 1×10^{-5}, ensuring that it remains within a reasonable range for effective training. This dynamic adjustment helps optimize model training by preventing the learning rate from becoming too small too quickly, which could hinder convergence. The loss function differed between the two tasks, with binary cross-entropy used for binary classification and categorical cross-entropy applied for multi-class classification. Accuracy was the metric utilized to assess model performance in both scenarios.

Table 17. Autoencoder model hyperparameters.

Parameter	Binary Classifier	Multi-Class Classifier
Batch size	128	128
Learning rate	Scheduled: Initial = 0.001, Factor = 0.5, Min = 1×10^{-5} (ReduceLROnPlateau)	Scheduled: Initial = 0.001, Factor = 0.5, Min = 1×10^{-5} (ReduceLROnPlateau)
Optimizer	Adam	Adam
Loss function	Binary cross-entropy	Categorical cross-entropy
Metric	Accuracy	Accuracy

3.3.3. Deep Neural Network (DNN)

The DNN model has been designed to handle both binary and multi-class classification tasks. It starts with an input block, consisting of a dense layer with ReLU activation, where the input dimension matches the number of features in the dataset. This layer introduces non-linearity, allowing the model to capture complex patterns in the data. The hidden layers are divided into two blocks. The first hidden block includes a dropout layer to prevent overfitting, followed by a dense layer with ReLU activation and then batch normalization to improve training stability and speed. The second hidden block mirrors this structure, adding another dropout layer and batch normalization. The output block is customized based on the classification task. For binary classification, it includes a dense layer with a single neuron and a sigmoid activation function to compute probabilities for the two classes. For multi-class classification, it features a dense layer with a number of neurons equal to the target classes, along with a softmax activation function to predict probabilities for each class. The model was compiled using the Adam optimizer, configured with an ExponentialDecay learning rate schedule to gradually reduce the learning rate during training. Binary cross-entropy was used as the loss function for binary classification, and categorical cross-entropy for multi-class classification. Accuracy served as the evaluation metric in both scenarios.

In a DNN, the feed-forward process involves transmitting the input through successive layers to generate the final output. This operation for layer l can be mathematically represented as shown in Equation (20) [54].

$$a^{(l)} = f\left(W^{(l)}\, a^{(l-1)} + b^{(l)}\right) \tag{20}$$

where $a^{(l)}$ represents the activation of the current layer l, the weight matrix for this layer is denoted by $W^{(l)}$, and $b^{(l)}$ stands for the bias vector of layer l. The activation function f is applied element-wise and can include non-linear functions such as ReLU or sigmoid, enabling the model to capture complex relationships.

The ReLU activation function presented in Equation (21) [55] is a straightforward and effective non-linear function. It outputs zero for any negative input values, encourages sparsity in activations, and facilitates smooth gradient flow, which makes it highly suitable for deep learning models.

$$ReLU(x) = max(0, \mathrm{x}) \tag{21}$$

For binary classification tasks, the sigmoid function is employed to produce a probability score that reflects the likelihood of an instance belonging to the positive class. It converts the raw score into a value between 0 and 1, acting as a threshold for making the classification decision. This process is mathematically expressed in Equation (22) [53].

$$\sigma(Z) = \frac{1}{1 + e^{-z}} \tag{22}$$

where Z represents the output from the final dense layer.

For multi-class classification tasks, the output layer uses the softmax function, allowing the model to generate probability distributions over multiple classes. This function transforms a vector of raw scores (logits) into normalized values within the range of 0 to 1, ensuring that the sum of all probabilities equals 1. The mathematical representation of the softmax function is provided in Equation (23) [53].

$$Softmax(Z_i) = \frac{e^{z_i}}{\sum_j e^{z_i}} \tag{23}$$

where Z_i represents the raw score for the i-th class.

(i) Binary Classification

The architecture outlined in Table 18 describes the structure of a DNN model tailored for binary classification using both the CICIDS2017 and NF-BoT-IoT-v2 datasets. The input layer processes the features of the datasets, with the CICIDS2017 dataset having 69 features and the NF-BoT-IoT-v2 dataset having 18 features. The input layer includes a dense layer with 1024 neurons, activated by ReLU, introducing non-linearity and enabling complex decision boundaries. To combat overfitting, the first hidden block includes a dropout layer with a minimal dropout rate of 0.0000001, followed by another dense layer with 768 neurons, activated by ReLU. This layer is enhanced through batch normalization to stabilize training via normalizing activations. The second hidden block mirrors the first, featuring an additional dropout layer with the same minimal rate and another batch normalization layer. The architecture concludes with a single-neuron output layer activated by the sigmoid function, producing a probability score representing the likelihood of the input belonging to the positive class. This design was optimized for the binary classification tasks, with separate input layers tailored to the CICIDS2017 and NF-BoT-IoT-v2 datasets.

Table 18. DNN model layers for binary classification.

Dataset	Block	Layers	Layer Size	Activation
CICIDS2017	Input block	Input layer	69	ReLU
NF-BoT-IoT-v2	Input block	Input layer	18	ReLU
Shared structure		Dense layer	1024	-
	Hidden block 1	Dropout layer	0.0000001	-
		Dense layer	768	ReLU
		Batch normalization	-	-
	Hidden block 2	Dropout layer	0.0000001	-
		Batch normalization	-	-
	Output block	Output layer	1	Sigmoid

(ii) Multi-Class Classification

The architecture outlined in Table 19 describes the structure of a DNN model tailored for multi-class classification on both the CICIDS2017 and NF-BoT-IoT-v2 datasets. The input layer processes the features of the datasets, with the CICIDS2017 dataset having 69 features and the NF-BoT-IoT-v2 dataset having 28 features. The input layer includes a dense layer with 1024 neurons, activated by ReLU, introducing non-linearity and enabling the model to capture more complex patterns. To combat overfitting, the first hidden block includes a dropout layer with a minimal dropout rate of 0.0000001, followed by another dense layer with 768 neurons, activated by ReLU. This layer is enhanced with batch normalization to stabilize training by normalizing activations. The second hidden block mirrors the first, featuring an additional dropout layer with the same minimal rate and another batch normalization layer. The architecture concludes with an output layer consisting of 14 neurons for the CICIDS2017 dataset and 4 neurons for the NF-BoT-IoT-v2 dataset, each activated by the softmax function. This configuration produces class probabilities, empowering the model to accurately handle multi-class classification tasks with high precision and reliability.

Table 19. DNN model layers for multi-class classification.

Dataset	Block	Layers	Layer Size	Activation
CICIDS2017	Input block	Input layer	69	ReLU
NF-BoT-IoT-v2	Input block	Input layer	28	ReLU
Shared structure		Dense layer	1024	-
	Hidden block 1	Dropout layer	0.0000001	-
		Dense layer	768	ReLU
		Batch normalization	-	-
	Hidden block 2	Dropout layer	0.0000001	-
		Batch normalization	-	-
CICIDS2017	Output block	Output layer	14	Softmax
NF-BoT-IoT-v2	Output block	Output layer	4	Softmax

(iii) Hyperparameter Configuration for the DNN Model

The hyperparameters for the DNN models, as shown in Table 20, are configured for both binary and multi-class classification tasks. Both classifiers use a batch size of 128 and the Adam optimizer. The learning rate for both the binary and multi-class classifiers is scheduled using Exponential Decay. The initial learning rate is set to 0.0003, with a decay factor of 0.9 and decay steps of 10,000. This exponential decay schedule reduces the learning rate progressively during training, helping to stabilize the optimization process and ensure more effective convergence over time. The loss function applied is binary cross-entropy for the binary classifier and categorical cross-entropy for the multi-class classifier. Accuracy is used as the evaluation metric for both classifiers.

Table 20. DNN model hyperparameters.

Parameter	Binary Classifier	Multi-Class Classifier
Batch size	128	128
Learning rate	Scheduled: Initial = 0.0003, Factor = 0.9, Decay Steps = 10,000 (Exponential Decay)	Scheduled: Initial = 0.0003, Factor = 0.9, Decay Steps = 10,000 (Exponential Decay)
Optimizer	Adam	Adam
Loss function	Binary cross-entropy	Categorical cross-entropy
Metric	Accuracy	Accuracy

3.3.4. Autoencoder–Convolutional Neural Network (Autoencoder–CNN)

The Autoencoder component of the model is crucial for reshaping input data, preparing it for enhanced classification in the subsequent CNN layers. Starting with an input layer that processes the entire feature set individually, the Autoencoder's Encoder block reduces dimensionality through a dense layer with ReLU activation, capturing essential patterns while filtering out less relevant information. This compression emphasizes key characteristics in the data, effectively highlighting meaningful patterns. The Decoder block then reconstructs this compressed data back to the original feature space via a layer with linear activation, retaining significant features and reducing noise. This reshaping results in a refined dataset that optimally represents the underlying data patterns. Compiled using the Adam optimizer and mean squared error loss function, the Autoencoder minimizes reconstruction error, enhancing the dataset's suitability for further classification. The output of the Autoencoder serves as the input for the CNN component, where additional feature extraction and classification take place. The proposed model architecture combines CNN

and MLP to address both binary and multi-class classification tasks in a structured sequence of feature extraction and classification layers. The CNN component starts with an input layer designed to process sequential data in a one-dimensional format. This is followed by an initial CNN block that includes convolution operations with ReLU activation, followed by batch normalization, max pooling, and dropout layers, tailored for binary classification. For multi-class classification, an initial CNN block with convolution operations, followed by batch normalization, ReLU activation, max pooling, and dropout layers is used. This combination refines key features while preventing overfitting and is repeated in subsequent CNN layers with varying kernel sizes and pool sizes, enabling the model to capture diverse patterns and relationships within the input data. The different kernel sizes allow the model to detect features at multiple scales along the sequence, while the varying pool sizes facilitate downsampling and reduce the sequence's spatial dimensions, enhancing the model's ability to generalize across different input features. After passing through the CNN layers, the output is flattened and processed through an MLP block. This block comprises a dense layer with an activation function, followed by batch normalization and dropout, to enhance the model's capability to learn intricate patterns for binary classification while minimizing overfitting. Similarly, for multi-classification tasks, the output is flattened and passed through an MLP block that includes a dense layer, batch normalization, activation function, and dropout, effectively strengthening the model's ability to represent complex features and reducing the risk of overfitting. Features from both the CNN and MLP components are then combined into a comprehensive unified feature set. Finally, an output layer employs a sigmoid activation function for binary classification or a softmax activation function for multi-class classification, with binary cross-entropy or categorical cross-entropy as the respective loss functions. The model is optimized with the Adam optimizer to ensure efficient learning and robust convergence across both classification objectives.

The encoder layers progressively reduce the dimensionality of the input data, enabling the model to extract key features through dense layers. This process of dimensionality reduction and feature extraction can be mathematically expressed as shown in Equation (24) [50].

$$h^{(l)} = f\left(W^{(l)}\, a^{(l-1)} + b^{(l)}\right) \tag{24}$$

where $h^{(l)}$ represents the output of encoder layer l and $a^{(l-1)}$ denotes the output of the preceding layer, which acts as the input to the first layer. The weight matrix for layer l is denoted by $W^{(l)}$, and the bias vector for that layer is given by $b^{(l)}$. In this context, the activation function represented by f is specifically the ReLU function.

In a standard Autoencoder, the decoder layer is responsible for reconstructing the original input from the compressed representation generated by the encoder. This process of reconstructing the input from the encoded features can be mathematically described by Equation (25) [50].

$$\acute{\alpha} = g\left(W^{(d)}\, h^{(l)} + b^{(d)}\right) \tag{25}$$

where $\acute{\alpha}$ represents the reconstructed output, $h^{(l)}$ is the output from the final encoder layer, the weight matrix for the decoder layer is represented by $W^{(d)}$, and $b^{(d)}$ denotes the bias vector for the decoder layer. The activation function g, generally selected to be linear, is applied to facilitate reconstruction.

The convolution operation in the CNN layers plays a critical role in extracting features by applying filters to the input data. It works by sliding the convolution kernel across the input feature map, computing the dot product at each position and producing a transformed output known as the feature map. The mathematical representation of this convolution process is given in Equation (26) [50].

$$Z_{i,j} = (X * K)_{i,j} = \sum_m \sum_n Z_{i+m,j+n} k_{m,n} \tag{26}$$

where Z denotes the output feature map and X represents the input feature map. The convolution kernel denoted by K is applied during the convolution operation to convert the input features into the output features.

The ReLU activation function illustrated in Equation (27) [51] is a simple but powerful non-linear transformation. It sets all negative inputs to zero while maintaining positive values. This mechanism not only mitigates the problem of vanishing gradient but also promotes sparse activations, both of which contribute to faster training and enhanced model performance.

$$ReLU(x) = max(0,x) \tag{27}$$

The max pooling operation reduces the spatial dimensions of the input feature map, effectively downsampling the data while retaining the most important features. This is accomplished by selecting the maximum value from within a specified pooling window, ensuring that the most significant information is preserved while less relevant details are discarded. The mathematical representation of this operation is provided in Equation (28) [50].

$$P_{i,j} = max\left(X_{i:i+p,\ j:j+q}\right) \tag{28}$$

where P represents the output resulting from the pooling process; p and q denote the dimensions of the pooling window applied to aggregate the input features into the pooled output.

The dropout layer serves as a regularization technique by randomly deactivating a fraction p of the input units during training. This randomness prevents the model from relying too heavily on any particular input feature, thereby reducing the risk of overfitting. As a result, the model is encouraged to learn more robust and generalized patterns. The impact of this regularization process is mathematically described in Equation (29) [52].

$$\text{Dropout(x)} = \begin{cases} x \text{ with probability } 1-p \\ 0 \text{ with probability } p \end{cases} \tag{29}$$

For binary classification tasks, the output layer utilizes the sigmoid function to produce a probability score that indicates the likelihood of an instance belonging to the positive class. This function outputs a value between 0 and 1, with values near 1 indicating a stronger likelihood of the instance being classified as positive, while values closer to 0 suggest a lower probability. The mathematical representation of this process is shown in Equation (30) [53].

$$\sigma(Z) = \frac{1}{1+e^{-z}} \tag{30}$$

where Z denotes the result produced by the last dense layer in the network.

The output layer employs the softmax function for multi-class classification, converting the raw outputs from the model into a probability distribution across all potential classes. This transformation ensures that the probabilities for each class sum to one, effectively representing the model's level of certainty regarding each class. The mathematical representation of this operation is given in Equation (31) [53].

$$Softmax(Z_i) = \frac{e^{z_i}}{\sum_j e^{z_i}} \tag{31}$$

where Z_i represents the output from the final dense layer for class i.

(i) Binary Classification

The architecture outlined in Table 21 describes an Autoencoder-CNN model de-signed for binary classification on the CICIDS2017 and NF-BoT-IoT-v2 datasets. The model begins with an input layer, processing 69 features for the CICIDS2017 dataset and 18 features for the NF-BoT-IoT-v2 dataset. The encoder block employs a dense layer with 64 neurons and ReLU activation to learn compressed representations of the input data. The decoder block features a dense layer with a size matching the Autoencoder input and a linear activation function to reconstruct the original input. The output of the Autoencoder is then fed into the CNN, which begins with a 1D CNN layer comprising 256 filters and ReLU activation, followed by a 1D max pooling layer with a pool size of 2. To prevent overfitting, a dropout layer with a minimal rate of 0.00000001 is included. The second hidden block mirrors this structure, with a pool size of 4 in the max pooling layer. The third hidden block features a dense layer with 256 neurons and ReLU activation, followed by another dropout layer. Finally, a single neuron output layer with a sigmoid activation function produces a probability score, enabling effective binary classification of inputs.

Table 21. Autoencoder–CNN model layers for binary classification.

Dataset	Block	Layers	Layer Size	Activation
CICIDS2017	Input block	Input layer	69	-
NF-BoT-IoT-v2	Input block	Input layer	18	-
Shared structure	Encoder	Dense layer	64	ReLU
	Decoder	Dense layer	Autoencoder input	Linear
	Input block	Input layer	Autoencoder output	-
	Hidden block 1	1D CNN layer	256	ReLU
		1D max pooling layer	2	-
		Dropout layer	0.00000001	-
	Hidden block 2	1D CNN layer	256	ReLU
		1D max pooling layer	4	-
		Dropout layer	0.00000001	-
	Hidden block 3	Dense layer	256	ReLU
		Dropout layer	0.00000001	-
	Output block	Output layer	1	Sigmoid

(ii) Multi-Class Classification

The architecture described in Table 22 details the Autoencoder–CNN model designed for multi-class classification on the CICIDS2017 and NF-BoT-IoT-v2 datasets. The model begins with an input block that processes 69 features for CICIDS2017 and 28 features for NF-BoT-IoT-v2. The encoder block consists of a dense layer with 64 units activated by ReLU, compressing the input data into a lower-dimensional representation. The decoder block mirrors this structure, employing a dense layer with the same size as the Autoencoder input and a linear activation function to reconstruct the original features. The output of the Autoencoder serves as the input to the CNN segment. The CNN segment begins with an input block that directly utilizes the output of the Autoencoder as its input. Hidden block 1 incorporates a 1D CNN layer with 256 filters followed by ReLU activation, 1D max pooling layer with a pool size of 2 and a dropout layer with a rate of 0.0000001 to mitigate overfitting. Hidden block 2 repeats this structure, with the max pooling layer adjusted to a pool size of 4. Hidden block 3 retains the same configuration but uses a max

pooling layer with a pool size of 8. Hidden block 4 features a dense layer with 1024 units followed by ReLU activation and a dropout layer with a rate of 0.0000001. Hidden block 5 contains a dense layer with 768 units, followed by softmax function and a dropout layer with the same minimal rate. Finally, the output block comprises 14 units for CICIDS2017 and 4 units for NF-BoT-IoT-v2, each employing a softmax activation function to generate class probabilities, effectively enabling the model to address multi-class classification tasks.

Table 22. Autoencoder–CNN model layers for multi-class classification.

Dataset	Block	Layers	Layer Size	Activation
CICIDS2017	Input block	Input layer	69	-
NF-BoT-IoT-v2	Input block	Input layer	28	-
Shared structure	Encoder	Dense layer	64	ReLU
	Decoder	Dense layer	Autoencoder input	Linear
	Input block	Input layer	Autoencoder output	-
	Hidden block 1	1D CNN layer	256	ReLU
		1D max pooling layer	2	-
		Dropout layer	0.0000001	-
	Hidden block 2	1D CNN layer	256	ReLU
		1D max pooling layer	4	-
		Dropout layer	0.0000001	-
	Hidden block 3	1D CNN layer	256	ReLU
		1D max pooling layer	8	-
		Dropout layer	0.0000001	-
	Hidden block 4	Dense layer	1024	ReLU
		Dropout layer	0.0000001	-
	Hidden block 5	Dense layer	768	Softmax
		Dropout layer	0.0000001	-
CICIDS2017	Output block	Output layer	14	Softmax
NF-BoT-IoT-v2	Output block	Output layer	4	Softmax

(iii) Hyperparameter Configuration for the Autoencoder–CNN Model

The hyperparameters for the Autoencoder model are detailed in Table 23. For both binary and multi-class classification tasks, the model operates with a batch size of 64, determining the number of samples processed before the model updates its internal weights. The Adam optimizer was utilized with its default learning rate of 0.001, ensuring efficient and stable convergence during training. The model employed mean squared error (MSE) as the loss function for both tasks. Accuracy was used as the evaluation metric for both tasks, representing the percentage of correctly classified instances out of the total predictions.

The hyperparameters for the CNN model, as outlined in Table 24, were optimized for both binary and multi-class classification tasks. The model operated with a batch size of 128, indicating the number of samples processed before the model's parameters were updated, and this configuration was consistent across both classification types. The learning rate for both the binary and multi-class classifiers was managed using ReduceLROnPlateau callback. The initial learning rate was set to 0.001. The callback monitored the validation loss, and if the loss rate showed no improvement for two consecutive epochs, the learning

rate was reduced by a factor of 0.5. The learning rate is capped at a minimum value of 1×10^{-5}, ensuring it remained within an effective range for training. This adaptive learning rate strategy enabled more efficient convergence, particularly as the model approached the optimal solution, and helped prevent overshooting during the training process. The Adam optimizer was chosen for its robust adaptive learning capabilities which have been proved effective in both binary and multi-class scenarios. For binary classification, the model employed binary cross-entropy as the loss function, which measured the error between predicted probabilities and actual binary labels. Conversely, the multi-class classification model utilized categorical cross-entropy, evaluating the discrepancy between predicted class probabilities and true class labels across multiple categories. Both models used accuracy as the evaluation metric, reflecting the proportion of correctly predicted instances relative to the total number of predictions made. This metric provided a clear indication of the models' performance, showcasing how well the predicted labels aligned with the actual labels.

Table 23. Hyperparameters of the Autoencoder model.

Parameter	Binary Classifier	Multi-Class Classifier
Batch size	64	64
Learning rate	0.001	0.001
Optimizer	Adam	Adam
Loss function	Mean_squared_error	Mean_squared_error
Metric	Accuracy	Accuracy

Table 24. Hyperparameters of the CNN model.

Parameter	Binary Classifier	Multi-Class Classifier
Batch size	128	128
Learning rate	Scheduled: Initial = 0.001, Factor = 0.5, Min = 1×10^{-5} (ReduceLROnPlateau)	Scheduled: Initial = 0.001, Factor = 0.5, Min = 1×10^{-5} (ReduceLROnPlateau)
Optimizer	Adam	Adam
Loss function	Binary cross-entropy	Categorical cross-entropy
Metric	Accuracy	Accuracy

3.3.5. Transformer–Deep Neural Network (Transformer–DNN)

The model architecture integrates transformer and DNN components to enhance classification performance by processing sequential data, where each data point is represented as a 1D vector. The transformer block plays a crucial role in capturing intricate relationships within the input data, using its multi-head attention mechanism, dynamically assigning varying levels of importance to different parts of the input sequence and aiding in the identification of complex patterns and dependencies. After the attention mechanism, the output is passed through a feed-forward network (FFN) consisting of dense layers with non-linear transformations, thereby refining the feature representations further. Residual connections maintain gradient flow during training, improving convergence, while layer normalization stabilizes the output. After the transformer block, the output is passed to DNN layers, where multiple fully connected layers with ReLU activation progressively refine the feature representations learned by the transformer, enabling the model to learn more complex and abstract patterns. Dropout layers are applied to regularize the network and prevent overfitting by randomly disabling a proportion of neurons during training.

The output is then flattened and further processed by the DNN. The final output layer uses a sigmoid activation function for binary classification, producing a probability score between 0 and 1, indicating the likelihood of an instance belonging to the positive class. For multi-class classification, the output layer employs a softmax activation function with a number of neurons equal to the number of classes, generating a probability distribution over all classes and ensuring the sum of the probabilities equals 1. This distribution reflects the likelihood of an instance belonging to each class, with the highest probability indicating the predicted class. The combination of the global context and long-range dependencies captured by the transformer along with the high-level feature extraction and non-linear transformations provided by the DNN results in an effective architecture for both binary and multi-class classification tasks, enabling the model to leverage both global and local patterns within the data for classification into either two or multiple categories.

The multi-head attention mechanism efficiently identifies intricate relationships within the input data, as demonstrated by Equation (32) [56].

$$Attention(Q,\ K,\ V) = softmax(\frac{QK^T}{\sqrt{d_k}}) \tag{32}$$

where Q represents the query matrix, K denotes the key matrix, and V signifies the value matrix. The variable d_k refers to the dimension of the keys, which plays a crucial role in the computation of attention scores within the model.

Each attention head computes its attention independently, and the results are then concatenated according to Equation (33) [56].

$$MultiHead(Q,\ K,\ V) = Concat(head_1,\ \ldots\ldots,\ head_h)W^0 \tag{33}$$

where W^0 refers to the output weight matrix, which is utilized to transform the output of the preceding layer into the final output of the model.

Layer normalization stabilizes the output of each layer, as described in Equation (34) [57].

$$LayerNorm(x) = \frac{X -}{\sigma} \times \gamma + \beta \tag{34}$$

where μ represents the mean of the inputs, σ denotes the standard deviation, and γ and β are learnable parameters that are utilized in the normalization process.

The FFN processes the output from the attention mechanism according to Equation (35) [56].

$$FNN(X) = max(0,\ xW_1 + b_1)W_2 + b_2 \tag{35}$$

where W_1 and W_2 refer to the weight matrices, while b_1 and b_2 represent the corresponding biases associated with the layers in the model.

In a DNN, the feed-forward process entails passing the input through multiple layers to produce the final output. This process for layer l is mathematically expressed in Equation (36) [54].

$$a^{(l)} = f\left(W^{(l)}\, a^{(l-1)} + b^{(l)}\right) \tag{36}$$

where $a^{(l)}$ represents the activation of the current layer l, the weight matrix for this layer is denoted by $W^{(l)}$, and $b^{(l)}$ stands for the bias vector of layer l. The activation function f is applied element-wise and can include non-linear functions such as ReLU or sigmoid, enabling the model to capture complex relationships.

The ReLU activation function shown in Equation (37) [55] is a simple yet powerful non-linear function. It outputs zero for negative input values, promotes sparsity in activations, and ensures smooth gradient flow, making it particularly effective for deep learning models.

$$ReLU(x) = max(0, x) \tag{37}$$

For binary classification tasks, the sigmoid function is used to generate a probability score indicating the likelihood of an instance belonging to the positive class. It transforms the raw score into a value between 0 and 1, serving as a threshold for the classification decision. This operation is mathematically represented in Equation (38) [53].

$$\sigma(Z) = \frac{1}{1 + e^{-z}} \tag{38}$$

where Z refers to the output from the last dense layer.

For multi-class classification tasks, the output layer employs the softmax function, enabling the model to produce probability distributions across multiple classes. This function converts a vector of raw scores (logits) into normalized values between 0 and 1, ensuring that the sum of all probabilities equals 1. The mathematical expression of the softmax function is given in Equation (39) [53].

$$Softmax(Z_i) = \frac{e^{z_i}}{\sum_j e^{z_i}} \tag{39}$$

where Z_i denotes the raw score for the i-th class.

(i) Binary Classification

The architecture of the transformer model designed for binary classification is detailed in Table 25. For the CICIDS2017 dataset, the model begins with an input layer that processes data structured as (69, 1), accommodating input with 69 features. Similarly, for the NF-BoT-IoT-v2 dataset, the input layer processes data structured as (18, 1), effectively handling input with 18 features. The transformer block employs a multi-head attention mechanism with eight heads and a key dimension of 64, capturing complex relationships within the input data. The output from the attention layer is normalized using layer normalization with an epsilon value of 1×10^{-6}. A residual connection is implemented to add the original input data back to the attention output for stability. The feed-forward block consists of a dense layer with 512 units and a ReLU activation function, followed by another dense layer with 512 units. The output from the feed-forward network is added back to the previous block's output via another residual connection, followed by another layer normalization step with epsilon = 1×10^{-6} to normalize the combined output.

The DNN model designed for binary classification consisted of several blocks, as outlined in Table 26. The input block takes the output from the transformer model, with the size matching the transformer output. In hidden block 1, a dense layer with 256 units and ReLU activation is employed for feature extraction, followed by a dropout layer configured with an extremely low dropout rate of 0.0000001 to mitigate overfitting. Hidden block 2 includes a dense layer with 128 units, ReLU activation, and another dropout layer with the same rate. Hidden block 3 features a dense layer with 64 units and ReLU activation. The output block is composed of a dense layer with a single unit and a sigmoid activation function, producing the final binary classification output.

(ii) Multi-Class Classification

The architecture of the transformer model designed for multi-class classification is outlined in Table 27. For the CICIDS2017 dataset, the model processed input data with 69 features through an input layer, structured as (69, 1). Similarly, for the NF-BoT-IoT-v2 dataset, the input layer accommodated input data with 28 features, structured as (28, 1). The transformer block utilizes a multi-head attention mechanism, configured with 8 attention heads and a key dimension of 64, enabling the model to capture intricate dependencies within the input data. Following the attention mechanism, the output is subjected to layer normalization with an epsilon value of 1×10^{-6}, stabilizing the model's learning

process. To further enhance training stability, a residual connection is employed, allowing the original input to be added to the attention output. The feed-forward block consists of a dense layer with 512 units and ReLU activation, followed by a second dense layer with 512 units. The output from the feed-forward network is then combined with the previous block's output through another residual connection. Finally, another normalization step with epsilon = 1×10^{-6} is applied to the combined output, ensuring the model's stability during the learning phase.

Table 25. Transformer model layers for binary classification.

Dataset	Block	Layer Type	Output Size	Activation Function	Parameters	Description
CICIDS2017	Input block	Input layer	(69, 1)	-	-	Accepts input data with 69 features (as per the input shape).
NF-BoT-IoT-v2	Input block	Input layer	(18, 1)	-	-	Accepts input data with 18 features (as per the input shape).
Shared structure	Transformer block	Multi-head attention	-	-	num_heads = 8, key_dim = 64	Captures complex relationships within input data.
		Layer normalization	-	-	epsilon = 1×10^{-6}	Normalizes the output from the attention layer.
		Add (residual connection)	-	-	-	Adds input data to the attention output for stability.
	Feed-forward block	Dense layer	512	ReLU	units = 512, activation = 'ReLU'	Applies a dense transformation with ReLU activation.
		Dense layer	512	-	units = 512	Another dense transformation without activation.
		Add (residual connection)	-	-	-	Adds feed-forward output to the previous block output.
		Layer normalization	-	-	epsilon = 1×10^{-6}	Normalizes the combined output for stability.

Table 26. DNN model layers for binary classification.

Dataset	Block	Layers	Layer Size	Activation
Shared structure	Input block	Input layer	Transformer output	-
	Hidden block 1	Dense layer	256	ReLU
		Dropout layer	0.0000001	-
	Hidden block 2	Dense layer	128	ReLU
		Dropout layer	0.0000001	-
	Hidden block 3	Dense layer	64	ReLU
	Output block	Output layer	1	Sigmoid

The DNN model for multi-class classification, as shown in Table 28, begins with the input block, where the output from the transformer model is used as input. The hidden block 1 consists of a dense layer with 256 units, activated by ReLU, followed by a dropout layer with a rate of 0.3 for regularization. In hidden block 2, a dense layer with 128 units and a ReLU activation is applied, along with another dropout layer at a rate of 0.3. Hidden block 3 includes a dense layer with 64 units and a ReLU activation function. The output

block features a dense layer with a softmax activation function, producing the final multi-class classification output. For the CICIDS2017 dataset, the output layer comprised 14 units, corresponding to 14 different classes. Similarly, for the NF-BoT-IoT-v2 dataset, the output layer consisted of 4 units, corresponding to 4 distinct classes.

Table 27. Transformer model layers for multi-class classification.

Dataset	Block	Layer Type	Output Size	Activation Function	Parameters	Description
CICIDS2017	Input block	Input layer	(69, 1)	-	-	Accepts input data with 69 features (as per the input shape).
NF-BoT-IoT-v2	Input block	Input layer	(28, 1)	-	-	Accepts input data with 28 features (as per the input shape).
Shared Structure	Transformer block	Multi-head attention	-	-	num_heads = 8, key_dim = 64	Captures complex relationships within input data.
		Layer normalization	-	-	epsilon = 1×10^{-6}	Normalizes the output from the attention layer.
		Add (residual connection)	-	-	-	Adds input data to the attention output for stability.
	Feed-forward block	Dense layer	512	ReLU	units = 512, activation = 'ReLU'	Applies a dense transformation with ReLU activation.
		Dense layer	512	-	units = 512	Another dense transformation without activation.
		Add (residual connection)	-	-	-	Adds feed-forward output to the previous block output.
		Layer normalization	-	-	epsilon = 1×10^{-6}	Normalizes the combined output for stability.

Table 28. DNN model layers for multi-class classification.

Dataset	Block	Layers	Layer Size	Activation
Shared Structure	Input block	Input layer	Transformer output	-
	Hidden block 1	Dense layer	256	ReLU
		Dropout layer	0.3	-
	Hidden block 2	Dense layer	128	ReLU
		Dropout layer	0.3	-
	Hidden block 3	Dense layer	64	ReLU
CICIDS2017	Output block	Output layer	14	Softmax
NF-BoT-IoT-v2	Output block	Output layer	4	Softmax

(iii) Hyperparameter Configuration for the Transformer–DNN Model

The hyperparameters for the Transformer-DNN model, detailed in Table 29, have been meticulously optimized for effectiveness in both binary and multi-class classification tasks. The model operates with a batch size of 128, which defines the number of samples processed before the model's weights are updated, ensuring consistency across both classification scenarios. The learning rate is adjustable, set at 0.001, allowing for fine-tuning of the speed at which weights are modified during training. The Adam optimizer is utilized due to its

robust adaptive learning features, demonstrating effectiveness in both binary and multi-class contexts. In the case of binary classification, the model leverages binary cross-entropy as its loss function, quantifying the divergence between predicted probabilities and actual binary outcomes. In contrast, the multi-class classification model employs categorical cross-entropy, assessing the difference between predicted class probabilities and the true class labels across multiple categories. For performance evaluation, both models utilize accuracy as their primary metric, which reflects the ratio of correctly predicted instances to the total number of predictions made. This metric serves as a straightforward indicator of model performance, illustrating the extent to which predicted labels correspond with actual labels.

Table 29. Hyperparameters of the Transformer–DNN model.

Parameter	Binary Classifier	Multi-Class Classifier
Batch size	128	128
Learning rate	0.001	0.001
Optimizer	Adam	Adam
Loss function	Binary cross-entropy	Categorical cross-entropy
Metric	Accuracy	Accuracy

4. Results and Experiments

In this section, the performances of the CNN, Autoencoder, DNN, and proposed Autoencoder–CNN and Transformer–DNN models were evaluated using the CICIDS2017 and NF-BoT-IoT-v2 datasets. The experimental results demonstrated that our models outperformed the state-of-the-art methods in anomaly detection.

4.1. Dataset Description and Preprocessing Overview

This study employed the CICIDS2017 and NF-BoT-IoT-v2 datasets, both established as pivotal benchmarks for IDSs, encompassing a broad spectrum of network behaviors and attack patterns. However, these datasets present inherent challenges that include missing entries, redundant data, outliers, and imbalances across classes. In this study, to mitigate these issues, a series of preprocessing techniques were implemented that included handling missing values, eliminating duplicates, addressing anomalous data points, and correcting class distribution disparities. These measures ensured that the datasets were effectively prepared for accurate and robust model evaluation in both binary and multi-class classification contexts.

4.1.1. CICIDS2017 Dataset

The CICIDS2017 dataset, as described in Section 3.1, provides a comprehensive representation of network activities, encompassing both legitimate and malicious traffic across diverse attack types, making it a critical resource for IDS research. Despite its value, the dataset poses challenges such as incomplete records, duplicate data, and significant class imbalances. These issues were systematically addressed through the preprocessing techniques outlined in Section 3.2. The process involved managing missing values, eliminating duplicates, and employing sophisticated methods for detecting and removing outliers, including LOF. Additionally, numerical features were scaled using MinMaxScaler to ensure uniformity. To address class imbalance, advanced resampling techniques were integrated into the model training pipeline, including enhanced hybrid ADASYN-SMOTE for oversampling in binary classification, advanced SMOTE for oversampling in multi-class classification, and ENN for undersampling. These comprehensive preprocessing steps

significantly improved the dataset's robustness and reliability for binary and multi-class classification tasks.

4.1.2. NF-BoT-IoT-v2 Dataset

An IoT NetFlow-based dataset was developed by extending the NF-BoT-IoT dataset, featuring data meticulously extracted from publicly available pcap files and flows categorized based on their corresponding attack types. This dataset comprises 37,763,497 data flows, of which 37,628,460 (99.64%) are identified as attack instances, while 135,037 (0.36%) represent benign traffic [13]. It encompasses five distinct classes, including one benign category and four unique attack types. Data preprocessing is a critical component of machine learning workflows, transforming raw data into a structured and analysis-ready format to optimize model performance. For the NF-BoT-IoT-v2 dataset, this process began with handling missing values by removing incomplete records, followed by eliminating duplicate entries to prevent redundancy. Outliers were identified and addressed using a combination of Z-score and LOF methods, ensuring a more robust detection of anomalies and improving data integrity. Feature selection based on correlation analysis reduced dimensionality by retaining only the most significant features, while MinMaxScaler standardized the numerical data for consistent representation. Principal component analysis (PCA) was employed in the binary classification to minimize redundancy and retain essential features. The dataset was initially split into training and testing subsets then recombined for the application of the ADASYN method, generating synthetic samples to address class imbalances. After this augmentation, the dataset was re-split, keeping the test set intact. The ENN method was applied for undersampling, removing noisy or borderline samples, and class weights were adjusted during training to further address imbalances. This comprehensive approach ensured the model could effective learn from the data, enhancing its ability to generalize and deliver accurate predictions for detecting anomalies and classifying network traffic.

4.2. Experiment's Establishment

The models were constructed using TensorFlow 2.17.0 and Keras 3.4.1 on the Kaggle platform 1.6.17. The experimental setup included hardware featuring an Nvidia GeForce RTX 1050 graphics card and Windows 10. During data resampling, only the training set was utilized. The test dataset was kept untouched, reserved exclusively for evaluating the model's performance. Training involved 500 epochs, with each epoch optimizing the model to improve performance and minimize loss, ensuring thorough learning and generalization from the dataset.

4.3. Evaluation Metrics

Confusion matrices are commonly used for evaluating machine learning models, comparing actual and predicted class information in a structured table format, as described in reference [58]. A confusion matrix simplifies the computation of various performance metrics by providing the following information:

- True Positives (TPs): Correctly predicted positive instances.
- False Negatives (FNs): Instances wrongly predicted as negative.
- True Negatives (TNs): Correctly predicted negative instances.
- False Positives (FPs): Instances wrongly predicted as positive.

Equation (40) [59] is used to calculate accuracy, the simplest and most fundamental metric that can be derived from the confusion matrix.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{40}$$

In addition to accuracy, it is crucial to assess any model using a range of performance metrics, particularly when working with imbalanced datasets. Therefore, the current models were also evaluated using additional metrics such as recall, precision, and F1 score. Precision was calculated by dividing the number of correctly identified positive instances by the total number of instances predicted as positive, including both true positives and false positives. These metrics provided a more comprehensive understanding of the models' performance across all classes. Precision is also referred to as positive predictive value and is calculated using Equation (41) [59]. Recall, calculated using Equation (42) [59], measures the proportion of actual positive samples correctly identified by the model among all samples that should have been identified as positive. Equation (43) [60] computes the F-score, which represents the harmonic mean of accuracy and recall.

$$Precision = \frac{TP}{TP + FP} \tag{41}$$

$$Recall = \frac{TP}{TP + FN} \tag{42}$$

$$Fscore = \frac{2 * precision * recall}{precision + recall} \tag{43}$$

In this case, the objective was to optimize the metrics accuracy, precision, recall, and F-score as specified by the evaluation criteria.

4.4. Results

The evaluation of the proposed models was conducted in two key phases, training and testing, using the train and test subsets of the CICIDS2017 dataset, with additional evaluation on other datasets like NF-BoT-IoT-v2 to demonstrate the models' generalizability. The experiments focused on both binary and multi-class classification tasks, aimed at effectively detecting attacks and accurately identifying various attack types. The performance of the proposed models was benchmarked against existing intrusion detection systems as reported in the literature, providing a contextual understanding of their strengths and limitations. The results on the CICIDS2017 and NF-BoT-IoT-v2 datasets highlight the effectiveness of the models in both binary and multi-class classification tasks. The Transformer-DNN and Autoencoder-CNN models consistently demonstrated superior performance, excelling in accurately detecting attacks and correctly identifying various attack types. Other models, such as CNN, Autoencoder and DNN, also performed well, but the Transformer-DNN and the Autoencoder-CNN model stood out as the most robust across all evaluation metrics. These findings underscore the impact of the implemented preprocessing techniques and the strength of the proposed models in handling the classification challenges presented by the datasets.

(i) Binary Classification

The performance metrics for binary classification across different models on the CICIDS2017 and NF-BoT-IoT-v2 datasets are presented in Table 30. On the CICIDS2017 dataset, the CNN model achieved accuracy, precision, recall, and F-score of 99.83% across all metrics, while the Autoencoder model recorded slightly lower values, at 99.73%. The DNN model performed better, achieving 99.88% for all metrics. The Autoencoder–CNN model demonstrated notable improvements, achieving accuracy, precision, recall, and F-score of 99.90%. The Transformer–DNN model further outperformed the other models on this dataset, with all metrics reaching 99.92%. With the NF-BoT-IoT-v2 dataset, the CNN model performed exceptionally well, with accuracy, precision, recall, and F-score values of 99.97%. The Autoencoder model followed closely with 99.94% across all metrics, while

the DNN model achieved slightly higher results at 99.96%. The Autoencoder–CNN model delivered superior performance, achieving 99.98% for all metrics, matched only by the Transformer–DNN model, which also achieved 99.98% for accuracy, precision, recall, and F-score. These results emphasize the effectiveness and reliability of our Autoencoder–CNN and Transformer–DNN models in achieving consistently high metrics across both datasets for binary classification.

Table 30. Performance metrics in binary classification.

Dataset	Metric	Accuracy	Precision	Recall	F-Score
CICIDS2017	CNN	99.83%	99.83%	99.83%	99.83%
	Autoencoder	99.73%	99.73%	99.73%	99.73%
	DNN	99.88%	99.88%	99.88%	99.88%
	Autoencoder–CNN	99.90%	99.90%	99.90%	99.90%
	Transformer–DNN	99.92%	99.92%	99.92%	99.92%
NF-BoT-IoT-v2	CNN	99.97%	99.97%	99.97%	99.97%
	Autoencoder	99.94%	99.94%	99.94%	99.94%
	DNN	99.96%	99.96%	99.96%	99.96%
	Autoencoder–CNN	99.98%	99.98%	99.98%	99.98%
	Transformer–DNN	99.98%	99.98%	99.98%	99.98%

(ii) Multi-Class Classification

The performance metrics for multi-class classification across various models on the CICIDS2017 and NF-BoT-IoT-v2 datasets are summarized in Table 31, highlighting the results of our proposed models, Autoencoder-CNN and Transformer-DNN. On the CICIDS2017 dataset, the CNN and DNN models achieved accuracy, precision, recall, and F-score values of 99.94%, while the Autoencoder model recorded slightly lower values at 99.93%. Autoencoder-CNN model outperformed these, achieving 99.95% for all metrics, with the Transformer-DNN model demonstrating the best performance at 99.96% across accuracy, precision, recall, and F-score. For the NF-BoT-IoT-v2 dataset, the CNN model delivered an accuracy of 97.87%, precision of 97.91%, and recall and F-score values of 97.87%. The Autoencoder model showed slightly lower performance, with 97.81% for all metrics except precision, which was 97.87%. The DNN model achieved an accuracy of 97.83%, precision of 97.88%, recall of 97.83%, and an F-score of 97.82%. Autoencoder-CNN model demonstrated the best overall performance, achieving 97.95% for accuracy, recall, and F-score, and 97.97% for precision. The Transformer-DNN model closely followed, achieving accuracy, recall, and F-score values of 97.90% and precision of 97.98%.These results underscore the superior performance of our proposed models, Autoencoder-CNN and Transformer-DNN, in multi-class classification, demonstrating their reliability and effectiveness across both datasets.

Table 31. Performance metrics in multi-class classification.

Dataset	Metric	Accuracy	Precision	Recall	F-Score
CICIDS2017	CNN	99.94%	99.94%	99.94%	99.94%
	Autoencoder	99.93%	99.93%	99.93%	99.93%
	DNN	99.94%	99.94%	99.94%	99.94%
	Autoencoder–CNN	99.95%	99.95%	99.95%	99.95%
	Transformer–DNN	99.96%	99.96%	99.96%	99.96%

Table 31. *Cont.*

Dataset	Metric	Accuracy	Precision	Recall	F-Score
NF-BoT-IoT-v2	CNN	97.87%	97.91%	97.87%	97.87%
	Autoencoder	97.81%	97.87%	97.81%	97.81%
	DNN	97.83%	97.88%	97.83%	97.82%
	Autoencoder–CNN	97.95%	97.97%	97.95%	97.95%
	Transformer–DNN	97.90%	97.98%	97.90%	97.90%

5. Discussion

This section provides a comprehensive evaluation of the Autoencoder–CNN and Transformer–DNN models, benchmarking their performance against other classification techniques including CNN, Autoencoder, and DNN, across both binary and multi-class classification tasks. A detailed analysis of confusion matrices and performance metrics including accuracy, precision, recall, and F1 score highlights the comparative strengths and limitations of each approach. By focusing on results from the CICIDS2017 and NF-BoT-IoT-v2 datasets, we aim to demonstrate how the Autoencoder-CNN model, with its integration of Autoencoder and CNN architectures, and the Transformer-DNN model, with its combination of Transformer and DNN architectures, excel in detecting different classes. These integrations can contribute to measurable improvements in the detection and differentiation of diverse attack types, addressing key challenges in network intrusion detection. Through this in-depth discussion, we underscore the practical implications of our findings, particularly regarding the enhanced accuracy and reliability these models offer for intrusion detection systems in real-world applications. These results demonstrate the models' potential to improve both early threat detection and response, thereby elevating the overall robustness of modern cyber security solutions.

(i) Binary Classification

On the CICIDS2017 dataset, the Autoencoder–CNN and Transformer–DNN models exhibited exceptional performance in binary classification. The Autoencoder-CNN achieved accuracy, precision, recall, and F1 scores of 99.90%, correctly detecting 13,927 normal instances and 11,039 attack instances, while incorrectly detecting 14 normal instances and 10 attack instances. The Transformer-DNN outperformed these results slightly, achieving metrics of 99.92% and correctly detecting 13,930 normal instances and 11,039 attack instances, with 11 normal instances and 10 attack instances incorrectly detected. On the NF-BoT-IoT-v2 dataset, both models achieved identical metrics of 99.98% across accuracy, precision, recall, and F1 score. The Autoencoder–CNN and Transformer–DNN correctly detected 1263 normal instances and 10,795 attack instances, with 3 attack instances incorrectly detected, as shown in Figure 3. This high level of performance underscores the models' robustness, especially in handling imbalanced datasets such as are typical in real-world applications. The high precision and recall values demonstrate their reliability in detecting attacks while minimizing both false positives and false negatives, making them effective solutions for intrusion detection systems.

The comparative performance of the Transformer–DNN and Autoencoder–CNN models in binary classification on the CICIDS2017 and NF-BoT-IoT-v2 datasets, as shown in Figures 4 and 5, demonstrates their outstanding effectiveness. With the CICIDS2017 dataset, the Transformer–DNN model exceled, achieving perfect scores across all metrics and 99.92% accuracy, precision, recall, and F1 score. The Autoencoder-CNN was close behind, attaining 99.90% across all metrics, reflecting its strong performance. The DNN model also performed well, achieving 99.88% in all metrics, while the CNN and Autoencoder models showed slightly lower results at 99.83% and 99.73%, respectively. With the NF-BoT-IoT-v2

dataset, both the Transformer–DNN and Autoencoder-CNN models achieved flawless performance, with 99.98% across all metrics, highlighting their excellent ability in binary classification tasks. The DNN model trailed slightly with a score of 99.96%, while the CNN and Autoencoder models achieved 99.97% and 99.94%, respectively. These results further emphasize the exceptional classification capabilities of the Transformer–DNN and Autoencoder–CNN models, proving their suitability for real-world application in intrusion detection, particularly when dealing with imbalanced datasets.

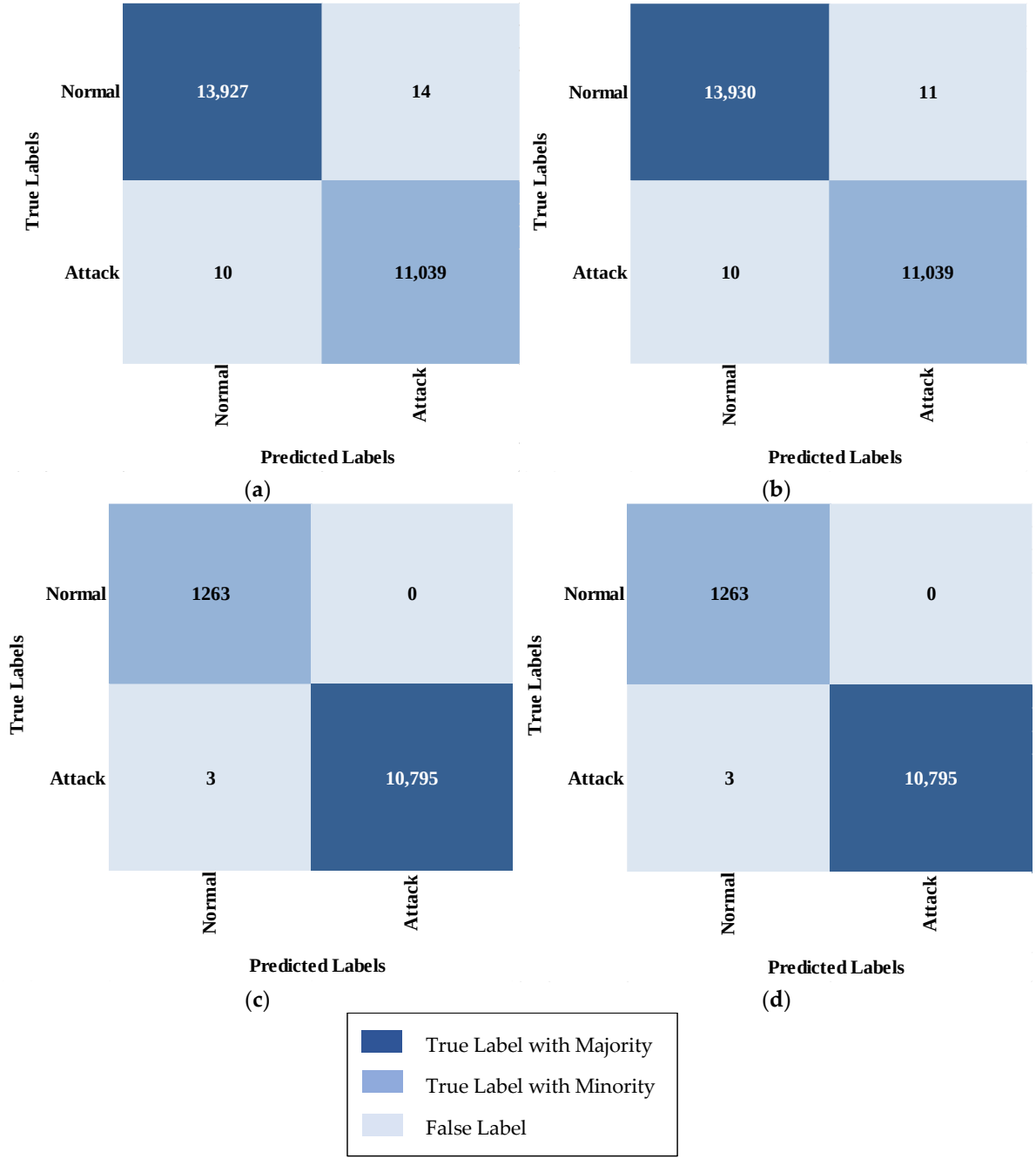

Figure 3. Confusion matrices for binary classification: (**a**) Autoencoder–CNN on the CICIDS2017 dataset, (**b**) Transformer–DNN on the CICIDS2017 dataset, (**c**) Autoencoder-CNN on the NF-BoT-IoT-v2 dataset, and (**d**) Transformer–DNN on the NF-BoT-IoT-v2 dataset.

The performance metrics in Table 32 highlight the exceptional effectiveness of the Autoencoder–CNN and Transformer–DNN models for binary classification across different classes within the CICIDS2017 and NF-BoT-IoT-v2 datasets. With the CICIDS2017 dataset,

the Autoencoder–CNN achieved an accuracy of 99.90% for the Normal class, with precision of 99.93%, recall of 99.90%, and an F1 score of 99.91%. For the Attack class, it attained an accuracy of 99.91%, precision of 99.87%, recall of 99.91%, and an F1 score of 99.89%. The Transformer–DNN slightly outperformed this, with 99.92% accuracy for the Normal class, achieving a precision of 99.93%, recall of 99.92%, and an F1 score of 99.92%. For the Attack class, it reached 99.91% accuracy with metrics of 99.90% precision, 99.91% recall, and 99.90% F1 score. With the NF-BoT-IoT-v2 dataset, both models demonstrated outstanding performance, highlighting their robustness and effectiveness in intrusion detection. The Autoencoder–CNN and Transformer–DNN models achieved 100% accuracy for the Normal class, with precision of 99.76%, recall of 100%, and F1 score of 99.88%. For the Attack class, both models delivered accuracy of 99.97%, achieving 100% precision, 99.97% recall, and an F1 score of 99.99%. These metrics showcase the balanced performance of the Autoencoder–CNN and Transformer–DNN models across classes, making them highly effective for real-world intrusion detection scenarios where accurate identification of both normal and attack traffic is critical.

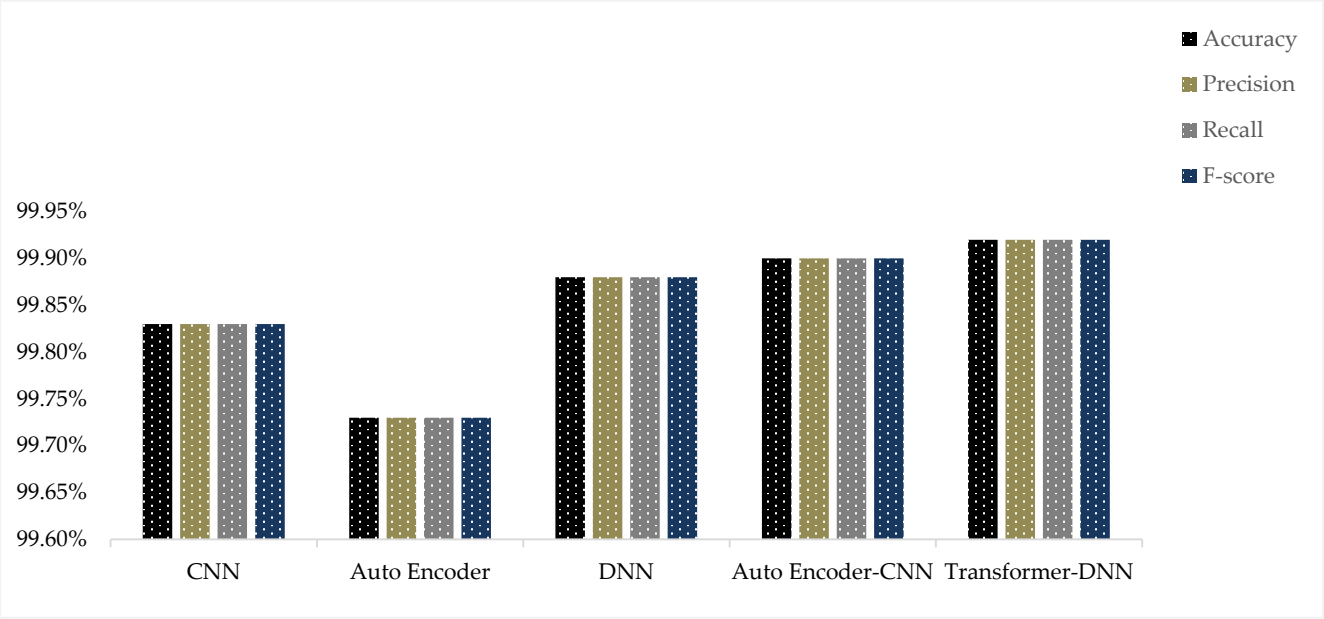

Figure 4. Proposed Transformer–DNN and Autoencoder–CNN versus binary classifiers, on the CICIDS2017 dataset.

(ii) Multi-Class Classification

In multi-class classification on the CICIDS2017 dataset, the Autoencoder–CNN model demonstrated exceptional performance across accuracy, precision, recall, and F1 score, all reaching 99.95%. The confusion matrix shown in Figure 6 reflects the model's ability to effectively distinguish between a wide range of attack types with minimal misclassifications. For example, the model correctly classified 1811 instances of PortScan, with only one misclassification incorrectly labeled as DoS GoldenEye. Similarly, it correctly classified all 2025 instances of DDoS, with no misclassifications. The model also accurately classified 5467 instances of DoS Hulk, with one misclassification incorrectly labeled as DDoS. Regarding more challenging attack types such as DoS Slowloris, the model correctly identified 227 instances, with a single misclassification incorrectly labeled as DoS Slowhttptest. Likewise, it accurately classified 155 instances of DoS Slowhttptest, with 1 misclassification incorrectly labeled as a Web Attack—XSS. In the case of Bot attacks, the model correctly classified 98 instances, with one misclassification incorrectly labeled as

a Web Attack—Brute Force. The model also excelled in classifying rare attack types like Web Attack—Brute Force or Web Attack—XSS, infiltration, Web Attack—Sql Injection, and Heartbleed, all with no misclassifications. The confusion matrix highlights the model's ability to handle both frequent and rare attack types with minimal errors, demonstrating its effectiveness in addressing multi-class classification challenges, particularly in imbalanced datasets. These results validate the model's potential for real-world intrusion detection systems, where both high accuracy and the ability to distinguish among diverse attack types are essential.

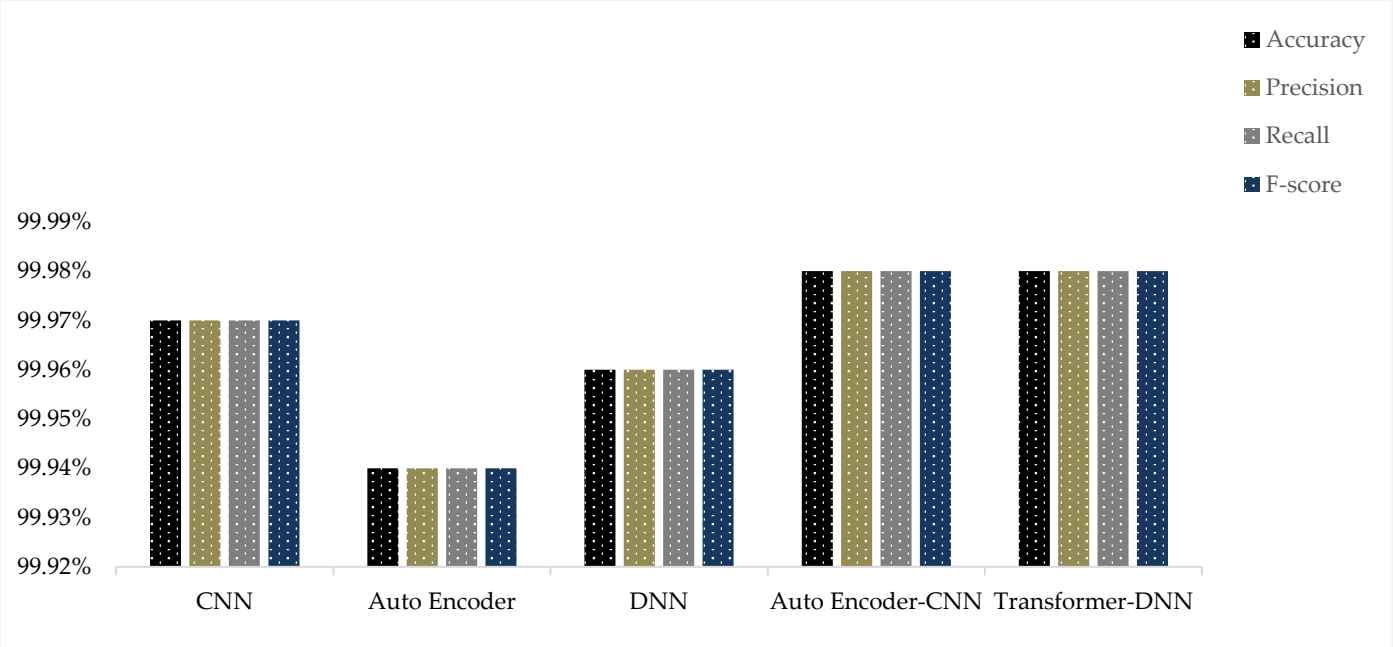

Figure 5. Proposed Transformer–DNN and Autoencoder–CNN versus binary classifiers, on the NF-BoT-IoT-v2 dataset.

Table 32. Performance metrics for Autoencoder–CNN and Transformer–DNN in binary classification across several classes.

Dataset	Model	Label	Accuracy	Precision	Recall	F-Score
CICIDS2017	Autoencoder–CNN	Normal	99.90%	99.93%	99.90%	99.91%
		Attack	99.91%	99.87%	99.91%	99.89%
	Transformer–DNN	Normal	99.92%	99.93%	99.92%	99.92%
		Attack	99.91%	99.90%	99.91%	99.90%
NF-BoT-IoT-v2	Autoencoder–CNN	Normal	100%	99.76%	100%	99.88%
		Attack	99.97%	100%	99.97%	99.99%
	Transformer–DNN	Normal	100%	99.76%	100%	99.88%
		Attack	99.97%	100%	99.97%	99.99%

In multi-class classification on the CICIDS2017 dataset, the Transformer-DNN model demonstrates exceptional performance, achieving outstanding accuracy, precision, recall, and F1-scores across multiple attack classes. As illustrated by the confusion matrix in Figure 7, the model effectively distinguishes between various attack types with minimal misclassifications. For instance, the model correctly identified all 1811 instances of PortScan, with only 1 misclassification, where it was classified as DDoS, and accurately classified all 2025 instances of DDoS and 5468 instances of DoS Hulk, with no misclassifications. For DoS GoldenEye, the model correctly classified 536 instances without errors, and similarly,

the model classified 305 instances of FTP-Patator, with just 1 instance incorrectly labeled as DoS Slowloris. Additionally, for more challenging attack types, such as DoS Slowloris and Web Attack–Brute Force, the model achieved perfect accuracy, correctly identifying all 228 instances of DoS Slowloris and all 63 instances of Web Attack–Brute Force, with no misclassifications. The Web Attack-SQL Injection, Infiltration, and Heartbleed attack classes were all perfectly identified, with 100% precision, recall, and F1-scores. The model also exhibited strong performance across less frequent classes like Bot, with only 1 misclassification out of 99 instances, showcasing its ability to handle imbalanced datasets effectively. With an overall accuracy of 99.96%, precision of 99.96%, recall of 99.96%, and F1-score of 99.96%, these results highlight the Transformer-DNN model's robust ability to manage multi-class classification challenges, making it a reliable candidate for real-world intrusion detection systems where precision and accuracy in classifying diverse attack types are essential.

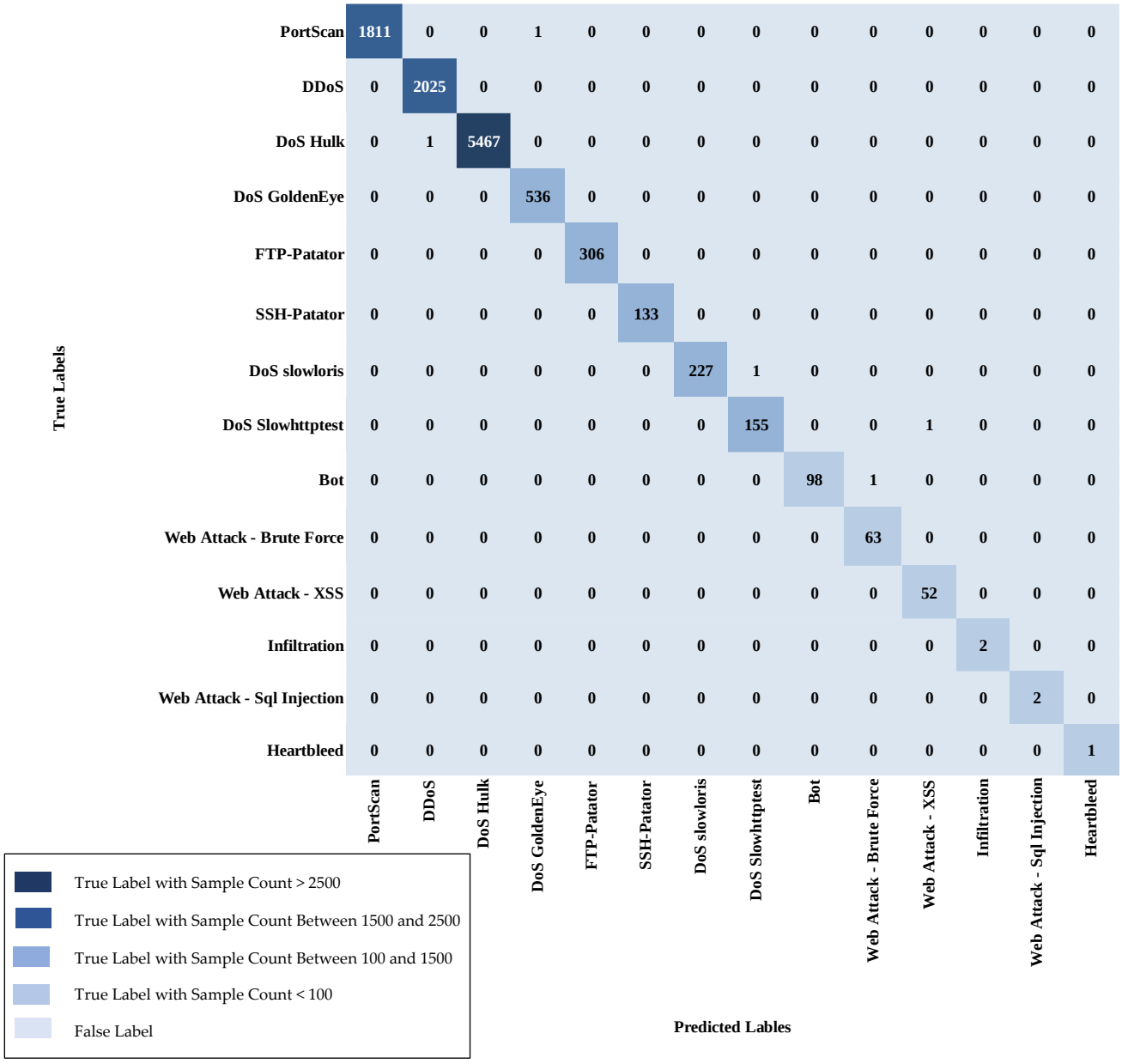

Figure 6. Confusion matrix for multi-class classification using Autoencoder–CNN on the CI-CIDS2017 dataset.

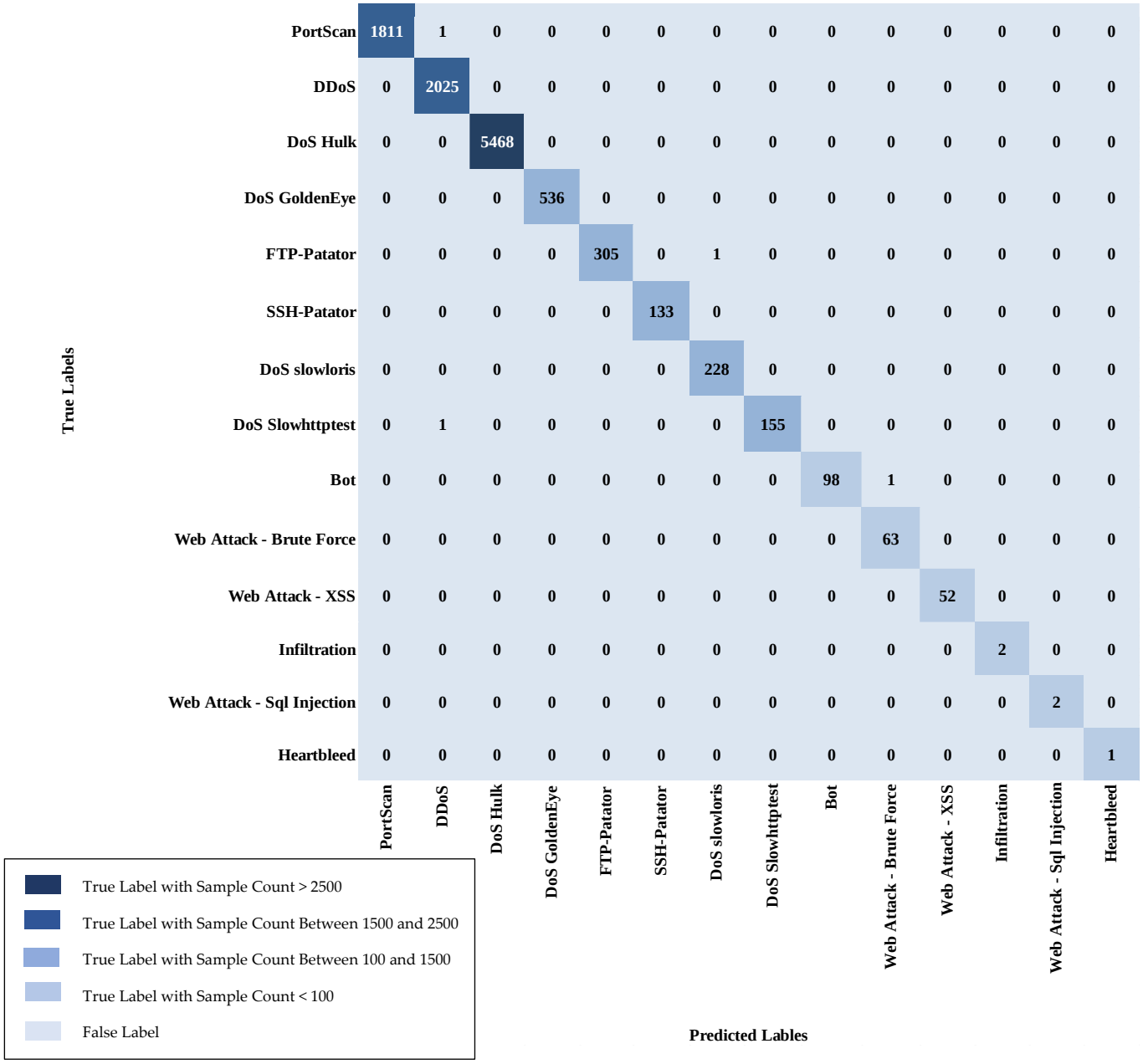

Figure 7. Confusion matrix for multi-class classification using Transformer–DNN on the CI-CIDS2017 dataset.

The Autoencoder–CNN and Transformer–DNN models demonstrates impressive performance in multi-class classification on the NF-BoT-IoT-v2 dataset. The Autoencoder–CNN achieved an overall accuracy of 97.95%, with precision, recall, and F1 score values of 97.97%, 97.95%, and 97.95%, respectively. The Autoencoder–CNN correctly identifies 2670 instances of Reconnaissance, 4154 instances of DDoS, 3475 instances of DoS, and 62 instances of Theft. The Transformer–DNN, with a slightly lower accuracy of 97.90%, achieved precision, recall, and F1 score values of 97.98%, 97.90%, and 97.90%, respectively. It correctly identifies 2633 instances of Reconnaissance, 4154 instances of DDoS, 3507 instances of DoS, and 62 instances of Theft. Both models misclassified some instances among the classes, as shown in Figure 8. Both models effectively minimized false positives and false negatives, demonstrating their robustness in real-world applications, especially when handling the complexities of imbalanced datasets.

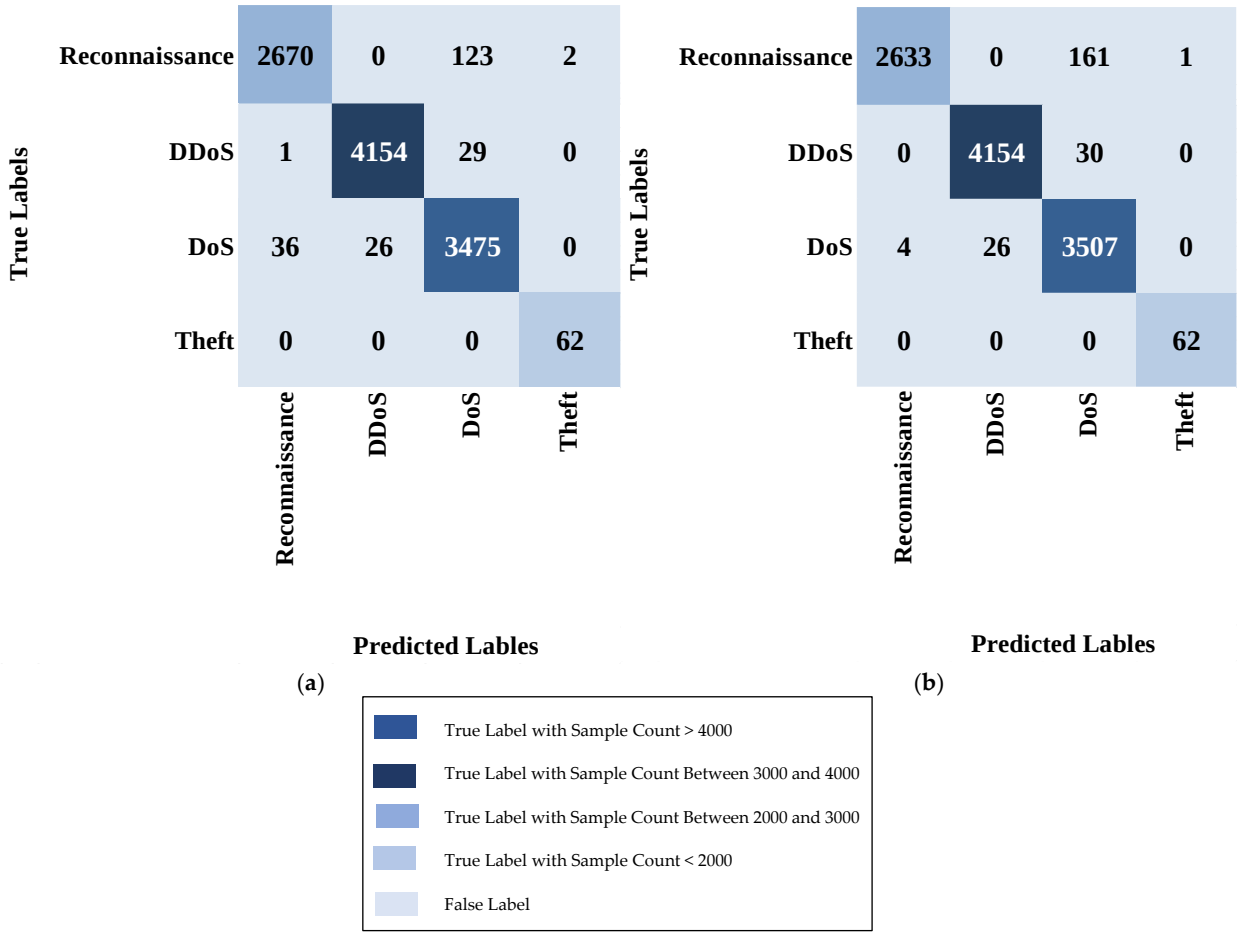

Figure 8. Confusion matrix for multi-class classification on the NF-BoT-IoT-v2 dataset using (**a**) Autoencoder–CNN and (**b**) Transformer–DNN.

The comparative performances of the proposed Transformer–DNN and Autoencoder–CNN models against other multi-class classifiers demonstrates their exceptional capability in handling complex classification tasks on the CICIDS2017 and NF-BoT-IoT-v2 datasets, as shown in Figures 9 and 10. The evaluation metrics, including accuracy, precision, recall, and F1 score, highlight the outstanding performance of both models. On the CICIDS2017 dataset, the Transformer–DNN achieved the highest scores across all metrics, with an impressive 99.96% accuracy, precision, recall, and F1 score. The Autoencoder-CNN model followed closely with scores of 99.95% in each metric. The CNN and DNN models each achieved 99.94% across all metrics, while the Autoencoder model scored slightly lower at 99.93%. On the NF-BoT-IoT-v2 dataset, the Autoencoder–CNN outperformed the Transformer–DNN, with slightly higher accuracy of 97.95% and precision, recall, and F1 scores of 97.97%, 97.95%, and 97.95%, respectively. The Transformer–DNN achieved accuracy of 97.90%, with precision, recall, and F1 scores of 97.98%, 97.90%, and 97.90%, respectively. The CNN model achieved 97.87% accuracy, with precision, recall, and F1 scores of 97.91%, 97.87%, and 97.87%, respectively. The DNN model achieved 97.83% accuracy, with precision, recall, and F1 scores of 97.88%, 97.83%, and 97.82%, respectively. The standalone Autoencoder scored 97.81% for accuracy, with precision, recall, and F1 scores of 97.87%, 97.81%, and 97.81%, respectively. These results underscore the effectiveness of integrating advanced architectures to enhance classification performance when using imbalanced datasets. The results also demonstrate the superiority of the Transformer–DNN and Autoencoder–CNN models in terms of reliable and efficient multi-class classification on both datasets.

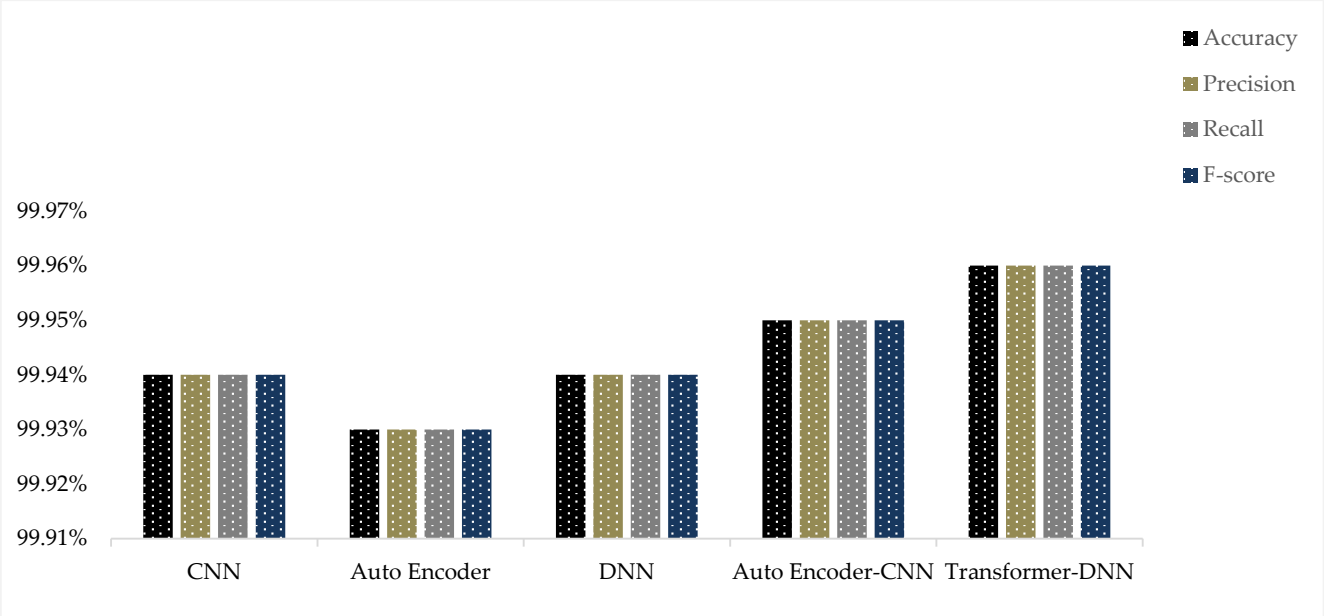

Figure 9. Proposed Transformer–DNN and Autoencoder–CNN versus multi-class classifiers on the CICIDS2017 dataset.

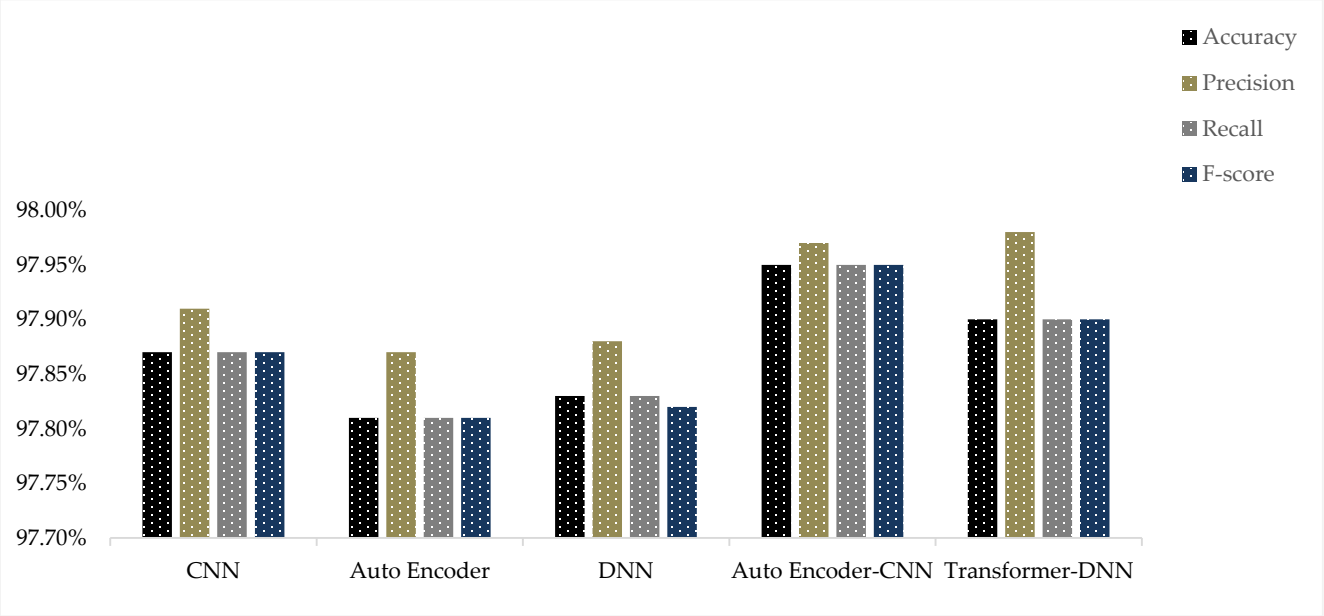

Figure 10. Proposed Transformer–DNN and Autoencoder–CNN versus multi-class classifiers on the NF-BoT-IoT-v2 dataset.

The Autoencoder–CNN model demonstrated exceptional performance in multi-class classification on the CICIDS2017 dataset, achieving remarkable accuracy and reliability across diverse attack categories, as detailed in Table 33. For certain classes such as FTP-Patator, SSH-Patator, infiltration, Web Attack—Sql Injection, and Heartbleed, the model achieved perfect scores in all metrics, including 100% accuracy, precision, recall, and F1 score, underscoring its capability to detect these threats flawlessly. For challenging attack types such as DoS Slowloris and DoS Slowhttptest, the model maintained robust performance with F1 scores of 99.78% and 99.36%, respectively. For classes like PortScan and DDoS, which often involve complex patterns, the model exceled with F1 scores of 99.97% and 99.98%, respectively, indicating its effectiveness in handling intricate attack

scenarios. Notably, F1 scores of 99.21% and 99.05% were achieved for Web Attack—Brute Force and Web Attack—XSS, reflecting the model's adeptness at identifying less frequent attack types. Overall, these comprehensive results affirm the Autoencoder–CNN model's robustness and suitability for real-world intrusion detection systems where precise and consistent performance is paramount.

Table 33. Performance metrics for Autoencoder–CNN across several classes in multi-class classification on the CICIDS2017 dataset.

Label	Accuracy	Precision	Recall	F-Score
PortScan	99.94%	100%	99.94%	99.97%
DDoS	100%	99.95%	100%	99.98%
DoS Hulk	99.98%	100%	99.98%	99.99%
DoS GoldenEye	100%	99.81%	100%	99.91%
FTP-Patator	100%	100%	100%	100%
SSH-Patator	100%	100%	100%	100%
DoS Slowloris	99.56%	100%	99.56%	99.78%
DoS Slowhttptest	99.36%	99.36%	99.36%	99.36%
Bot	98.99%	100%	98.99%	99.49%
Web Attack—Brute Force	100%	98.44%	100%	99.21%
Web Attack—XSS	100%	98.11%	100%	99.05%
Infiltration	100%	100%	100%	100%
Web Attack—Sql Injection	100%	100%	100%	100%
Heartbleed	100%	100%	100%	100%

The Transformer–DNN model demonstrated exceptional performance in multi-class classification on the CICIDS2017 dataset, achieving outstanding metrics across various attack classes, as detailed in Table 34. The model achieved perfect scores for several categories, including DoS Hulk, DoS GoldenEye, SSH-Patator, Web Attack—XSS, infiltration, Web Attack—Sql Injection, and Heartbleed, with 100% accuracy, precision, recall, and F1 scores. These results underscore the model's ability to detect these attack types with complete reliability and no misclassifications. For other attack classes, such as DDoS and PortScan, the model maintained exceptional performance, achieving F1 scores of 99.95% and 99.97%, respectively. Strong metrics were also displayed for challenging attack types including DoS Slowloris and DoS Slowhttptest, with F1 scores of 99.78% and 99.68%, respectively. Additionally, FTP-Patator and Bot attacks were associated with F1 scores of 99.84% and 99.49%, demonstrating the model's capacity to handle diverse patterns of malicious behavior. Even for Web Attack—Brute Force, the model achieved an impressive F1 score of 99.21%. These comprehensive metrics highlight the Transformer–DNN model's robustness in handling complex multi-class classification tasks, making it highly suitable for real-world intrusion detection systems where accurate and reliable detection across various attack types is essential.

The performance metrics presented in Table 35 demonstrate the remarkable capabilities of the Autoencoder–CNN and Transformer–DNN models for multi-class classification using the NF-BoT-IoT-v2 dataset. The Autoencoder–CNN model demonstrated strong performance across all classes, with the highest accuracy of 100% for the Theft class, precision of 96.88%, and recall of 100%, yielding an F1 score of 98.41%. For the Reconnaissance class, it achieved 95.53% accuracy, with precision of 98.63%, recall of 95.53%, and an F1 score of 97.06%. The DDoS class was associated with an accuracy of 99.28%, with precision and recall values of 99.38% and 99.28%, respectively, resulting in an F1 score of 99.33%. For the DoS class, accuracy was 98.25%, with precision of 95.81%, recall of 98.25%, and an F1

score of 97.01%. The Transformer–DNN model performed comparably, with impressive metrics. For the Theft class, it achieved 100% accuracy, 98.41% precision, and 100% recall, with an F1 score of 99.20%. In the Reconnaissance class, it showed 94.20% accuracy, 99.85% precision, and 94.20% recall, with an F1-score of 96.94%. In the DDoS class, it matched the Autoencoder–CNN's results with an accuracy of 99.28%, precision of 99.38%, recall of 99.28%, and an F1 score of 99.33%. It also demonstrated strong performance in the DoS class, achieving 99.15% accuracy, 94.84% precision, 99.15% recall, and an F1 score of 96.95%. Overall, both models exhibited robust performance in identifying and classifying different types of attacks, with the Autoencoder–CNN excelling in certain classes and the Transformer–DNN showing slightly better results in others.

Table 34. Performance metrics for Transformer–DNN across several classes in multi-class classification on the CICIDS2017 dataset.

Label	Accuracy	Precision	Recall	F-Score
PortScan	99.94%	100%	99.94%	99.97%
DDoS	100%	99.90%	100%	99.95%
DoS Hulk	100%	100%	100%	100%
DoS GoldenEye	100%	100%	100%	100%
FTP-Patator	99.67%	100%	99.67%	99.84%
SSH-Patator	100%	100%	100%	100%
DoS Slowloris	100%	99.56%	100%	99.78%
DoS Slowhttptest	99.36%	100%	99.36%	99.68%
Bot	98.99%	100%	98.99%	99.49%
Web Attack—Brute Force	100%	98.44%	100%	99.21%
Web Attack—XSS	100%	100%	100%	100%
Infiltration	100%	100%	100%	100%
Web Attack—Sql Injection	100%	100%	100%	100%
Heartbleed	100%	100%	100%	100%

Table 35. Performance metrics for Autoencoder–CNN and Transformer–DNN across several classes in multi-class classification on the NF-BoT-IoT-v2 dataset.

Model	Label	Accuracy	Precision	Recall	F-Score
Autoencoder–CNN	Reconnaissance	95.53%	98.63%	95.53%	97.06%
	DDoS	99.28%	99.38%	99.28%	99.33%
	DoS	98.25%	95.81%	98.25%	97.01%
	Theft	100%	96.88%	100%	98.41%
Transformer–DNN	Reconnaissance	94.20%	99.85%	94.20%	96.94%
	DDoS	99.28%	99.38%	99.28%	99.33%
	DoS	99.15%	94.84%	99.15%	96.95%
	Theft	100%	98.41%	100%	99.20%

6. Limitations

The Transformer–DNN and Autoencoder–CNN represent advanced hybrid deep learning architectures designed to enhance performance in both binary and multi-class classification tasks. While this approach addresses key challenges in intrusion detection systems, such as improving accuracy and managing class imbalances, several limitations and challenges remain to be addressed, including the following:

- Scalability: As dataset sizes and the complexities of network traffic increase, computational requirements may escalate, potentially affecting the models' efficiency and ability to handle larger datasets or adapt to evolving network environments.
- Generalization: While the Transformer–DNN and Autoencoder–CNN demonstrated strong performance on the CICIDS2017 and NF-BoT-IoT-v2 datasets, their effectiveness on different types of network traffic or emerging attack vectors remains uncertain. Evaluating the models on a broader range of datasets, such as NSL-KDD, KDDCup99, UNSW-NB15, or newer ones like CSE-CIC-IDS2018 and NF-ToN-IoT-v2, is essential to gauge their robustness and generalization capability.
- Data Preprocessing: Implementing data preprocessing on different datasets is a critical step that involves tasks such as handling missing values, encoding categorical variables, normalizing or standardizing numerical features, and removing irrelevant information. The effectiveness of the models depends heavily on how well these preprocessing steps are executed.
- Model Adaptation: Adapting the models to different datasets involves a trial-and-error process for hyperparameter optimization. This process is crucial for fine-tuning the models to align with the characteristics of new datasets.

7. Conclusions

Developing an effective IDS is essential for businesses and organizations committed to safeguarding their networks. This study presents a robust IDS solution that leverages two advanced hybrid deep learning models, Transformer–DNN and Autoencoder–CNN, specifically engineered to tackle class imbalance issues. Integrating advanced data resampling techniques including enhanced hybrid ADASYN-SMOTE for binary classification and enhanced SMOTE for multi-class classification alongside ENN, the proposed model delivers exceptional performance. The Transformer–DNN achieved 99.92% accuracy in binary classification and 99.96% in multi-class classification on the CICIDS2017 dataset, while the Autoencoder–CNN model reached 99.90% accuracy in binary classification and 99.95% in multi-class classification. With the NF-BoT-IoT-v2 dataset, the Autoencoder–CNN reached 99.98% accuracy in binary and 97.95% in multi-class classification, and the Transformer–DNN achieved 99.98% and 97.90%, respectively. These remarkable results underscore the superiority of our approach over existing state-of-the-art methods. Through comprehensive evaluation using the CICIDS2017 and NF-BoT-IoT-v2 datasets, this study demonstrates the effectiveness and reliability of the proposed models, positioning it as an advanced solution for addressing class imbalance and enhancing the overall performance of IDSs.

8. Future Work

To overcome the limitations and challenges mentioned in Section 6, future research should focus on the following areas:

- Broader Dataset Evaluation: Future work should include evaluating the Transformer–DNN and Autoencoder–CNN with a wider range of datasets, including NSL-KDD, KDDCup99, UNSW-NB15, and newer ones like CSE-CIC-IDS2018 and NF-ToN-IoT-v2, to assess the models' robustness, generalization capability, and effectiveness against emerging attack vectors.
- Data Preprocessing Refinement: The data preprocessing steps should be further refined and tailored for each specific dataset, to ensure optimal model performance. This will involve experimenting with different preprocessing techniques and understanding their impact on model outcomes. Detailed analyses of these preprocessing operations are thoroughly discussed in Sections 3.2 and 4.1 of this article.

- Model Adaptation and Hyperparameter Optimization: Continued exploration of model adaptation techniques is necessary, with a focus on refining the hyperparameter optimization process for different datasets. This process should be systematically analyzed to identify the best practices for adapting the models to diverse data environments. These details are discussed in Section 3, particularly in Sections 3.3.4 and 3.3.5.
- Scalability and Computational Efficiency: Efforts should be made to optimize the models' computational efficiency and scalability, ensuring that they can handle larger datasets and more complex network traffic scenarios without compromising performance.

Author Contributions: Conceptualization, H.K. and M.M.; Methodology, H.K. and M.M.; Software, H.K. and M.M.; Validation, H.K. and M.M.; Writing—original draft, H.K. and M.M.; Supervision, M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The datasets used in our study, CICIDS2017 and NF-BoT-IoT-v2, are publicly available. Below are the URLs for the datasets: CICIDS2017: https://www.unb.ca/cic/datasets/ids-2017.html (accessed on 16 January 2025), NF-BoT-IoT-v2: https://staff.itee.uq.edu.au/marius/NIDS_datasets/#RA8 (accessed on 16 January 2025).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. TechTarget. 6 Common Types of Cyber Attacks and How to Prevent Them. 2023. Available online: https://www.techtarget.com/searchsecurity/tip/6-common-types-of-cyber-attacks-and-how-to-prevent-them (accessed on 21 January 2025).
2. Tariq, U.; Ahmed, I.; Bashir, A.K.; Shaukat, K. A critical cybersecurity analysis and future research directions for the internet of things: A comprehensive review. *Sensors* **2023**, *23*, 4117. [CrossRef] [PubMed]
3. BlackBerry. Quarterly Global Threat Report—September 2024. 2024. Available online: https://www.blackberry.com/us/en/solutions/threat-intelligence/threat-report (accessed on 21 January 2025).
4. Conti, M.; Dargahi, T.; Dehghantanha, A. *Cyber Threat Intelligence: Challenges and Opportunities*; Springer International Publishing: New York, NY, USA, 2018.
5. Osama, F.; Dogdu, E. Intrusion detection using big data and deep learning techniques. In *Proceedings of the 2019 ACM Southeast Conference*; AMC: New York, NY, USA, 2019; pp. 86–93.
6. Kaur, G.; Lashkari, A.H.; Rahali, A. Intrusion traffic detection and characterization using deep image learning. In Proceedings of the 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Calgary, AB, Canada, 17–22 August 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 55–62.
7. Internet Security Threat Report. Available online: https://docs.broadcom.com/doc/istr-23-2018-en (accessed on 18 July 2022).
8. Cybersecurity Ventures. Cybercrime to Cost the World $9 Trillion Annually in 2024. Available online: https://cybersecurityventures.com/cybercrime-to-cost-the-world-9-trillion-annually-in-2024/ (accessed on 21 January 2025).
9. Zhang, X.; Xie, J.; Huang, L. Real-Time Intrusion Detection Using Deep Learning Techniques. *J. Netw. Comput. Appl.* **2020**, *140*, 45–53.
10. Kumar, S.; Kumar, R. A Review of Real-Time Intrusion Detection Systems Using Machine Learning Approaches. *Comput. Secur.* **2020**, *95*, 101944.
11. Smith, A.; Jones, B.; Taylor, C. Enhancing Network Security with Real-Time Intrusion Detection Systems. *Int. J. Inf. Secur.* **2021**, *21*, 123–135.
12. UNB. Intrusion Detection Evaluation Dataset (CICIDS2017), University of New Brunswick. Available online: https://www.unb.ca/cic/datasets/ids-2017.html (accessed on 21 January 2025).
13. Sarhan, M.; Layeghy, S.; Portmann, M. Towards a standard feature set for network intrusion detection system datasets. *Mob. Netw. Appl.* **2022**, *27*, 357–370. [CrossRef]
14. Anderson, J.P. *Computer Security Threat Monitoring and Surveillance*; Technical Report; James P. Anderson Company: Washington, DC, USA, 1980.
15. Vinayakumar, R.; Alazab, M.; Soman, K.P.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access* **2019**, *7*, 41525–41550. [CrossRef]

16. Farhana, K.; Rahman, M.; Ahmed, M.T. An intrusion detection system for packet and flow based networks using deep neural network approach. *Int. J. Electr. Comput. Eng.* **2020**, *10*, 5514–5525. [CrossRef]
17. Razan, A.; Faezipour, M.; Musafer, H.; Abuzneid, A. Efficient network intrusion detection using PCA-based dimensionality reduction of features. In Proceedings of the 2019 International Symposium on Networks, Computers and Communications (ISNCC), Istanbul, Turkey, 18–20 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6.
18. Alsharaiah, M.A.; Abualhaj, M.; Baniata, L.H.; Al-Saaidah, A.; Kharma, Q.M.; Al-Zyoud, M.M. An innovative network intrusion detection system (NIDS): Hierarchical deep learning model based on Unsw-Nb15 dataset. *Int. J. Data Netw. Sci.* **2024**, *8*, 709–722. [CrossRef]
19. Jouhari, M.; Benaddi, H.; Ibrahimi, K. Efficient Intrusion Detection: Combining χ^2 Feature Selection with CNN-BiLSTM on the UNSW-NB15 Dataset. *arXiv* **2024**, arXiv:2407.14945.
20. Prabhakaran, V.; Kulandasamy, A. Hybrid semantic deep learning architecture and optimal advanced encryption standard key management scheme for secure cloud storage and intrusion detection. *Neural Comput. Appl.* **2021**, *33*, 14459–14479. [CrossRef]
21. Kao, M.-T.; Sung, D.-Y.; Kao, S.-J.; Chang, F.-M. A novel two-stage deep learning structure for network flow anomaly detection. *Electronics* **2022**, *11*, 1531. [CrossRef]
22. Fu, Y.; Du, Y.; Cao, Z.; Li, Q.; Xiang, W. A deep learning model for network intrusion detection with imbalanced data. *Electronics* **2022**, *11*, 898. [CrossRef]
23. Kamal, H.; Mashaly, M. Advanced Hybrid Transformer-CNN Deep Learning Model for Effective Intrusion Detection Systems with Class Imbalance Mitigation Using Resampling Techniques. *Future Internet* **2024**, *16*, 481. [CrossRef]
24. Alzughaibi, S.; El Khediri, S. A cloud intrusion detection systems based on dnn using backpropagation and pso on the cse-cic-ids2018 dataset. *Appl. Sci.* **2023**, *13*, 2276. [CrossRef]
25. Arslan, R.S. FastTrafficAnalyzer: An efficient method for intrusion detection systems to analyze network traffic. *Dicle Üniversitesi Mühendislik Fakültesi Mühendislik Derg.* **2021**, *12*, 565–572. [CrossRef]
26. Yaras, S.; Dener, M. IoT-Based Intrusion Detection System Using New Hybrid Deep Learning Algorithm. *Electronics* **2024**, *13*, 1053. [CrossRef]
27. ElKashlan, M.; Elsayed, M.S.; Jurcut, A.D.; Azer, M. A machine learning-based intrusion detec-tion system for iot electric vehicle charging stations (evcss). *Electronics* **2023**, *12*, 1044. [CrossRef]
28. Othman, T.S.; Abdullah, S.M. An intelligent intrusion detection system for internet of things attack detection and identification using machine learning. *ARO-THE Sci. J. KOYA Univ.* **2023**, *11*, 126–137. [CrossRef]
29. Wang, Y.; Li, J.; Zhao, W.; Han, Z.; Zhao, H.; Wang, L.; He, X. N-STGAT: Spatio-Temporal Graph Neural Network Based Network Intrusion Detection for Near-Earth Remote Sensing. *Remote Sens.* **2023**, *15*, 3611. [CrossRef]
30. Xu, R.; Wu, G.; Wang, W.; Gao, X.; He, A.; Zhang, Z. Applying self-supervised learning to network intrusion detection for network flows with graph neural network. *Comput. Netw.* **2024**, *248*, 110495. [CrossRef]
31. Li, F.; Shen, H.; Mai, J.; Wang, T.; Dai, Y.; Miao, X. Pre-trained language model-enhanced conditional generative adversarial networks for intrusion detection. *Peer-to-Peer Netw. Appl.* **2024**, *17*, 227–245. [CrossRef]
32. Silivery, A.K.; Kovvur, R.M.R.; Solleti, R.; Kumar, L.S.; Madhu, B. A model for multi-attack classification to improve intrusion detection performance using deep learning approaches. *Meas. Sens.* **2023**, *30*, 100924. [CrossRef]
33. Umair, M.B.; Iqbal, Z.; Faraz, M.A.; Khan, M.A.; Zhang, Y.-D.; Razmjooy, N.; Kadry, S. A network intrusion detection system using hybrid multilayer deep learning model. *Big Data* **2022**, *12*, 367–376. [CrossRef] [PubMed]
34. Türk, F. Analysis of intrusion detection systems in UNSW-NB15 and NSL-KDD datasets with machine learning algorithms. *Bitlis Eren Üniversitesi Fen Bilim. Derg.* **2023**, *12*, 465–477. [CrossRef]
35. Farhan, B.I.; Jasim, A.D. Performance analysis of intrusion detection for deep learning model based on CSE-CIC-IDS2018 dataset. *Indones. J. Electr. Eng. Comput. Sci.* **2022**, *26*, 1165–1172. [CrossRef]
36. Sarhan, M.; Layeghy, S.; Moustafa, N.; Portmann, M. Cyber threat intelligence sharing scheme based on federated learning for network intrusion detection. *J. Netw. Syst. Manag.* **2023**, *31*, 3. [CrossRef]
37. Ring, M.; Wunderlich, S.; Scheuring, D.; Landes, D.; Hotho, A. A survey of network-based intrusion detection data sets. *Comput. Secur.* **2019**, *86*, 147–167. [CrossRef]
38. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp 1* **2018**, *1*, 108–116.
39. Breunig, M.M.; Kriegel, H.-P.; Ng, R.T.; Sander, J. LOF: Identifying Density-Based Local Outliers. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 15–18 May 2000; Volume 29, pp. 93–104. [CrossRef]
40. Patro, S.G.K.; Sahu, K.K. Normalization: A preprocessing stage. *Int. Adv. Res. J. Sci. Eng. Technol.* **2015**, *2*, 20–22. [CrossRef]
41. Bagui, S.; Li, K. Resampling imbalanced data for network intrusion detection datasets. *J Big Data.* **2021**, *8*, 6. [CrossRef]
42. Elmasry, W.; Akbulut, A.; Zaim, A.H. Empirical study on multiclass classifcation-based network intrusion detection. *Comput Intell* **2019**, *35*, 919–954. [CrossRef]

43. Mbow, M.; Koide, H.; Sakurai, K. Handling class imbalance problem in intrusion detection system based on deep learning. *Int. J. Netw. Comput.* **2022**, *12*, 467–492. [CrossRef]
44. He, H.; Bai, Y.; Garcia, E.A.; Li, S. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In Proceedings of the 2008 IEEE international Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–8 June 2008; IEEE: Piscataway, NJ, USA, 1–8 June 2008; pp. 1322–1328.
45. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic Minority Over-sampling Technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. [CrossRef]
46. Wilson, D.L. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Trans. Syst. Man Cybern.* **1972**, *3*, 408–421. [CrossRef]
47. El-Habil, B.Y.; Abu-Naser, S.S. Global climate prediction using deep learning. *J. Theor. Appl. Inf. Technol.* **2022**, *100*, 4824–4838.
48. Song, Z.; Ma, J. Deep learning-driven MIMO: Data encoding and processing mechanism. *Phys Commun.* **2022**, *57*, 101976. [CrossRef]
49. Zhou, X.; Zhao, C.; Sun, J.; Yao, K.; Xu, M. Detection of lead content in oilseed rape leaves and roots based on deep transfer learning and hyperspectral imaging technology. *Spectroch Acta Part A Mol. Biomole Spectrosc.* **2022**, *290*, 122288. [CrossRef]
50. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
51. Nair, V.; Hinton, G.E. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, 21–24 June 2010; pp. 807–814.
52. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
53. Bishop, C.M.; Nasser, M.N. *Pattern Recognition and Machine Learning*; Springer: New York, NY, USA, 2006; Volume 4.
54. Nielsen, A. Neural Networks and Deep Learning. 2015. Available online: https://books.google.com.hk/books/about/Neural_Networks_and_Deep_Learning.html?id=STDBswEACAAJ&redir_esc=y (accessed on 16 January 2025).
55. Glorot, X.; Bordes, A.; Bengio, Y. Deep sparse rectifier neural networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 315–323.
56. Vaswani, A.; Noam, S.; Niki, P.; Jakob, U.; Llion, J.; Aidan, N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need.(Nips). *arXiv* **2017**, arXiv:1706.03762.
57. Ba, J.L. Layer normalization. *arXiv* **2016**, arXiv:1607.06450.
58. Jyothsna, V.; Prasad, K.M. Anomaly-Based Intrusion Detection System. Computer and Network Security 10. 2019. Available online: https://www.intechopen.com/chapters/67618 (accessed on 16 January 2025).
59. Chen, C.; Song, Y.; Yue, S.; Xu, X.; Zhou, L.; Lv, Q.; Yang, L. FCNN-SE: An Intrusion Detection Model Based on a Fusion CNN and Stacked Ensemble. *Appl. Sci.* **2022**, *12*, 8601. [CrossRef]
60. Powers, D.M.W. Evaluation: From Precision, Recall, and F-Measure to ROC, Informedness, Markedness & Correlation. *J. Mach. Learn. Technol.* **2011**, *2*, 37–63.