# ENHANCING INTRUSION DETECTION SYSTEMS USING DEEP LEARNING TECHNIQUES: A COMPARATIVE STUDY ON CICIDS 2017 DATASET

A Thesis

by

BYUNGKAK JEON

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

| | |
|---|---|
| Chair of Committee, | Drew Hamilton |
| Committee Members, | Martin Carlisle |
| | Jeyavijayan Rajendran |
| Head of Department, | Scott Schaefer |

August 2023

Major Subject: Computer Engineering

# ABSTRACT

Intrusion Detection Systems (IDS) play a vital role in detecting and thwarting harmful activities on the network. While traditional machine learning techniques have been employed in IDS, recent advancements in deep learning offer promising results for enhancing their performance. This study compares the effectiveness of various deep learning models applied to the CICIDS 2017 dataset, focusing on the impact of different sampling techniques on improving IDS efficiency and accuracy.

We evaluated seven deep learning models, including CNN Simple, CNN Deep, ANN, DNN, LSTM, GRU, and a hybrid model that we named CLAttNet, which combines Convolution, LSTM, and Attention mechanisms. The name CLAttNet was chosen for the sake of convenience in this study. We also investigated the impact of sampling techniques, such as No Resampling, Selective Oversampling, Selective Undersampling, Combined Sampling (SMOTE+Selective Undersampling), and SMOTE, on the performance of each model.

We assessed the models using a 5-fold cross-validation method, looking at Precision, Recall, Accuracy, F1-Score, ROC-AUC and Precision-Recall Curve AUC. Our findings demonstrate the potential of deep learning models to improve IDS performance by showing that the CLAttNet model consistently outperforms the other models, delivering the F1-Score (0.99288). Additionally, Our experiment demonstrates that the SMOTE sampling method is a highly effective sampling strategy that improves the

performance of most models, increasing IDS efficiency and accuracy, except for some models.

This study advances the field by showcasing how deep learning models and sampling strategies can improve IDS performance. The results provide a strong basis for further investigation to create deep learning-based IDS solutions that are more effective, precise, and scalable. The applications of our research indicate that employing CLAttNet can increase the performance of the IDS, hence boosting the ability to defend against the constantly changing environment of cyber threats.

# DEDICATION

To my beloved parents, wife, and children,

This thesis is dedicated to you, who have always been my greatest source of support and inspiration. Your unwavering love, encouragement, and sacrifices have been the driving force behind my academic journey, and I am forever grateful.

I would also like to dedicate this work to the Republic of Korea Army, which has provided me with invaluable opportunities to learn and grow as a researcher and a member of society.

Lastly, I dedicate this thesis to my advisor, Dr. Hamilton, for his guidance, mentorship, and unwavering support throughout my graduate studies. His insight and proficiency have been crucial in molding my scholarly pursuits and professional trajectory, and I can't express my gratitude enough.

With deepest gratitude and love,

Byungkak Jeon

# ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Hamilton, for his continuous guidance, support, and encouragement throughout my research journey. As a former U.S. Army officer who transitioned to academia, Dr. Hamilton has been a source of inspiration, drawing on his wealth of experience in both military and academic fields.

Our shared background as army officers, despite serving in different countries, has created a strong bond between us and provided many opportunities for valuable insights and advice. Dr. Hamilton's long-standing affection for Korea and the Korean military, combined with his extensive service in the country, has enriched our discussions and enhanced my research.

His expertise, knowledge, and enthusiasm for research have been truly inspiring, and I am deeply grateful for the opportunity to learn from him and work under his guidance. I would also like to thank him for sharing his military experiences and wisdom, which have been of great help and motivation throughout my academic journey.

With utmost appreciation,

Byungkak Jeon

# CONTRIBUTORS AND FUNDING SOURCES

## Contributors

This thesis was supported by a thesis committee consisting of Professor Drew Hamilton, my advisor, Professor Martin Carlisle of the Department of Computer Science and Engineering, and Professor Jeyavijayan Rajendran of the Department of Electrical & Computer Engineering at Texas A&M University. I would like to express my sincere gratitude for their guidance, insightful feedback, and contributions to this work.

## Funding Sources

All of my graduate research has been supported by the Republic of Korea Army and the Texas A&M University Cybersecurity Center. The financial support provided by these organizations was essential in conducting this research, and I would like to express my sincere appreciation for their support. No other sources of funding were received for this study.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

# CHAPTER I

# INTRODUCTION

## 1.1 Background and Motivation

The exponential rise of the internet and digital technologies has led to an escalated frequency and complexity of cyber threats. [1] These threats imperil the integrity of information systems and networks, inflicting substantial monetary and reputational losses on individuals, corporations, and governments. Consequently, intrusion detection systems (IDS) have become indispensable for tracking and pinpointing malicious activities within network traffic. However, conventional signature-based IDSs struggle with identifying new and unidentified attacks, underscoring the need for more innovative and flexible intrusion detection methods. [1], [2]

Deep learning and machine learning techniques have come to the fore as robust solutions to the challenges faced by traditional IDS. Their ability to discern known and previously unseen threats stems from their capacity to learn from extensive datasets, recognizing patterns and correlations indicative of cyber threats. [3] Recent studies suggest deep learning models can detect intrusion with impressive accuracy rates and minimal false alarms. However, considerable scope still exists for enhancing these models' performance, utility, and adaptability.

The CICIDS 2017 dataset, with its comprehensiveness, meticulous annotations, and diverse attack patterns, makes for an excellent resource for assessing the efficacy of machine learning and deep learning models in intrusion detection scenarios. [4] This

research aims to evaluate and compare various deep learning models and sampling techniques to identify the most promising strategies for detecting and classifying cyberattacks within the CICIDS 2017 dataset. [4] Through this endeavor, we aim to develop more accurate and efficient IDS, thereby fortifying networks and information systems against the onslaught of cyber threats.

## 1.2 Research Objectives

The primary purpose of this study revolves around exploring the efficiency of various deep learning models and sampling methods in intrusion detection, utilizing the CICIDS 2017 dataset. To fulfill this purpose, we focus on these specific research aims:

1. We aim to conduct an extensive analysis of the CICIDS 2017 dataset, which includes understanding the different types of attacks, their implications in the real world, and the crucial features linked to each attack type.
2. Our study focuses on implementing and evaluating seven distinct deep learning models (namely CNN-Simple, CNN-Deep, ANN, DNN, LSTM, GRU, and CLAttNet) for intrusion detection. We aim to compare their performance concerning accuracy, recall, and other pertinent metrics.
3. We strive to scrutinize the effects of different sampling methods (No Resampling, Selective Oversampling, Selective Undersampling, Combined Sampling, and SMOTE) on the effectiveness of deep learning models for intrusion detection, particularly considering the challenge of imbalanced data.

4. Lastly, we seek to pinpoint the shortcomings of current deep learning models for intrusion detection, proposing potential enhancements and future research directions to boost their proficiency and precision.

By fulfilling these research aims, we aspire to contribute to the development of more efficient and versatile intrusion detection systems, thereby strengthening the protection of our information systems and networks against a myriad of cyber threats.

### 1.3 Organization of the Paper

This paper is organized into the following chapters:

- **Chapter I - Introduction**: This chapter provides background information and motivation for the research, outlines the research objectives.
- **Chapter II - Literature Review**: We summarize existing research that we consulted to gain knowledge for our study, including machine learning and deep learning models, sampling techniques, and model performance evaluation measures.
- **Chapter III - CICIDS 2017 Dataset**: We will describe how the CICIDS 2017 dataset was generated and its attack labels. We will also describe the preprocessing required to apply this dataset to the deep learning model we will run.
- **Chapter IV - Methodology**: Describe the general experimental methodology. This includes the structure of the deep learning model, the sampling technique applied, the model performance evaluation measures, and the experimental environment settings.

- **Chapter V - Results and Discussion**: We analyze the experimental results from various angles here. Along with the results of the Cross-Validation and Test Set, each model's most optimized sampling method will also be shown.
- **Chapter VI - Conclusion**: This is the concluding section of the study. We summarize our findings from our experiments and discuss future research directions.

Additionally, the paper includes a list of references and appendices that provide supplementary information related to the study.

CHAPTER II

LITERATURE REVIEW

## 2.1 Intrusion Detection System (IDS)

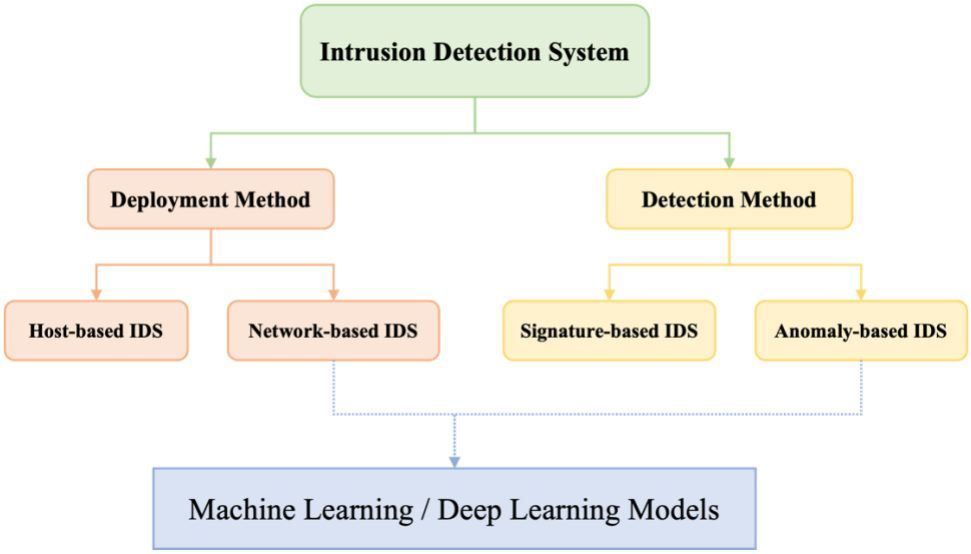


**Figure 1 IDS Classification Taxonomy Reprinted from [6, Fig 4]**

### *2.1.1 IDS Overview*

IDS is an abbreviation for intrusion detection system. [2] When someone gains illegal access to data stored on a computer or network system, its integrity, confidentiality, or availability may be jeopardized. [5] While a detection system is a security measure for identifying such criminal activities. IDS is a type of security technology that constantly monitors host and network traffic for irregularities that can violate security policy and endanger the confidentiality, integrity, and availability of data. [3], [5] The host or network administrators will get notifications from the IDS regarding any identified malicious activities. [3], [5]

IDS can be classified according to the deployment and detection methods, and the specific classification is given in Figure 1. IDS is again split into host-based and network-based IDS according to the deployment method. Additionally, the detection method's IDS is split into two categories: Signature-based and Detection-based IDS. [3], [6]

### *2.1.2 Deployment Method Classification*

From the perspective of deployment-based IDS, IDS is further split into host-based-IDS (HIDS) and network-based-IDS (NIDS). [3] On the single information host, HIDS is installed. It is responsible for monitoring all activity on this one host and scanning for any infractions of security guidelines or suspicious activity. [3] The most significant disadvantage is that it must be installed on every host that needs intrusion protection, which adds extra processing costs to every node and lowers the IDS's overall performance. [3] In contrast, NIDS is set up on the network to protect each device and the entire network against intrusions. [3] Continuous network traffic monitoring by the NIDS will look for any potential security violations. [3]

### *2.1.3 Detection Method Classification*

From a detection IDS standpoint, SIDS and AIDS are subdivided. [3] SIDS is centered on creating a signature for attack patterns. [3] These signatures are maintained in the signature database and used to identify attacks. [3] Due to signatures, known attacks may be detected efficiently. With signature patterns, this approach can identify new threats. A massive signature database is kept and examined with data packets for potential

intrusions, which is resource intensive. AIDS, often dubbed "behavior-based IDS," defines typical conduct. Any variation from this usual profile is abnormal. [6] AIDS can identify unknown and new assaults and customizes its typical activity profile for different networks and apps. [6] The significant FAR (False Alarm Rate) makes distinguishing between normal and aberrant incursion characteristics difficult. [3], [6]

### *2.1.4 Signature-based Detection*

Misuse detection, also known as signature-based detection, looks for potential incidents by comparing signatures to observed events. [3] A signature is a pattern that is associated with a recognized threat. It has the potential to be very effective in identifying known risks, but it is largely ineffective at identifying unidentified threats. Signature-based solutions require assistance to track and comprehend the status of complex communications as well as a deeper comprehension of numerous network or application protocols. [3]

### *2.1.5 Anomaly-based Detection*

To find significant deviations, anomaly-based detection contrasts definitions of what behavior is normal against observed events. [3] Many behavioral characteristics, such as the volume of emails a user writes, are the subject of profiles. A length of time, commonly referred to as a training period, is used to build an initial profile (usually days, but occasionally weeks). Anomaly-based IDS products usually produce a substantial number of false positives since harmless behavior regularly deviates greatly from profiles,

especially in environments that are more diverse or dynamic. Despite being vulnerable to evasion efforts from attackers, dynamic profiles do not have this issue. Small-scale malicious activity may be infrequently performed by an attacker before the frequency and volume of the activity gradually rise. [3]

### 2.2 Machine Learning and Deep Learning Techniques for IDS

In recent years, various machine learning and deep learning techniques have been applied to intrusion detection systems (IDS) to improve their detection capabilities. [3] This section briefly describes some of the most common techniques and their potential impacts on IDS dataset analysis.

- **Naïve Bayes**: Naïve Bayes is a probabilistic classifier based on Bayes' theorem. It has been widely used in IDS due to its simplicity and efficiency. Studies have shown that Naïve Bayes can perform well on IDS datasets, especially in detecting specific types of attacks with high accuracy. [3]

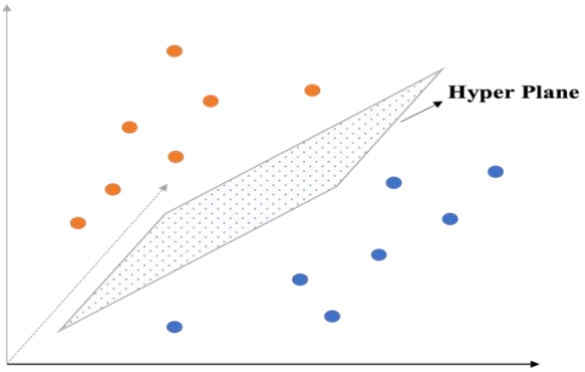


**Figure 2 Support Vector Machine (SVM)**

- **Support Vector Machines (SVM):** SVM is a high-dimensional space efficient supervised learning technique. [7]IDS has used it to classify network traffic as usual or malicious. [3] The algorithm's ability to handle large feature sets makes it suitable for IDS dataset analysis. [3], [7]
- **K-Nearest Neighbors(kNNs)**: A straightforward machine learning technique called kNNs is employed in intrusion detection systems to categorize unlabeled observations by contrasting their features with those of tagged samples. [3], [8] It can identify the category of an observation based on the category that receives the most votes from the nearest samples. It is a straightforward method for classification tasks. [3], [8]
- **Decision Tree (DT)**: In intrusion detection systems, classification systems or prediction algorithms are constructed up using the machine learning technique known as a decision tree. [3], [9] It can handle large, complex datasets without parameters by dividing data into branches. A dataset that has been divided into training and validation sets can be used to train the algorithm and choose the ideal tree size. It is a flexible tool for accurate and efficient detection of intrusion attempts. [3], [9]

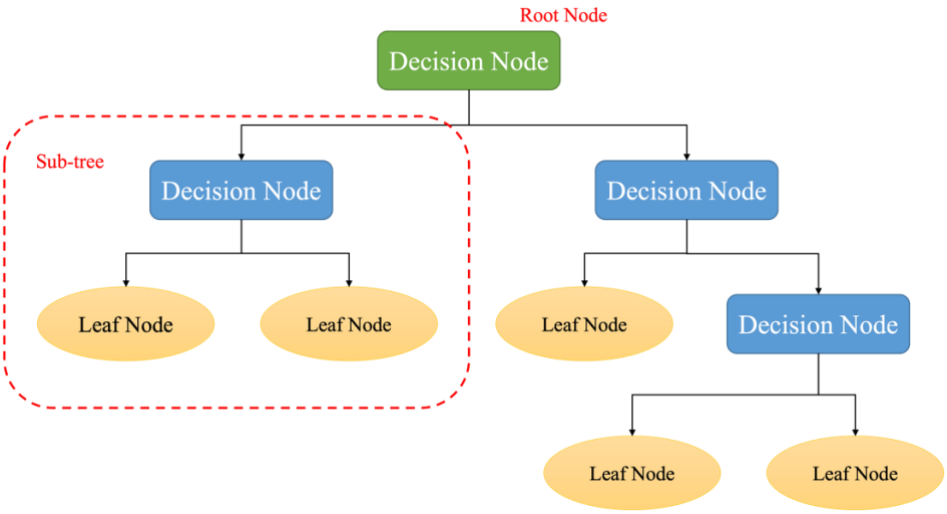


**Figure 3 Decision Tree (DT) Reprinted from [48, Fig 1]**

- **Artificial Neural Networks (ANN)**: ANN is a computational model inspired by the structure and functioning of biological neural networks. [10] ANN models have been employed in IDS to capture complex patterns and relationships in network traffic data, leading to improved detection performance. [10], [11]
- **Deep Neural Networks (DNN)**: DNNs are multilayer ANNs with more complex architectures, allowing for better representation learning. [10] DNNs have been applied to IDS to capture intricate relationships in network traffic data and improve detection accuracy. [10]

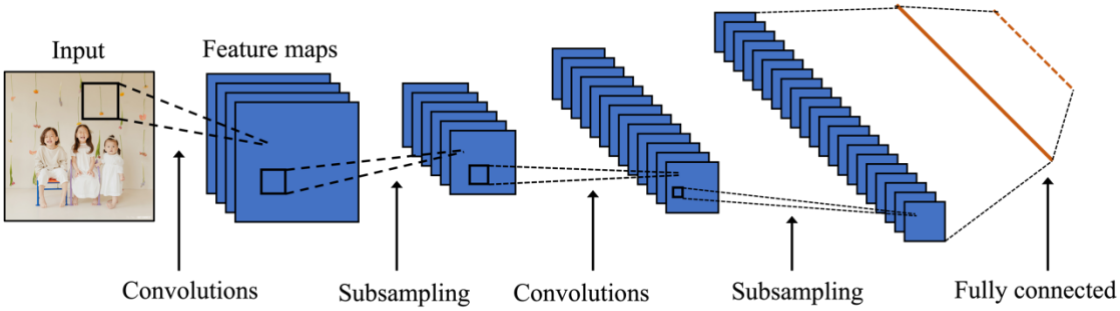


**Figure 4 Convolutional Neural Networks (CNN)**

- **Convolutional Neural Networks (CNN)**: Originally designed for image processing, CNNs have been widely adapted for IDS datasets to detect patterns in network traffic. [12] Some studies suggest that CNNs can effectively recognize patterns in network traffic data, making them a promising technique for IDS. [12]
- **Long Short-Term Memory (LSTM)**: LSTM is a type of recurrent neural network (RNN) designed to address the vanishing gradient problem in RNNs. [13] LSTM models have been used in IDS to capture temporal dependencies in network traffic data, which can lead to better detection of attack patterns over time. [13]
- **Gated Recurrent Units (GRU)**: GRU is another type of RNN that aims to solve the vanishing gradient problem. [14] Like LSTM, GRU models have been employed in IDS to capture temporal patterns in network traffic data, potentially improving detection performance. [14]

This study focuses on deep learning techniques, specifically CNN, ANN, DNN, LSTM, and GRU models. While the fundamental principles and characteristics of these models are briefly described here, we will further discuss their impact on IDS dataset

analysis, including their performance on the CICIDS 2017 dataset, in the subsequent sections of the paper. [3], [4]

### 2.3 Sampling Techniques for Imbalanced Data

Handling imbalanced data is crucial in intrusion detection systems (IDS), as it can significantly impact the performance of machine learning and deep learning models. [1] This section will offer a quick summary of the common sampling procedures used to address imbalanced data. Then, we will focus on the theoretical background of these techniques and their general application. [15]

1. **No Resampling**: This approach involves using the original dataset without any modifications. This strategy preserves the original data distribution, however it may not be the best choice for datasets that are imbalanced because it could produce biased models that perform poorly for minority classes. [2]
2. **Oversampling**: Oversampling includes adding additional samples to minority classes in order to achieve a balanced distribution of classes. This can be done by randomly duplicating instances from the minority class or generating synthetic instances. The Synthetic Minority Oversampling Technique (SMOTE) is a typical oversampling method that produces synthetic instances by interpolating between existing minority class instances and their nearest neighbors. [15], [16]
3. **Undersampling**: Undersampling involves decreasing the number of instances in majority classes to balance the class distribution. This can be done by randomly removing instances from the majority class or using more sophisticated techniques,

such as Tomek Links or Neighborhood Cleaning Rule, which consider the distribution and relationship between instances. [15], [17]

4. **Combined Resampling**: This approach involves a combination of Oversampling and Undersampling techniques. It aims to balance the class distribution while maintaining sufficient data for model training. Combined resampling can help create more robust models that perform well on both majority and minority classes. [18]
5. **Adaptive Sampling**: Adaptive sampling techniques, such as Adaptive Synthetic Sampling (ADASYN), dynamically adjust the sampling strategy based on the learning difficulty of each instance. These methods are designed to generate more synthetic instances for minority classes that are harder to learn, thus helping the model perform better on such instances. [15]

### 2.4 Performance Metrics for Model Evaluation

Model evaluation requires various performance metrics to be considered. In this part, we will present an overview of popular performance indicators used to evaluate machine learning models.

1. **Accuracy**: The ratio of cases that were successfully categorized to all instances is known as accuracy. Although it is a widely used metric, it may not be suitable for unbalanced datasets since it can be deceiving when the dominant class is the majority. [19]

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

2. **Precision**: Precision is defined as the ratio of true positives (properly identified positive cases) to the sum of true positives and false positives (instances incorrectly classified as positive). If its accuracy is high, the model is effective in avoiding false positives. [19]

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

3. **Recall (Sensitivity)**: Recall is the ratio of true positives to the sum of true and false negatives (instances incorrectly classified as unfavorable). High recall indicates that the model detects positive occurrences effectively. Recall is important in some applications, such as Intrusion Detection Systems, since it underlines the model's capacity to identify hostile actions, even if some false positives occur. [19]

$$Recall(Sensitivity) = \frac{TP}{TP + FN}$$

4. **F1-Score**: This performance measurement metric is the harmonic mean of Precision and Recall. The ratio of true positives to the sum of true and false negatives is referred to as recall (instances incorrectly classified as unfavorable). A high recall indicates that the model is able to recognize positive occurrences. It is beneficial for imbalanced datasets where the minority class is of interest. [19]

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

5. **Confusion Matrix**: A confusion matrix is a table that displays the number of TP(True Positives), FP(False Positives), TN(True Negatives), and FN(False Negatives) for each class. It offers a thorough look at the model's performance and can be used to pinpoint particular areas that need work. [20]

6. **ROC (Receiver Operating Characteristic) Curve**: For various decision thresholds, the ROC curve shows the true positive rate (recall) versus the false positive rate. The model's capacity to distinguish between positive and negative examples is indicated by the area under the ROC curve (AUC-ROC). AUC-ROC values of 1.0 and 0.5 are equivalent to perfect models and random guessing, respectively. [21]
7. **Precision-Recall Curve**: It is a valuable evaluation tool in machine learning, especially for imbalanced datasets. It describes the trade-off between two essential metrics for classification tasks: precision and recall. The curve provides a comprehensive view of model performance by plotting precision against recall for various threshold settings. It is beneficial in domains such as information retrieval, natural language processing, and bioinformatics, where precision and recall are more critical than mere accuracy. [21]

These performance metrics will serve as a foundation for understanding how the models are evaluated and compared in this study's experimental and analysis sections.

# CHAPTER III

# CICIDS 2017 DATASET*

## 3.1 Overview of the Dataset

The CICIDS 2017 dataset was created to overcome the shortcomings of obsolete and unreliable datasets while assessing anomaly-based intrusion detection systems. This dataset comprises up-to-date and realistic benign traffic and common attack scenarios, providing a reliable benchmark for researchers and practitioners. [4], [22]

The data gathering period they've had covered five days, from July 3 to July 7, 2017, with the first day consisting just of good traffic and the following days including a variety of attacks, including Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DdoS. [4], [22]

The dataset was built by simulating essential human interactions and background traffic using a B-Profile system. The network traffic is comprised of 25 users' HTTP, HTTPS, FTP, SSH, and email connections. The dataset is shown as labeled flows according to timestamps, source and destination IP addresses, ports, protocols, and attack types. CICFlowMeter was used to analyze network traffic. [4], [22]

---

* Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization", 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018

The publicly accessible dataset comprises of CSV files appropriate for machine learning and deep learning, annotated network flows, and pcap files with entire packet contents.

## 3.2 Attack Types and Scenarios

### *3.2.1 Description of Each Attack Types*

**Table 1 CICIDS 2017 Initial Label Distribution**

| ***Label*** | ***Entries*** |
|---|---|
| *Benign* | 2,271,320 |
| *DoS Hulk* | 230,124 |
| *Port Scan* | 158,804 |
| *DdoS* | 128,025 |
| *DoS GoldenEye* | 10,293 |
| *FTP-Patator* | 7,935 |
| *SSH-Patator* | 5,897 |
| *DoS Slowloris* | 5,796 |
| *DoS Slowhttptest* | 5,499 |
| *Bot* | 1,956 |
| *Web Attack: Brute Force* | 1,507 |
| *Web Attack: XSS* | 652 |
| *Infiltration* | 36 |
| *Web Attack: SQL Injection* | 21 |
| *Heartbleed* | 11 |

In the CICIDS 2017 dataset, various attack types are represented, providing a comprehensive and diverse evaluation platform. The descriptions of each attack type and its related label are provided below: [4], [22]

- **DoS Hulk**: A Denial-of-Service (DoS) attack designed to overwhelm web servers by generating a high volume of unique and seemingly legitimate traffic, eventually exhausting the server's resources, and rendering it unresponsive. [4]

- **PortScan**: This attack scans a range of IP addresses and ports to identify open ports and running services, which can then be exploited by attackers to gain unauthorized access or launch further attacks. [4]
- **DDoS**: Distributed Denial-of-Service, or DDoS attacks, represent a concerted effort by multiple systems aiming at a singular target system. The intent is to inundate the target with excessive demands, thereby exhausting its resources and creating a service disruption for its authentic users. [4]
- **DoS GoldenEye**: A DoS attack that exploits vulnerabilities in the HTTP protocol by sending numerous partial requests, effectively exhausting a server's resources, and causing it to become unresponsive. [4]
- **FTP-Patator**: A brute force attack targeting File Transfer Protocol (FTP) services, attempting multiple username and password combinations to gain unauthorized access. [4]
- **SSH-Patator**: Like FTP-Patator, this attack focuses on Secure Shell (SSH) services, attempting multiple username and password combinations to gain unauthorized access. [4]
- **DoS Slowloris**: A Slowloris attack is a type of DoS assault that gradually transmits incomplete HTTP requests, slowly exhausting a server's resources until it can no longer respond. [4]
- **DoS Slowhttptest**: A variant of Slowloris, this DoS attack sends partial HTTP requests slowly to exhaust server resources and render it unresponsive. [4]

- **Bot**: A botnet attack involves a network of compromised systems controlled by a central entity (botmaster) that carries out various malicious activities, such as DdoS attacks, spamming, or data theft. [4]
- **Web Attack: Brute Force**: This attack targets web applications by attempting numerous username and password combinations to gain unauthorized access to sensitive data or administrative privileges. [4]
- **Web Attack: XSS (Cross-Site Scripting)**: An attack that injects malicious scripts into legitimate websites, enabling attackers to steal sensitive user data, deface websites, or redirect users to malicious sites. [4]
- **Infiltration**: An attack in which a malicious actor gains unauthorized access to a network or system and remains undetected to perform reconnaissance, exfiltrate data, or launch further attacks. [4]
- **Web Attack**: **SQL Injection**: An assault on web applications that takes advantage of database systems by introducing harmful SQL instructions, thereby enabling unsanctioned access, data leakage, or data corruption. [4]
- **Heartbleed**: A critical vulnerability in the OpenSSL cryptographic software library allows an attacker to read the memory of systems using vulnerable versions of OpenSSL and potentially steal sensitive information, such as user credentials or encryption keys. [4]

### *3.2.2 Real-world Scenarios of Attacks*

Considering my background as a military officer (Army Major) in the South Korean Army, where I previously worked in the cyber security department of the Defense

Counterintelligence Command, I am familiar with various attack types and real-life scenarios. The Defense Counterintelligence Command works closely with South Korea's Cyber Command, which is responsible for formulating cyber security policies within the Ministry of National Defense, investigating security incidents, and conducting regular security checks. In this section, I will present real-world attack scenarios specifically related to the South Korean military, considering my experience as a South Korean military officer. [23]

**Scenario 1: Compromised Internal Network**

In 2016, the South Korean military faced a substantial cyber attack that compromised its internal network. Based on this incident, let us consider a scenario where an attacker injects malicious code into the military's internal network by exploiting a vulnerability in the network's security system. This attack could lead to unauthorized access and exfiltration of sensitive military information. [23]

In this case, the attacker might use a combination of the following attack types:

- **PortScan**: To identify open ports and running services within the military's network infrastructure.
- **Infiltration**: To gain unauthorized access and remain undetected in the network.
- **Web Attack: SQL Injection**: To manipulate database systems and exfiltrate sensitive data.

**Scenario 2: DdoS Attack on Military Communication Systems**

In this scenario, an attacker launches a large-scale DdoS attack targeting the military's communication systems, aiming to disrupt communication between various units and cause chaos during a critical operation. This attack could employ the following attack types:[23]

- **DoS Hulk**: To overwhelm the military's web servers and render them unresponsive.
- **DoS GoldenEye**: To exploit HTTP protocol vulnerabilities and exhaust server resources.
- **Bot**: To use a botnet to carry out a massive DDoS attack on the communication systems.

**Scenario 3: Spear-phishing Campaign**

A spear-phishing campaign targets high-ranking military officials with well-crafted and seemingly legitimate emails containing malicious attachments or links. These emails aim to trick the officials into providing their login credentials or downloading malware that grants unauthorized access to sensitive information. The attacker could use the following attack types:[24]

- **Web Attack: Brute Force**: To attempt multiple username and password combinations to gain unauthorized access.
- **Web Attack: XSS (Cross-Site Scripting)**: To insert harmful scripts into trustworthy websites, allowing the attacker to steal user information or reroute visitors to dangerous domains.

These real-world scenarios, while fictional, are based on my experience as a South Korean military officer and emphasize the importance of understanding various attack types and being prepared to defend against them. We can better secure our military networks and systems against potential threats by studying these attack types and their real-world applications.

## 3.3 Data Preprocessing and Feature Selection

This section elaborates on the data preprocessing and feature selection methods applied to the CICIDS 2017 dataset. These processes are essential for assuring the accuracy of the input data, lowering dimensionality, and improving the performance of the machine learning models on the supplied dataset.

### *3.3.1 Data Preprocessing*

In order to make raw data ready for analysis, it must first be cleaned, transformed, and organized. For example, for the CICIDS 2017 dataset, we performed the following preprocessing steps:

1. **Data Consolidation**: The CICIDS 2017 dataset is initially provided in multiple CSV files. We combined these files into a single pandas data frame to facilitate further analysis and processing.
2. **Column Naming**: The dataset consists of numerous features, each representing a different aspect of network traffic. We assigned appropriate column names to these features, ensuring the dataset was organized and easily understandable.

3. **Duplicate Columns**: In some cases, the dataset may contain duplicate columns, which can negatively impact the performance of machine learning models. We identified and removed any such duplicate columns from the dataset.
4. **Handling Missing Values**: The presence of missing or infinite values in the dataset can lead to biased or inaccurate results. We replaced infinite values with NaN and dropped rows containing missing values to maintain data integrity.
5. **Reducing Imbalanced Classes**: The CICIDS 2017 dataset suffers from class imbalance, with some minor classes having very few samples. We addressed this issue by removing classes such as Heartbleed, Web Attack – SQL Injection, and Infiltration, which had insufficient samples. [15]
6. **Renaming Labels**: We simplified the labels for Web Attack – Brute Force and Web Attack – XSS by renaming them to Brute Force and XSS, respectively. This change made the dataset more consistent and easier to work with.
7. **Saving Processed Dataset**: After completing the preprocessing steps, we saved the resulting dataset into a new CSV file for further analysis and used in the subsequent stages of our study.

**Table 2 CICIDS 2017 Label Distribution after Preprocessing**

| *Label* | *Entries* |
|---|---|
| *Benign* | 2,271,320 |
| *DoS Hulk* | 230,124 |
| *Port Scan* | 158,804 |
| *DdoS* | 128,025 |
| *DoS GoldenEye* | 10,293 |
| *FTP-Patator* | 7,935 |
| *SSH-Patator* | 5,897 |
| *DoS Slowloris* | 5,796 |
| *DoS Slowhttptest* | 5,499 |
| *Bot* | 1,956 |
| *Brute Force* | 1,507 |
| *XSS* | 652 |

### *3.3.2 Feature Selection for CICIDS 2017*

Feature selection selects a subset of the most relevant features significantly contributing to the model's performance. [25] By applying feature selection techniques to the CICIDS 2017 dataset, we aimed to reduce overfitting, improve accuracy, and minimize training time. [26] The following feature selection techniques were used:

1. **Splitting the dataset**: Using stratified sampling, we divided the CICIDS 2017 dataset into training, testing, and validation sets. This method ensures that each subset's class distribution is maintained, providing a more accurate representation of the overall dataset.
2. **Removing Constant Features**: Features with only a single unique value do not contribute meaningful information to the model and can hinder its performance. We identified and removed such features from the CICIDS 2017 dataset. [15]

3. **Data Normalization**: We applied MinMaxScaler to normalize the feature values in the CICIDS 2017 dataset, transforming them to a range of [0, 1]. This step ensures that features with different scales do not disproportionately impact the performance of the machine-learning models. [27], [28]

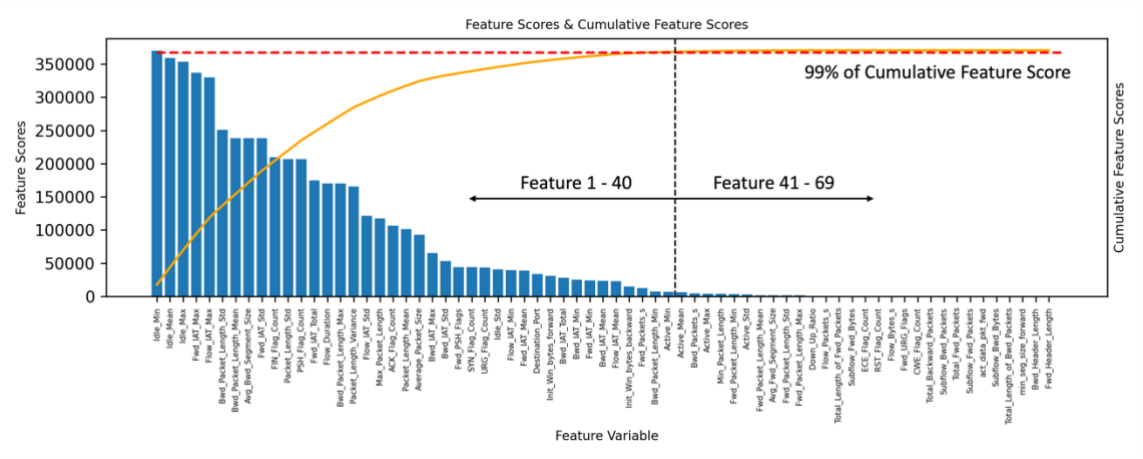


**Figure 5 Feature Scores & Cumulative Feature Scores**

4. **SelectKBest**: We employed the SelectKBest method, utilizing the chi-squared ($\chi 2$) statistical test to rank the features in the CICIDS 2017 dataset based on their importance. [29] As illustrated in Figure 9, which presents the Feature Scores & Cumulative Feature Scores, we observed that the top 40 features account for 99% of the cumulative importance. This observation suggests that the first 40 features exert a substantial impact on the model's performance, whereas features ranked beyond the 40$^{th}$ position contribute minimally. [30] Consequently, we opted to select the top 40 features using SelectKBest as a dimensionality reduction technique, thereby enabling our machine learning models to concentrate on the most pertinent features. [31]

After applying these preprocessing and feature selection techniques to the CICIDS 2017 dataset, the resulting datasets were ready for training and evaluating machine learning models.

# CHAPTER IV

# METHODOLOGY

In this chapter, we present the methodology employed in our study, focusing on deep learning models used for intrusion detection using the CICIDS 2017 dataset. In addition, we go over how to sample, evaluate models, and set up experiments.

## 4.1 Deep Learning Models

In our study, we employ a variety of deep learning models, including Convolutional Neural Networks (CNN), Artificial Neural Networks (ANN), Deep Neural Networks (DNN), Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and CLAttNet (Convolutional + LSTM + Attention). These models were chosen to explore different architectures and techniques to identify the best-performing model for the intrusion detection task.

To ensure a fair and consistent comparison between the models, all were subjected to the same evaluation metrics, resampling methods, and cross-validation strategy. Furthermore, to maintain consistency, the hyperparameters defining the internal structure of each model were determined and fixed arbitrarily for this experiment by us. The specific architectural configurations, such as the number of layers and neurons in each layer, were chosen based on commonly accepted configurations in the field. It is important to note that these hyperparameters were not explicitly optimized for the intrusion detection task but were chosen as a starting point for this study.

We split the dataset into an 80:20 ratio. Then, we apply five different resampling techniques to the 80% portion of the dataset to address class imbalance: No Resampling,

Selective Oversampling, Selective Undersampling, Combined Sampling (SMOTE + Undersampling), and SMOTE (Synthetic Minority Over-sampling Technique).

For each resampled dataset, we utilize 5-fold cross-validation to assess and adjust the model's performance. After completing each fold, we evaluate the model's performance using the untouched 20% of the data, which serves as our test set. This process helps to provide an honest assessment of the model's performance on unseen data.

### *4.1.1 Convolutional Neural Networks (CNN)*

This section explores using Convolutional Neural Networks (CNNs) for intrusion detection tasks, particularly for the CICIDS 2017 dataset. Two distinct CNN architectures were examined: the simpler CNN model (CNN-Simple) and the deeper CNN model (CNN-Deep). The performance of these two models is compared to gain insights into the advantages of a simpler versus a more profound architecture. [10], [12].

- **CNN-Simple Model**:

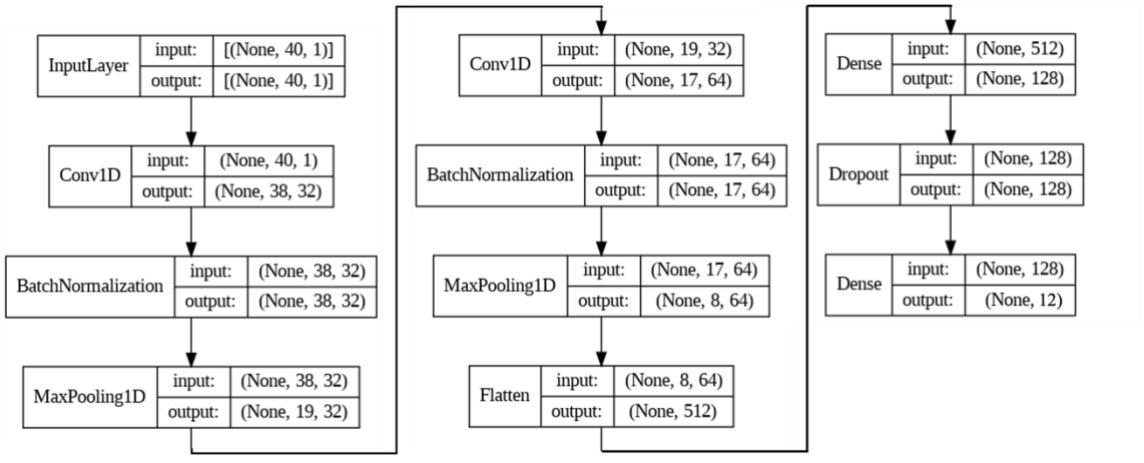


**Figure 6 CNN Simple Model**

Figure 10 illustrates the CNN-Simple model, which comprises an input layer, two 1D convolutional layers, batch normalization layers, max pooling layers, a flatten layer, a dense layer, a dropout layer, and an output layer. The simple architecture aims to learn features from the data while minimizing computational complexity efficiently.

- **CNN-Deep Model**:

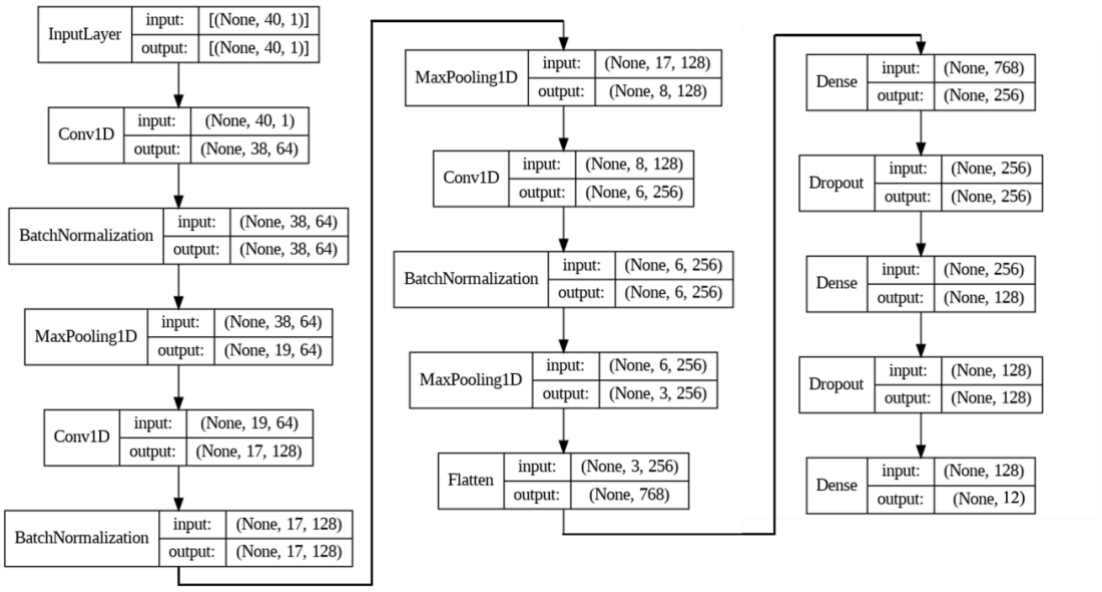


**Figure 7 CNN Deep Model**

On the other hand, Figure 11 displays the CNN-Deep model, which has a more intricate structure with additional layers and units. The model consists of an input layer, three 1D convolutional layers, batch normalization layers, max pooling layers, a flatten layer, two dense layers, two dropout layers, and an output layer. The more profound architecture captures more complex patterns in the data, potentially leading to improved performance. The Adam optimizer, categorical cross-entropy loss, and recall and accuracy measures are used to create the models.

We experimented with both CNN-Simple and CNN-Deep models to investigate which architecture is more effective for the CICIDS 2017 dataset. By comparing the performance of these two models, we can gain insights into the benefits of a simpler versus a more profound architecture for intrusion detection tasks.

### *4.1.2 Artificial Neural Networks (ANN)*

Artificial neural networks (ANNs) are machine learning models that discover links and patterns in data by interacting with other nodes or neurons that are interconnected. ANNs can be applied to intrusion detection systems (IDS) to identify patterns indicating intrusions or malicious activities. In this study, we designed an ANN model with the architecture depicted in Figure 12[10]:

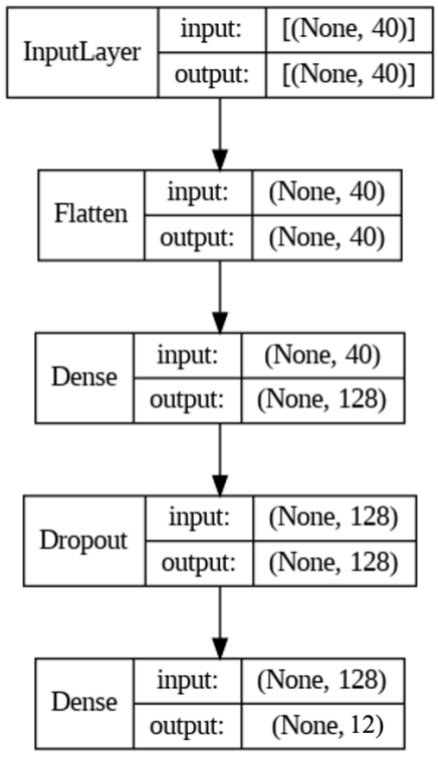


**Figure 8 ANN Model**

The categorical cross-entropy loss, accuracy, and recall metrics are included in the ANN model's construction. This model aims to efficiently learn features from the data and detect intrusion patterns with a relatively simple architecture.

### *4.1.3 Deep Neural Networks (DNN)*

Deep Neural Networks (DNNs) are an advancement of Artificial Neural Networks (ANNs), possessing multiple hidden layers.[33] This multi-layered structure enhances their capacity to encapsulate complex representations, boosting their performance across various tasks. DNNs, like ANNs, can also be employed for intrusion detection tasks by identifying patterns indicative of intrusions or malicious activities. In this study, we designed a DNN model with the architecture illustrated in Figure 13:

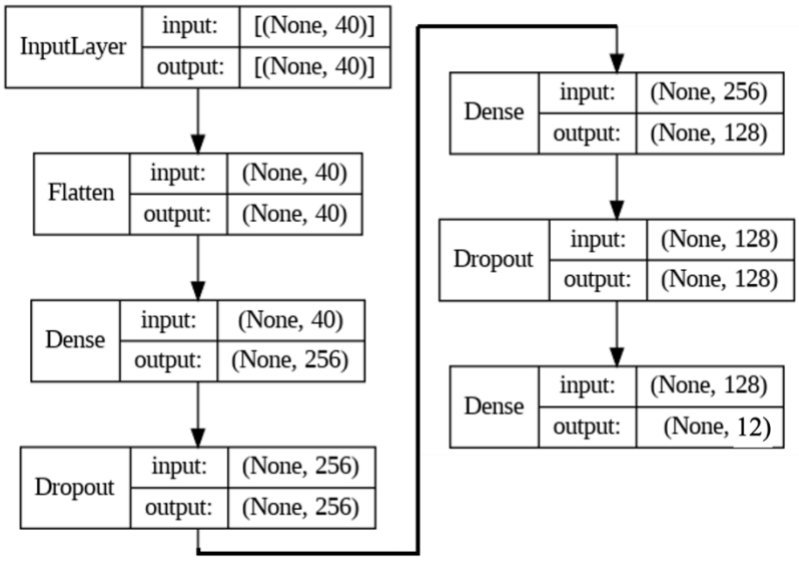


**Figure 9 DNN Model**

The DNN model is compiled with the Adam optimizer, categorical cross-entropy loss, and accuracy and Recall metrics. The more profound architecture of the DNN model

allows for better performance on intrusion detection tasks by capturing more complex patterns in the data.

### *4.1.4 Long Short-Term Memory (LSTM)*

Long Short-Term Memory (LSTM), a specialized form of recurrent neural network (RNN), was devised to solve the pervasive problem of vanishing gradients often encountered in conventional RNNs. [13] LSTMs are particularly helpful for time series analysis and tasks involving natural language processing because they can capture long-term dependencies in sequential data. [13] In the context of intrusion detection systems (IDS), LSTMs can analyze temporal patterns and detect malicious activities over time. We designed an LSTM model with the architecture shown in Figure 14:

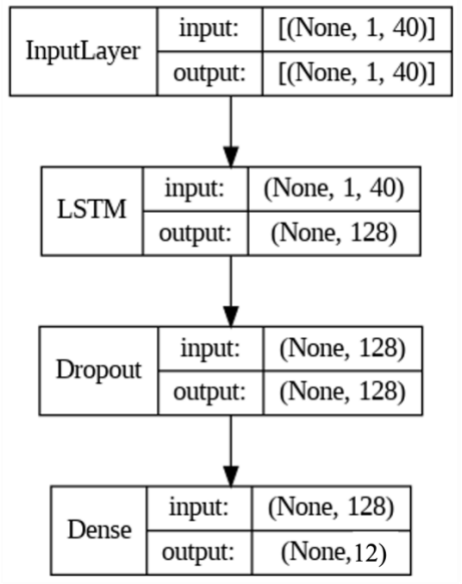


**Figure 10 LSTM Model**

The LSTM model is compiled with the Adam optimizer, categorical cross-entropy loss, and accuracy and Recall metrics. The LSTM model can effectively detect intrusion patterns that unfold over time by capturing long-term dependencies in the data.

### *4.1.5 Gated Recurrent Units (GRU)*

Another sort of RNN that tackles the vanishing gradient issue and is computationally more effective than LSTMs is the gated recurrent unit (GRU) [14]. GRUs may capture long-term dependencies in sequential data, just like LSTMs, making them appropriate for comparable tasks [14]. In this study, we designed a GRU model for intrusion detection with the architecture depicted in Figure 15:

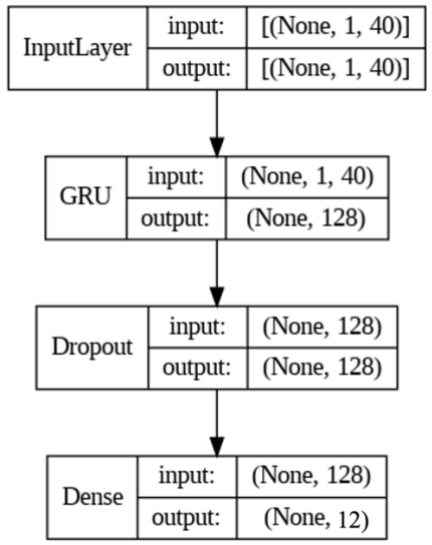


**Figure 11 GRU Model**

The GRU model is compiled with the Adam optimizer, categorical cross-entropy loss, and accuracy and Recall metrics.

### *4.1.6 CLAttNet (Convolutional + LSTM + Attention)*

The CLAttNet model is a relatively recent hybrid deep learning architecture utilized in various domains but is now being applied to the Intrusion Detection System (IDS) field. We has informally coined the term “CLAttNet” to represent the integration of three well-known deep learning components in this research: Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Attention mechanisms [13], [34]. The unique combination of these three elements in the IDS domain is a novel approach to improving intrusion detection performance. The CLAttNet model’s architecture consists of the following layers depicted in Figure 16:

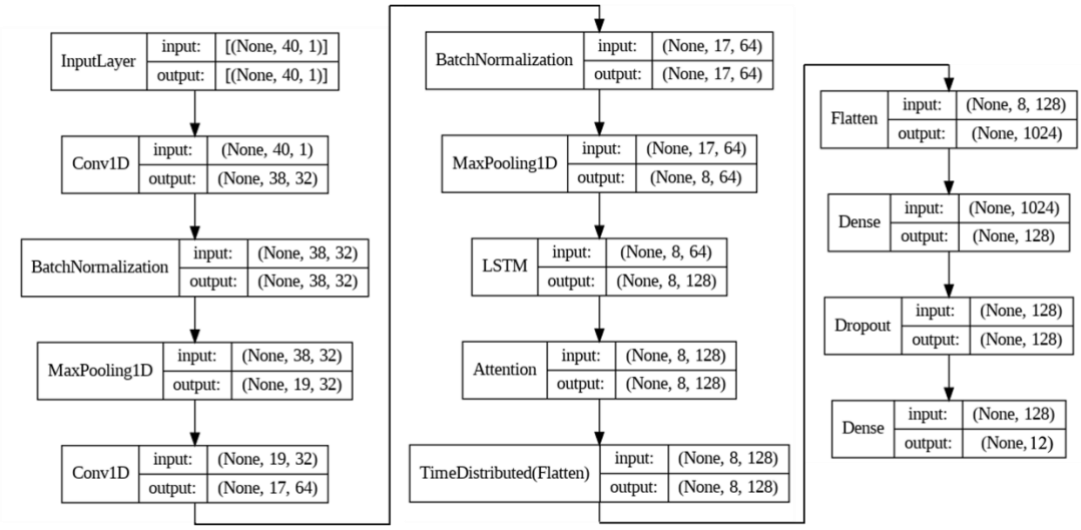


**Figure 12 Hybrid Model**

The Attention mechanism is particularly captivating because it empowers the model to prioritize specific sections of the input data most pertinent to the task. It does this by learning to assign weights to each input element based on their importance for the current

prediction. This ability to capture more context and dependencies between features can lead to better performance in intrusion detection. [35]

In the context of IDS, the CLAttNet model aims to capitalize on the strengths of its three core components:

1. **CNN layers**: Efficient at extracting local and spatial features from input data, making them suitable for detecting patterns in network traffic data.
2. **LSTM layer**: Capable of capturing long-term dependencies in sequential data, which can help detect malicious activities that span multiple time steps. [13]
3. **Attention layer**: This enables the model to emphasize relevant features and filter out less critical information, resulting in better decision-making during intrusion detection. [35]

The Adam optimizer, categorical cross-entropy loss, accuracy, and recall measures are used to create the model. As a result, the CLAttNet model is expected to provide enhanced intrusion detection performance compared to models that use a single type of architecture because it combines these three powerful techniques.

It is crucial to note that CLAttNet is not an officially recognized name but rather an informal designation given by the author to reflect the integration of CNN, LSTM, and Attention mechanisms. While this model has been experimented with in other fields, its application in the IDS domain is novel, and its effectiveness in detecting intrusions remains to be explored through experimentation. Nevertheless, integrating these three components offers promising potential for improved performance in intrusion detection systems.

### 4.2 Sampling Techniques Applied to the CICIDS 2017 Dataset

In this section, we discuss applying various sampling techniques to the CICIDS 2017 dataset, aiming to address the challenges of imbalanced data. [36] The performance of the intrusion detection models will be compared across these different sampling methods to evaluate their effectiveness.

1. **No Resampling**: The original CICIDS 2017 dataset is used without any modifications. By utilizing the raw dataset, we can establish a baseline for model performance and compare the effects of the other sampling techniques.
2. **Selective Oversampling**: Instances from minority classes in the CICIDS 2017 dataset are selectively oversampled to balance the class distribution. RandomOverSampler from the imbalanced-learn library is used for this purpose. This technique aims to improve the model's performance on underrepresented classes while maintaining the integrity of the majority classes.
3. **Selective Undersampling**: Instances from the majority classes in the CICIDS 2017 dataset are selectively undersampled to balance the class distribution. RandomUnderSampler from the imbalanced-learn library is used for this purpose. This technique helps reduce the potential bias toward majority classes by creating a more balanced dataset for model training.
4. **Combined Resampling**: This approach involves a combination of SMOTE and undersampling applied to the CICIDS 2017 dataset. SMOTE oversamples minority classes while RandomUnderSampler undersamples majority classes. The

combined resampling aims to provide a more balanced dataset that facilitates the development of robust IDS models.

5. **SMOTE for Minority Classes**: The Synthetic Minority Over-sampling Technique (SMOTE) is applied selectively to specific minority classes in the CICIDS 2017 dataset, such as 'Bot,' 'Brute Force,' and 'XSS.'[37] Using the imbalanced-learn library, synthetic instances are generated for these minority classes to improve the model's performance on them without affecting the majority classes. [38]

The impact of these sampling techniques on the performance of IDS models will be analyzed in the subsequent sections of this study. By comparing the results, we aim to identify the most effective approach for handling imbalanced data in the CICIDS 2017 dataset and improving the overall performance of intrusion detection systems.

## 4.3 Model Evaluation

In this section, we describe the methods used to evaluate the performance of the proposed models in detecting intrusion activities. Various evaluation techniques ensure a comprehensive assessment of the models' performance, including cross-validation, test set evaluation, performance metrics, confusion matrix, ROC and Precision-Recall Curves.

### *4.3.1 Cross-Validation*

As it reduces the risk of overfitting and gives a more accurate estimate of the generalization abilities of machine learning models, cross-validation is a widely used and crucial technique for evaluating the performance of these models. [39] When a model

becomes overly attuned to the training data, it can mistakenly incorporate noise or random variances. Over-optimization can hinder the model's effectiveness when faced with new, unseen data. This is known as overfitting. Cross-validation addresses this issue by training and validating the models on different subsets of the data, ensuring that a specific data partition does not bias their performance. [28], [39]

The K-fold cross-validation method randomly partitions the dataset into K equal-sized subsets. During each iteration, the model is trained on K-1 subsets and then validated on the subset not used for training. This cycle is repeated K times, each subset serving as the validation set once. The performance metrics from all iterations are then averaged to yield a more reliable assessment of the model's capabilities. A cross-validation is a valuable tool for gauging a model's ability to generalize to unseen data by using multiple training and validation subsets. [39]

In this study, we employ 5-fold cross-validation to assess the performance of the proposed models in detecting intrusion activities. The choice of 5 folds was made not only to strike a balance between computational efficiency and the reliability of the performance estimates but also as a fixed experimental setup for the scope of this particular research. It should be noted that future research may consider varying the number of folds in cross-validation, depending on the specific requirements and objectives of the study, to explore further the impact of different fold numbers on model performance and optimization.

During the cross-validation process, we focus on the models' ability to classify instances accurately, minimize false positives, and detect intrusion activities confidently. The performance metrics obtained from the cross-validation process, such as precision,

recall, accuracy, and F-score, provide valuable insights into the models' capabilities and help us identify the most effective model for intrusion detection. [39]

### *4.3.2 Performance Metrics*

In this part, we delve into the method employed to calculate the critical performance metrics – precision, recall, accuracy, and F1-Score, for assessing our IDS model [19]. These metrics are essential in gauging our model's efficacy in detecting malicious activities and generalizing these findings to real-world circumstances. Here is the step-by-step process we used:

1. **True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN)**: We initiate our process by quantifying the true positives, false positives, true negatives, and false negatives in our test set. These numbers are foundational in computing the performance metrics and evaluating our model's proficiency in correctly classifying instances. [19]
2. **Accuracy**: Accuracy is the ratio of correctly identified instances (TP and TN) to all cases in the test set. Although it assesses the overall performance of our model in classifying network traffic instances, other metrics might be more suitable for imbalanced datasets. [19]
3. **Precision**: Precision is computed as the True Positives (TP) ratio to the True Positives and False Positives (FP) sum. It provides insight into our model's proficiency in minimizing false alarms, or in other words, preventing the mislabeling of regular traffic as malicious. For IDS, it is vital to minimize false

positives to prevent system administrators from being overwhelmed by false alarms. [19]

4. **Recall (Sensitivity)**: Recall is determined by the proportion of TP against the sum of TP and FN. Observing a high recall score signifies that our model is adept at pinpointing malicious activities within the network traffic. We stress the importance of recall as it directly relates to the model's ability to detect threats and safeguard the network from potential intrusions. [19]
5. **F1-score**: The F1-score represents a balanced average of precision and recall, which helps to address their inherent trade-off. Particularly in datasets with an imbalance, such as ours, where the focus is on the minority class (malicious traffic), the F1-score provides an all-inclusive evaluation of how well the model performs. We compute the F1-score to ensure our model effectively detects malicious activities while minimizing false alarms. [19]

By measuring our model's performance using these metrics, we can assess its ability to detect and classify network intrusions. In addition, the test set evaluation results further offer valuable insights into the model's real-world applicability, contributing to the continual enhancement of our IDS.

### *4.3.4 Confusion Matrix, ROC and Precision-Recall Curves*

We take this opportunity to detail the methodology for creating the confusion matrix, ROC and Precision-Recall curves, vital tools for visualizing and understanding our IDS model's performance on the test set. These visual aids allow us to pinpoint areas of

enhancement and better grasp the trade-offs between true positive and false positive rates. [40]

1. **Confusion Matrix**: To construct the confusion matrix, we initially compute the count of accurate positive (TP), accurate negative (TN), inaccurate positive (FP), and inaccurate negative (FN) predictions from our model's assessment of the test set. [40] We then arrange these figures in a 12 x 12 matrix, with rows representing actual classes (regular and malicious) and columns representing predicted classes. [20] This matrix offers a detailed overview of our model's performance, emphasizing its capability to differentiate between normal and malicious network traffic. By inspecting the confusion matrix, we can pinpoint specific issues with our model and devise targeted improvement plans.
2. **ROC Curves**: In order to generate the ROC curves for our intrusion detection system, we first determine both the true positive rate (often called recall or sensitivity) and the false positive rate at various decision thresholds. [21] We then plot TPR against FPR for each threshold, creating a curve that visualizes the trade-offs between correctly identifying malicious instances (TPR) and incorrectly classifying regular instances as malicious (FPR). [41] By examining the ROC curve, we can pinpoint the optimal decision threshold that maximizes our model's intrusion detection capability while minimizing false alarms.
3. **Area Under the ROC Curve (AUC-ROC)**: In order to gauge the effectiveness of our IDS model, we calculate the area under the Receiver Operating Characteristic curve, also known as AUC-ROC. [21] The values of AUC-ROC

span between 0 and 1, with 1 signifying an optimal classifier, while a score of 0.5 is suggestive of a model with a performance level equivalent to that of random guessing. [40] By comparing the AUC-ROC values of different models or configurations, we can identify the most effective method for detecting network intrusions in our experimental setup.

4. **Precision-Recall Curves:** To generate Precision-Recall curves for the Intrusion Detection System (IDS) applied to the CICIDS 2017 dataset, we first determine precision and recall at various decision thresholds [21]. Precision refers to correctly identified infestations out of all instances classified as infestations, and recall refers to correctly identified infestations out of all actual infestations. We then plot precision against recall for each threshold to generate a curve visualizing the balance between accurately detecting malicious activity (accuracy) and comprehensively identifying all intrusions (recall) [21].
5. Area Under the Precision-Recall Curve (AP): To evaluate the effectiveness of an IDS model, the area under the Precision-Recall curve, often referred to as AUC-PR, can be calculated and utilized [21]. This plot looks similar to ROC-AUC, but is substantially different. AP values range from 0 to 1, with 1 representing an optimal model [21]. We explored the most efficient approach to detect network intrusion by comparing AP values for each label for different deep-learning models and sampling methods.

By leveraging these visualization techniques and performance metrics, we comprehensively understand our IDS model's performance on the test set. This

information aids us in optimizing the model and addressing its shortcomings, ensuring it effectively detects and classifies network intrusions in real-world scenarios.

## 4.4 Experiment Setup

In this section, we describe the experiment setup, including the virtual environment, programming language, libraries, and their respective versions for implementing and evaluating our IDS model.

### *4.4.1 Google Colab Virtual Environment*

All of our tests were performed using Google Colaboratory (Colab), a platform in the cloud that offers a Jupyter Notebook environment for creating and running Python programs. Google Colab offers a convenient and accessible way to work with data and run machine learning experiments. It requires no local installation and can be accessed from any device with an internet connection.

To ensure a smooth and uninterrupted experimental process, we subscribed to Colab Pro Plus, which provides additional benefits such as faster GPUs, longer runtimes, and priority resource access. In addition, this subscription enabled us to allocate the necessary computational resources for training and evaluating our deep-learning models more efficiently.

### *4.4.2 Python and Libraries*

Our entire codebase for this study was implemented in Python 3.9.16 (Dec 7, 2022), a popular programming language widely used in machine learning and data science.

We chose Python due to its extensive ecosystem of libraries and tools, which significantly facilitate the implementation and evaluation of machine learning models.

For building and training our IDS model, we utilized the following libraries and their respective versions:

1. **TensorFlow**: An open-source machine learning framework developed by Google for building and training our deep learning model.
2. **NumPy**: A Python numerical computing library used to efficiently manage massive multidimensional arrays and matrices.
3. **Pandas**: a library for handling and processing structured data, like the network traffic dataset utilized in our study. Pandas provides DataFrame objects, which are two-dimensional, size-mutable, and heterogeneous tabular data structures with labeled axes. This differs from NumPy's arrays, as DataFrames allow more flexibility in handling data with other data types and offer advanced indexing capabilities.
4. **Scikit-learn (sklearn)**: A machine learning library that provides data preprocessing, model evaluation, and performance metric calculation tools.
5. **Matplotlib**: A Python plotting toolkit for producing animated, interactive, and static visualizations of the outcomes and performance data of our studies.

By leveraging the capabilities of Google Colab, Python 3.9.16, TensorFlow, and other essential libraries, we were able to effectively implement and evaluate our IDS model, obtaining valuable insights into its performance and potential areas for improvement.

# CHAPTER V

# RESULTS AND DISCUSSION

In this chapter, we present a comprehensive analysis of the performance of various deep learning models for intrusion detection. We evaluated the models using five sampling techniques: No Sampling, Selective Oversampling, Selective Undersampling, SMOTE, and Combined Sampling (SMOTE + Selective Undersampling). The models used in this study include CNN-Simple, CNN-Deep, ANN, DNN, LSTM, GRU, and CLAttNet. We assessed their performance based on precision, recall, accuracy, and F-score using 5-fold cross-validation. In addition, we derived and analyzed the ROC-AUC and Precision-Recall Curve AUC scores for each model to compare the performance of each model in detail.

## 5.1 Results by Sampling Technique

The cross-validation results for each model across the five sampling techniques are presented in Tables 3 to 7. We calculated the mean of the performance metrics for each model to compare their performance.

### *5.1.1 No Sampling Technique*

**Table 3 Model Performance with No sampling technique in 5-fold CV**

| *Model* | *Precision* | *Recall* | *F1-Score* | *Accuracy* |
|---|---|---|---|---|
| *CNN-S* | 0.9864 | 0.9865 | **0.98592** | 0.9865 |
| *CNN-D* | 0.95501 | 0.95466 | **0.95252** | 0.95466 |
| *ANN* | 0.98251 | 0.98085 | 0.98075 | 0.98085 |
| *DNN* | 0.98326 | 0.98169 | 0.98164 | 0.98169 |
| *LSTM* | 0.98598 | 0.98544 | 0.98507 | 0.98544 |
| *GRU* | 0.98429 | 0.98349 | 0.98314 | 0.98349 |
| *CLAttNet* | 0.98561 | 0.98587 | 0.98551 | 0.98587 |

In the No Sampling Technique setting, the CNN-Simple model demonstrated the highest F1-Score (0.98592), followed by the CLAttNet, LSTM, GRU, DNN, ANN, and CNN-Deep models. The CNN-Deep model showed the lowest F1-Score(0.95252). The detailed cross-validation results can be found in Table 3.

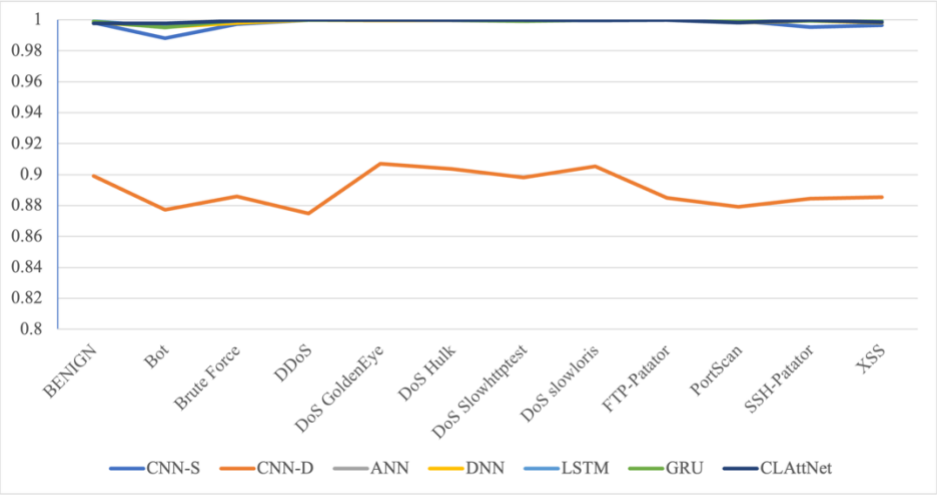


**Figure 13 ROC-AUC Score by Label in No Resampling**

Figure 17 above compares the ROC-AUC scores for each model in the No Resampling Technique by the label. In this result, as in comparing the F1-Score, it can be seen that the CNN-Deep model has the lowest score.

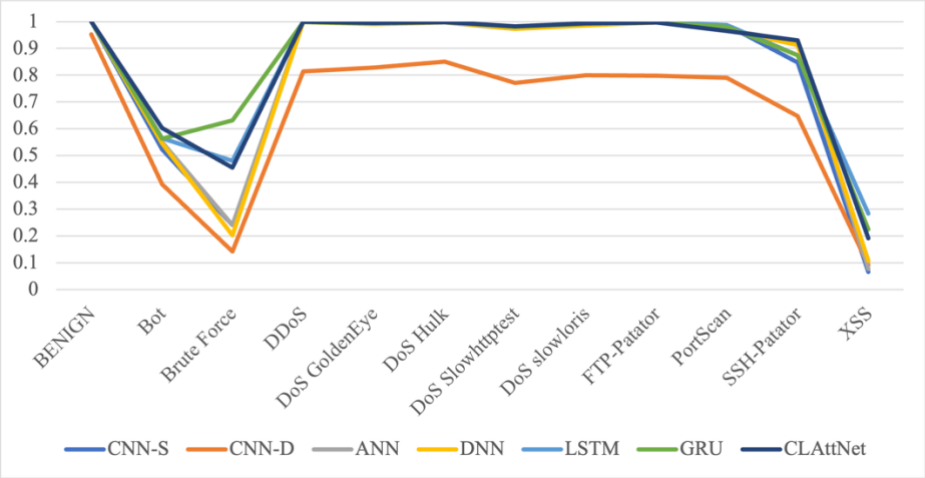


**Figure 14 Precision Recall-AUC Score by Label in No Resampling**

Similarly, Figure 18 compares the AUC scores of the Precision-Recall Curve by the model in No Resampling. Again, it can be seen that the CNN-Deep model shows the lowest score as in the previous results.

This result shows that the too-complicated model might have yet to effectively capture the underlying patterns and features that differentiate the classes.

*5.1.2 Selective Oversampling*

**Table 4 Model Performance with Selective Oversampling in 5-fold CV**

| *Model* | *Precision* | *Recall* | *F1-Score* | *Accuracy* |
|---|---|---|---|---|
| *CNN-S* | 0.96571 | 0.96316 | **0.96257** | 0.96316 |
| *CNN-D* | 0.96699 | 0.96413 | 0.9632 | 0.96413 |
| *ANN* | 0.9845 | 0.98092 | 0.9819 | 0.98092 |
| *DNN* | 0.9845 | 0.98025 | 0.98149 | 0.98025 |
| *LSTM* | 0.98852 | 0.98515 | 0.98609 | 0.98515 |
| *GRU* | 0.98816 | 0.98533 | **0.98623** | 0.98533 |
| *CLAttNet* | 0.97694 | 0.97453 | 0.975 | 0.97453 |

With Selective Oversampling, the GRU model achieved the highest F1-Score (0.98623). The LSTM, ANN, DNN, CLAttNet, CNN-Deep and CNN-Simple models

followed in descending order. The detailed cross-validation results can be found in Table 4. The CNN-Simple model, which showed the best performance in No Resampling, shows the lowest score(0.96257) this time. Furthermore, similarly, the CNN-Deep model also shows relatively poor performance.

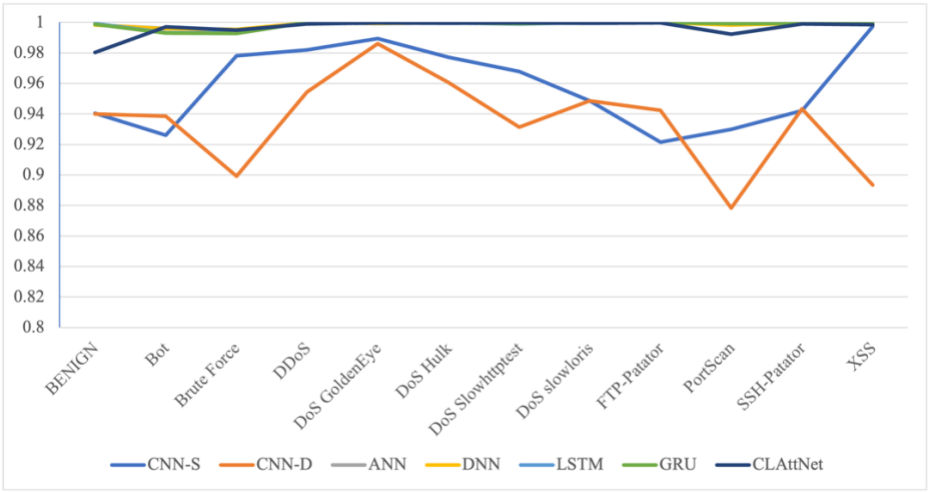


**Figure 15 ROC-AUC Score by Label in Selective Oversampling**

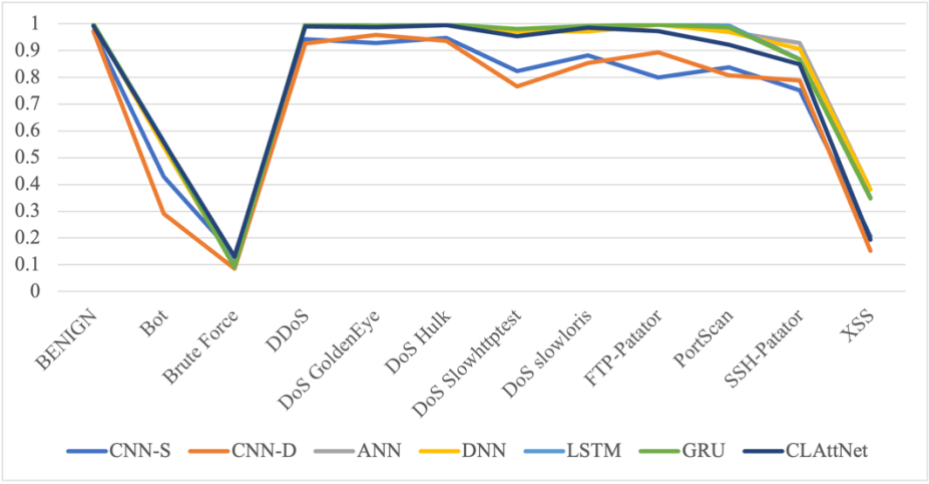


**Figure 16 Precision Recall-AUC Score by Label in Selective Oversampling**

Figure 19 shows that the CNN-Deep and CNN-Simple models show relatively poor scores compared to other models, similar to the F1-Score result. Moreover, this shows the same aspect in the AUC Score of the Precision-Recall Curve in Figure 20.

### *5.1.3 Selective Undersampling*

**Table 5 Model Performance with Selective Undersampling in 5-fold CV**

| *Model* | *Precision* | *Recall* | *F1-Score* | *Accuracy* |
|---|---|---|---|---|
| *CNN-S* | 0.93996 | 0.84085 | **0.8747** | 0.84085 |
| *CNN-D* | 0.92263 | 0.7663 | 0.81333 | 0.7663 |
| *ANN* | 0.91464 | 0.75074 | 0.79658 | 0.75074 |
| *DNN* | 0.91406 | 0.74846 | 0.79484 | 0.74846 |
| *LSTM* | 0.92704 | 0.77343 | 0.82175 | 0.77343 |
| *GRU* | 0.91984 | 0.74989 | 0.80007 | 0.74989 |
| *CLAttNet* | 0.94367 | 0.87027 | **0.89516** | 0.87027 |

Overall, all models show poor performance in the Selective Undersampling Technique compared to other sampling methods. Since it shows an average of 15% or lower scores than other samplings, this sampling does not significantly improve model performance. However, we confirmed that the CLAttNet and CNN-Simple models scored close to 90% (0.89516 / 0.8747) among the poor performances of other models.

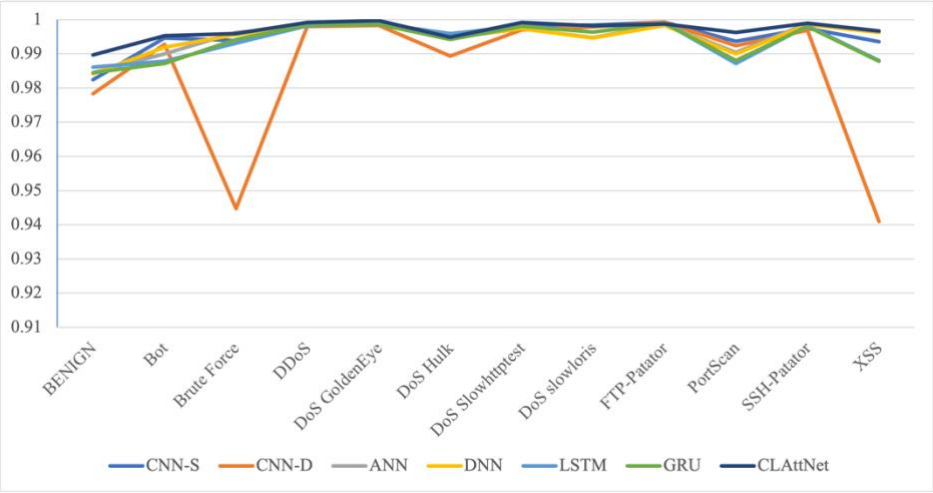


**Figure 17 ROC-AUC Score by Label in Selective Undersampling**

In Figure 21 and Figure 22, we can see the overall low-scoring models. Furthermore, in the two figures, it can also be found that the CLAttNet and CNN-Simple models show relatively good scores.

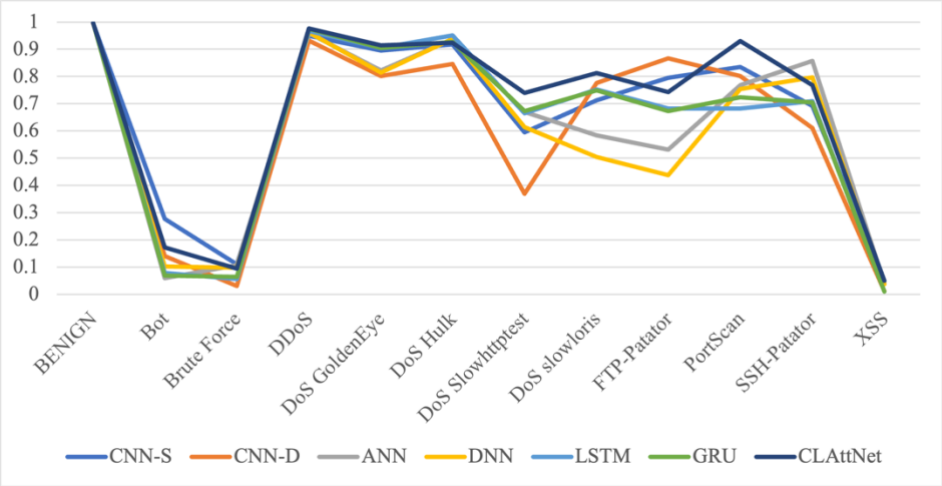


**Figure 18 Precision Recall-AUC Score by Label in Selective Underampling**

### *5.1.4 SMOTE*

**Table 6 Model Performance with SMOTE in 5-fold CV**

| *Model* | *Precision* | *Recall* | *F1-Score* | *Accuracy* |
|---|---|---|---|---|
| *CNN-S* | 0.98144 | 0.97994 | **0.97998** | 0.97994 |
| *CNN-D* | 0.98776 | 0.98592 | 0.98594 | 0.98592 |
| *ANN* | 0.98325 | 0.97936 | 0.98045 | 0.97936 |
| *DNN* | 0.98457 | 0.98097 | 0.98199 | 0.98097 |
| *LSTM* | 0.98852 | 0.98633 | 0.98687 | 0.98633 |
| *GRU* | 0.98598 | 0.98274 | 0.98355 | 0.98274 |
| *CLAttNet* | 0.99391 | 0.9926 | **0.99288** | 0.9926 |

In the SMOTE setting, the CLAttNet model performed the best in terms of F1-Score (0.99288). The LSTM, CNN-Deep, GRU, DNN, ANN, and CNN-Simple models followed in descending order. The detailed cross-validation results can be found in Table

6. This result indicates that generating synthetic examples of the minority classes using SMOTE can significantly improve the performance of deep learning models for intrusion detection, particularly for the CLAttNet model in this case.

The model that showed the poorest performance in SMOTE sampling was the CNN-Simple model (0.97998), which performed well in the previous sampling technique. However, it is difficult to see this as a significant result because there is no significant difference from other models.

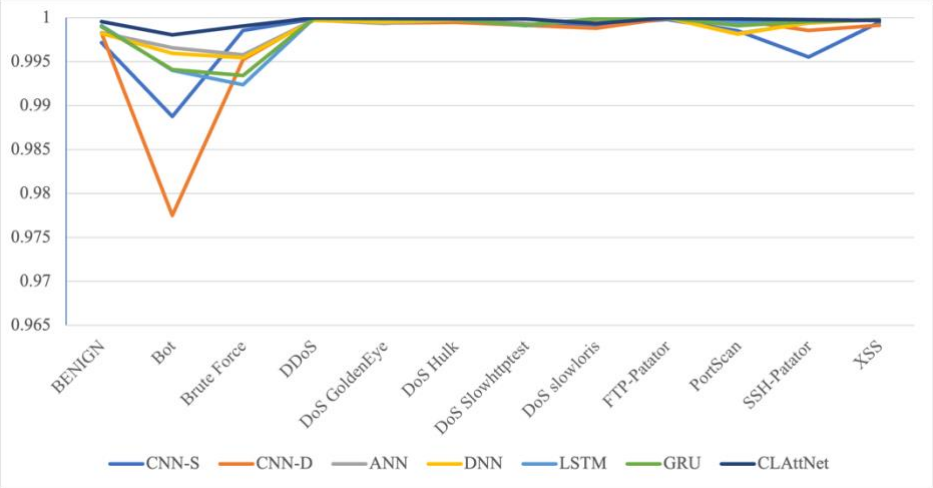


**Figure 19 ROC-AUC Score by Label in SMOTE**

As with the F1-Score result, in Figure 23 and Figure 24, we can see that CLAttNet shows the best score. However, other models show similar scores. In other words, it is difficult to distinguish clearly from the graph that the CNN-Simple model performs poorly.

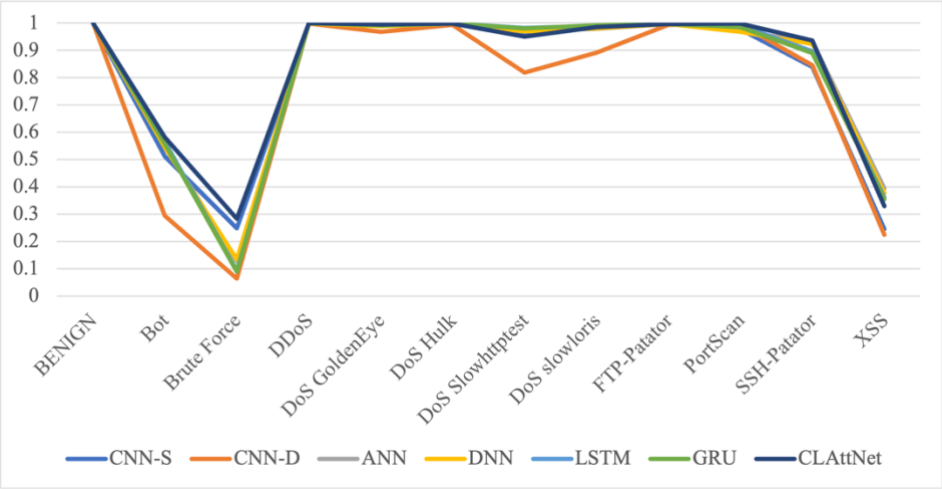


**Figure 20 Precision Recall-AUC Score by Label in SMOTE**

### *5.1.5 Combined Sampling(SMOTE + Selective Undersampling)*

**Table 7 Model Performance with Combined Sampling in 5-fold CV**

| *Model* | *Precision* | *Recall* | *F1-Score* | *Accuracy* |
|---|---|---|---|---|
| *CNN-S* | 0.93856 | 0.85166 | 0.88143 | 0.85166 |
| *CNN-D* | 0.91433 | 0.6323 | 0.70605 | 0.6323 |
| *ANN* | 0.93479 | 0.82329 | 0.8606 | 0.82329 |
| *DNN* | 0.93267 | 0.8224 | 0.85853 | 0.8224 |
| *LSTM* | 0.94985 | 0.8414 | 0.88225 | 0.8414 |
| *GRU* | 0.94825 | 0.83114 | 0.87493 | 0.83114 |
| *CLAttNet* | 0.94356 | 0.8659 | 0.89388 | 0.8659 |

Even with the combined sampling technique, we can confirm that the models show poor overall performance. For example, compared to the No Resampling Technique, which did not apply the sampling technique, it was confirmed that the performance was lowered by as much as 0.25 (CNN-Deep: 0.95252 ⇒ 0.70605). Moreover, even in the case of CLAttNet, which has the best performance in this technique, the score was about 0.09 lower than that of No Resampling.(0.98551 ⇒ 0.89388)

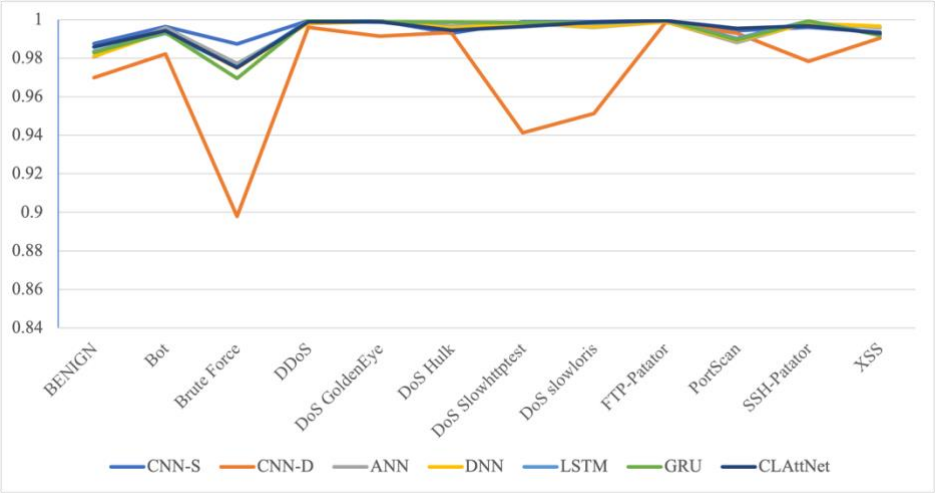


**Figure 21 ROC-AUC Score by Label in Combined Sampling**

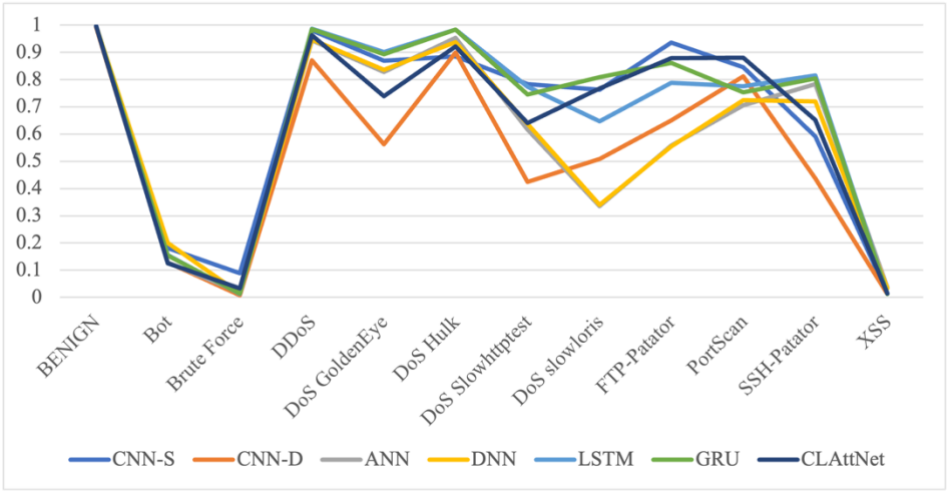


**Figure 22 Precision Recall-AUC Score by Label in Combined Sampling**

Figure 25 and Figure 26 show the performance of the two models by comparing the AUC scores for each label in terms of ROC and Precision-Recall Curve. Among them, Figure 26 clearly shows the overall low performance of the models.

### 5.2 Best Sampling Technique for Each Deep Learning Model Based on F1-Score

In this section, we present the best sampling method for each deep learning model according to their F1-Scores. We will include the label classification report, ROC graphs, and Confusion Matrices for each combination, accompanied by an interpretation and explanation of the results.

**Table 8 Best Sampling Technique and F1-Score for Each model**

| | *Best Sampling* | *F1-Score* | *2nd Sampling* | *F1-Score* | F1-Score of *No Resampling* |
|---|---|---|---|---|---|
| *CNN - S* | No Sampling | 0.98592 ( - ) | SMOTE | 0.97998 (▼0.00594) | 0.98592 |
| *CNN - D* | SMOTE | 0.98594 (▲0.03342) | Oversampling | 0.96320 (▲0.01068) | 0.95252 |
| *ANN* | Oversampling | 0.98190 (▲0.00115) | No Sampling | 0.98075 ( - ) | 0.98075 |
| *DNN* | SMOTE | 0.98199 (▲0.00035) | No Sampling | 0.98164 ( - ) | 0.98164 |
| *LSTM* | SMOTE | 0.98687 (▲0.00180) | Oversampling | 0.98609 (▲0.00102) | 0.98507 |
| *GRU* | SMOTE | 0.98355 (▲0.00041) | Oversampling | 0.98623 (▲0.00309) | 0.98314 |
| *CLAttNet* | SMOTE | 0.99288 (▲0.00737) | No Sampling | 0.98551 ( - ) | 0.98551 |

5454545454

#### *5.2.1 CNN - Simple with No Sampling (F1-Score: 0.98592)*

Referring to Table 12, the CNN Simple model shows the highest F1-Score(0.98592) when no sampling technique is applied. In addition, Table 13 shows the label classification report of the CNN Simple model in No Sampling, among which Bot (0.47237), Brute Force (0.12042), and XSS (0) show low F1-Scores. The reason for this is that the number of instances of the corresponding label is tiny.

In particular, the fact that the F1-Score is zero(0) even though the Precision of XSS is one(1) implies why we should not use performance metrics such as Precision and Recall alone when evaluating model performance. Instead, F1-Score means the harmonic average

of Precision and Recall, and by using it, we can understand the model's performance in a more balanced way.

**Table 9 CNN Simple with No Sampling Label Classification Report**

| *Label* | *Precision* | *Recall* | *F1-Score* | *Support* |
|---|---|---|---|---|
| *BENIGN* | 0.99016 | 0.99356 | 0.99186 | 2271320 |
| *Bot* | 0.98699 | 0.31049 | 0.47237 | 1955 |
| *Brute Force* | 0.91509 | 0.06445 | 0.12042 | 1505 |
| *DDoS* | 0.99575 | 0.98911 | 0.99242 | 128025 |
| *DoS GoldenEye* | 0.9949 | 0.96552 | 0.97999 | 10295 |
| *DoS Hulk* | 0.96927 | 0.97525 | 0.97225 | 230125 |
| *DoS Slowhttptest* | 0.94998 | 0.87709 | 0.91208 | 5500 |
| *DoS slowloris* | 0.99129 | 0.92355 | 0.95623 | 5795 |
| *FTP-Patator* | 0.99634 | 0.99509 | 0.99571 | 7935 |
| *PortScan* | 0.95246 | 0.9425 | 0.94745 | 158805 |
| *SSH-Patator* | 0.93488 | 0.58881 | 0.72255 | 5900 |
| *XSS* | 1 | 0 | 0 | 650 |
| | | | | |
| *accuracy* | | | 0.9865 | 2827810 |
| *macro avg* | 0.97309 | 0.71879 | 0.75528 | 2827810 |
| *weighted avg* | 0.9864 | 0.9865 | 0.98592 | 2827810 |

The model's performance across all classes is effectively demonstrated by the Confusion Matrix (Figure 27), which exhibits accurate classification with minimal misclassifications. The Figure 27 also shows that this model classifies almost all XSS labels as Benign.

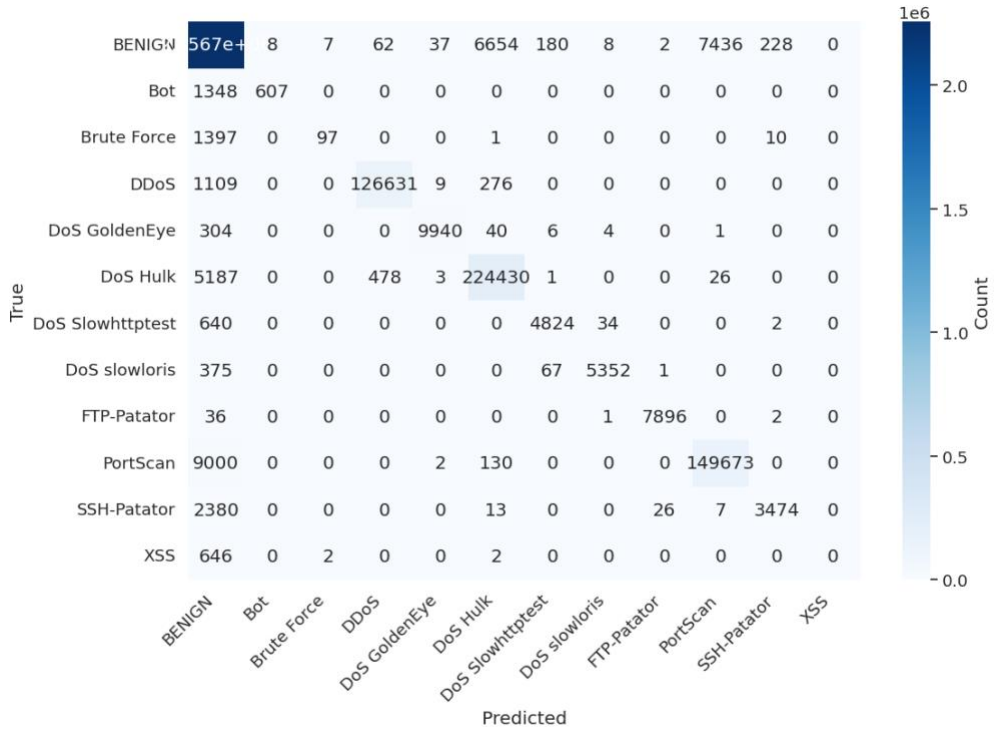


**Figure 23 CNN - S Confusion Matrix**

In addition, as depicted in the Receiver Operating Characteristic (ROC) graph (Figure 28), the model maintains high True Positive Rates (TPR) for all classes while successfully minimizing the False Positive Rates (FPR).

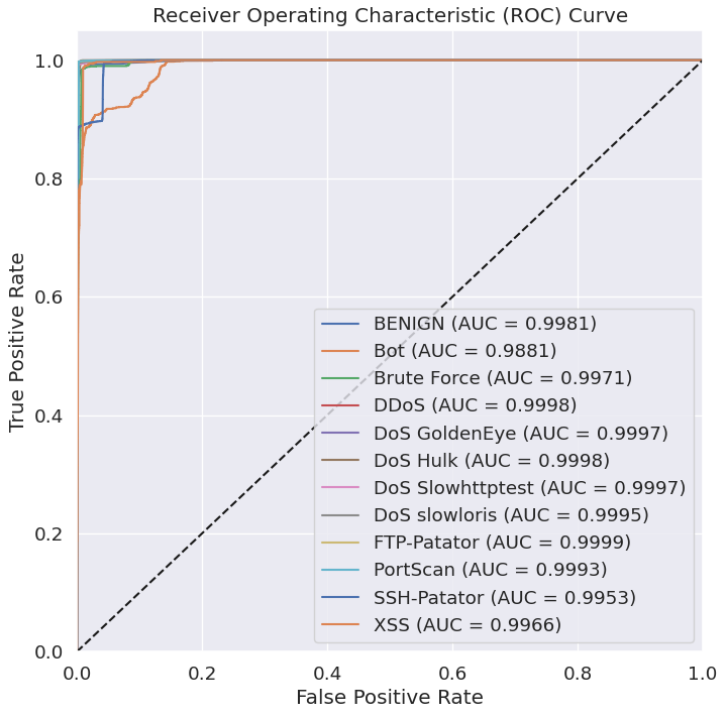


**Figure 24 CNN - S ROC Curve**

Moreover, Figure 28 is the ROC curve for each label of this model, and Figure 29 shows the Precision-Recall curve for each label. In the ROC curve, most of the labels show good performance. Even Bot, Brute Force, and XSS labels show that the AUC is over 0.98, confirming the results far from the performance judged by the F1-Score. However, in Figure 29, as in the previous performance evaluation results, it can be seen that labels with low F1-Scores still show low Precision-Recall Curve AUC scores.

In other words, it is strongly assumed that the model's performance will be more balanced when comprehensively considering F1-Score and Precision-Recall Curve.

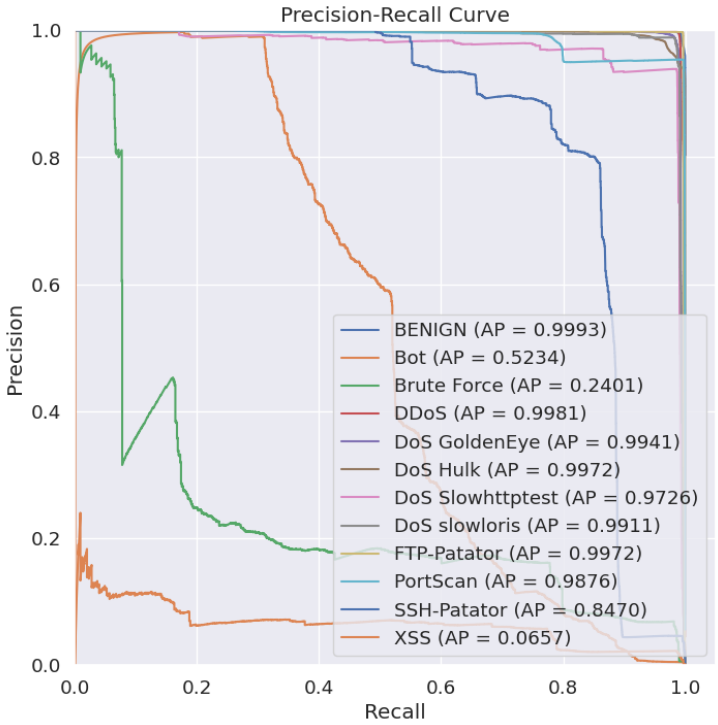


**Figure 25 CNN - S Precision-Recall Curve**

### *5.2.2 CNN-Deep with SMOTE (F1-Score: 0.98594)*

Referring to Table 12, the CNN Deep model shows the highest F1-Score (0.98594) when the SMOTE technique is applied, which is a result showing a performance improvement of about 0.03342 higher than when no sampling was used. In addition, Table 14 shows the label classification report of the CNN Deep model in SMOTE, among which Bot (0.20468), Brute Force (0), and XSS (0.14105) show low F1-Scores. The reason for this is that the number of instances of the corresponding label is tiny, which is the same reason as before.

In particular, the fact that the F1-Score is zero(0) even though the Precision of Brute Force is one(1) implies why we should not use performance metrics such as Precision and Recall alone when evaluating model performance.

**Table 10 CNN - D Label Classification Report**

| *Label* | *Precision* | *Recall* | *F1-Score* | *Support* |
|---|---|---|---|---|
| *BENIGN* | 0.98956 | 0.99469 | 0.99212 | 2271320 |
| *Bot* | 0.99554 | 0.11407 | 0.20468 | 1955 |
| *Brute Force* | 1 | 0 | 0 | 1505 |
| *DDoS* | 0.99819 | 0.98016 | 0.9891 | 128025 |
| *DoS GoldenEye* | 0.98871 | 0.9017 | 0.9432 | 10295 |
| *DoS Hulk* | 0.96926 | 0.95727 | 0.96323 | 230125 |
| *DoS Slowhttptest* | 0.75256 | 0.52091 | 0.61567 | 5500 |
| *DoS slowloris* | 0.98663 | 0.63676 | 0.77399 | 5795 |
| *FTP-Patator* | 0.99058 | 0.88784 | 0.9364 | 7935 |
| *PortScan* | 0.99156 | 0.98503 | 0.98828 | 158805 |
| *SSH-Patator* | 0.99931 | 0.49102 | 0.65848 | 5900 |
| *XSS* | 0.07727 | 0.80769 | 0.14105 | 650 |
| *accuracy* | | | 0.98592 | 2827810 |
| *macro avg* | 0.89493 | 0.68976 | 0.68385 | 2827810 |
| *weighted avg* | 0.98776 | 0.98592 | 0.98594 | 2827810 |

The model's performance across all classes is effectively demonstrated by the Confusion Matrix (Figure 30), which exhibits accurate classification with minimal misclassifications. In the case of Brute Force, where the Precision was 1, but the F1-Score was 0, looking at Figure 30, it can be seen that all classifications were classified with labels other than Brute Force.

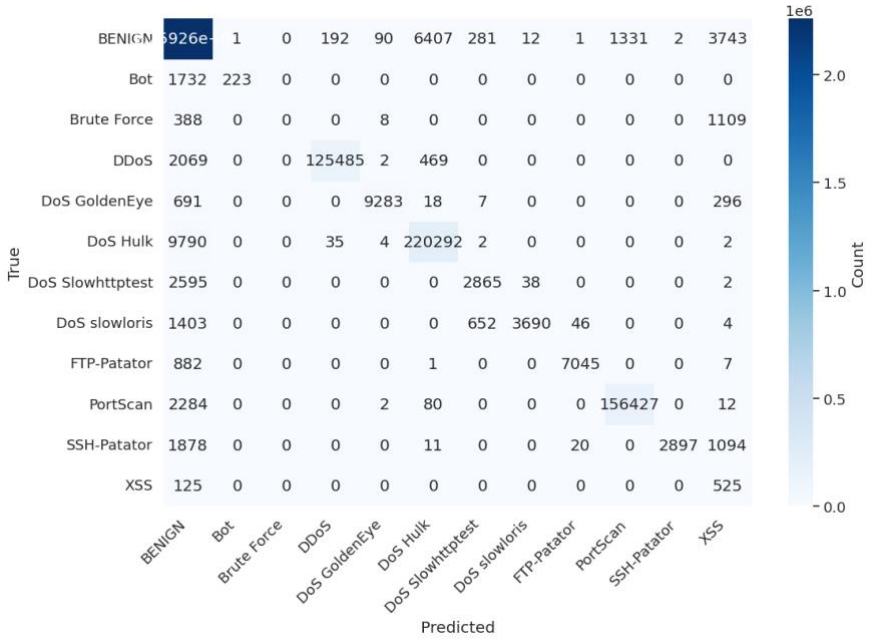


**Figure 26 CNN - D Confusion Matrix**

The Confusion Matrix (Figure 30) supports the model's performance across all classes, showing effective classification with minimal misclassifications. The Receiver Operating Characteristic (ROC) graph (Figure 31) also demonstrates high True Positive Rates (TPR) for all classes while maintaining low False Positive Rates (FPR).

Moreover, Figure 31 is the ROC curve of this model, but like the previous results, it does not correctly reflect the performance of the model. However, looking at Figure 32, it can be seen that the AUC score of the Precision-Recall Curve shows the performance of the model and the F1-Score.

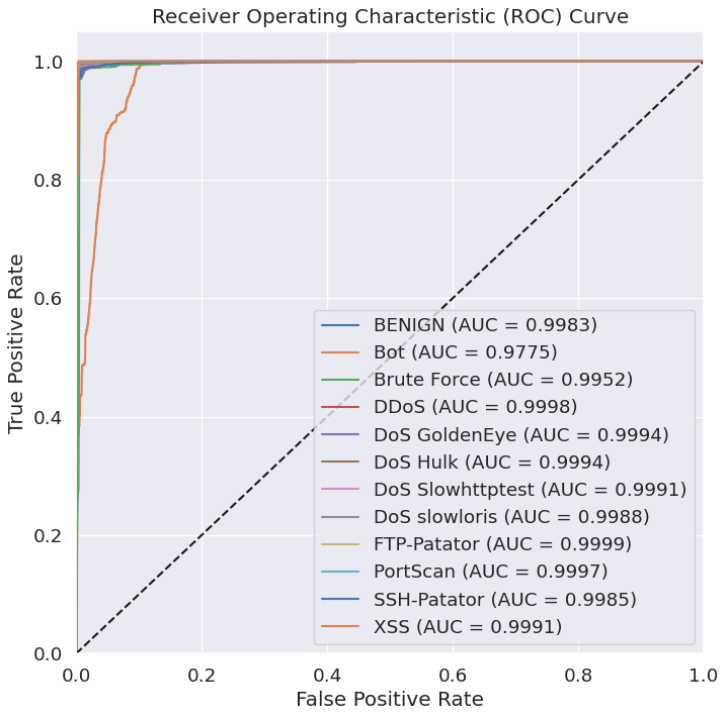


**Figure 27 CNN - D ROC Curve**

In particular, Figure 32 shows that the score of Brute Force, whose F1-Score was 0, is 0.0640, which is close to 0, contrary to the ROC AUC score of Figure 31, which is 0.9952. In addition, it can be confirmed that the three labels with the lowest scores in Figure 32 show the same results when comparing the performance of the model with the F1-Score metric: Bot (0.2936), Brute Force (0.0640), and XSS (0.2239).

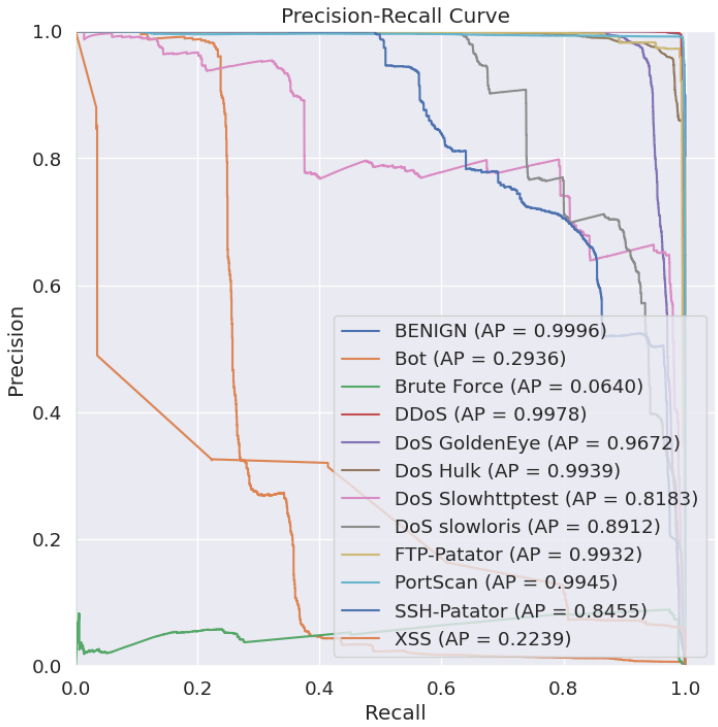


**Figure 28 CNN - D Precision-Recall Curve**

### *5.2.3 ANN with Selective Oversampling (F1-Score: 0.9819)*

Referring to Table 12, the ANN model shows the highest F1-Score (0.9819) when the Selective Oversampling technique is applied, which is a result showing a slight performance improvement of about 0.00115 higher than when no sampling was used. In addition, Table 15 shows the label classification report of the ANN model in Selective Oversampling, among which Bot (0.47054), Brute Force (0.05178), and XSS (0.1674) show low F1-Scores. The reason for this is that the number of instances of the corresponding label is tiny, which is the same reason as before.

**Table 11 ANN Label Classification Report**

| *Label* | *Precision* | *Recall* | *F1-Score* | *Support* |
|---|---|---|---|---|
| *BENIGN* | 0.99662 | 0.9811 | 0.9888 | 2271320 |
| *Bot* | 0.95171 | 0.31253 | 0.47054 | 1955 |
| *Brute Force* | 1 | 0.02658 | 0.05178 | 1505 |
| *DDoS* | 0.99883 | 0.98358 | 0.99115 | 128025 |
| *DoS GoldenEye* | 0.98984 | 0.94628 | 0.96757 | 10295 |
| *DoS Hulk* | 0.96151 | 0.99563 | 0.97827 | 230125 |
| *DoS Slowhttptest* | 0.96344 | 0.97255 | 0.96797 | 5500 |
| *DoS slowloris* | 0.98985 | 0.89215 | 0.93846 | 5795 |
| *FTP-Patator* | 0.96924 | 0.99672 | 0.98279 | 7935 |
| *PortScan* | 0.84321 | 0.97943 | 0.90623 | 158805 |
| *SSH-Patator* | 0.83844 | 0.91915 | 0.87694 | 5900 |
| *XSS* | 0.09134 | 1 | 0.1674 | 650 |
| *accuracy* | | | 0.98092 | 2827810 |
| *macro avg* | 0.88284 | 0.83381 | 0.77399 | 2827810 |
| *weighted avg* | 0.9845 | 0.98092 | 0.9819 | 2827810 |

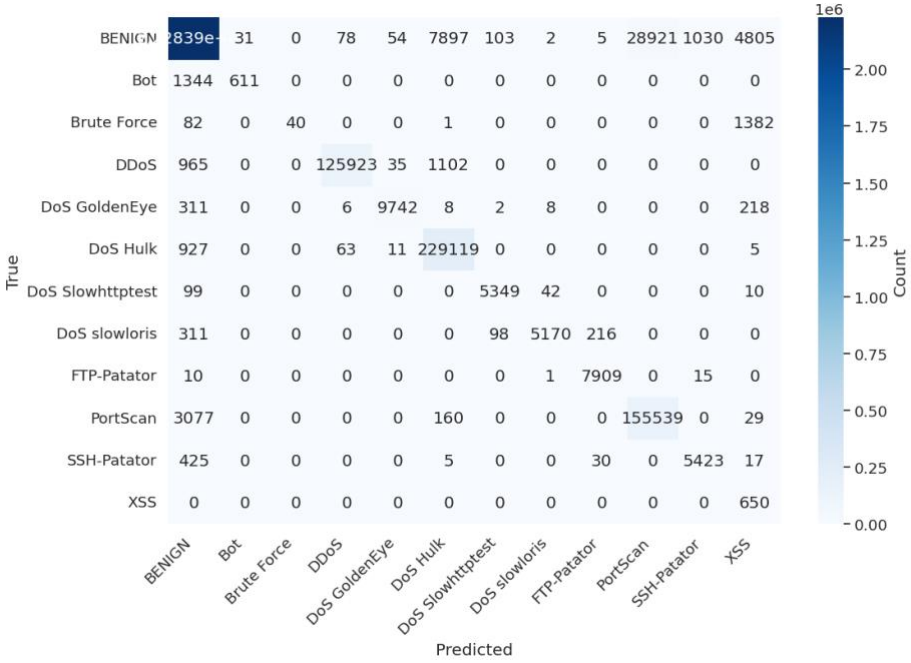


**Figure 29 ANN Confusion Matrix**

The model's performance across all classes is effectively demonstrated by the Confusion Matrix (Figure 33), which exhibits accurate classification with minimal misclassifications. The Confusion Matrix supports the model's performance across all classes, showing effective classification with minimal misclassifications. The Receiver Operating Characteristic (ROC) graph (Figure 34) also demonstrates high True Positive Rates (TPR) for all classes while maintaining low False Positive Rates (FPR).

Moreover, Figure 34 is the ROC curve of this model, but like the previous results, it does not correctly reflect the performance of the model. However, looking at Figure 35, it can be seen that the AUC score of the Precision-Recall Curve shows the performance of the model and the F1-Score.

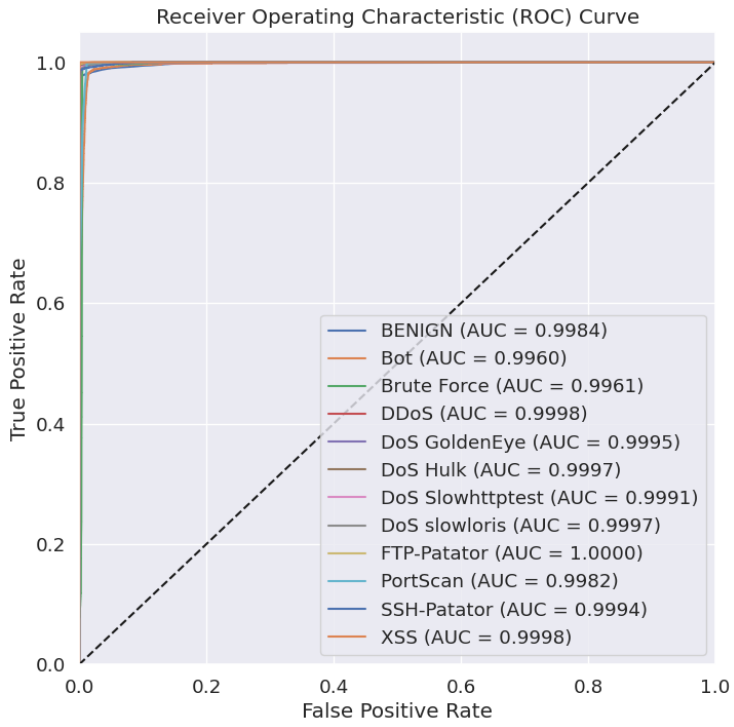


**Figure 30 ANN ROC Curve**

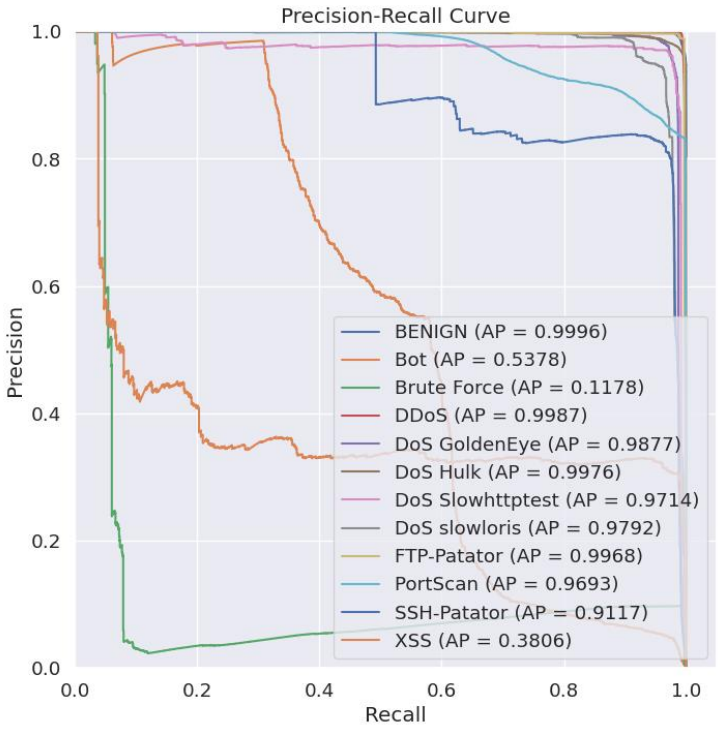


**Figure 31 ANN Precision-Recall Curve**

In particular, Figure 35 shows that the score of Brute Force, whose F1-Score was 0.05178, is 0.1178, contrary to the ROC AUC score of Figure 34, which is 0.9961. In addition, it can be confirmed that the three labels with the lowest scores in Figure 35 show the same results when comparing the performance of the model with the F1-Score metric: Bot (0.5378), Brute Force (0.1178), and XSS (0.3806).

### *5.2.4 DNN with SMOTE (F1-Score: 0.98199)*

Referring to Table 12, the DNN model shows the highest F1-Score (0.98199) when the SMOTE is applied, which is a result showing a slight performance improvement of about 0.00035 higher than when no sampling was used. In addition, Table 16 shows the

label classification report of the DNN model in SMOTE, among which Bot (0.428), Brute Force (0.04291), and XSS (0.15482) show low F1-Scores. The reason for this is that the number of instances of the corresponding label is tiny, which is the same reason as before.

**Table 12 DNN Label Classification Report**

| *Label* | *Precision* | *Recall* | *F1-Score* | *Support* |
|---|---|---|---|---|
| *BENIGN* | 0.99549 | 0.98221 | 0.98881 | 2271320 |
| *Bot* | 0.98165 | 0.27366 | 0.428 | 1955 |
| *Brute Force* | 1 | 0.02193 | 0.04291 | 1505 |
| *DDoS* | 0.99831 | 0.98385 | 0.99103 | 128025 |
| *DoS GoldenEye* | 0.98978 | 0.95017 | 0.96957 | 10295 |
| *DoS Hulk* | 0.97097 | 0.98964 | 0.98022 | 230125 |
| *DoS Slowhttptest* | 0.96696 | 0.96327 | 0.96512 | 5500 |
| *DoS slowloris* | 0.98618 | 0.91113 | 0.94717 | 5795 |
| *FTP-Patator* | 0.97702 | 0.9966 | 0.98671 | 7935 |
| *PortScan* | 0.84658 | 0.9729 | 0.90535 | 158805 |
| *SSH-Patator* | 0.83721 | 0.91525 | 0.87449 | 5900 |
| *XSS* | 0.0839 | 1 | 0.15482 | 650 |
| *accuracy* | | | 0.98097 | 2827810 |
| *macro avg* | 0.88617 | 0.83005 | 0.76952 | 2827810 |
| *weighted avg* | 0.98457 | 0.98097 | 0.98199 | 2827810 |

The model's performance across all classes is effectively demonstrated by the Confusion Matrix (Figure 36), which exhibits accurate classification with minimal misclassifications. The Confusion Matrix supports the model's performance across all classes, showing effective classification with minimal misclassifications. The Receiver Operating Characteristic (ROC) graph (Figure 37) also demonstrates high True Positive Rates (TPR) for all classes while maintaining low False Positive Rates (FPR).

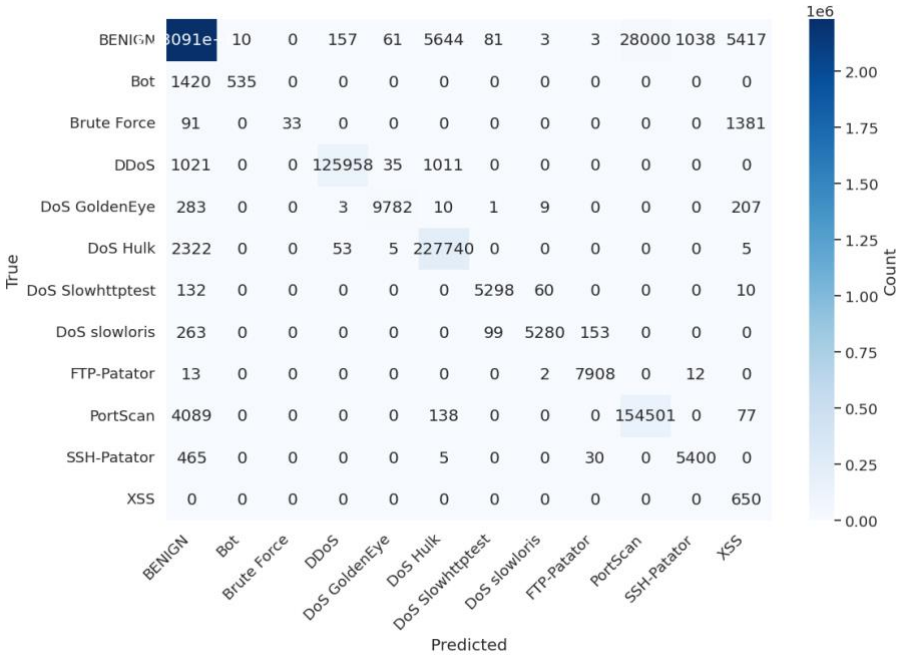


**Figure 32 DNN Confusion Matrix**

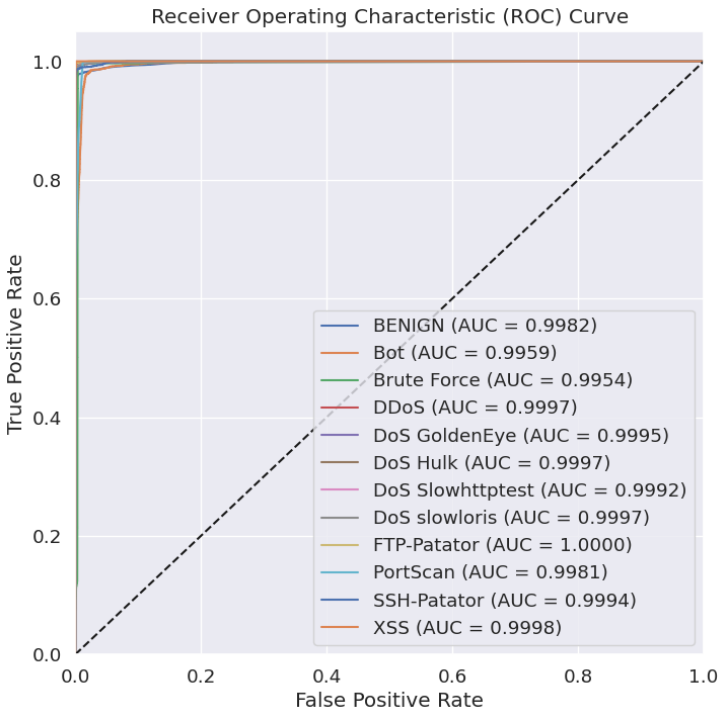


**Figure 33 DNN ROC Curve**

Moreover, Figure 37 is the ROC curve of this model, but like the previous results, it does not correctly reflect the performance of the model. However, looking at Figure 38, it can be seen that the AUC score of the Precision-Recall Curve shows the performance of the model and the F1-Score.

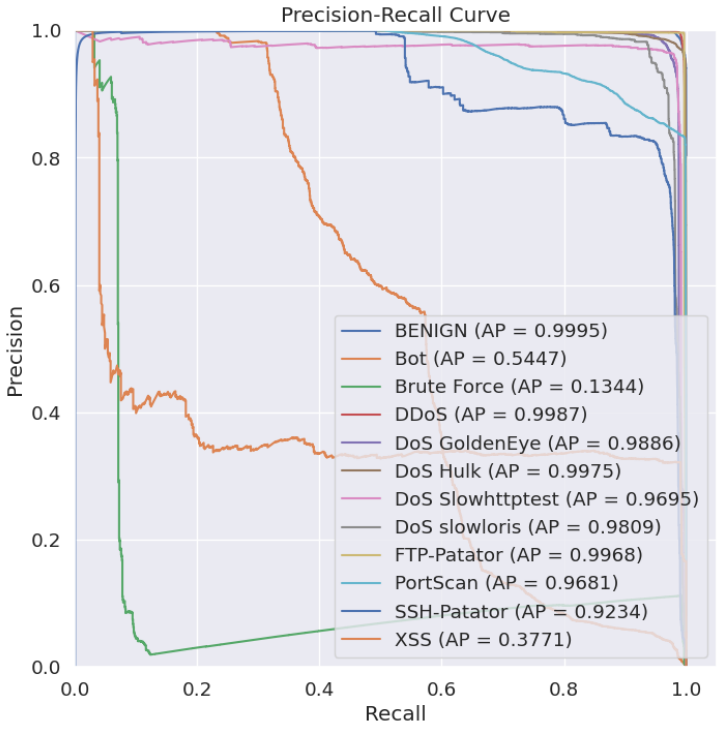


**Figure 34 DNN Precision-Recall Curve**

In particular, Figure 38 shows that the score of Brute Force, whose F1-Score was 0.04291, is 0.1344, contrary to the ROC AUC score of Figure 37, which is 0.9954. In addition, it can be confirmed that the three labels with the lowest scores in Figure 38 show the same results when comparing the performance of the model with the F1-Score metric: Bot (0.5447), Brute Force (0.1344), and XSS (0.3771).

### *5.2.5 LSTM with SMOTE (F1-Score: 0.98687)*

Referring to Table 12, the LSTM model shows the highest F1-Score (0.98687) when the SMOTE is applied, which is a result showing a slight performance improvement of about 0.00180 higher than when no sampling was used. In addition, Table 17 shows the label classification report of the LSTM model in SMOTE, among which Bot (0.50143), Brute Force (0.0728), and XSS (0.20734) show low F1-Scores. However, when experimenting with the LSTM model, it can be confirmed that the classification score for the Bot label, which previously had a low score, appears as 0.50143, which is more than half.

**Table 13 LSTM Label Classification Report**

| *Label* | *Precision* | *Recall* | *F1-Score* | *Support* |
|---|---|---|---|---|
| *BENIGN* | 0.99771 | 0.98674 | 0.99219 | 2271320 |
| *Bot* | 0.83353 | 0.35857 | 0.50143 | 1955 |
| *Brute Force* | 0.93443 | 0.03787 | 0.0728 | 1505 |
| *DDoS* | 0.99784 | 0.98585 | 0.99181 | 128025 |
| *DoS GoldenEye* | 0.98534 | 0.9594 | 0.97219 | 10295 |
| *DoS Hulk* | 0.96576 | 0.99262 | 0.97901 | 230125 |
| *DoS Slowhttptest* | 0.9641 | 0.98618 | 0.97501 | 5500 |
| *DoS slowloris* | 0.98758 | 0.96083 | 0.97402 | 5795 |
| *FTP-Patator* | 0.99484 | 0.99672 | 0.99578 | 7935 |
| *PortScan* | 0.89684 | 0.99534 | 0.94353 | 158805 |
| *SSH-Patator* | 0.7847 | 0.85864 | 0.82001 | 5900 |
| *XSS* | 0.11566 | 1 | 0.20734 | 650 |
| *accuracy* | | | 0.98633 | 2827810 |
| *macro avg* | 0.87153 | 0.84323 | 0.78543 | 2827810 |
| *weighted avg* | 0.98852 | 0.98633 | 0.98687 | 2827810 |

The model's performance across all classes is effectively demonstrated by the Confusion Matrix (Figure 40), which exhibits accurate classification with minimal

misclassifications. The Confusion Matrix supports the model's performance across all classes, showing effective classification with minimal misclassifications.

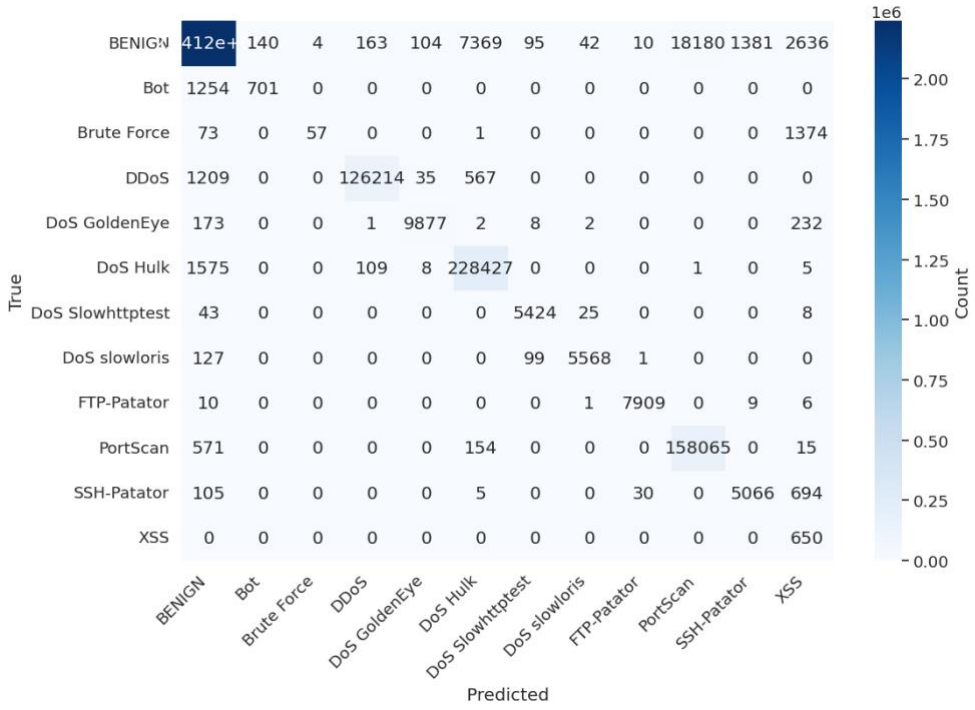


**Figure 35 LSTM Confusion Matrix**

The Receiver Operating Characteristic (ROC) graph (Figure 41) also demonstrates high True Positive Rates (TPR) for all classes while maintaining low False Positive Rates (FPR).

Moreover, Figure 41 is the ROC curve of this model, but like the previous results, it does not correctly reflect the performance of the model. However, looking at Figure 38, it can be seen that the AUC score of the Precision-Recall Curve shows the performance of the model and the F1-Score.

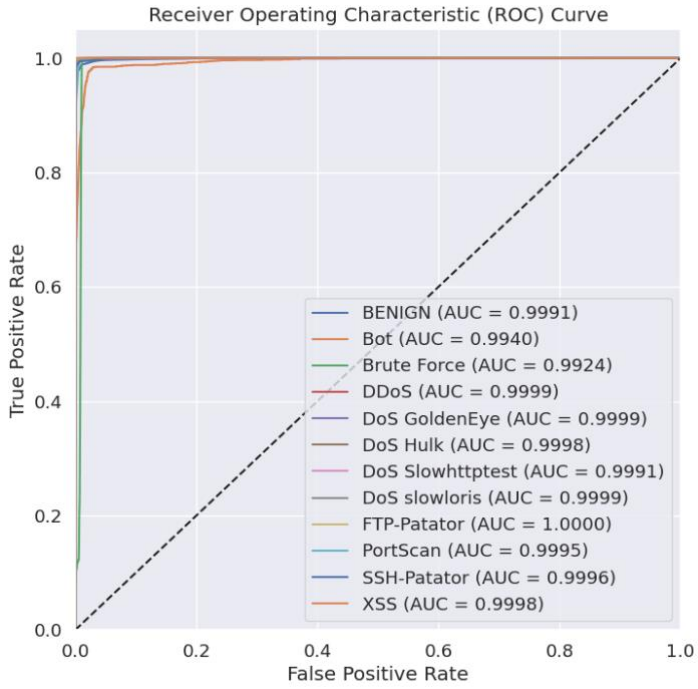


**Figure 36 LSTM ROC Curve**

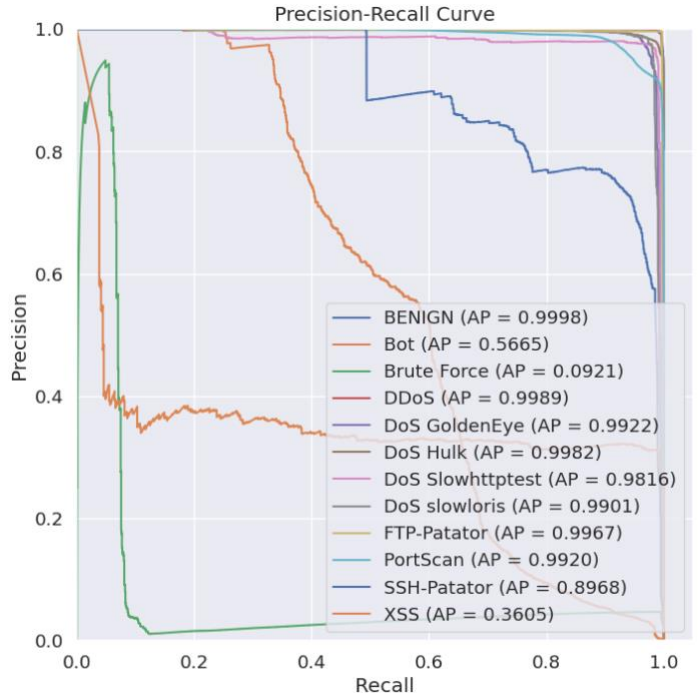


**Figure 372 LSTM Precision-Recall Curve**

In particular, Figure 42 shows that the score of Brute Force, whose F1-Score was 0.0728, is 0.0921, contrary to the ROC AUC score of Figure 41, which is 0.9924. In addition, it can be confirmed that the three labels with the lowest scores in Figure 42 show the same results when comparing the performance of the model with the F1-Score metric: Bot (0.5665), Brute Force (0.0921), and XSS (0.3605).

### *5.2.6 GRU with SMOTE (F1-Score: 0.98355)*

Referring to Table 12, the GRU model shows the highest F1-Score (0.98355) when the SMOTE is applied, which is a result showing a slight performance improvement of about 0.00041 higher than when no sampling was used. In addition, Table 18 shows the label classification report of the GRU model in SMOTE, among which Bot (0.51059), Brute Force (0.06174), and XSS (0.17253) show low F1-Scores. However, when experimenting with the GRU model like LSTM, it can be confirmed that the classification score for the Bot label, which previously had a low score, appears as 0.51059, which is more than half.

**Table 14 GRU Label Classification Report**

| *Label* | *Precision* | *Recall* | *F1-Score* | *Support* |
|---|---|---|---|---|
| *BENIGN* | 0.99739 | 0.98301 | 0.99015 | 2271320 |
| *Bot* | 0.76931 | 0.3821 | 0.51059 | 1955 |
| *Brute Force* | 0.96 | 0.03189 | 0.06174 | 1505 |
| *DDoS* | 0.99811 | 0.98452 | 0.99127 | 128025 |
| *DoS GoldenEye* | 0.98786 | 0.94823 | 0.96764 | 10295 |
| *DoS Hulk* | 0.96461 | 0.99255 | 0.97838 | 230125 |
| *DoS Slowhttptest* | 0.96664 | 0.98509 | 0.97578 | 5500 |
| *DoS slowloris* | 0.99048 | 0.95168 | 0.97069 | 5795 |
| *FTP-Patator* | 0.99522 | 0.99685 | 0.99603 | 7935 |
| *PortScan* | 0.85386 | 0.99684 | 0.91983 | 158805 |
| *SSH-Patator* | 0.89613 | 0.58492 | 0.70782 | 5900 |
| *XSS* | 0.09441 | 1 | 0.17253 | 650 |
| *accuracy* | | | 0.98274 | 2827810 |
| *macro avg* | 0.87283 | 0.81981 | 0.7702 | 2827810 |
| *weighted avg* | 0.98598 | 0.98274 | 0.98355 | 2827810 |

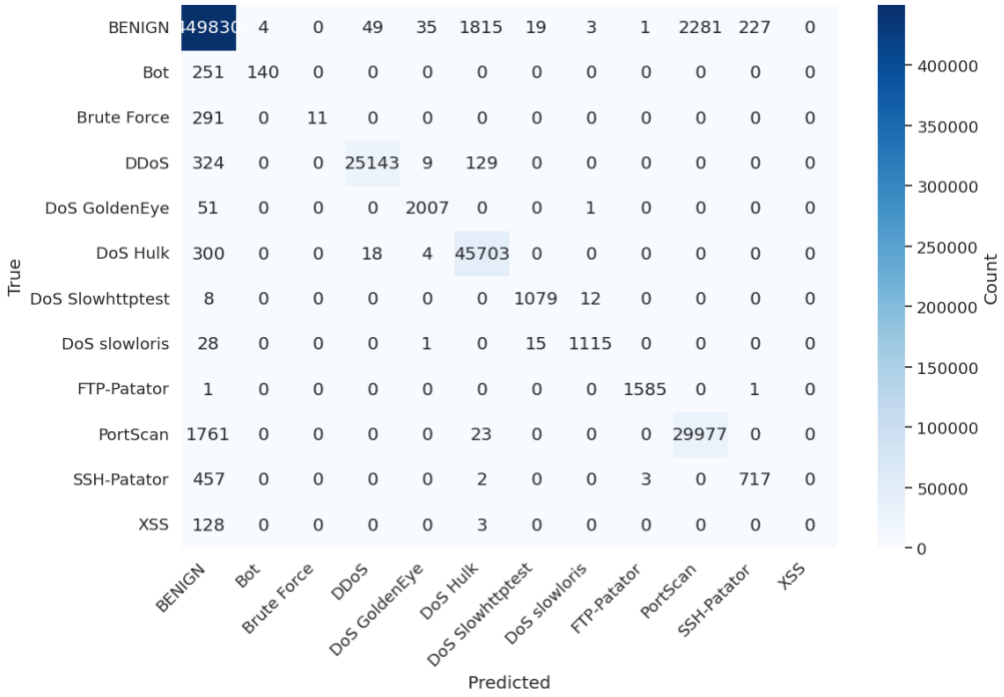


**Figure 38 GRU Confusion Matrix**

The model's performance across all classes is effectively demonstrated by the Confusion Matrix (Figure 42), which exhibits accurate classification with minimal misclassifications. The Confusion Matrix supports the model's performance across all classes, showing effective classification with minimal misclassifications.

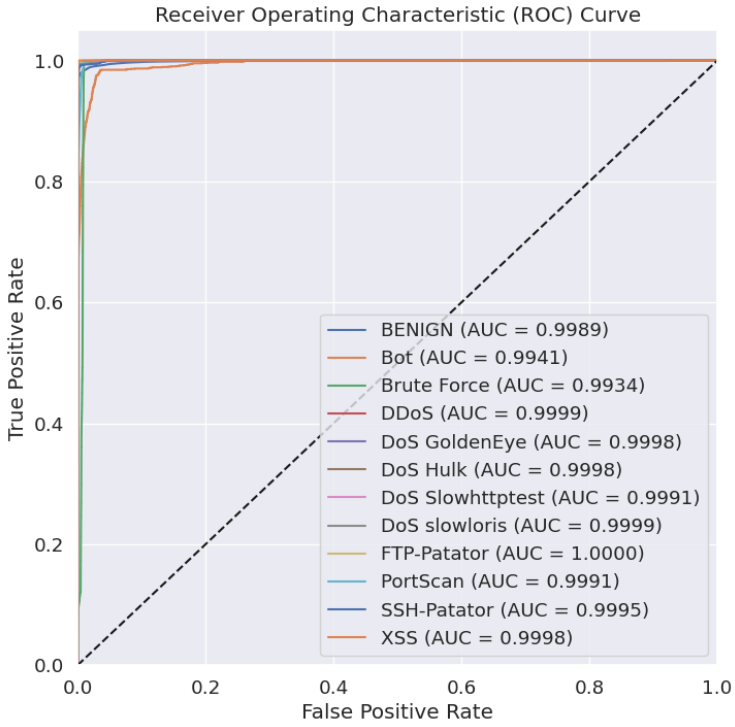


**Figure 39 GRU ROC Curve**

The Receiver Operating Characteristic (ROC) graph (Figure 43) also demonstrates high True Positive Rates (TPR) for all classes while maintaining low False Positive Rates (FPR).

Moreover, Figure 43 is the ROC curve of this model, but like the previous results, it does not correctly reflect the performance of the model. However, looking at Figure 44,

it can be seen that the AUC score of the Precision-Recall Curve shows the performance of the model and the F1-Score.

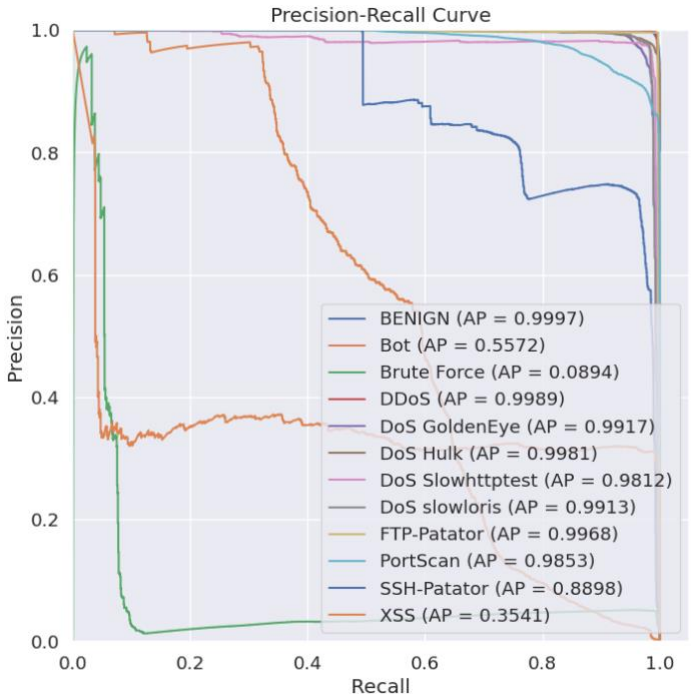


**Figure 404 LSTM Precision-Recall Curve**

In particular, Figure 44 shows that the score of Brute Force, whose F1-Score was 0.06174, is 0.0894, contrary to the ROC AUC score of Figure 43, which is 0.9934. In addition, it can be confirmed that the three labels with the lowest scores in Figure 44 show the same results when comparing the performance of the model with the F1-Score metric: Bot (0.5572), Brute Force (0.0894), and XSS (0.3541).

### *5.2.7 CLAttNet with SMOTE (F1-Score: 0.99288)*

In Table 12, the CLAttNet model showed the highest F1-Score (0.99288) when SMOTE was applied, showing a slight performance improvement of about 0.00737 higher than when sampling was not used. Moreover, it has the highest performance score among all other models.

In addition, Table 19 shows the label classification report of the CLAttNet model in SMOTE, among which Bot (0.50683), Brute Force (0.16888), and XSS (0.23147) show low F1-Scores. However, when experimenting with the CLAttNet model like LSTM and GRU, it can be confirmed that the classification score for the Bot label, which previously had a low score, appears as 0.50683, which is more than half.

**Table 15 CLAttNet Label Classification Report**

| *Label* | *Precision* | *Recall* | *F1-Score* | *Support* |
|---|---|---|---|---|
| *BENIGN* | 0.99773 | 0.99453 | 0.99612 | 2271320 |
| *Bot* | 0.98091 | 0.34169 | 0.50683 | 1955 |
| *Brute Force* | 0.91503 | 0.09302 | 0.16888 | 1505 |
| *DDoS* | 0.99478 | 0.99502 | 0.9949 | 128025 |
| *DoS GoldenEye* | 0.99114 | 0.96668 | 0.97876 | 10295 |
| *DoS Hulk* | 0.98121 | 0.9893 | 0.98524 | 230125 |
| *DoS Slowhttptest* | 0.90318 | 0.86836 | 0.88543 | 5500 |
| *DoS slowloris* | 0.99119 | 0.95168 | 0.97104 | 5795 |
| *FTP-Patator* | 0.98714 | 0.9966 | 0.99185 | 7935 |
| *PortScan* | 0.97046 | 0.99725 | 0.98368 | 158805 |
| *SSH-Patator* | 0.8539 | 0.84102 | 0.84741 | 5900 |
| *XSS* | 0.13094 | 0.99692 | 0.23147 | 650 |
| *accuracy* | | | 0.9926 | 2827810 |
| *macro avg* | 0.89147 | 0.83601 | 0.79513 | 2827810 |
| *weighted avg* | 0.99391 | 0.9926 | 0.99288 | 2827810 |

The model's performance across all classes is effectively demonstrated by the Confusion Matrix (Figure 44), which exhibits accurate classification with minimal misclassifications. The Confusion Matrix supports the model's performance across all classes, showing effective classification with minimal misclassifications.

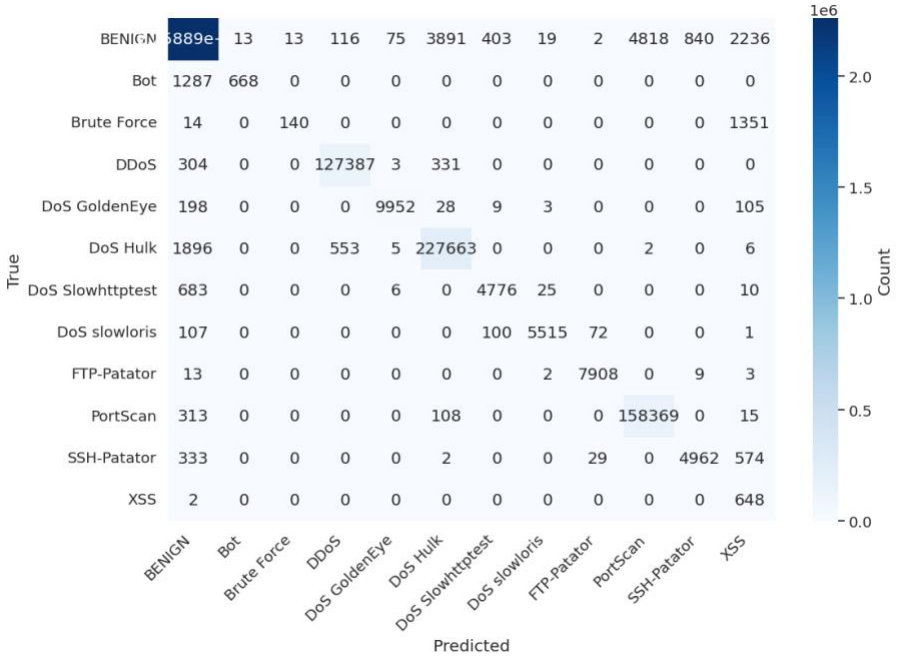


**Figure 41 CLAttNet Confusion Matrix**

The Receiver Operating Characteristic (ROC) graph (Figure 43) also demonstrates high True Positive Rates (TPR) for all classes while maintaining low False Positive Rates (FPR).

Moreover, Figure 45 is the ROC curve of this model, but like the previous results, it does not correctly reflect the performance of the model. However, looking at Figure 46, it can be seen that the AUC score of the Precision-Recall Curve shows the performance of the model and the F1-Score.

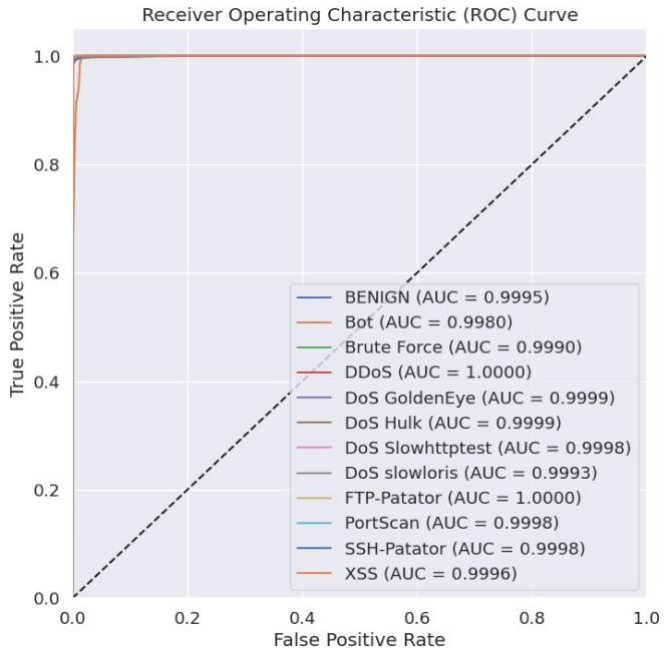


**Figure 42 CLAttNet ROC Curve**

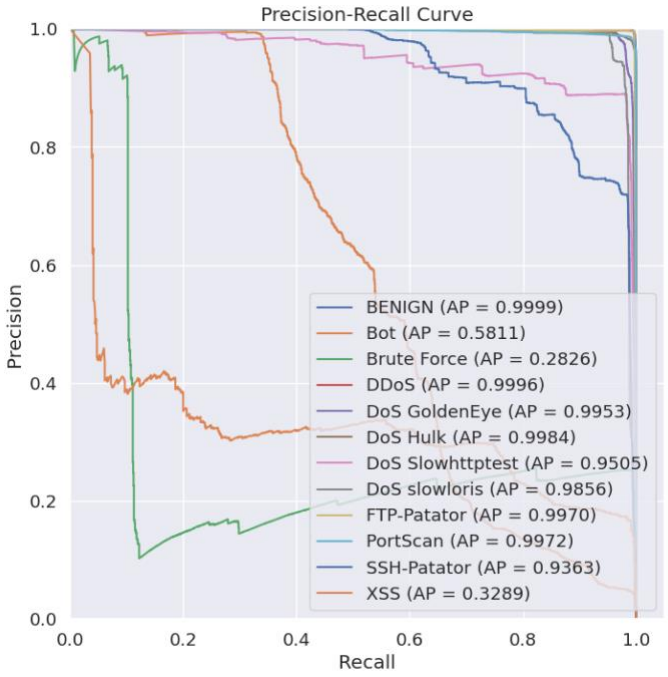


**Figure 436 CLAttNet Precision-Recall Curve**

In particular, Figure 46 shows that the score of Brute Force, whose F1-Score was 0.16888, is 0.2826, contrary to the ROC AUC score of Figure 45, which is 0.9990, almost perfect score. In addition, it can be confirmed that the three labels with the lowest scores in Figure 46 show the same results when comparing the performance of the model with the F1-Score metric: Bot (0.5811), Brute Force (0.2826), and XSS (0.3289).

## 5.3 Limitations and Future Work

This work has shed light on how different deep learning models and sampling strategies might improve intrusion detection systems using the CICIDS 2017 dataset. The important findings must be balanced, however, by acknowledging and addressing some limitations.

### *5.3.1 Hyperparameter Optimization and Experimental Conditions*

One of the main limitations of this study is the constraint on hyperparameter optimization due to the experimental environment. The experiments were conducted using Google Colab Pro Plus, which imposes a maximum runtime limit of 24 hours. As a result, some more complex models, such as the CLAttNet, could not be fully optimized concerning their hyperparameters, as the time required for 5-fold cross-validation and GridSearchCV exceeded the allotted time.

To overcome this limitation, future research should aim to secure better experimental environments that allow for more extensive hyperparameter optimization. This would enable a more comprehensive comparison of the deep learning models, considering their performance with default settings and their potential for improvement

through fine-tuning. Additionally, the availability of more powerful computer resources would enable researchers to use more exact search methods, such random search and Bayesian optimization, which would result in hyperparameter setups with higher efficacy. In addition, hyperparameter optimization should be extended to other models in the study, further enhancing the overall comparison and ensuring that each model is evaluated at its best possible performance. This would involve fine-tuning each model's architecture and considering other hyperparameters, such as the learning rate, weight initialization, and optimization algorithms, among others.

### *5.3.2 Feature Selection and Engineering*

The limited number of traits that were employed in the experiments for the study is another drawback. Using the SelectKBest approach, the number of features was limited to 40 due to the restrictions of the experimental setting. Because of this strategy, some models might not have been able to properly capture the complexity of the dataset and perform at their best. [29]

The number of features utilized for each model should be varied in future study, and other feature selection and feature engineering strategies should be tested. [25], [29] This would make it possible to gain a deeper knowledge of how various feature sets affect each deep learning model's performance and perhaps find new combinations that improve intrusion detection systems' precision and effectiveness. In addition, more complex feature selection methods, such as Recursive Feature Elimination (RFE) or feature significance analysis based on tree-based models, could be employed to comprehend the relationships between features and their effects on prediction accuracy. [42]

Future study might also look into semi-supervised or unsupervised feature learning techniques like autoencoders or variational autoencoders, which can be used to find hidden patterns in the data and provide more useful feature representations. By providing more robust features that are resistant to noise and irrelevant data, this could improve the performance of deep learning models for intrusion detection.

### *5.3.3 Extension and Generalization*

The CICIDS 2017 dataset served as the foundation for this study's conclusions. It is crucial to validate and generalize these findings across numerous datasets and realistic circumstances, even though they offer insightful information about the effectiveness of various deep learning models and sampling approaches for intrusion detection. In order to ensure the models' resilience and flexibility, future research should examine the efficiency of these models and methodologies on different datasets, such as the NSL-KDD or the more recent CSE-CIC-IDS2018 dataset. [43], [44]

Additionally, investigating how these models operate in other network environments, such as IoT networks, cloud-based systems, or industrial control systems, can offer insightful information about their suitability and ability to address the particular problems faced by these environments. [45] The ultimate goal of this would be to create an intrusion detection system that is more reliable and flexible and can effectively address the constantly changing landscape of cyber threats.

### *5.3.4 Ensemble and Hybrid Models*

In this study, the CLAttNet model, which integrates the LSTM, Convolution, and Attention processes, performed better than other models. This result raises the possibility

of investigating other ensemble and hybrid models that integrate the benefits of several deep learning architectures. [46] Future studies might look into the creation of novel hybrid models that combine several deep learning methods, such as CNNs and GRUs or other recurrent architectures with attention mechanisms. [32]

Stacking, bagging, and boosting are examples of ensemble approaches that can improve the efficiency of intrusion detection systems. The accuracy of ensemble techniques, which aggregate the results of multiple models, can be improved while reducing the impact of each model's shortcomings. However, this strategy would necessitate the creation of effective and efficient methods for model variety and combination, as well as rigorous testing to identify the best ensemble configurations for intrusion detection.

### *5.3.5 Real-Time Intrusion Detection and Scalability*

While the precision, recall, accuracy, and F-score performance of deep learning models were the main emphasis of this study, it is equally critical to take into account the scalability and real-time application of these models in actual IDS installations. [45] The appropriateness of each model for real-time intrusion detection in diverse network contexts should be assessed in future study by taking into account each model's computing efficiency, latency, and resource requirements. [47]

Additionally, the scalability of these models should be evaluated in terms of their propensity to adjust and function well as network traffic volume and attack complexity rise. [47] This could entail examining methods for parallelizing, distributed learning, and model compression, as well as looking at the potential of online learning and incremental

learning strategies to allow the models to be continuously improved as new data becomes available. [32]

In conclusion, utilizing the CICIDS 2017 dataset, our study has shed important light on how different deep learning models and sampling strategies perform when used to improve intrusion detection systems. The results lay a strong platform for future studies aimed at creating deep learning-based IDS systems that are more effective, precise, and scalable. Future research can support ongoing efforts to defend our digital world against the constantly changing panorama of cyber threats by addressing the constraints and investigating the potential directions for improvement described in this section.

# CHAPTER VI

# CONCLUSIONS

Our study aimed to evaluate the performance of different deep learning models and sampling techniques for intrusion detection using the CICIDS 2017 dataset. We performed a comprehensive analysis based on Precision, Recall, F-score, Accuracy, ROC-AUC, and Precision-Recall Curve AUC and compared the results.

Studies have shown that the CLAttNet model combined with the SMOTE sampling technique consistently outperforms the other models in terms of F1-Score. Utilizing convolution, LSTM, and attention processes, this model has shown excellent capabilities in detecting cyber threats. Moreover, rather than comparing performance measurement metrics such as Precision, Recall, and Accuracy alone, we found that the F1-Score, the harmonic average of Precision and Recall, was more suitable for performance measurement. In addition, the performance of each model was further confirmed through the analysis of the ROC curve and the Precision-Recall curve. It was also confirmed in the process that the Precision-Recall Curve reflects model performance better than the ROC Curve.

It is also worth noting that the SMOTE sampling technique consistently improved performance in most models except CNN-Simple and ANN. Furthermore, applying SMOTE to each model improved its performance, highlighting its effectiveness as a sampling technique.

However, our study has the following limitations. Further studies are needed in the future, focusing on optimizing hyperparameters, exploring alternative feature selection techniques, validating results using different data sets and real-world examples, exploring other ensemble and hybrid models, and evaluating the scalability and real-time applicability of IDS deployments.

In conclusion, our study based on the CICIDS 2017 dataset provides valuable insights into the performance of different deep-learning models and sampling techniques for intrusion detection. Including ROC and Precision-Recall curves in the analysis improves the comprehensiveness of the results. Our work highlights the effectiveness of the SMOTE sampling technique in achieving performance improvements for most models. It serves as the basis for future efforts to develop more accurate, accurate, and scalable deep learning-based IDS systems that will ultimately serve as a basis for evolving cyberspace.

REFERENCES

[1] A. Momand, S. U. Jan, and N. Ramzan, “A Systematic and Comprehensive Survey of Recent Advances in Intrusion Detection Systems Using Machine Learning: Deep Learning, Datasets, and Attack Taxonomy,” *Journal of Sensors*, vol. 2023. Hindawi Limited, 2023. doi: 10.1155/2023/6048087.

[2] D. E. Denning, "An intrusion-detection model," in IEEE Transactions on Software Engineering, vol. SE-13, no. 2, pp. 222-232, Feb. 1987, doi: 10.1109/TSE.1987.232894.

[3] H. Liu and B. Lang, “Machine learning and deep learning methods for intrusion detection systems: A survey,” *Applied Sciences (Switzerland)*, vol. 9, no. 20. MDPI AG, Oct. 01, 2019. doi: 10.3390/app9204396.

[4] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, SciTePress, 2018, pp. 108–116. doi: 10.5220/0006639801080116.

[5] K. A. Scarfone and P. M. Mell, “Guide to Intrusion Detection and Prevention Systems (IDPS),” Gaithersburg, MD, 2007. doi: 10.6028/NIST.SP.800-94.

[6] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, “Network intrusion detection system: A systematic study of machine learning and deep learning approaches,” *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, Jan. 2021, doi: 10.1002/ett.4150.

[7] A. Pradhan, "Support vector machine-A survey," International Journal of Emerging Technology and Advanced Engineering, vol. 2, no. 8, pp. 82-85, 2012. [Online]. Available: www.ijetae.com

[8] Z. Zhang, “Introduction to machine learning: K-nearest neighbors,” *Ann Transl Med*, vol. 4, no. 11, Jun. 2016, doi: 10.21037/atm.2016.03.37.

[9] Y. Y. Song and Y. Lu, “Decision tree methods: applications for classification and prediction,” *Shanghai Arch Psychiatry*, vol. 27, no. 2, pp. 130–135, Apr. 2015, doi: 10.11919/j.issn.1002-0829.215044.

[10] R. Y. Choi, A. S. Coyner, J. Kalpathy-Cramer, M. F. Chiang, and J. Peter Campbell, “Introduction to machine learning, neural networks, and deep learning,” *Transl Vis Sci Technol*, vol. 9, no. 2, 2020, doi: 10.1167/tvst.9.2.14.

[11] S.-H. Han, K. W. Kim, S. Kim, and Y. C. Youn, “Artificial Neural Network: Understanding the Basic Concepts without Mathematics,” *Dement Neurocogn Disord*, vol. 17, no. 3, p. 83, 2018, doi: 10.12779/dnd.2018.17.3.83.

[12] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," arXiv preprint arXiv:1511.08458, 2015. [Online]. Available: https://arxiv.org/abs/1511.08458.

[13] H. Sak, A.W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in Proceedings of INTERSPEECH, 2014, pp. 338-342.

[14] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” Dec. 2014, [Online]. Available: http://arxiv.org/abs/1412.3555

[15] G. Karatas, O. Demir, and O. K. Sahingoz, “Increasing the Performance of Machine Learning-Based IDSs on an Imbalanced and Up-to-Date Dataset,” *IEEE Access*, vol. 8, pp. 32150–32162, 2020, doi: 10.1109/ACCESS.2020.2973219.

[16] M. R. Parsaei, S. M. Rostami, and R. Javidan, “A Hybrid Data Mining Approach for Intrusion Detection on Imbalanced NSL-KDD Dataset,” 2016. [Online]. Available: www.ijacsa.thesai.org

[17] T. Elhassan, A. M. Aljourf, F. Al-Mohanna, and M. Shoukri, "Classification of Imbalance Data using Tomek Link (T-Link) Combined with Random Under-sampling (RUS) as a Data Reduction Method," Global Journal of Technology and Optimization, vol. 01, 2016, doi: 10.4172/2229-8711.S1111.

[18] S. Gazzah, A. Hechkel, and N. E. Ben Amara, “A hybrid sampling method for imbalanced data,” in *12th International Multi-Conference on Systems, Signals and Devices, SSD 2015*, Institute of Electrical and Electronics Engineers Inc., Dec. 2015. doi: 10.1109/SSD.2015.7348093.

[19] D. M. W. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," arXiv preprint arXiv:2010.16061, Oct. 2020, doi: 10.48550/arXiv.2010.16061.

[20] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Inf Process Manag*, vol. 45, no. 4, pp. 427–437, Jul. 2009, doi: 10.1016/j.ipm.2009.03.002.

[21] T. Saito and M. Rehmsmeier, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," in PLOS ONE, vol.10, no.3, e0118432, March 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0118432

[22] Z. Pelletier and M. Abualkibash, “Evaluating the CIC IDS-2017 Dataset Using Machine Learning Methods and Creating Multiple Predictive Models in the Statistical Computing Language R,” *International Research Journal of Advanced Engineering and Science*, vol. 5, no. 2, pp. 187–191, 2020.

[23] Hyeong-wook Boo, “An Assessment of North Korean Cyber Threats,” *JOEAA*, vol. 31, no. Institute for National Security Strategy, pp. 97–117, Jun. 2017.

[24] T. Lin *et al.*, “Susceptibility to spear-phishing emails: Effects of internet user demographics and email content,” *ACM Transactions on Computer-Human Interaction*, vol. 26, no. 5, Sep. 2019, doi: 10.1145/3336141.

[25] I. Guyon, A. Elisseeff, and L. P. Kaelbling, Eds., "An introduction to variable and feature selection," Journal of Machine Learning Research, vol. 3, no. 7-8, pp. 1157–1182, 2003, doi: 10.1162/153244303322753616.

[26] R. Kohavi and G. H. John, "Wrappers for feature subset selection," Artificial Intelligence, vol. 97, no. 1–2, pp. 273-324, 1997, doi: 10.1016/S0004-3702(97)00043-X.

[27] M. Abdulraheem and N. Al-Dabagh, "A detailed analysis of new intrusion detection dataset," Journal of Theoretical and Applied Information Technology, vol. 97, pp. 4519-4537, 2019.

[28] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 436–444, May 27, 2015. doi: 10.1038/nature14539.

[29] G. Chandrashekar and F. Sahin, “A survey on feature selection methods,” *Computers and Electrical Engineering*, vol. 40, no. 1, pp. 16–28, Jan. 2014, doi: 10.1016/j.compeleceng.2013.11.024.

[30] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," in IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 4, pp. 491-502, April 2005, doi: 10.1109/TKDE.2005.66.

[31] M. E. Mihailescu *et al.*, “The proposition and evaluation of the roedunet-simargl2021 network intrusion detection dataset,” *Sensors*, vol. 21, no. 13, Jul. 2021, doi: 10.3390/s21134319.

[32] A. L. Buczak and E. Guven, “A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection,” *IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1153–1176, Apr. 2016, doi: 10.1109/COMST.2015.2494502.

[33] A. Koutsoukas, K. J. Monaghan, X. Li, and J. Huan, “Deep-learning: Investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data,” *J Cheminform*, vol. 9, no. 1, Jun. 2017, doi: 10.1186/s13321-017-0226-y.

[34] Y. Zhang, J. Zheng, Y. Jiang, G. Huang, and R. Chen, “A text sentiment classification modeling method based on coordinated CNN-LSTM-attention model,” *Chinese Journal of Electronics*, vol. 28, no. 1, pp. 120–126, Jan. 2019, doi: 10.1049/cje.2018.11.004.

[35] Z. Niu, G. Zhong, and H. Yu, “A review on the attention mechanism of deep learning,” *Neurocomputing*, vol. 452, pp. 48–62, Sep. 2021, doi: 10.1016/j.neucom.2021.03.091.

[36] F. Provost, "Machine Learning from Imbalanced Data Sets 101," in AAAI'2000 Workshop on Imbalanced Data Sets, Jul. 2000, CeDER-PP-2000-02. [Online]. Available: http://hdl.handle.net/2451/27763

[37] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," arXiv preprint arXiv:1106.1813, 2011, doi: 10.48550/arXiv.1106.1813.

[38] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning," in Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I (ICIC'05), vol. Part I, Springer-Verlag, Berlin, Heidelberg, pp. 878-887, 2005, doi: 10.1007/11538059_91

[39] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2 (IJCAI'95), Morgan Kaufmann Publishers Inc.,

San Francisco, CA, USA, pp. 1137-1143, 1995. [Online]. Available: https://ai.stanford.edu/~ronnyk/accEst.pdf

[40] D.J. Hand and R.J. Till, "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems," Machine Learning, vol. 45, pp. 171-186, Nov. 2001, doi: 10.1023/A:1010920819831.

[41] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in Proceedings of the 23rd International Conference on Machine Learning (ICML '06), New York, NY, USA, Association for Computing Machinery, 2006, pp. 233-240, doi: 10.1145/1143844.1143874.

[42] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 2, no. 1, pp. 41-50, Feb. 2018, doi: 10.1109/TETCI.2017.2772792.

[43] N. Moustafa and J. Slay, “UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set),” in *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Dec. 2015. doi: 10.1109/MilCIS.2015.7348942.

[44] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of tor traffic using time based features,” in *ICISSP 2017 - Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, SciTePress, 2017, pp. 253–262. doi: 10.5220/0006105602530262.

[45] Q. Niyaz, W. Sun, A. Y. Javaid, and M. Alam, “A deep learning approach for network intrusion detection system,” in *EAI International Conference on Bio-inspired Information and Communications Technologies (BICT)*, 2015. doi: 10.4108/eai.3-12-2015.2262516.

[46] D. P. Gaikwad and R. C. Thool, “Intrusion detection system using bagging ensemble method of machine learning,” in *Proceedings - 1st International Conference on Computing, Communication, Control and Automation, ICCUBEA 2015*, Institute of Electrical and Electronics Engineers Inc., Jul. 2015, pp. 291–295. doi: 10.1109/ICCUBEA.2015.61.

[47] F. Pierazzi, G. Apruzzese, M. Colajanni, A. Guido, and M. Marchetti, "Scalable architecture for online prioritisation of cyber threats," in Proceedings of the 2017 IEEE 9th International Conference on Cyber Conflict (CYCON), May 2017, pp. 1-18, doi: 10.23919/CYCON.2017.8240337.

[48] M. Hafeez, M. Rashid, H. Tariq, Z. Abideen, S. Alotaibi, and M. Sinky, "Performance Improvement of Decision Tree: A Robust Classifier Using Tabu Search Algorithm," in Applied Sciences, vol. 11, no. 15, p. 6728, 2021, doi: 10.3390/app11156728.

APPENDIX A

SOURCE CODE – PREPROCESSING & SAMPLING

## A.1 Import Modules and Initial Preprocessing

```
import os
import glob
import time

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn import preprocessing
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier,
RandomForestRegressor
from sklearn.feature_selection import SelectKBest, chi2,
mutual_info_classif
from sklearn.metrics import (accuracy_score, completeness_score,
confusion_matrix,
                             homogeneity_score,
precision_recall_fscore_support as score, v_measure_score)
from sklearn.model_selection import train_test_split, KFold,
StratifiedKFold
from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Model
from tensorflow.keras.layers import (Input, Conv1D, MaxPooling1D,
LSTM,
                                     Flatten, Dense, Dropout,
Multiply, Activation, BatchNormalization, TimeDistributed)
from tensorflow.keras.layers import BatchNormalization

from sklearn.metrics import precision_recall_fscore_support,
accuracy_score, classification_report, confusion_matrix, roc_curve,
auc, roc_auc_score
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import RandomOverSampler
```

```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
```

```
# Initialize an empty list to store DataFrames
dataframes = []

# Iterate through files in the specified directory
for dirname, _, filenames in os.walk('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/MLcsv/'):
    for i, filename in enumerate(filenames):
        # Print the file path
        print(os.path.join(dirname, filename))

        # Read CSV file and append it to the list of DataFrames
        dataframes.append(pd.read_csv(os.path.join(dirname,
filename)))

# Concatenate all DataFrames in the list
dataset = pd.concat(dataframes)

# Clean up memory by deleting the list of DataFrames
del dataframes

# Remove duplicate columns
dataset = dataset.loc[:, ~dataset.columns.duplicated()]

# Replace infinite values with NaN
dataset = dataset.replace([np.inf, -np.inf], np.nan)

# Drop rows with missing values
dataset = dataset.dropna()
```

### A.2 Remove unimportant columns

```
dataset = dataset.applymap(lambda x: np.nan if x in ['Heartbleed',
'Web Attack \ufffd Sql Injection', 'Infiltration'] else x)
dataset = dataset.dropna()
dataset['Label'] = dataset['Label'].replace({'Web Attack \ufffd Brute
Force': 'Brute Force', 'Web Attack \ufffd XSS': 'XSS'})
```

### A.3 Save and Load the dataset

```
dataset.to_csv("dataset.csv", index=False)
dataset = pd.read_csv("/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/dataset.csv")
```

### A.4 Split the dataset

```
# xs=feature vectors, ys=labels
xs = dataset.drop('Label', axis=1)
ys = dataset['Label']

# split dataset - stratified
X_train, X_test, y_train, y_test = train_test_split(xs, ys,
test_size=0.2, random_state=0, stratify=ys)


# Remove columns with only identical values
column_names = np.array(list(X_train))
to_drop = [x for x in column_names if
len(X_train.groupby([x]).size().unique()) == 1]
X_train = X_train.drop(to_drop, axis=1)
X_test = X_test.drop(to_drop, axis=1)
```

### A.5 Normalization of features

```
min_max_scaler = MinMaxScaler().fit(X_train)
X_train = min_max_scaler.transform(X_train)
X_test = min_max_scaler.transform(X_test)
```

### A.6 Feature Selection using SelectKBest

```
kbest = SelectKBest(score_func=chi2, k=40).fit(X_train, y_train)
X_train = kbest.transform(X_train)
X_test = kbest.transform(X_test)
```

## A.7 Sampling

```
# Instantiate a LabelEncoder for label transformation
label_encoder = LabelEncoder()

# Perform label encoding
y_train_encoded = label_encoder.fit_transform(y_train)

# Resampling strategies

# 1. No Resampling - original data
X_train_no_resampling = X_train
y_train_no_resampling = y_train_encoded

# 2. Selective Oversampling - increase minority class
oversampler = RandomOverSampler(sampling_strategy='minority')
X_train_selective_oversampling, y_train_selective_oversampling =
oversampler.fit_resample(X_train, y_train_encoded)

# 3. Selective Undersampling - decrease majority class
undersampler = RandomUnderSampler(sampling_strategy='majority')
X_train_selective_undersampling, y_train_selective_undersampling =
undersampler.fit_resample(X_train, y_train_encoded)

# 4. Combined Resampling - combination of SMOTE and
RandomUnderSampler
resample_steps = Pipeline([
    ('oversampling', SMOTE(sampling_strategy='minority')),
    ('undersampling',
RandomUnderSampler(sampling_strategy='majority'))
])

# Implement the resampling pipeline on the training data
X_train_combined, y_train_combined =
resample_steps.fit_resample(X_train, y_train_encoded)

# 5. SMOTE Resampling (Standard) - Synthetic Minority Over-sampling
Technique
smote_resampler = SMOTE(sampling_strategy='minority',
random_state=42)
X_train_smote, y_train_smote =
smote_resampler.fit_resample(X_train, y_train_encoded)
```

```
# Save the result
np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/X_train_no_resampling.npz',
X_train_no_resampling)
np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/y_train_no_resampling.npz',
y_train_no_resampling)

np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/X_train_selective_oversampling.npz',
X_train_selective_oversampling)
np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/y_train_selective_oversampling.npz',
y_train_selective_oversampling)

np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/X_train_selective_undersampling.npz',
X_train_selective_undersampling)
np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/y_train_selective_undersampling.npz',
y_train_selective_undersampling)

np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/X_train_combined.npz', X_train_combined)
np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/y_train_combined.npz', y_train_combined)

np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/X_train_smote.npz', X_train_smote)
np.savez('/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/y_train_smote.npz', y_train_smote)
```

### A.8 Reshaping Data for Deep Learning Models

```
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

# Load resampled data from earlier
resampling_methods = ['no_resampling', 'selective_oversampling',
'selective_undersampling', 'combined', 'smote']
```

```
X_train_resampled = {}
y_train_resampled = {}

y_train_encoded = {}
y_train_onehot = {}

for method in resampling_methods:
    data = np.load(f'/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/X_train_{method}.npz')
    X_train_resampled[method] = data['arr_0']

    data = np.load(f'/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/y_train_{method}.npz')
    y_train_resampled[method] = data['arr_0']

# Apply label encoding and one-hot encoding for labels
encoder = LabelEncoder()

for method in resampling_methods:
    y_train_encoded[method] =
encoder.fit_transform(y_train_resampled[method])
    y_train_onehot[method] =
to_categorical(y_train_encoded[method])

# Convert class labels to a list of strings
label_names = encoder.classes_

# Adjust one-hot encoding to y_test
y_test_encoded = encoder.fit_transform(y_test)
y_test_onehot = to_categorical(y_test_encoded)

# Adjust input data shape for ANN and DNN
X_train_ann_dnn = {}
X_test_ann_dnn = X_test.reshape(X_test.shape[0], X_test.shape[1])

for method in resampling_methods:
    X_train_ann_dnn[method] =
X_train_resampled[method].reshape(X_train_resampled[method].shape[0
], X_train_resampled[method].shape[1])

# Adjust input data shape for CNN
```

```
X_train_cnn = {}
X_test_cnn = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

for method in resampling_methods:
    X_train_cnn[method] =
X_train_resampled[method].reshape(X_train_resampled[method].shape[0
], X_train_resampled[method].shape[1], 1)

# Adjust input data shape for LSTM and GRU
X_train_lstm_gru = {}
X_test_lstm_gru = X_test.reshape(X_test.shape[0], 1,
X_test.shape[1])

for method in resampling_methods:
    X_train_lstm_gru[method] =
X_train_resampled[method].reshape(X_train_resampled[method].shape[0
], 1, X_train_resampled[method].shape[1])
```

APPENDIX B

SOURCE CODE – DEEP LEARNING MODEL

## B.1 CNN-Simple & Deep

### *B.1.1 Create the Models*

```
output_dir = '/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/Results/'
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D,
Flatten, Dense, Dropout
from tensorflow.keras.layers import BatchNormalization
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report,
confusion_matrix, roc_curve, auc, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import os
from sklearn.metrics import precision_recall_fscore_support


# Define the CNN model
def create_cnn_simple_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    x = Conv1D(32, 3, activation='relu')(inputs)
    x = BatchNormalization()(x)
    x = MaxPooling1D(pool_size=2)(x)
    x = Conv1D(64, 3, activation='relu')(x)
    x = BatchNormalization()(x)
    x = MaxPooling1D(pool_size=2)(x)
    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)

    output_label = Dense(num_classes, activation='softmax',
name='output_label')(x)
```

```
    model = Model(inputs=inputs, outputs=output_label)

    model.compile(optimizer='adam',
loss=tf.keras.losses.CategoricalCrossentropy(),
metrics=[tf.keras.metrics.Recall(), 'accuracy'])

    return model

# Define the CNN model
def create_cnn_deep_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    x = Conv1D(64, 3, activation='relu')(inputs)
    x = BatchNormalization()(x)
    x = MaxPooling1D(pool_size=2)(x)

    x = Conv1D(128, 3, activation='relu')(x)
    x = BatchNormalization()(x)
    x = MaxPooling1D(pool_size=2)(x)

    x = Conv1D(256, 3, activation='relu')(x)
    x = BatchNormalization()(x)
    x = MaxPooling1D(pool_size=2)(x)

    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)

    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)

    output_label = Dense(num_classes, activation='softmax',
name='output_label')(x)

    model = Model(inputs=inputs, outputs=output_label)
    model.compile(optimizer='adam',
loss=tf.keras.losses.CategoricalCrossentropy(),
metrics=[tf.keras.metrics.Recall(), 'accuracy'])

    return model
```

```
def create_cnn_model(input_shape, num_classes, model_type):
    if model_type == 'simple':
        return create_cnn_simple_model(input_shape, num_classes)
    elif model_type == 'deep':
        return create_cnn_deep_model(input_shape, num_classes)
    else:
        raise ValueError('Invalid model type')
```

### *B.1.2 Cross Validation and Save Results*

```
def evaluate_and_save_results(model, X_test, y_test_encoded,
y_test_onehot, method, label_names, fold_number, output_dir,
fold_results, roc_auc_scores, ap_scores, num_classes):
    # Evaluate the model
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_test_onehot = to_categorical(y_test_encoded,
num_classes=num_classes)  # Convert to one-hot encoding

    y_pred = model.predict(X_test)

    # Use the integer-encoded labels for generating the
classification report
    report = classification_report(y_test_encoded, y_pred_classes,
target_names=label_names, zero_division=1, digits=5)

    # Save the classification report and confusion matrix
    output_method_dir =
f"{output_dir}/{model_type}/{method}_fold_{fold_number}/"
    os.makedirs(output_method_dir, exist_ok=True)

    with
open(f'{output_method_dir}report_{method}_fold_{fold_number}.txt',
'w') as f:
        f.write(report)

    # Plot and save confusion matrix
    plot_confusion_matrix(y_test_encoded, y_pred_classes,
label_names, output_method_dir, f"{method}_fold_{fold_number}")

    # Calculate FPR, TPR and AUC for each class
```

```
    y_test_roc = np.zeros((len(y_test_encoded), num_classes))
    for i, label in enumerate(label_names):
        y_test_roc[:, i] = (y_test_encoded == i).astype(int)

    # Calculate ROC AUC and AP for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    precision = dict()
    recall = dict()
    average_precision = dict()

    for i, label in enumerate(label_names):
        y_test_binary = np.where(y_test_encoded == i, 1, 0)
        y_pred_binary = y_pred[:, i]
        fpr[i], tpr[i], _ = roc_curve(y_test_binary, y_pred_binary)
        roc_auc[i] = auc(fpr[i], tpr[i])
        precision[i], recall[i], _ =
precision_recall_curve(y_test_binary, y_pred_binary)
        average_precision[i] =
average_precision_score(y_test_binary, y_pred_binary)

    # Save the scores
    for i, label in enumerate(label_names):
        if label not in roc_auc_scores:
            roc_auc_scores[label] = []
        if label not in ap_scores:
            ap_scores[label] = []
        roc_auc_scores[label].append(roc_auc[i])
        ap_scores[label].append(average_precision[i])

    # Plot and save ROC curve
    plot_roc_curve(y_test_roc, y_pred, label_names,
output_method_dir, f"{method}_fold_{fold_number}")


    # Plot and save Precision-Recall curve
    plot_pr_curve(y_test_onehot, y_pred, label_names,
output_method_dir, f"{method}_fold_{fold_number}")

    # Update fold_results with the scores
```

```
    precision, recall, fscore, _ =
precision_recall_fscore_support(y_test_encoded, y_pred_classes,
average='weighted', zero_division=1)
    accuracy = accuracy_score(y_test_encoded, y_pred_classes)
    fold_results['precision'].append(precision)
    fold_results['recall'].append(recall)
    fold_results['fscore'].append(fscore)
    fold_results['accuracy'].append(accuracy)

    # Save the results after each fold
    fold_results_df = pd.DataFrame(fold_results)
    fold_results_df.to_csv(f'{output_method_dir}fold_results.csv',
index=False)


def evaluate_model_with_cross_validation(X, y, y_encoded, X_test,
y_test_encoded, input_shape, num_classes, label_names, n_splits=5,
model_type='simple'):
    # Perform cross-validation
    cv = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    fold_number = 0
    fold_results = {'recall': [], 'precision': [], 'accuracy': [],
'fscore': []}
    roc_auc_scores = {}
    ap_scores = {}

    # Lists to store prediction results and true labels across all
folds
    y_preds = []
    y_trues = []

    for train_index, val_index in cv.split(X, y_encoded):
        X_train, X_validate = X[train_index], X[val_index]
        y_train, y_validate = y[train_index], y[val_index]
        y_train_encoded, y_validate_encoded =
y_encoded[train_index], y_encoded[val_index]

        # Create a new model for each fold
        model = create_cnn_model(input_shape, num_classes,
model_type)
```

```
        model.fit(X_train, y_train, validation_data=(X_validate,
y_validate), epochs=20, batch_size=64, verbose=1, workers=8,
use_multiprocessing=True)

        # Evaluate and save results
        output_dir = '/content/drive/MyDrive/ML-
DL/Research/CICIDS2017/Results/'
        evaluate_and_save_results(model, X_test, y_test_encoded,
y_test_onehot, method, label_names, fold_number, output_dir,
fold_results, roc_auc_scores, ap_scores, num_classes)

        # Store predictions and true labels
        y_pred = model.predict(X_test)
        y_preds.append(y_pred)
        y_trues.append(y_test_encoded)

        fold_number += 1

    # Concatenate all predictions and true labels
    y_preds = np.concatenate(y_preds, axis=0)
    y_trues = np.concatenate(y_trues, axis=0)

    # Generate final report and confusion matrix
    y_pred_classes = np.argmax(y_preds, axis=1)
    report = classification_report(y_trues, y_pred_classes,
target_names=label_names, zero_division=1, digits=5)

    output_final_dir =
f"{output_dir}/{model_type}/{method}_final_results/"
    os.makedirs(output_final_dir, exist_ok=True)

    with open(f'{output_final_dir}final_report.txt', 'w') as f:
        f.write(report)

    plot_confusion_matrix(y_trues, y_pred_classes, label_names,
output_final_dir, 'final')

    # Calculate FPR, TPR and AUC for each class
    y_test_roc = np.zeros((len(y_trues), num_classes))

    for i, label in enumerate(label_names):
        y_test_roc[:, i] = (y_trues == i).astype(int)
```

```
    plot_roc_curve(y_test_roc, y_preds, label_names,
output_final_dir, 'final')

    # Calculate precision, recall for each class
    precision = dict()
    recall = dict()
    average_precision = dict()
    for i, label in enumerate(label_names):
        precision[i], recall[i], _ =
precision_recall_curve(y_test_roc[:, i], y_preds[:, i])
        average_precision[i] =
average_precision_score(y_test_roc[:, i], y_preds[:, i])

    plot_pr_curve(y_test_roc, y_preds, label_names,
output_final_dir, 'final')

    # Calculate ROC AUC and AP for the final results
    final_fpr = dict()
    final_tpr = dict()
    final_roc_auc = dict()
    final_precision = dict()
    final_recall = dict()
    final_average_precision = dict()

    y_test_roc = np.zeros((len(y_trues), len(label_names)))
    for i, label in enumerate(label_names):
        y_test_roc[:, i] = (y_trues == i).astype(int)

    for i, label in enumerate(label_names):
        final_fpr[i], final_tpr[i], _ = roc_curve(y_test_roc[:, i],
y_preds[:, i])
        final_roc_auc[i] = auc(final_fpr[i], final_tpr[i])
        final_precision[i], final_recall[i], _ =
precision_recall_curve(y_test_roc[:, i], y_preds[:, i])
        final_average_precision[i] =
average_precision_score(y_test_roc[:, i], y_preds[:, i])

    save_scores_to_csv(final_roc_auc, output_dir,
f"{model_type}_{method}_final_roc_auc_scores.csv")
    save_scores_to_csv(final_average_precision, output_dir,
f"{model_type}_{method}_final_ap_scores.csv")
```

```
    # Compute the mean and standard deviation of the scores
    mean_results = {metric: np.mean(values) for metric, values in
fold_results.items()}
    std_results = {metric: np.std(values) for metric, values in
fold_results.items()}

    save_scores_to_csv(roc_auc_scores, output_dir,
f"{model_type}_{method}_roc_auc_scores.csv")
    save_scores_to_csv(ap_scores, output_dir,
f"{model_type}_{method}_ap_scores.csv")

    return mean_results, std_results
```

### *B.1.3 Plot Function Code*

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import precision_recall_fscore_support,
accuracy_score
import numpy as np
import os
import time
from sklearn.metrics import classification_report,
confusion_matrix, roc_curve, auc, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import csv
from sklearn.metrics import precision_recall_curve,
average_precision_score

def plot_confusion_matrix(y_true, y_pred, label_names, output_dir,
suffix):
    cm = confusion_matrix(y_true, y_pred)
    cm_df = pd.DataFrame(cm, index=label_names,
columns=label_names)

    plt.figure(figsize=(12, 8))
    sns.set(font_scale=1.2)
    ax = sns.heatmap(cm_df, annot=True, fmt='g', cmap='Blues',
cbar_kws={'label': 'Count'})
```

```
    ax.set_xlabel('Predicted')
    ax.set_ylabel('True')
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45,
horizontalalignment='right')
    plt.savefig(f'{output_dir}confusion_matrix_{suffix}.png',
bbox_inches='tight')

def plot_roc_curve(y_test_roc, y_pred, label_names, output_dir,
suffix):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i, label in enumerate(label_names):
        fpr[i], tpr[i], _ = roc_curve(y_test_roc[:, i], y_pred[:,
i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure(figsize=(8, 8))
    for i, label in enumerate(label_names):
        plt.plot(fpr[i], tpr[i], label='%s (AUC = %0.4f)' % (label,
roc_auc[i]))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc="lower right")
    plt.savefig(f'{output_dir}roc_curve_{suffix}.png',
bbox_inches='tight')

def plot_pr_curve(y_true, y_pred, label_names, output_dir, suffix):
    precision = dict()
    recall = dict()
    average_precision = dict()
    for i, label in enumerate(label_names):
        precision[i], recall[i], _ =
precision_recall_curve(y_true[:, i], y_pred[:, i])
        average_precision[i] = average_precision_score(y_true[:,
i], y_pred[:, i])

    plt.figure(figsize=(8, 8))
```

```
    for i, label in enumerate(label_names):
        plt.plot(recall[i], precision[i], label='%s (AP = %0.4f)' %
(label, average_precision[i]))
    plt.xlim([0.0, 1.05])
    plt.ylim([0.0, 1.0])
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve')
    plt.legend(loc="lower right")
    plt.savefig(f'{output_dir}pr_curve_{suffix}.png',
bbox_inches='tight')

def save_scores_to_csv(scores, output_dir, file_name):
    # Convert scalar scores to lists
    scores = {k: [v] for k, v in scores.items()}
    df = pd.DataFrame(scores)
    df.to_csv(os.path.join(output_dir, file_name))
```

While each model does require its own optimized Cross Validation and Test Set code for proper execution, we will primarily focus on documenting the 'Create the Model' section for subsequent models. This decision is made to avoid redundancy, as the Cross Validation and Test Set codes are generally similar in structure across different models. Therefore, we aim to reduce repetition while maintaining essential information for each model's implementation.

### B.2 ANN & DNN

```
# Define the ANN model
def create_ann_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    x = Flatten()(inputs)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
```

```
    output_label = Dense(num_classes, activation='softmax',
name='output_label')(x)

    model = Model(inputs=inputs, outputs=output_label)

    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy',
tf.keras.metrics.Recall()])

    return model


# Define the DNN model
def create_dnn_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    x = Flatten()(inputs)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)

    output_label = Dense(num_classes, activation='softmax',
name='output_label')(x)

    model = Model(inputs=inputs, outputs=output_label)

    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy',
tf.keras.metrics.Recall()])

    return model
```

### B.3 LSTM & GRU

```
# Define the LSTM model
def create_lstm_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    x = LSTM(128)(inputs)
    x = Dropout(0.5)(x)
```

```
    output_label = Dense(num_classes, activation='softmax',
name='output_label')(x)

    model = Model(inputs=inputs, outputs=output_label)

    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy',
tf.keras.metrics.Recall()])

    return model

# Define the GRU model
def create_gru_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    x = GRU(128)(inputs)
    x = Dropout(0.5)(x)

    output_label = Dense(num_classes, activation='softmax',
name='output_label')(x)

    model = Model(inputs=inputs, outputs=output_label)

    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy',
tf.keras.metrics.Recall()])

    return model
```

### B.4 CLAttNet

```
# Attention layer
class Attention(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        self.W = self.add_weight(shape=(input_shape[-1],
input_shape[-1]), initializer='random_normal', trainable=True)
        self.b = self.add_weight(shape=(input_shape[-1],),
initializer='zeros', trainable=True)
```

```
        super(Attention, self).build(input_shape)

    def call(self, x):
        q = tf.nn.tanh(tf.linalg.matmul(x, self.W) + self.b)
        a = tf.nn.softmax(q, axis=1)
        return Multiply()([x, a])

    def compute_output_shape(self, input_shape):
        return input_shape

def create_hybrid_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    # CNN Layers
    x = Conv1D(32, 3, activation='relu')(inputs)
    x = BatchNormalization()(x)
    x = MaxPooling1D(pool_size=2)(x)
    x = Conv1D(64, 3, activation='relu')(x)
    x = BatchNormalization()(x)
    x = MaxPooling1D(pool_size=2)(x)

    # LSTM Layer
    x = LSTM(128, return_sequences=True)(x)

    # Attention Layer
    x = Attention()(x)
    x = TimeDistributed(Flatten())(x)

    # Dense Layers
    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)

    output_label = Dense(num_classes, activation='softmax',
name='output_label')(x)

    model = Model(inputs=inputs, outputs=output_label)
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy',
tf.keras.metrics.Recall()])

    return model
```