NEAR EAST UNIVERSITY

INSTITUTE OF GRADUATE STUDIES

DEPARTMENT OF MECHATRONICS ENGINEERING

A COMPARATIVE STUDY OF SUPPORT VECTOR MACHINE AND CONVOLUTIONAL NEURAL NETWORK MODELS FOR INTRUSION DETECTION

MASTER THESIS

OKAH ONYEKACHI MICHAEL

NICOSIA

2023

NEAR EAST UNIVERSITY

INSTITUTE OF GRADUATE STUDIES

DEPARTMENT OF MECHATRONICS ENGINEERING

A COMPARATIVE STUDY OF SUPPORT VECTOR MACHINE AND CONVOLUTIONAL NEURAL NETWORK MODELS FOR INTRUSION DETECTION

MASTER THESIS

OKAH ONYEKACHI MICHAEL

NICOSIA

2023

Approval

We certify that we have read the thesis submitted by Okah Onyekachi Michael titled **"A Comparative Study Support Vector Machine and Convolutional Neural Network Models for Intrusion Detection"** and that in our combined opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Educational Sciences.

Examining Committee	Name-Surname	Signature
Head of the Committee:	Assist. Prof. Dr. John Bush IDOKO	.....
Committee Member:	Assist. Prof. Dr. Mohammad Karimzadeh Kolamroudi	.....
Supervisor:	Prof. Dr. Rahib ABİYEV	.....

Approved by the Head of the Department

...../...../20.....

.....

Assoc. Prof. Dr. Hüseyin HACI

Head of Department

Approved by the Institute of Graduate Studies

...../...../20.....

Prof. Dr. Kemal Hüsnü Can Başer

Head of the Institute

Declaration

I hereby declare that this thesis is the result of my original research work conducted under the supervision of Prof. Dr. Rahib H. Abiyev and it has not been previously submitted for any degree or examination in any other university or institution.

I affirm that all sources used, including published or unpublished works of others, have been duly acknowledged and referenced in accordance with the guidelines provided by Near East University. Any contributions made by others to this research have also been appropriately credited.

I take full responsibility for the content and findings presented in this thesis and confirm that all data, figures, and results are authentic and accurately represent the outcomes of the conducted research.

Furthermore, I understand that any misrepresentation or failure to abide by academic integrity and ethical standards could result in severe consequences, including the rejection of this thesis or the revocation of any degrees obtained as a result of this work.

Okah Onyekachi Michael

20th December 2023

Acknowledgement

I would like to express my heartfelt gratitude to the Almighty God for granting me the wisdom, strength, and perseverance throughout this research journey. Without His blessings and guidance, this thesis would not have been possible.

I extend my sincere appreciation to my supervisor, for his invaluable support, patience, and expert guidance. His mentorship and constructive feedback have been instrumental in shaping the direction of this research.

I would like to express my gratitude to Dr. Ibrahim Shuaibu for his valuable insights and contributions to this research. His expertise and constructive feedback have immensely enriched the quality of this work.

I am deeply indebted to my parents for their unwavering love, encouragement, and continuous belief in my abilities. Their sacrifices and unwavering support have been the driving force behind my academic pursuits.

Furthermore, I am also thankful to my friends and family for their encouragement, motivation, and understanding during the course of this endeavor. Their unwavering support and words of encouragement have been a constant source of inspiration.

Lastly, I acknowledge the contributions of all individuals and institutions whose work and research I have cited and utilized in this thesis. Their scholarly endeavors have laid the foundation for my own exploration and understanding.

Thank you all for being an integral part of this journey. Your support and encouragement have been pivotal in making this thesis a reality.

Okah Onyekachi Michael.

Abstract

A Comparative Study of Support Vector Machine and Convolutional Neural Network Models for Intrusion Detection

Okah Onyekachi Michael

MSc Department of Mechatronics Engineering

Supervisor Prof. Dr. Rahib H. Abiyev

January, 2023 112 Pages

In this rapidly evolving era of technology and automation, Mechatronics systems are more exposed to the threat of cyber-attacks and intrusions as there are more increasingly interconnected and reliant on digital communication. Failure to safeguard network security can lead to data breaches and unauthorized access by potential hackers. In this study, we delve into the development and evaluation of two deep learning models and we compare their performance on an internet firewall dataset which was collected from a school library. This analysis in this study was conducted using Python programming language in the Google Colabaratory environment which encompasses data preprocessing, splitting the data into train set and test set and feature scaling. CNN and SVM model training and evaluation with the test data. The models were used to classify the data into classes such as 'Allow', 'Deny', 'Drop' and 'Reset-Both' and we used various evaluation metrics such as accuracy, precision, recall, F1-score, and ROC AUC, to assess the performance of each models. In this study, we set out to thoroughly investigate the classification of network traffic action using deep learning models. The effective creation and assessment of Convolutional Neural Network (CNN) and Support Vector Machine (SVM) models, both of which shown excellent classification accuracy and performance, are among our significant findings. The SVM model showed robust classification abilities even in settings with non-linear data separability, whereas the CNN model, distinguished by its neural architecture, excelled better in capturing subtle data patterns. By integrating our models into a real-time intrusion detection system, our research was also made applicable to the outside world, demonstrating the usefulness and potential significance of our work. Finally, our study highlights the value of deep learning in boosting cybersecurity measures while also making significant contributions to the field of network security. The specific needs of

the activity at hand determine whether to use the CNN or SVM models. Organizations can strengthen their network security and proactively protect against threats by utilizing these methods.

Keywords **Intrusion Detection, CNN Model, SVM Model, Mechatronics**.

A Comparative Study of Support Vector Machine and Convolutional Neural Network Models for Intrusion Detection

Okah Onyekachi Michael

MSc Department of Mechatronics Engineering

Supervisor Prof. Dr. Rahib H. Abiyev

January, 2023 112 Pages

Hızla gelişen teknoloji ve otomasyon çağında, Mekatronik sistemler giderek daha fazla birbirine bağlı hale geldikçe ve dijital iletişime bağımlı hale geldikçe siber saldırı ve izinsiz giriş tehdidine daha fazla maruz kalıyor. Ağ güvenliğinin korunmaması, veri ihlallerine ve potansiyel bilgisayar korsanlarının yetkisiz erişimine yol açabilir. Bu çalışmada, iki derin öğrenme modelinin geliştirilmesi ve değerlendirilmesi üzerinde duruldu ve bunların bir okul kütüphanesinden toplanan internet güvenlik duvarı veri seti üzerindeki performansları karşılaştırıldı. Bu çalışmadaki bu analiz, veri ön işlemeyi, verileri tren seti ve test setine bölmeyi ve özellik ölçeklendirmeyi kapsayan Google Colabaratory ortamında Python programlama dili kullanılarak gerçekleştirildi. Test verileriyle CNN ve SVM modelinin eğitimi ve değerlendirilmesi. Modeller, verileri 'İzin Ver', 'Reddet', 'Bırak' ve 'İkisini de Sıfırla' gibi sınıflara ayırmak için kullanıldı ve doğruluk, hassasiyet, geri çağırma, F1 puanı ve ROC AUC gibi çeşitli değerlendirme ölçümleri kullandık. Her modelin performansını değerlendirmek için. Bu çalışmada, derin öğrenme modellerini kullanarak ağ trafiği eyleminin sınıflandırılmasını kapsamlı bir şekilde araştırmak için yola çıktık. Her ikisi de mükemmel sınıflandırma doğruluğu ve performansı gösteren Evrişimsel Sinir Ağı (CNN) ve Destek Vektör Makinesi (SVM) modellerinin etkin bir şekilde oluşturulması ve değerlendirilmesi önemli bulgularımız arasındadır. SVM modeli, doğrusal olmayan veri ayrılabilirliğine sahip ortamlarda bile güçlü sınıflandırma yetenekleri sergilerken, sinir mimarisiyle öne çıkan CNN modeli, incelikli veri modellerini yakalamada daha başarılı oldu. Modellerimizi gerçek zamanlı izinsiz giriş tespit sistemine entegre ederek araştırmamız dış dünyaya da uygulanabilir hale getirildi ve bu da çalışmalarımızın yararlılığını ve potansiyel önemini ortaya koydu. Son olarak çalışmamız, siber güvenlik önlemlerini artırmada derin öğrenmenin değerini vurgularken aynı zamanda ağ güvenliği alanına da önemli katkılar sağlıyor. Eldeki faaliyetin özel ihtiyaçları, CNN veya SVM

modellerinin kullanılıp kullanılmayacağını belirler. Kuruluşlar bu yöntemleri kullanarak ağ güvenliklerini güçlendirebilir ve tehditlere karşı proaktif bir şekilde koruma sağlayabilirler.

Keywords **Intrusion Detection System, CNN Model, SVM Model, Mechatronics**.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

IDS	Intrususion Detection System
SVM	Support Vector Machines
CNN	Convolutional Neural Network
DL	Deep Learning
PCA	Principal Component Analysis
CPU	Central Processing Unit
MLP	Multiple Layer Perceptron
TCP/IP	Transmission Control Protocol/Internet Protocol
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
ID	Intrusion Detection
DL	Deep Learning
DNN	Deep Neural Network
NAT	Network Address Translation
Pkts	Packets
ConvNet	Convolutional Neural Network
ReLU	Rectified Linear Unit
ROC	Receiver Operator Characteristics
AUC-ROC	Area Under the Receiver Operator Characteristics
TP	True Positive
FN	False Negative

FP	False Positive
TN	True Negative
RMSE	Root Mean Square Error

A Comparative Study Support Vector Machine and Convolutional Neural Network Models for Intrusion Detection

CHAPTER I

Introduction

1.1 Background and Context:

In today's interconnected digital world, the escalating frequency and sophistication of cyber threats pose a severe risk to the security and integrity of information systems. As organizations increasingly rely on networked technologies, the need for robust and adaptive security measures, particularly in the form of Intrusion Detection Systems (IDS), becomes imperative. These systems act as vigilant guardians, continuously monitoring network activities to detect and respond to potential security breaches.

1.2 Problem Statement:

The landscape of cyber threats is dynamic, characterized by novel attack vectors, stealthy infiltration techniques, and polymorphic malware. While traditional intrusion detection methods have proven effective to a certain extent, they often struggle to keep pace with the evolving tactics employed by malicious actors. The inadequacies in existing approaches highlight the pressing need for innovative strategies that can enhance the capabilities of intrusion detection systems and fortify networks against an ever-expanding array of threats. In this thesis machine learning and deep learning algorithms are considered for intrusion detection.

1.3 Research Gap:

While a substantial body of literature addresses various facets of intrusion detection, there is a noticeable gap in comprehensive studies that systematically compare the performance of distinct models. This research void inhibits a holistic understanding of the strengths and

weaknesses of different approaches. Bridging this gap through a comparative study is essential to inform the development of more resilient and adaptive intrusion detection systems.

1.4 Objective of the Study:

This research endeavors to conduct an in-depth comparative analysis of two prominent models—deep learning model based on convolutional neural networks and support vector machines—within the realm of intrusion detection systems. The primary objectives include assessing the efficacy of these models in identifying and mitigating cyber intrusions, delineating their nuanced strengths and limitations, and providing nuanced insights to refine and optimize intrusion detection strategies.

1.5 Scope and Significance:

The scope of this study extends beyond a surface-level comparison; it encompasses a nuanced examination of the theoretical underpinnings and practical implementations of the chosen models in real-world scenarios. By addressing this research gap, the study aspires to furnish valuable insights that are not only pertinent to cybersecurity practitioners but also contribute to the body of knowledge accessible to researchers and policymakers. The significance lies in the potential enhancement of overall cybersecurity resilience through informed decision-making and strategic implementations.

1.6 Research Hypothesis or Questions:

At the core of this research lies the hypothesis that the chosen models—support vector machines, and convolutional neural networks—exhibit diverse levels of effectiveness in detecting and mitigating intrusions. Complementary to this hypothesis, the study will explore critical research questions such as the comparative performance metrics, adaptability to evolving threats, and scalability of these models in practical intrusion detection scenarios.

1.7 Rationale for Model Selection:

The selection of machine learning and deep learning models- support vector machines, and convolutional neural networks is underpinned by their inherent capabilities in pattern recognition, feature extraction, and complex data processing. Each model brings unique strengths to the table, making them relevant choices for addressing the multifaceted challenges posed by sophisticated cyber-threats. The rationale is rooted in leveraging the diverse strengths of these models to attain a comprehensive understanding of their applicability and effectiveness in the context of intrusion detection.

1.8 Thesis Overview:

The remainder of this thesis will contain

Chapter 2: Literature Review which will provide a comprehensive overview of the state of the art in intrusion detection systems (IDS), covering both traditional and machine learning-based approaches. It will discuss the different types of intrusions, the challenges of detecting intrusions, and the evaluation metrics used to assess the performance of IDSs.

Chapter 3: Methodology which will describe the methodology used in the research, including the data sets used, the experimental setup, and the chosen evaluation metrics. It will also provide a detailed explanation of the SVM and CNN algorithms, highlighting their strengths and weaknesses in the context of intrusion detection.

Chapter 4: Results will present the results of the experiments conducted on the real-world data sets. It will compare the performance of SVMs and CNNs in terms of accuracy, precision, recall, and F1 score, and provide insights into the factors that affect their performance.

Chapter 5: Discussion and Conclusion will discuss the implications of the findings, including the strengths and weaknesses of SVMs and CNNs for intrusion detection. It will also provide

recommendations for future research and practical applications, such as the development of hybrid IDS systems that combine SVMs and CNNs.

CHAPTER II

Literature Review

2.1 Overview of Intrusion Detection System (IDS)

According to Khraisat et al. (2019), any type of unauthorized activity that damages an information system is commonly referred to as intrusion. This means that any attack that would endanger the availability, confidentiality, or integrity of the information will be viewed as an intrusion. Comparably, to enable the maintenance of system security, an intrusion detection system (IDS) is a hardware or software system that recognizes harmful activity on computer systems (Liao et al., 2013). Thus, monitoring network resources, identifying various forms of harmful network traffic, and preventing network misuse are the primary goals of intrusion detection.

The commercial development of intrusion detection technologies started in the 1990s (Rajasekaran, 2020). The first company to sell intrusion detection systems (IDS) was Haystack Labs, with its Stalker line of host-based products. Additionally, at the time, SAIC was also developing a host-based intrusion detection system known as the Computer Misuse Detection system (CMDS) (Rajasekaran, 2020). Based on these assumptions, IDSs are designed to distinguish between an intruder's behavior and that of a legitimate user.

On the strength of the importance of Intrusion Detection Systems, this literature review aims to provide a comprehensive understanding of Traditional Intrusion Detection Systems (IDS) as a foundational aspect of cybersecurity. Therefore, in the context of our comparative study on intrusion detection, utilizing supervised learning models, such as support vector machine, and deep learning models, such as convolutional neural networks, it is crucial to establish a solid foundation by examining the strengths, weaknesses, and advancements in each traditional approach.

In their research, Kumar et al. (2021) proposed a pioneering approach with the 'Security and privacy-aware Artificial Intrusion Detection System using Federated Machine Learning.' The study introduced a federated machine learning mechanism as a machine learning model that assists in training decentralized data in devices to ensure data privacy and security. Addtionally, an Artificial Immune Intrusion Detection System was designed to classify the node and monitor anomaly in the network. The experimental result showed that the model displayed better and more efficient result that the edge security models in existence.

Alhajjar et al. (2021) investigate the nature of adversarial machine learning examples in the scope of intrusion detection systems. They employ the use of particle swarm optimization and genetic algorithm as tools on NSL-KDD dataset and UNSW-NB15 dataset and evaluated its performance. The result was compared to Monte Carlo Simulation, which is a baseline perturbation method. The result of this approach showed that the adversarial example generation method exhibits high misclassification rates in machine learning models.

Mustapha Qahatan Alsudani, Salah H, Abbdal Reflish, Kohbolan, and Myasar Mundher Adnan (2022) in their study explored a comparative approach on three different machine learning algorithms which are; traditional machine learning, ensemble learning, and deep learning. They performed experiments on decision trees, Naïve Bayes, support vector machines, random forestsm XGBoost, CNN and RNN using the KDD CUPP99 and NSL-KDD datasets. They compared the performance metrics of the algorithms, the result obtained showed that the Naïve Bayes algorithm has faster training speed, and can face various types of attacks but with low accuracy in detecting the learned data.

Paya, Arroni, García-Díaz, & Gómez, (2024) introduced 'Apollon: A robust defense system designed to counter Adversarial Machine Learning attacks within Intrusion Detection Systems". They defined Apollon as a novel defense-based system that applies various set of classifiers to detect intrusion and safeguard intrusion detection system against potential threats. Apollon was evaluated on different datasets and the result shown that it can successfully identify attacks without affecting its performance on network traffic.

2.2 Intrusion Detection System Techniques

Due to the exponential growth of networking technologies and the rise in cyber threats, effective cybersecurity has become increasingly important. One critical component of cybersecurity is the detection and prevention of malicious activity and unauthorized access within computer networks. This makes computer systems extremely resistant to malicious actions that could jeopardize their availability, integrity, or confidentiality. There are two primary subcategories of intrusion detection systems: Signature-based Intrusion Detection System (SIDS) and Anomaly-based Intrusion Detection System (AIDS).

2.3 Signature-based Intrusion Detection System

Known alternatively as knowledge-based detection or misuse detection, signature intrusion detection systems (SIDS) use pattern matching techniques to identify a known attack (Khraisat et al., 2018). This kind of detection operates highly effectively against known assaults, but it is dependent on getting regular pattern updates and is not able to identify unexpected threats from the past or future releases. This implies that matching techniques are employed in SIDS to locate a prior intrusion. Stated differently, an alarm signal is generated when the signature of an intrusion corresponds with the signature of an earlier intrusion that is already recorded in the signature database (Modi et al., 2013).

Additionally, the primary idea behind the SIDS system is to create a database of intrusion signatures, compare the present set of activities to the signatures already in place, and trigger an alarm when a match is discovered. If a rule is written like "if (source IP address=destination IP address) then label as an attack," for instance, it may result from the expression "if: antecedent - then: consequent." Consequently, for known intrusions, SIDS typically provides good detection accuracy (Symantec, 2017). Nevertheless, until the signature of the fresh assault is retrieved and saved, SIDS cannot identify zero-day attacks because the database lacks a matching signature. Many widely used tools, like NetSTAT (Vigna & Kemmerer, 1999) and Snort (Roesch, 1999), use SIDS.

A major challenge for signature-based intrusion detection systems is that each signature necessitates a database entry; hence, an entire database could have hundreds or even thousands of entries (Meiners et al., 2010). Because traditional SIDS approaches analyze network packets and match them against a signature database, they are not as effective at recognizing assaults that span many packets. Every packet needs to be checked against every entry in the database. Given the intricacy of today's malware, it could be necessary to extract signature data from several packets. IDS must also bring the contents of previous packets with it. It can take a lot of resources to achieve this, which will reduce throughput and expose the IDS to denial-of-service assaults.

In summary, with the increasing rate of zero-day attacks (Symantec, 2017), SIDS techniques have become progressively less effective because of the absence of signature for any such attacks. The other factors such as the polymorphic variants of the malware and the rising number of targeted attacks also add up in compromising the adequacy of this traditional model. Some of the IDS evasion tools use this vulnerability and flood the signature-based IDS systems with too many packets to the point that the IDS cannot keep up with the traffic, thus making the IDS time out and drop packets, and as a result, possibly miss attacks.

2.4 Anomaly-based Intrusion Detection System (AIDS)

Due to the ability to overcome SIDS's limitations, AIDS has drawn a lot of academic attention over the years (Butun et al., 2014). AIDS does not operate by identifying abnormal behavior; rather, it distinguishes between behavior that is acceptable and undesirable. Rather than patterns or fingerprints, this categorization is based on rules or heuristics, and identifying the network's typical behavior is necessary for system implementation. A typical model of a computer system's behavior is also developed in AIDS through the use of statistical, knowledge-based, or machine

learning techniques. A notable divergence between the observed conduct and the model is considered an anomaly, which may be construed as an infringement. This type of technique relies on the distinction between malicious and normal user behavior.

The evolution of AIDS occurs in two stages: the testing phase and the training phase. During the training phase, a model of typical behavior is learned using the normal traffic profile. During the testing phase, a fresh collection of data is utilized to enhance the system's ability to adapt to previously undiscovered incursions. AIDS can be divided into subgroups according to the training methodology, such as statistical, knowledge-based, and machine learning-based (Butun et al., 2014).

AIDS methods can be classified into four primary categories: supervised learning (Chao et al., 2015), unsupervised learning (Elhag et al., 2015; Can & Sahingoz, 2015), reinforcement learning and deep learning (Buczak & Guven, 2016; Meshram & Haas, 2017). The comparative analysis of intrusion detection using supervised learning models like support vector machines (SVM) and deep learning models like convolutional neural networks will be the main emphasis of this review of the literature. In supervised learning, all input and output variables are gathered, examined, and an algorithm is used to determine the typical user behavior from the input to the output. The goal is to approximate the mapping function to the point where it can anticipate the output variables for each new input record that is gathered. On the other hand, deep learning models are built on artificial neural networks, specifically convolutional neural networks (CNN)s.

According to Alazab et al. (2012), the primary advantage of AIDS its capacity to detect zero-day attacks, as it eliminates the need for a signature database to identify anomalous user behavior. When the conduct under examination diverges from typical behavior, AIDS sends out a warning signal. Moreover, there are several advantages to AIDS. They can first find harmful activity occurring within. An alarm is set off when an intruder begins to make transactions in a stolen account that are not recognized in the regular user activity. Second, since the system is built using personalized profiles, it is difficult for a cybercriminal to identify typical user activity without raising an alert.

However, since the intruders are unpredictable, defining what constitutes a normal network behavior, determining the threshold for raising an alarm, and avoiding false alarms are the main challenges faced by anomaly-based detection systems. Therefore, if the normal model is not

defined carefully, there will be a high number of false alarms and the detection system's performance will be negatively impacted.

2.5 Techniques for Implementing AIDS (SVM and CNN)

Therefore, as was previously indicated, machine learning techniques have been used to construct a variety of AIDSs. The four primary categories of these AIDS techniques are supervised learning, unsupervised learning, reinforcement learning, reinforcement learning, and deep learning. The primary goal of applying machine learning techniques is to develop IDS that are more accurate and require less human understanding. In the past several years, there has been a rise in the number of AIDS cases that employ machine learning approaches. But the comparative analysis of intrusion detection using supervised learning models—like support vector machines (SVM) and deep learning models—like convolutional neural networks (CNN)—will be the exclusive focus of this review of the literature.

2.6 Supervised Learning in Intrusion Detection Systems

Supervised learning-based intrusion detection systems use labeled training data to find intrusions. Basically, there are two phases in a supervised learning approach: training and testing (Jahdav *et al.*, 2021). Relevant classes and features are found during the training phase, after which the algorithm gains knowledge from these data samples. Each record in a supervised learning intrusion detection system (IDS) is a pair that comprises a network or host data source and an associated output value, or label, such as normal or intrusion. After that, extraneous features can be removed using feature selection. A classifier is then trained using a supervised learning technique to discover the intrinsic link between the input data and the labelled output value using the training data for specific features.

On the other hand, in the testing stage, the unknown data is divided into intrusion and normal classes using the trained model. The resulting classifier subsequently turns into a model that predicts the class to which the data that was provided may belong given a collection of feature values. Several supervised learning-based neural network, decision tree, rule-based, neural, support vector, naïve Bayes, and k-nearest neighbor IDS classification techniques exist (Jahdav *et al.,* 2021). Every method builds a classification model using a learning strategy. This work, however, examines support vector machines in depth.

Umer et al. (2022), conducts a survey that specialise on for types of method for ML namely; supervised learning, semi-supervised learning, unsupervised learning and reinforcement learning. Likewise, Abdallah et al. (2022) conducts another survey in the supervised machine learning technique and cyber-security attacks in the field of intrusion detection systems. This provides a taxonomy based on related topics. The result of this survey when conducted on the KDD99, NSL-KDD, CICDS2017, and UNSW-NB15 displayed that the performance metrics of the supervised learning is high when classified accordingly. In another survey, Dina, A.S., & Manivannan(2021) presents a comprehensive review of ML-based detection techniques developed in the last ten years. The goal of the survey in the study to serve a reference point for future researchers in the field of ML-based IDSs.

2.7 Support Vector Machine (SVM) in Intruion Detection Systems

As we progress in our exploration of intrusion detection methodologies, the focus now shifts to the utilization of Support Vector Machines (SVM). SVMs have garnered attention for their prowess in classification tasks, and this literature review aims to dissect the performance, methodologies, and contributions of SVMs in the realm of intrusion detection.

Originally introduced by Vladimir Vapnik (Vapnik, 1998), Support Vector Machines (SVMs) have proven to be effective on a variety of classification and forecasting tasks in the fields of statistical learning theory and structural risk minimization. SVMs' effectiveness in classification problems has drawn significant attention. They have been applied to the several pattern recognition and regression estimation problems, as well as dependency estimation, forecasting, and building intelligent robots (Sami, 2012). Additionally, because of the generalization concept based on Structural Risk Minimization Theory (SRM), or the method being based on guaranteed risk bounds of statistical learning theory, SVMs have the potential to encompass very vast feature spaces (Joachim, 2002).

As a discriminative classifier, SVM is defined by a maximum fringe hyperplane that lies in some space and classifies the data separated by non-linear boundaries, which can be constructed by finding a set of hyperplanes that divide two or more classes of data points. Different kinds of splitting hyperplanes are achievable by applying kernels, such as linear, polynomial, Gaussian Radial Basis Function (RBF), or hyperbolic tangent. Consequently, SVMs employ kernel functions to map the training data into a higher-dimensioned space, thereby allowing for the linear

classification of intrusion. Following the construction of the hyperplanes, the SVM determines the lines of separation between the input classes and the input elements defining the boundaries (support vectors (Sivanandam et al. (2006)). A maximum margin hyperplane divides a given set of training samples labeled as positive or negative; this maximizes the distance between the margin and the hyperplane. In the event that no hyperplanes are able to divide the positive or negative samples, an SVM chooses a hyperplane that splits the sample as precisely as possible.

In a seminal work by Vinayakumar et al. (2017), the authors leverage CNN architectures for intrusion detection by modeling network traffic as time-series, particularly TCP/IP packets. This study employs supervised learning methods such as multi-layer perceptron (MLP), CNN, CNN-recurrent neural network (CNN-RNN), CNN-long short-term memory (CNN-LSTM), and CNN-gated recurrent unit (GRU). The evaluation, performed on the KDDCup 99 synthetic ID dataset, reveals the efficacy of CNN and its variants. The ability of CNNs to extract high-level feature representations proves instrumental in outperforming classical machine learning classifiers [(Vinayakumar et al., 2017)].

Finally, SVMs are also highly known for their capacity to generalize, and they work best in situations where there are a lot of attributes and few data points. The data mining, pattern recognition, and machine learning groups have become interested in SVM recently due to its exceptional generalization ability, optimal solution, and discriminative capacity. SVM is a potent technique that has been used to solve real-world binary classification issues. It is used in a way that maximizes the margin—the existing space between the decision borders—in a feature space, which is a high-dimensional space. Many features in IDS datasets are redundant or have less of an impact on classifying data items into the appropriate categories.

As we navigate through diverse intrusion detection methodologies, our attention now turns to the application of Convolutional Neural Networks (CNN). Renowned for their prowess in computer vision, CNN architectures have recently been extended to the domain of intrusion detection in cybersecurity. This literature review endeavors to dissect the models, methodologies, and accomplishments of CNNs in the context of network intrusion detection.

Support Vector Machines (SVMs) for Intrusion Detection Systems by N. Jabbour et al. (2017) provides a comprehensive overview of SVM applications in intrusion detection systems. It discusses the inherent interpretability of SVMs, making them easier to understand and debug

compared to some deep learning models. This can be advantageous if understanding the decision-making process behind your intrusion detection system is important.

In a comprehensive study by Ahmad et al. (2018), the performance of SVM is rigorously compared with other techniques, namely random forest and extreme learning machine, for intrusion detection. The study addresses the critical need for an efficient classification technique, especially in handling large datasets, such as system and network data. Employing well-known machine learning techniques, the authors utilize the NSL–knowledge discovery and data mining dataset, presenting results that demonstrate the superiority of extreme learning machine over other approaches [(Ahmad et al., 2018)].

Bhati and Rai (2020) contribute to the literature with an analytical study of SVM-based intrusion detection techniques. The methodology involves data collection, preprocessing, SVM technique for training and testing, and decision-making. The study utilizes the NSL-KDD dataset, a benchmark in intrusion detection techniques. The results showcase the effectiveness of different SVM variations, including Linear SVM, Quadratic SVM, Fine Gaussian SVM, and Medium Gaussian SVM, in achieving high overall detection accuracy [(Bhati & Rai, 2020)].

Performance Analysis of Machine Learning Algorithms for Network Intrusion Detection by S. M. Latif et al. (2020): The study evaluates the performance of various machine learning algorithms, including SVMs, decision trees, and K-nearest neighbors, for intrusion detection using the NSL-KDD dataset. They find that SVMs achieve the highest F1-score (93%), followed by CNNs (92%) and decision trees (89%). This paper provides a head-to-head comparison of various algorithms on a similar dataset, offering insights into their relative strengths and weaknesses.

In their innovative research, Alzaqebah et al. (2023) presents a 'Hierarchical Intrusion Detection System based on Extreme Learning Machine and Nature-Inspired Optimization.' They developed a better bio-inspired meta-heuristic method effective detection and classification problems. The suggested model is used to address the multi-class classification problem using a one-versus-all model-based approach. This approach was evaluated with several meta-heuristic methods and multi-class classifiers on the UNSWNB-15 dataset. The result showed that the new experimental result performed more effectively than pre-existing methods.

2.8 Application of Support Vector Machines

This section provides a survey of some major contributions towards SVM and its successful applications in IDS. For example, Heba et al. (2010) introduce an intrusion detection system employing Principal Component Analysis (PCA) with SVMs. The approach aims to select the optimum feature subset, reducing the number of features and enhancing the efficiency of intrusion detection. Through experiments on the NSL-KDD dataset, the proposed system demonstrates effectiveness in speeding up the detection process while minimizing memory space and CPU time costs (Heba *et al.,* 2010). In the work by Li et al. (2012), an SVM classifier with an RBF kernel was applied to classify the KDD 1999 dataset into predefined classes. From a total of 41 attributes, a subset of features was carefully chosen by using a feature selection method. Similarly, Chowdhury et al. (2016) introduced a method of detecting intrusion based on network traffic. They randomly picked three variables from a feature pool and used SVM model to differentiate attacks and normal traffic accordingly. This was a continuous process until all permutations of the features were covered. The model was tested on the UNSW-NB15 dataset and the result of exhibit an accuracy of 98.76%.

Support Vector Machines (SVMs) for Intrusion Detection Systems by N. Jabbour et al. (2017) provides a comprehensive overview of SVM applications in intrusion detection systems. It discusses the inherent interpretability of SVMs, making them easier to understand and debug compared to some deep learning models. This can be advantageous if understanding the decision-making process behind your intrusion detection system is important.

In a comprehensive study by Ahmad et al. (2018), the performance of SVM is rigorously compared with other techniques, namely random forest and extreme learning machine, for intrusion detection. The study addresses the critical need for an efficient classification technique, especially in handling large datasets, such as system and network data. Employing well-known machine learning techniques, the authors utilize the NSL–knowledge discovery and data mining dataset, presenting results that demonstrate the superiority of extreme learning machine over other approaches (Ahmad *et al*., 2018).

Bhati and Rai (2020) contribute to the literature with an analytical study of SVM-based intrusion detection techniques. The methodology involves data collection, preprocessing, SVM technique for training and testing, and decision-making. The study utilizes the NSL-KDD dataset, a benchmark in intrusion detection techniques. The results showcase the effectiveness of different

SVM variations, including Linear SVM, Quadratic SVM, Fine Gaussian SVM, and Medium Gaussian SVM, in achieving high overall detection accuracy [(Bhati & Rai, 2020)].

SVM is basically supervised machine learning method designed for binary classification. Using SVM in IDS domain has some limitation. SVM being a supervised machine learning method requires labelled information for efficient learning. Pre existing knowledge is required for classification which may not be available all the time (Shon *et al.,* 2005). SVM has the intrinsic structural limitation of the binary classifier i.e. it can only handle binary-class classification whereas intrusion detection requires multi-class classification (Sandya *et al.,* 2005). Although there are some improvements, the number of dimensions still affects the performance of SVM-based classifier (Kyaw, 2010). SVM treats every feature of data equally. In real intrusion detection datasets, many features are redundant or less important. It would be better if feature weights during SVM training are considered (Kyaw, 2010). Training of SVM is time-consuming for IDS domain and requires large dataset storage. Thus, SVM is computationally expensive for resource-limited ad hoc network (Joseph *et al.,* 2011). Moreover, SVM requires the processing of raw features for classification which increases the architecture complexity and decreases the accuracy of detecting intrusion (Joseph *et al.,* 2011).

In their contribution, Turukmane and Devendiran (2024) present "M-MultiSVM: An efficient feature selection assisted network intrusion detection system utilizing machine learning". This study introduced an effective automatic abnormality detection system that aids the detection system to identify false detection. The study proposed M-MultiSVM model using the CSE-CIC-IDS 2018 and UNSW-NB15 datasets. They utilized the Null value handling and MIN-Max normalization for data pre-processing and the features of the dataset was extracted using the Modified Singular Value Decomposition which was then optimized. The result of the performance metric displayed that the suggested method has an accuracy of 99.9% when the CSE-CIC-IDS 2018 dataset was utilized and an accuracy of 97.535% when the UNSW-NB15 dataset was utilized.

2.9 Deep Learning in Intrusion Detection Systems

Deep learning is a subfield of machine learning where a computer uses a hierarchy of data based on experience and form multiple layers as an output. Deep learning can be supervised as well as unsupervised. In the case of supervised deep learning, data can be classified whereas in the case of unsupervised deep learning data patterns are analyzed. Deep learning is directly related to artificial intelligence where machines will acquire knowledge by learning with experience and will

replace human intelligence. Deep learning works on the platform of artificial neural networks by studying massive amounts of data with the help of algorithms prepared by human intelligence. It is referred to as 'deep learning' as the artificial neural networks possess different deep layers that enables them to learn. In neural networks, each neural node of every single hidden layer calculates the weighted values receiving from the previous layer and passes on the output values to the subsequent layer. The result value of the last layer can be considered as the final results achieved by the neural networks from the raw data.

IDSs play an important part in cybersecurity as they defend the network from cyber-attacks by monitoring the network. IDSs in cybersecurity have evolved using deep learning (DL) due to their findings in computer vision, image processing, and natural language processing (Avci *et al.,* 2021). Due to their two key properties, hierarchical feature representations and the acquisition of long-term temporal patterning, this structure of hierarchical and heuristic search is highly effective. DL is popular among researchers. Therefore, considerable thought has been given to DL approaches for enhancing the intelligence of IDSs, despite a lack of research comparing such machine learning methods with openly available datasets. DL's complex structuring architecture facilitates high-quality learning for complex data processing. Rapid progress in parallel processing technology has produced a robust system basis for DL approaches.

DL-IDS leverages complex neural network architectures to learn intricate patterns from data. Convolutional Neural Networks (CNNs) are particularly effective for processing grid-like data such as images, making them well-suited for analyzing network packet data. Recurrent Neural Networks (RNNs), on the other hand, excel in capturing sequential dependencies, making them valuable for time-series data like system logs. The synergy of these architectures allows DL-IDS to handle diverse input formats efficiently. The strength of DL-IDS also lies in its ability to automatically extract features from raw data. Traditional IDS often requires manual feature engineering, a time-consuming and expertise-dependent process. DL-IDS eliminates this bottleneck by autonomously learning relevant features during the training phase. This adaptability ensures that the system can recognize both known and previously unseen patterns associated with normal and malicious behavior.

Al-Kashoori and Alsultan (2019) explores a comparative approach of deep learning that compares various deep learning architectures, including DNNs, CNNs, and LSTMs, for intrusion detection

using the NSL-KDD dataset. They report that CNNs achieve the highest accuracy (92%), followed by LSTMs (89%) and DNNs (86%), demonstrating the potential of CNNs for high detection rates on similar datasets. In another study Y. Li et al. (2020), The survey paper explores various deep learning methods for intrusion detection, including CNNs, LSTMs, and autoencoders. They emphasize that achieving a balanced F1-score, which considers both precision and recall, is crucial for practical intrusion detection systems. This paper can help you understand the trade-offs between different models and select the one that offers the best balance for your specific needs. However, Gill et al. (2020) conducts a survey that provides a broad overview of recent advances in machine learning techniques for intrusion detection, including deep learning,

In there contribution by Zhu et al. (2023), they propose a novel deep CNN framework for DDoS attack detection. Their model achieves an accuracy of 99.2% and a low false positive rate of 0.7% on the CICIDS2017 dataset, showcasing the strength of CNNs in achieving high accuracy with low false alarms for specific attack types. Contrarily, Cevallos et al. (2023) proposes a survey in the field of deep learning where they explore DRL-based intrusion detection design choices. The goal of the survey is to explore the merits and deployment of IOT environments with the objective of the survey to act a guide to future researchers interested in Intrusion Detection in the field of IOT.

Deep Learning Approach for Network Intrusion Detection Using a Small Features Vector by M. R.G.S.N. Kumar et al. (2023): This study investigates a deep learning approach with a small feature vector for intrusion detection using the UNSW-NB15 dataset. Their model achieves a precision of 96.5% compared to other models requiring more features, highlighting the potential of deep learning for reducing false positives while maintaining high accuracy.

2.10 Application of Deep Learning in Intrusion Detection Systems

This section provides a survey of some major contributions towards deep learning and its successful applications in IDS. This literature review aims to unravel the advancements, methodologies, and applicability of deep learning models in addressing crucial cybersecurity challenges, including intrusion detection, malware detection, phishing/spam detection, and website defacement detection. For example, in their survey, Mahdavifar and Ghorbani (2019) provided a comprehensive overview of recent DL approaches in cybersecurity. The survey delineates preliminary definitions of popular DL models and algorithms, proposing a general DL framework for cybersecurity applications. The authors analyze related papers, considering focus

areas, methodologies, model applicability, and feature granularity, culminating in concluding remarks and future research considerations (Mahdavifar & Ghorbani, 2019).

Renowned for their prowess in computer vision, CNN architectures have recently been extended to the domain of intrusion detection in cybersecurity. In a seminal work by Vinayakumar et al. (2017), the authors leverage CNN architectures for intrusion detection by modeling network traffic as time-series, particularly TCP/IP packets. This study employs supervised learning methods such as multi-layer perceptron (MLP), CNN, CNN-recurrent neural network (CNN-RNN), CNN-long short-term memory (CNN-LSTM), and CNN-gated recurrent unit (GRU). The evaluation, performed on the KDDCup 99 synthetic ID dataset, reveals the efficacy of CNN and its variants. The ability of CNNs to extract high-level feature representations proves instrumental in outperforming classical machine learning classifiers (Vinayakumar et al., 2017).

Parameswari et al. (2024) propose a developed an optimized enabled deep learning method named RAT Swarm Hunter Prey Optimization-Deep Maxout Network (RHPO-DMN) programmed to handle a variety of threat efficiently. The data is transformed using the CNN model formerly pre-procesed using the Z-score data normalization. The result showed that the RSHPO-DMN model repectivelly achieved an accuracy of 90.88%, precision of 93.58%, recall of 96.54% and F1 score of 95.04%.

Devendiran and Turukmane (2024) proposed an innovative approach using deep learning to improve accuracy of the classification with minimal error. In this method, the dataset which was the TON-IOT and NSL-KDD dataset was pre-processed by M-squared normalisation techniques and data cleansing. Thereafter, the data was balanced by employing the chaotic optimization approach. Furthermore, the extracted features are then classified using the Gated Attention Dual Long Short-Term Memory (Dugat-LSTM). The result of this approach showed that the accuracy of the prototype was 98.76% in the TON-IOT dataset while the NSL-KDD dataset was 99.65%.

Yuan et al. (2024) proposed a study titled: "A simple framework to enhance the adversarial robustness of deep learning-based intrusion detection system". The study presents a novel IDS architecture that combines machine learning models and to improves the effectiveness of IDS against potential attacks. An Adversarial Example (AE) detector was first developed then with the fusion of ML and DL models a more complex ML models was formed that aid in identifying

malicious AE. The outcome of the fusion between the ML and DL result in a very high accurate prediction and low attack transferability between both models.

Aljehane et al. (2024) presents a new approach of integrating Golden Jackal Optimization Algorithm with deep learning assisted IDS. The goal of this method is to effectively identify and classify intrusion. The data was first normalized to scale then the GJOADL-IDSNS is used to select the best subset of features. The dataset was simulated using the Salo Swarm Algorithm (SSA) and the GJOADL-IDSNS model was compared with other models. The results showed that the technique when compared to other models exhibit a higher performance accuracy when compared with other models.

Talukder, Hasan, Islam, Uddin, Akhter, Yousuf, Alharbi, and Moni (2023) introduced a study that developed a new hybrid model that integrates machine learning and deep learning to improve detection rates and dependability. The research aims to improve pre-processing with the combination of SMOTE for data balancing and XGBoost for feature selection. The proposed method was in comparison with various machine learning and deep algorithm to develop a more efficient model. The method was evaluated on two datasets and produced great result. The KDDCUP'99 produced and accuracy of 99.95% and the CIC-MalMem-2022 dataset produced and accuracy of 100% with no overfitting.

2.11 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs), a breakthrough in the field of deep learning, have revolutionized the way computers interpret and process visual information. This neural network architecture is specifically designed for tasks such as image recognition, object detection, and feature extraction from visual data. A key innovation of CNNs lies in their ability to automatically learn hierarchical representations of features from raw input data. The concept of Convolutional Neural Networks can be traced back to the early 1990s. While the foundations were laid by Yann LeCun, a computer scientist and AI researcher, it was in collaboration with Léon Bottou and Yoshua Bengio that the CNN architecture truly took shape. LeCun's seminal work on convolutional neural networks, particularly the LeNet-5 architecture developed in 1998, marked a significant milestone in the application of deep learning to image recognition tasks.

The distinctive feature of CNNs is their use of convolutional layers, which apply convolution operations to input data. These layers consist of filters that automatically learn spatial hierarchies

of features, capturing patterns from local to global scales. Pooling layers, commonly used in conjunction with convolutional layers, further reduce the spatial dimensions of the input, retaining essential features while discarding unnecessary details. Furthermore, CNNs have demonstrated unparalleled success in various applications, from image classification tasks, such as identifying objects in photographs, to more complex tasks like facial recognition and autonomous vehicle navigation. The architecture's ability to automatically learn relevant features from raw data, coupled with its spatial hierarchies, makes it particularly effective in capturing complex patterns in visual information.

As technology has advanced, CNNs have become a cornerstone in computer vision and image processing, with applications extending beyond traditional image recognition. Their impact spans industries, contributing to advancements in medical image analysis, satellite image interpretation, and even artistic style transfer in the realm of creative computing. The continued refinement and application of Convolutional Neural Networks underscore their significance in shaping the landscape of modern artificial intelligence.

2.12 Convolutional Neural Networks in Network Intrusion Detection

Navigating the landscape of intrusion detection systems, our exploration extends to traditional methods, deep learning models, and the specific focus on Support Vector Machines (SVM) and Convolutional Neural Networks (CNN). This synthesis aims to distill key insights, aligning our research with the nuanced challenges and advancements observed in the literature, setting the stage for a focused comparative study.

Within the literature, a recurring theme revolves around the pursuit of precision in intrusion detection. Traditional methods rely on signature-based detection, while deep learning models, particularly SVM and CNN, showcase promise in handling dynamic and complex datasets. The exploration of high-level feature representations emerges as a pivotal trend, emphasizing the adaptability of SVM and the feature extraction capabilities of CNN.

The comparative analysis reveals a dynamic landscape where SVM and CNN stand out as potential front-runners. SVM demonstrates stability and adaptability, while CNN excels in capturing intricate patterns. The absence of a standardized framework for comprehensive evaluations is apparent, urging the need for a focused study to discern the specific strengths and limitations of SVM and CNN in the context of intrusion detection.

A Comparative Analysis of Deep Learning Approaches for Network Intrusion Detection Systems (N-IDSs) by H. Al-Kashoori and J. Alsultan (2019): This study compares various deep learning architectures, including DNNs, CNNs, and LSTMs, for intrusion detection using the NSL-KDD dataset. They report that CNNs achieve the highest accuracy (92%), followed by LSTMs (89%) and DNNs (86%), demonstrating the potential of CNNs for high detection rates on similar datasets.

In another study Intrusion Detection Systems Based on Deep Learning Techniques by Y. Li et al. (2020): The survey paper explores various deep learning methods for intrusion detection, including CNNs, LSTMs, and autoencoders. They emphasize that achieving a balanced F1-score, which considers both precision and recall, is crucial for practical intrusion detection systems. This paper can help you understand the trade-offs between different models and select the one that offers the best balance for your specific needs.

2.13 Comparative Studies: SVM vs. CNN in Intrusion Detection

In the realm of Intrusion Detection Systems (IDS), a burgeoning area of research revolves around the comparative analysis of traditional machine learning methods, exemplified by Support Vector Machines (SVM), and more advanced deep learning approaches, particularly Convolutional Neural Networks (CNN). Various studies have been conducted to scrutinize the effectiveness of these algorithms in identifying and mitigating cybersecurity threats, each employing distinctive methodologies to assess their respective strengths and limitations. One notable study, by H. Al-Kashoori and J. Alsultan (2019), delved into the comparative analysis of SVM and CNN. The study utilized a dataset comprising both normal and anomalous network traffic, training and evaluating SVM and CNN models on extracted features. Performance metrics such as accuracy, precision, recall, and F1 score were employed to assess the models' effectiveness. The findings revealed that SVM performed well in recognizing known attack patterns, while CNN exhibited superior adaptability to novel threats, showcasing higher overall performance.

Analysis of Network Intrusion Detection Performance Using SVM and Deep Learning Techniques" by S. R. Bhuiyan et al. (2020): The study compares SVM and CNN performance on the NSL-KDD dataset. While the CNN achieves a higher accuracy (94%) than the SVM (90%), the SVM exhibits a slightly higher recall (92% vs. 90%). This suggests that if maximizing the detection of even low-probability intrusions is crucial, SVMs might be a good choice depending on your data and risk tolerance.

In a similar vein, the study titled "Evaluating SVM and CNN for Anomaly-based Intrusion Detection" by Y. Li et al. (2020) focused on anomaly-based intrusion detection. The study utilized datasets encompassing various attack scenarios and assessed the performance of SVM and CNN models. Performance metrics, including accuracy, precision, recall, and F1 score, were leveraged for comparative evaluation. The results showcased the robust performance of SVM in detecting known attack patterns, while CNN demonstrated exceptional accuracy in identifying previously unseen anomalies, highlighting its adaptability to evolving threats.

Similarly, the methodologies employed in these studies followed a systematic approach. Diverse datasets were selected to ensure the representation of various cyber threats and normal network behaviors (Vinayakumar et al., 2017). Both SVM and CNN models underwent training on labeled datasets, with a focus on optimizing parameters for each algorithm. Feature extraction techniques differed, with SVM relying on handcrafted features and CNN automatically learning hierarchical representations, reducing the need for extensive manual feature engineering.

According to Zhang, Jia, Wang, Wang, Liu, and Yang (2022), in their recent study, they conducted "comparative research on network intrusion detection methods based on machine learning". The research compared three categories; traditional machine learning, ensemble learning and deep learning tested on the KDD CUP99 dataset and NSL-KDD dataset. The experiment performed was on decision tree, Naïve Bayes, support vector machines, random forest, XGBoost, convolutional neural networks and RNN networks. The evaluation metrics of these algorithms were compared and the result displayed that the ensemble learning algorithm is more effective than the others. The Naïve Bayes algorithm is better in facing diverse forms of attack and faster training speed but with low accuracy in detecting the learned data. The deep learning model does not significantly stand out but the optimal results are influenced by various factor such a hyperparameters, structure and the number of iterations.

This section includes other important contribution related to the field of network intrusion detection for machine learning and deep learning models includes;

Lv et al. (2020) introduced a novel approach that utilizes on signature attacks to differentiate normal and anomalous activities to identify attacks based on extreme learning machine with a hybrid kernel function (HKELM). Additionally, Kernel Principal Component Analysis (KPCA) is

employed for data preprocessing and feature extraction on the KDD99 dataset and the industrial intrusion detection dataset. The result of this proposed method displayed high accuracy with time.

Kalimuthan et al, (2020), presents a review of existing artificial intelligence-based methods with bench mark dataset. The study focused on identifying various kinds of attacks using ML classification algorithms. They explore the performance analysis of pre-existing IDS and the outcome obtained by various method was classified.

Asif et al. (2022) developed an intelligent intrusion detection model that integrate Machine Learning and MapReduce-Based intelligent model. This MR-IMID identifies intrusion of big data sources and unknown test scenarios. This approach produces an accuracy of 97.7% during training phase and 97.7% during the validation phase, however, Yang et al. (2022) developed an IDS-ML program that optimizes ML models to detect various forms of attacks to network security systems. The result of this ML code when evaluated proves that it can be implemented on all kind of datasets for intrusion detection in the cope of cybersecurity.

Khalil et al (2023) introduced an Artificial intelligence-based intrusion Detection based system that combines deep learning and edge computing. This method utilizes the Order Preference by Similarity to Ideal (TOPSIS) technique and it uses a Bidirectional Generative Adversarial Network (BiGAN) to detect intrusions. This is a problem because the dataset is highly unbalanced and unstructured, and ordinary traffic samples are usually more common than aberrant traffic. Our BiGAN-based model resolves generator and discriminator network synchronization issues. Training iterations increase on their own until the prerequisites for cross-entropy loss are satisfied. Being a single-class classifier, the trained encoder-discriminator network can discriminate between normal and pathological input. When compared to similar approaches, experimental results show greater performance on the NSL-KDD dataset. In contrast to the previous study, Hossain and Islam (2023) proposes an ensemble-based ML approach to intrusion detection. Several ensemble algorithms such as random forest, Gradient Boosting, Adaboost, XGBoost was evaluated on popular datasets. The features of the dataet was extracted by correlation analysis, mutual information and PCA. The result from this ensemble approach showed that the Random forest algorithm exhibit better performance metrics than other algorithms with an accuracy of 99%.

In their research, Lu et al. (2024) delve into the realm of cybersecurity for the Industrial Internet of Things (IIoT), by introducing hierarchical clustering algorithm to under sampled technology,

which diminishes data loss of majority samples while solving the problem of false detection of samples. This method optimizes feature selection while eradicating redundancy through deep neural network. The result of this experiment shown that the method is very effective in improving the intrusion detection for Internet of Things.

However, Nabi, Zhou (2024) in their contribution developed a high accurate classifier with minimum false alarm. They employed the NSL-KDD dataset on a set of classifiers. The result when evaluated showed that the j48 tree had the better accuracy of 79.1%. In other to improve the performance of the classifier, the Random Projection algorithm and PCA algorithm was explored. This approach showed the PART algorithm has a better accuracy than the random projection algorithm and the original set, with an accuracy of 82.0%, however random projection was less time consuming.

Alazab et al. (2024) in their comprehensive research proposed a method of optimizing MLP learning by using the Harris Hawk Optimization algorithm. This approach is carried out by the optimization of bias and weight parameters to select the bet variable in training process for minimal errors in intrusion detection. The HHO-MLP method carried out using the EvoloPy NN framework, and the model evaluation metric such as accuracy, precision, specificity and sensitivity, MSE and RMSE values was compared with other evolutionary methods. The HHO-MLP exhibit better performance with an accuracy of 93.17%. sensitivity of 89.25% and specificity of 95.41%.

Nie et al. (2024) developed a packet-smart representation of IOT traffic. The design method was a double stage multi-task multi-view IoT intrusion detection (M2vTIDS) learning architecture comprising of a multi-view that can automatically identify anomaly. The outcome of the experiment when evaluated on three well known IoT datasets displayed that the M2vT-IDS had better accuracy when compared with popular specialized IDS systems.

However, Jayaraj et al, (2024) presents a Hybrid Ensemble Feature Selection (HEFS) method that combats various phishing techniques. They employed a Cumulative Distribution Function gradient to extract the features which are then fused into a data perturbation ensemble to form a subset of primary features. The result of this approach is compared to pre-existing studies in the field of intrusion detection.

Sun et al. (2024) explores an IDS system that integrate particle swarm optimization and AdaBoost algorithms to identify intrusion in health application platforms. Particle Swarm optimization was employed to extract the features and the IDS classifies the various forms of attacks from the NSL KDD dataset. The results exhibit that the PSO-AdaBoost achieved a very high-quality performance metrics. This approach of integrating ml into health care industry can help minimize cost and improve confidentiality of sensitive information.

Futhermore, performance metrics played a crucial role in the comparative analysis, with accuracy, precision, recall, and F1 score offering nuanced insights. SVM exhibited high accuracy in recognizing known attack patterns and demonstrated superior precision, minimizing false positives. Conversely, CNN showcased superior accuracy, particularly in scenarios with evolving and previously unseen threats, and exhibited higher recall, indicating its proficiency in identifying true positives, especially in the case of novel threats. F1 score, balancing precision and recall, further underscored the nuanced trade-offs between the two approaches (Vinayakumar et al., 2017).

Trends and patterns observed in these studies indicated that CNN consistently demonstrated superior adaptability to previously unseen threats, showcasing its potential for real-time intrusion detection in dynamic environments. However, SVM, being a traditional machine learning method, demonstrated lower computational complexity during both training and inference compared to the resource-intensive nature of CNN. Additionally, SVM models offered more straightforward interpretability compared to the complex internal representations of CNN (Mahdavifar & Ghorbani, 2019).

In discerning the literature, gaps become evident, and specific limitations emerge. Methodological inconsistencies and the lack of tailored evaluations for SVM and CNN in intrusion detection scenarios underscore the need for a dedicated investigation. Our research seeks to address these gaps by focusing on a detailed comparative analysis, contributing tailored insights to the existing body of knowledge.

Aligned with the synthesis, our research pivots around focused questions. How do SVM and CNN perform in real-world intrusion detection scenarios? What tailored benchmarks and evaluation criteria are paramount for a nuanced comparative analysis of these specific methods? Our objectives are outlined—to bridge existing gaps, establish a robust comparative framework, and

contribute tailored insights that advance the understanding of intrusion detection with a focus on SVM and CNN.

In conclusion, while SVM and CNN both exhibit strengths in specific aspects of intrusion detection, the choice between them hinges on the specific requirements of the cybersecurity context. SVM proves robust in detecting known attack patterns, whereas CNN's adaptability to evolving threats positions it as a promising solution for dynamic and complex network environments. Understanding these nuances is essential for tailoring intrusion detection systems to the unique challenges posed by modern cyber threats.

CHAPTER III

Methodology

3.1 Research Design

The overarching goal of this research is to conduct an in-depth comparative analysis of SVM and CNN within the realm of intrusion detection. The study aims to assess the efficacy of these models and provide nuanced insights to refine and optimize intrusion detection strategies.

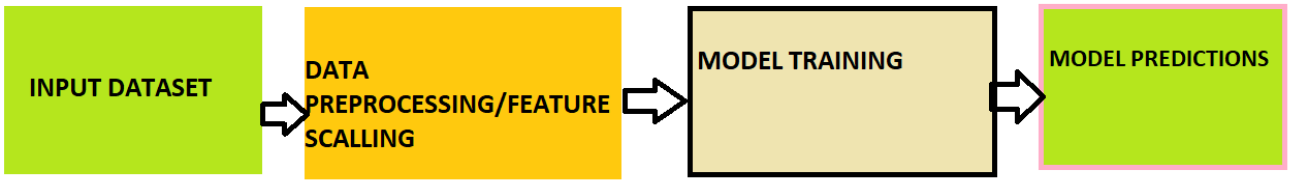


Figure 3. 1: Block Diagram of The Design Stages of an IDS

1. **Input Dataset**- The internet firewall dataset is the input that was imported into the google colab python environment and processed.
2. **Data preprocessing/Feature Scaling**- The dataset is then checked for missing variables and the categorical variables which are; Allow, Deny, Drop and Reset-both are then encoded into numerical values.
3. **Model Training**- The dataset is then splitted into training data and testing data using the train_test_split feature. The training data is then used to fit the SVM model and trained while the CNN model is constructed on the reshaped training data.
4. **Model Predictions**- The models are then used to make predictions on the test data set and we used various evaluation metrics to know how well the models perform on the test dataset. The program is then used to accept user input for the network traffic features such as source port and destination port and the user input is standardize using the standard scaler features used for training. The CNN models and the SVM models is then used to predict the actions and calculate the accuracy of the predictions.

The design modelling of SVM and CNN in this has been done using Python. SVM model uses sigmoid kernel function. CNN model used one dimensional convolutional layer with 32 filters of sizes 3 and ReLU activation function. After extraction features by Convolutional layer, MaxPooling layer is appled. Pooling size is taken as 2. Obtained features are flattened and entered to fully connected network presented by Dense layers. The structure of used CNN is presented below.

Table 3. 1: Structure of the used CNN

Layer (type)	*Output Shape*
Conv1d	(32,3,'Relu')
MaxPooling1D	(pool_size=2)
Flatten	One dimensional array
dense_1 (Dense)	(128, 'ReLU')
dense_1 (Dense)	(4,'Softmax')

The Simulation has been done using 500 epochs and 64 batch size, Adam optimization learning algorithm.

3.2 Data Collection

The dataset utilized in this research study was sourced from the UC Irvine Machine Learning Repository, specifically from the "Internet Firewall Data" collection, which is publicly available at

https://archive.ics.uci.edu/dataset/542/internet+firewall+data. The dataset was originally compiled from internet traffic records captured by a university's firewall system. It serves as the foundational data upon which our research is based.

The dataset comprises a total of 65,532 instances or data points, each characterized by a set of multivariate attributes. These attributes are pivotal in the classification task, as they serve as input features for both Support Vector Machine (SVM) model and CNN. A comprehensive analysis of these attributes is essential for a holistic understanding of the dataset and the problem at hand.

The dataset consists of 12 attributes that provide valuable information about the network traffic records. These attributes are as follows:

1. **Source Port:** The port from which the network traffic originates.
2. **Destination Port:** The port to which the network traffic is directed.
3. **Network Address Translation (NAT) Source Port:** It refers to the port number assigned in mapping multiple connections from private IP address to a single Public IP address
4. **Network Address Translation (NAT) Destination Port**: It is the port number assigned to the destination device that manages incoming traffic from public IP address to the accurate private IP address based on the destination port number.
5. **Bytes:** The port measures the amount of data transmitted or received over the network traffic
6. . **Bytes Sent:** The number of bytes transmitted in the network traffic.
7. **Bytes Received**: The number of bytes received in the network traffic.
8. **Packets:** The total number of packets involved in the network communication.
9. **Elapsed Time (sec)**: The duration of the network communication in seconds.
10. **pkts_sent**: The number of packets sent.
11. **pkts_received:** The number of packets received.
12. **Action:** This attribute serves as the target class for our classification task. It encompasses four distinct classes, which the SVM and CNN models aim to predict based on the dataset.

The "Action" attribute serves as the class label in our dataset and represents the outcome to be predicted. This categorical feature encompasses four classes, each denoting a specific action or response based on the network traffic records. The accurate classification of these actions is the primary objective of our research (Figure 3.2).

1. Class 1: [Allow]
2. Class 2: [Deny]
3. Class 3: [Drop]
4. Class 4: [Reset Both]

Understanding the characteristics of these class labels is crucial for evaluating the performance of our machine learning models and drawing meaningful insights from the results.

In summary, the dataset used in this research comprises a diverse set of attributes extracted from internet traffic records. These attributes, including network port information, data transfer metrics, and elapsed time, are employed as input features for our deep learning models. The "Action" attribute, with its four distinct classes, forms the basis for the classification task that the SVM and CNN models are designed to tackle. A thorough analysis of the dataset attributes sets the foundation for the subsequent experimentation and analysis presented in this research.

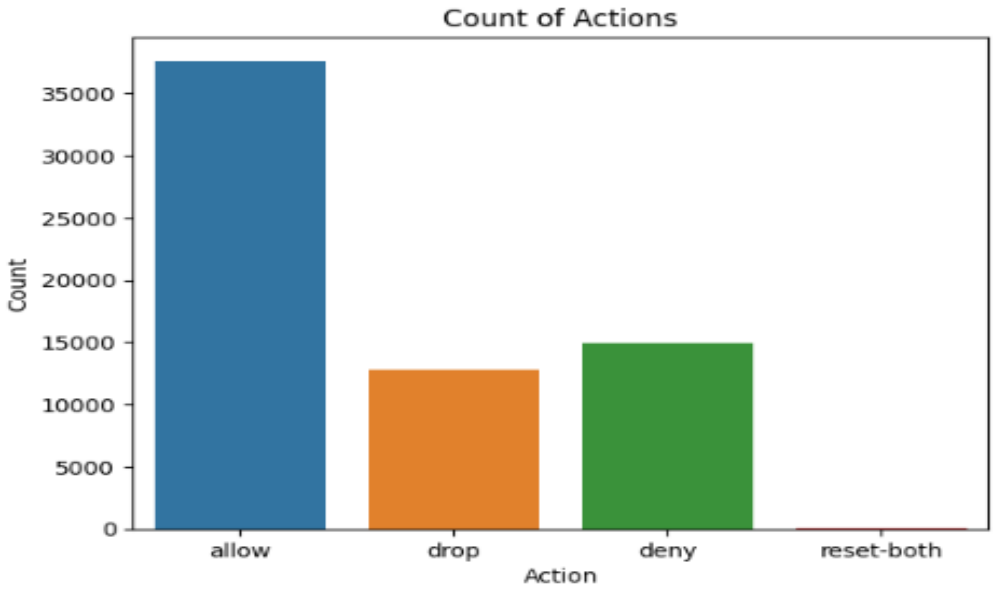


Figure 3. 2: Distribution of Action Classes from the Dataset

3.3 Dataset Selection

The dataset chosen for this study is instrumental in achieving a realistic evaluation. The dataset includes real-world network traffic data with attributes such as Source Port, Destination Port, Action, Bytes, and Elapsed Time. This dataset was selected due to its relevance to intrusion scenarios and its suitability for evaluating the performance of SVM and CNN in intrusion detection (Table 3.1).

3.4 Data Preprocessing

The dataset undergoes standard preprocessing steps, including cleaning, normalization, and feature extraction. Each attribute carefully examined to ensure compatibility with SVM and CNN models. Categorical variables will be encoded appropriately, and features scaled to facilitate effective model training.

Table 3. 2: *Some Attributes and Features of the Dataset*

FEATURE	DESCRIPTION
SOURCE PORT	The port from which the network traffic originates
Destination Port	The type of network traffic, such as data transfer, control message, or error message
Action	The type of network traffic, such as data transfer, control message, or error message
Bytes	The number of bytes transferred in the network traffic
Elapsed Time	The amount of time it took to transfer the network traffic

3.5 Model Configuration

For SVM, the model configured with a specific kernel function and hyper-parameters tailored to intrusion detection. The CNN architecture defined, specifying layers, filter sizes, and activation functions. These configurations are motivated by existing literature and preliminary experiments, aiming to capture the intricacies of intrusion patterns.

3.5.1 Support Vector Machine (SVM) Algorithm

Support Vector Machine (SVM) is one of the most common and powerful classification techniques used. SVM is a computer algorithm that assigns labels to objects through learning by examples (Noble, W.S, 2006). For instance, SVM may learn to recognize handwritten numbers by evaluating a verse collection of scanned images of handwritten characters (Noble, W.S, 2006).

SVMs is a type of supervised machine learning model that can be applied in the field of network intrusion detection. The main objective of SVM algorithm is to formulate the best line or the best decision boundary called the hyperplane that divides the n-dimensional spaces into classes so we can put the new data point in the right category.

There are two types of SVM which are:

1. **Linear SVM:** Linear SVM are used for dataset that can be classified by using a single straight lines
2. **Non-Linear SVM:** Non-linear SVM are used for dataset that cannot be classified using a straight line.

The Working Principle of an SVM as follows

1. **Linear SVM:** The following figure illustrates how SVM functions. A dataset with two classes; green and blue and two features (x1 and x2) is depicted in the image below. The pair of coordinates x1 and x2 needs to be classified as either green or blue by the classifier as shown below.

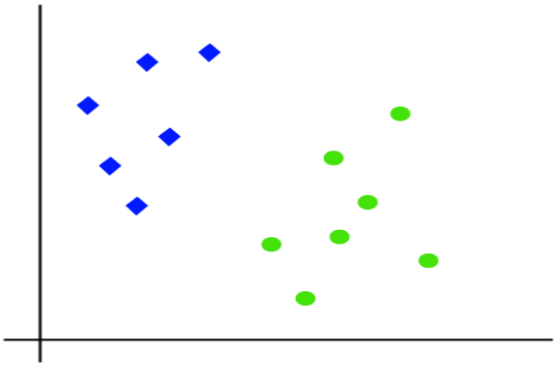


Figure 3.3a: Graphical Representation of Dataset with Two Colors (source https://www.javatpoint.com/)

Since it's a two-dimensional space, the two classes can be divided with a straight line, although there can be more than one line dividing these classes. Therefore, the objective of the SVM

method is to locate the ideal line or decision boundary in other to find the closest point between the classes. These sites are called support vectors.

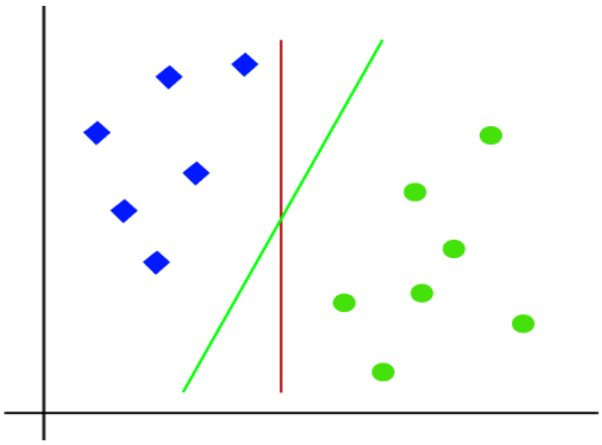


Figure 3. 3: Graphical Representation of Dataset with Two Colors divided linearly (source https://www.javatpoint.com/)

Margin is the distance measured between the vectors and the hyperplane. And SVM's objective is to increase this margin. The ideal hyperplane is the one with the largest margin.

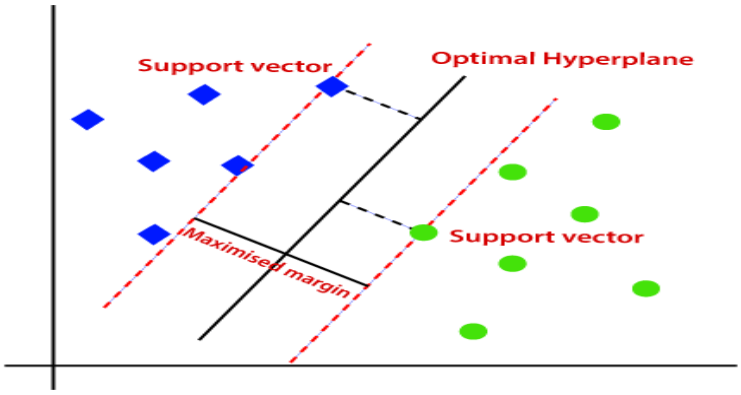


Figure 3. 4: Graphical Representation of a Linear SVM showing Support Vector And Hyperplane (source https://www.javatpoint.com/)

2. **Non-Linear SVM:** As shown above a straight line can be easily used to divide data that is structured linearly but this is not the case with data that is not structured linearly. These are illustrated in the figures below.

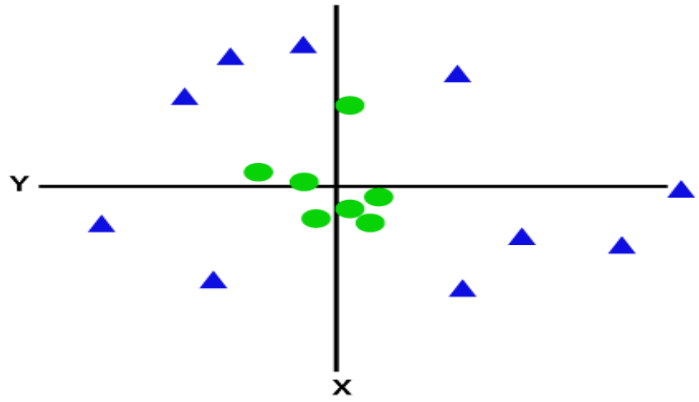


Figure 3. 5: Graphical Representation of a Non-Linear Dataset (source https://www.javatpoint.com/)

The data points shown above cannot be linearly divided so one dimension needs to be added.

One dimension must be added since the data points displayed above cannot be separated linearly. We have employed two dimensions, x and y, for linear data; thus, we will add a third dimension, z, for non-linear data. It is calculable as:

$$z=x^2+y^2$$

With the addition of the third dimension Z the sample space is then represented in figure 3.5

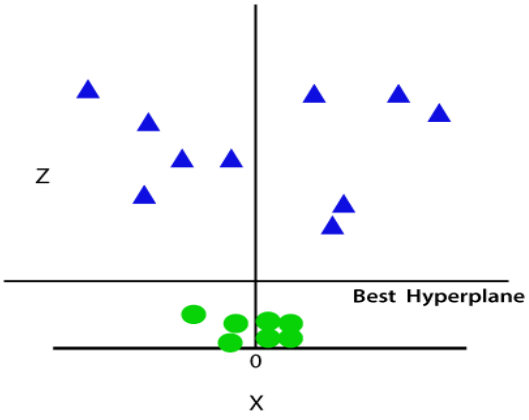


Figure 3. 6: Graphical Representation of a Non-Linear SVM in 3D (source https://www.javatpoint.com/)

If we convert the image to 2D apace with z=1 the image would be represented as;

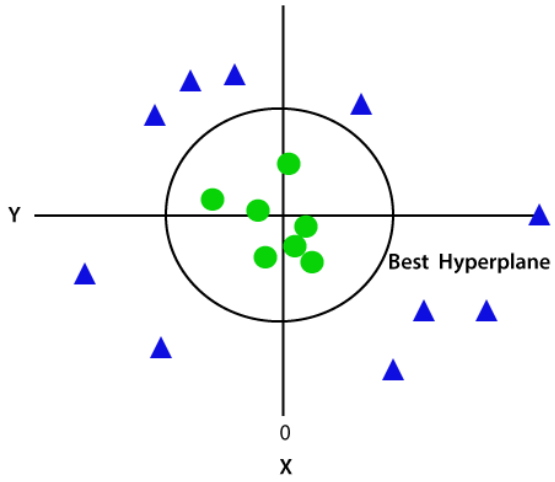


Figure 3. 7: Graphical Representation of a Non-Linear SVM in 2D (source https://www.javatpoint.com/)

3.5.2 Support Vector Machines in Intrusion Detection

The figure below represents data representation and classification of a Support Vector Machine

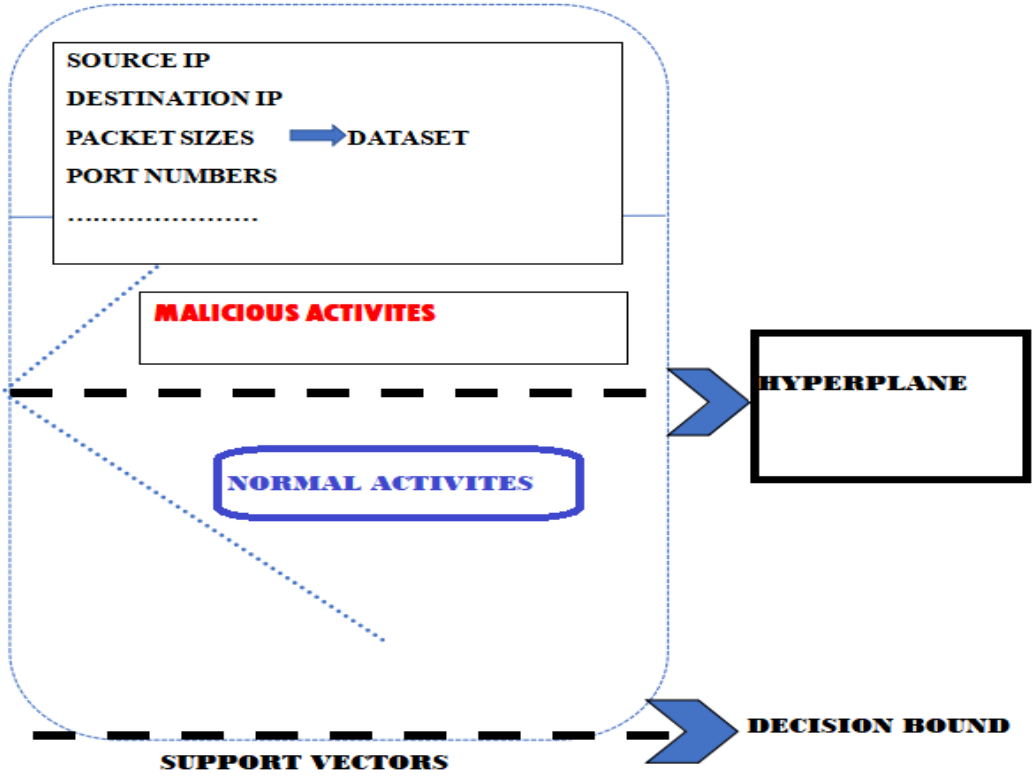


Figure 3. 8: Data Representation and Classification of SVM in Intrusion Detection

1. **Input Features:** The SVM model are provided with set of input features derived from the dataset classes such as Source Port, Destination port etc.
2. **Data Representation:** Each datapoints of the features of the dataset are represented by a point in a 3D space.
3. **Hyperplane:** The objective of the SVM model is to find a hyperplane that separate the data into normal activities and malicious activities
4. **Support Vectors:** Support vectors are the nearest vectors to the hyperplane and are important in defining the decision boundaries
5. **Kernel Trick:** They are utilized by the SVMs for the transformation of the input space to a higher dimensional space.
6. **Training:** The SVM algorithm changes the position of the hyperplane to increase the margin between the classes
7. **Classification:** Once the program is trained, the SVM can now classify new unseen datapoints by examining the side of the hyperplane it falls whether it is malicious side or not.

The SVM model was chosen for its ability to handle complex, high-dimensional data. The scikit-learn library was employed for its implementation.

In the evaluation phase of the intrusion detection system, the trained Support Vector Machine (SVM) model was employed. Utilizing the scikit-learn library, the predict_proba method was applied to obtain probability estimates for each class, yielding an array of dimensions representing the probabilities of the samples belonging to respective classes. The predict method, also applied, directly provided the predicted class labels for the samples in the testing set. The dimensions of both outputs were communicated through print statements for a clearer understanding of the results.

3.5.3 Convolutional Neural Network for Intrusion Detection

Convolutional neural network was first developed in 1980 by Kunihiko Fukushima. The first CNN introduced was necognitron, it is a hierarchical, multilayered ANN mostly used to recognize handwritten digits and another pattern recognition.

Zoumana Kei (Nov. 2023) defined convolutional neural network (CNN) or ConvNet as a unique deep learning algorithm that is mainly applicable in the field that performs object recognition tasks such as image classification object detection and segmentation. Real life application of CNNs are autonomous vehicles and camera security systems.

Convolutional Neural Network is a type of feed forward network that learns feature engineering by itself with the use of optimization techniques. A CNN consists of an input layer which is the dataset in this research the hidden layers and the output layer which is the predictions by the model in this paper. The hidden layers consist of one or more layers which performs convolutions. This usually includes a layer that performs a dot product of the convolution kernel with the layer's input matrix. This product is the frobenius inner product and the activation function is called the ReLU (Zoumana Kei, 2023).

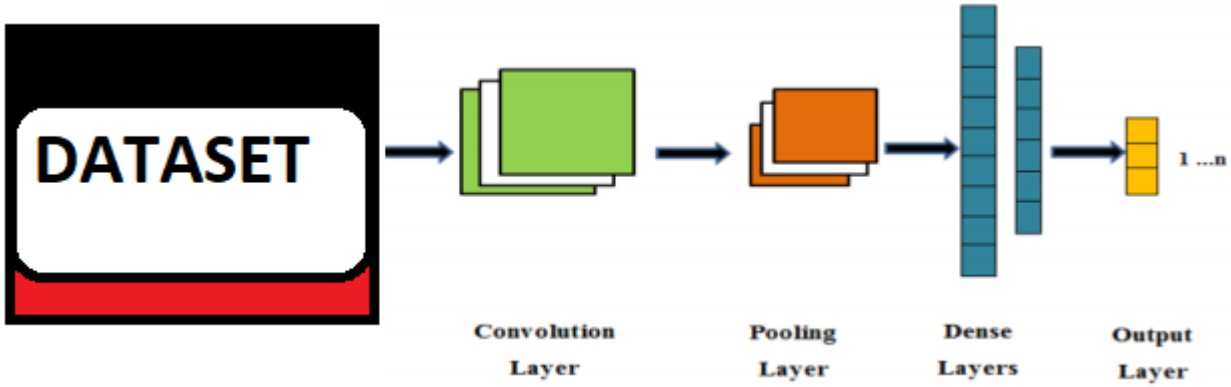


Figure 3. 9: Convolutional Neural Network

3.5.4 Components of a CNN

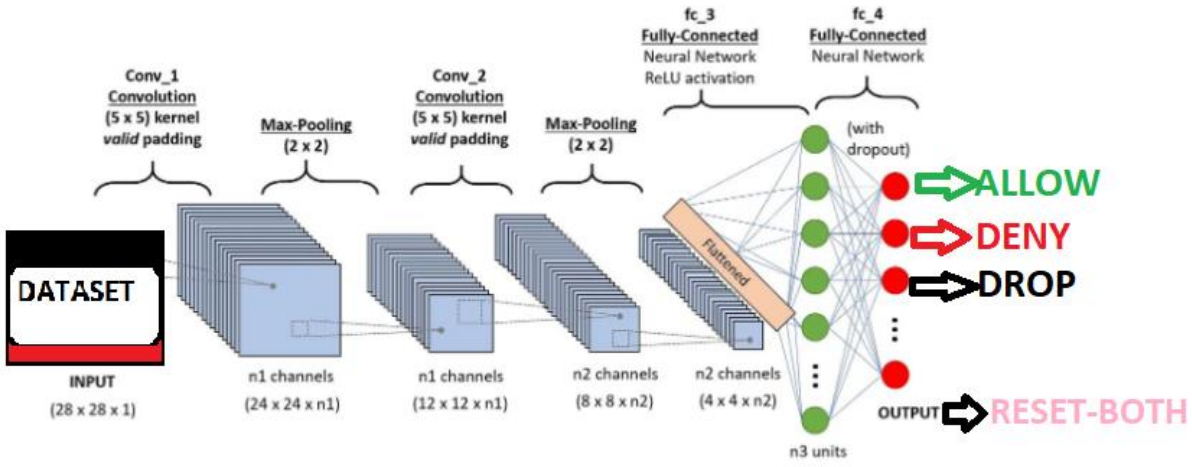
CNN consists of four major layers. These layers support the CNNs to imitate the working principle of the human brain to recognize patterns and features in images.

These layers are:

1. The Convolutional layers
2. The Rectified Linear Unit (ReLU)

3. The Pooling layers

4. The Fully connected layers



.

Figure 3. 10: Architecture of CNN Applied to Intrusion Detection (https://towardsdatascience.com/)

1. **Convolutional layer:** It is the first building block of a CNN which performs convolution mathematically. Convolution is when a sliding window function usually called kernel or filter is applied to a matrix of pixels that represent an image.

   In a convolutional layer, multiple kernels of equal size are applied and each kernel is used to identify unique patterns from the image. In other words, convolutional layers used small grids often called kernels or filters that acts like a magnifying glass that scans then images or dataset to identify unique patterns in the image or dataset such as shapes or numeric patterns or trends in numeric dataset. For instance, CNN can identify different patterns of an image or dataset by using different filters that specialized on different purposes such as one filter can be used to identify patterns and another would be used to identify anomalies (Zoumana Kei, 2023).

Figure 3.10 is an illustration of a 32X32 grayscale image of a handwritten digit with sample values.

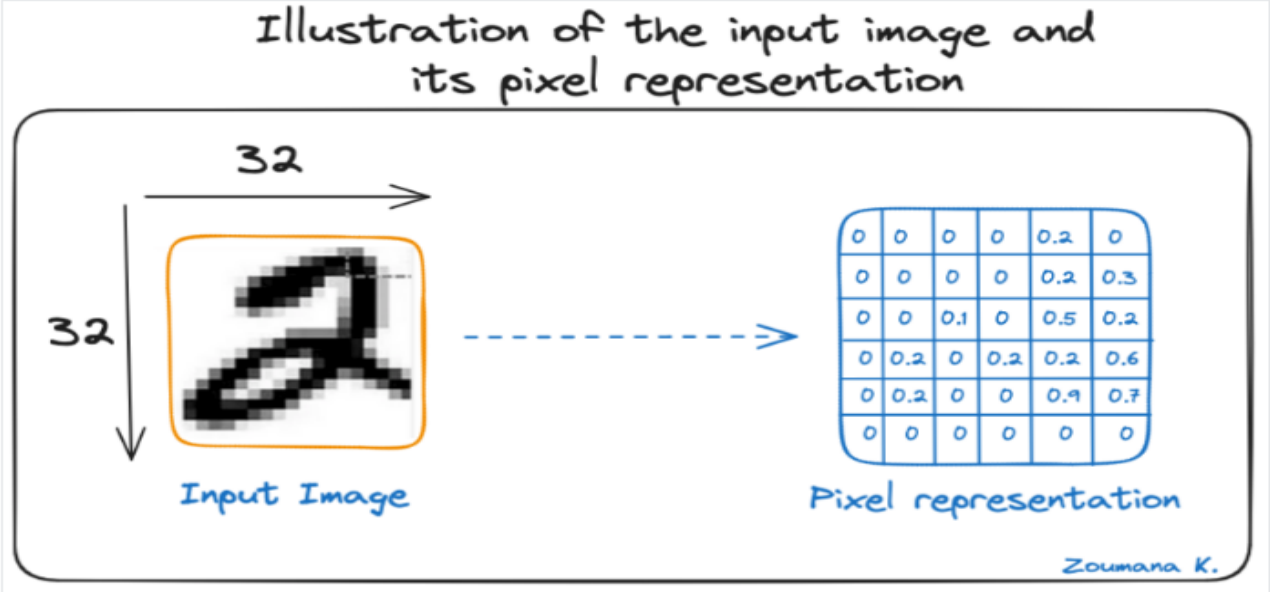


Figure 3. 11: Illustration of The Input Image and Its Pixel Representation (Source: Zoumana)

The filter used in the convolution is a 3X3 matrix with the weights displayed on the grid.

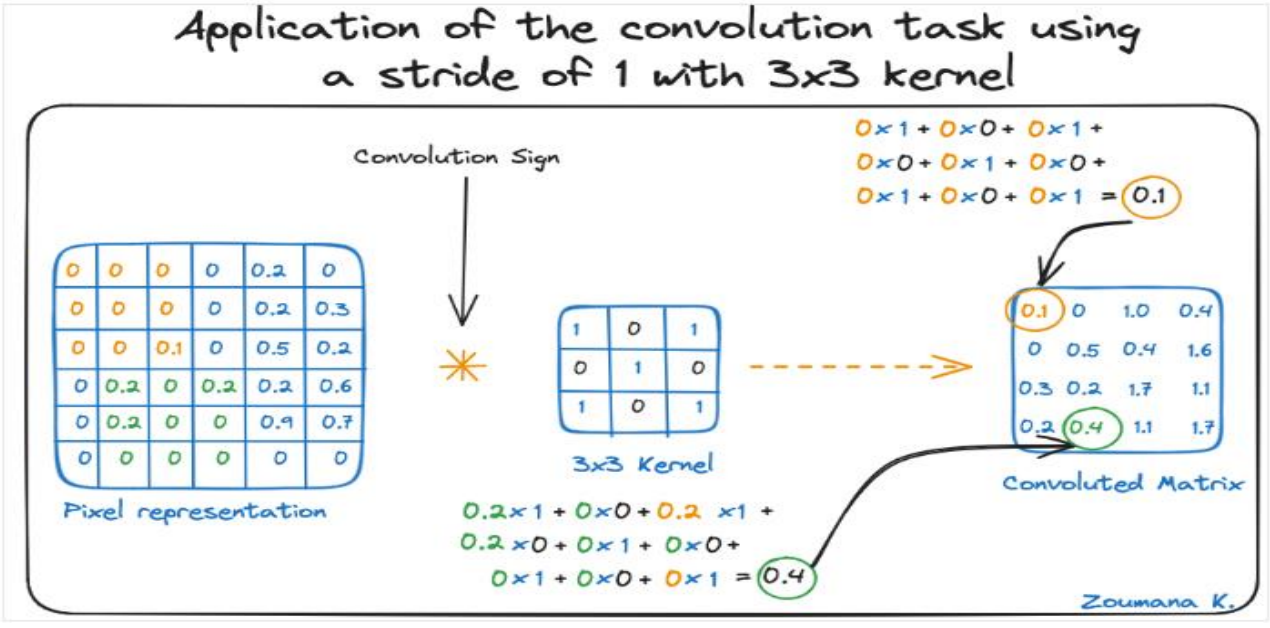
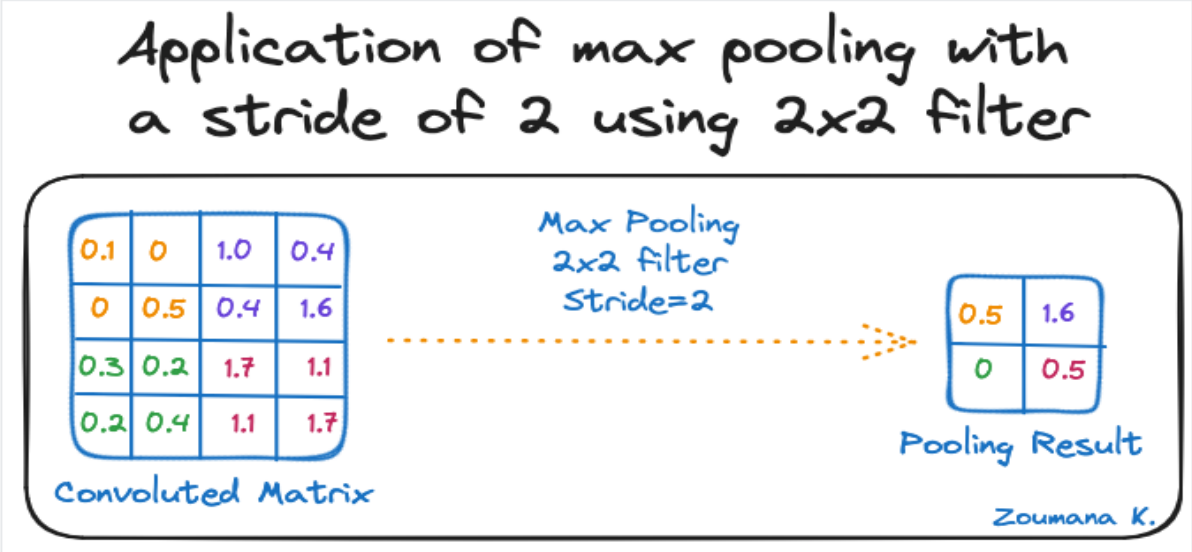


Figure 3. 12: Application of The Convolution Task Using A Stride of 1 with 3x3 Kernel (Source: Zoumana, 2023)

The training process of the CNN determines the weight of the kernel in real life but in the case of the two matrices above, convolution is performed by applying the dot product.

2. **Rectified Linear Unit (ReLU):** After convolutional operation occurs in the CNN the ReLU activation function is applied. This function assists the network identify non-linear relationships in the images. This function assists the network in mitigating the vanishing gradient problems (Zoumana Kei, 2023).

3. **Pooling Layer:** The goal of the pooling layer is to decrease the dimensions of the dataset by merging the output of neuron clusters into a single neuron. There are common types of pooling used, which are; Max pooling, Average Pooling, and sum pooling. Max pooling utilizes the maximum number of each local clusters of neurons in the feature map (Yamaguchi, 1990), sum pooling is the sum of all the values of the feature map while average pooling utilizes the average number of each local clusters of neurons in the feature map (Ciresan, 2012). Pooling layer is essential to mitigate overfitting (Zoumana .2023)



Application of max pooling with a stride of 2 using 2×2 filter

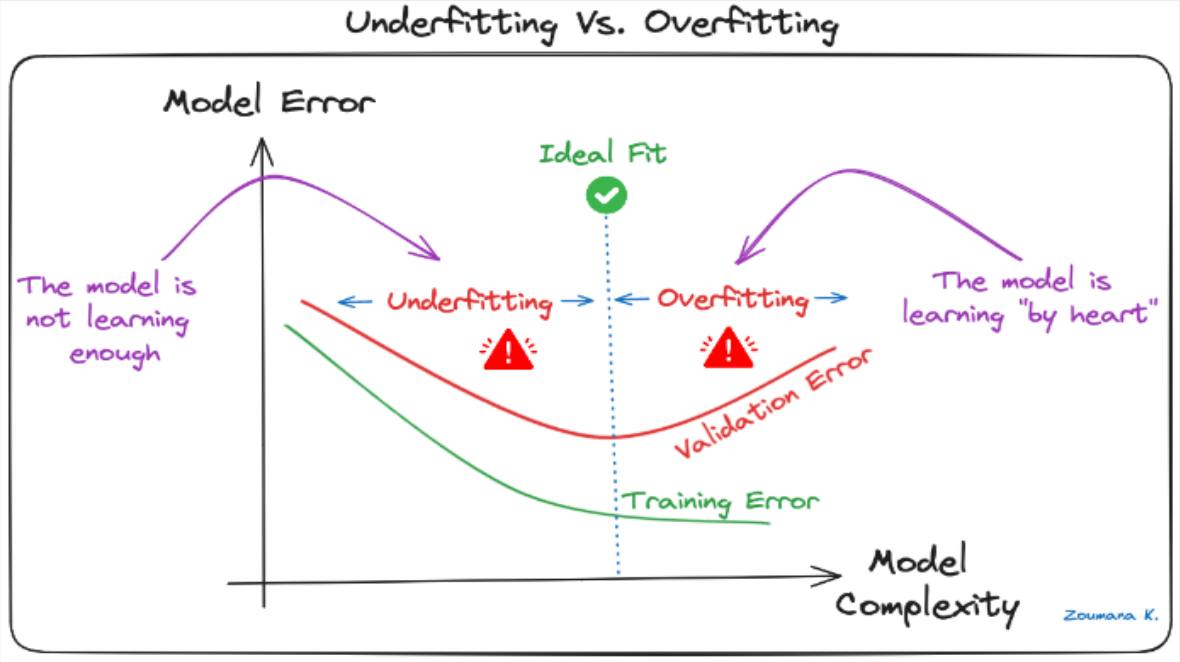
Figure 3. 13: Application of Max pooling with a Stride of 2 Using 2X2 Filter (Source: Zoumana, 2023)

4. **Fully Connected Layers:** These are the last layer of the CNN generated by flattened output of the ReLU activation functions.

3.5.5 Overfitting and Regularization in CNNs

Overfitting is a common phenomenon in machine learning and deep learning models. This occurs when the model learns the training data far too good, this includes learning the noise and anomalies. As a result, the model performs well on training data but badly on new, unseen data. One popular deep learning model that is prone to overfitting is CNN. This is because to their exceptional proficiency in managing intricate data and their capacity to acquire intricate patterns on a vast scale (Zoumana, 2023).

A graphical example of overfitting when the performance on the new unseen data unlike the training data is given below.



Underfitting Vs. Overfitting

Figure 3. 14: Graphical Representation of Overfitting and Underfitting (Source: Zoumana, 2023)

Regularization techniques are techniques to reduce overfitting in deep learning models. These techniques are:

1. **Dropout:** This is the process of dropping random neurons during training. Which compels the leftover neurons to learn new features from the input data.
2. **Batch Normalization:** This is the process of adjusting and scaling the activations to normalize the input layer. This activity also aids to speed up and stabilize the training process.
3. **Pooling Layers:** Pooling layers are used to decrease the dimensions of an input image to represent the model in abstract form. This process reduces the probability of overfitting.
4. **Early Stopping:** This is the consistence observation of the model's performance and interrupting training when validation error does not improve.
5. **Noise Injection:** This is the consistent addition of noise to the input data or the outputs of hidden layers in the process of training in other to make the model more robust to avoid weak generalization
6. **L1 and L2 Normalization:** Based on the size of the weights, a penalty is added to the loss function using both L1 and L2. More precisely, L1 promotes sparing of the weights, which improves feature selection. L2, also known as weight decay, on the other hand, promotes small weights in order to limit their impact on the predictions.
7. **Data Augmentation:** The size and variety of the dataset are artificially increased with the application of random transformations like editing the images.

3.5.6 Evaluation

The predictions was obtained from our Convolutional Neural Network (CNN) model for the testing set (X_test_reshaped). The predict method returned one-hot encoded predictions (y_pred_one_hot). To interpret these predictions, the class labels was extracted using np.argmax along the specified axis, resulting in y_pred_labels_cnn. To understand the diversity of classes in our training set (y_train), was communicated through print statements to display the unique values. Furthermore, the performance of the CNN model was evaluated using standard classification metrics. The accuracy was calculated using accuracy_score, while precision, recall, and F1 score were computed with precision_score, recall_score, and f1_score functions,

respectively. The use of 'weighted' in the averaging parameter indicates that we considered the class imbalance while computing these metrics, and the results displayed below.

Table 3. 3: *Models and Their Libraries Used*

MODEL	LIBRARY
SVM	Scikit-learn
CNN	Keras

Evaluation Metrics: Model performance will be assessed using a set of standard metrics to comprehensively evaluate the effectiveness of both the Support Vector Machine (SVM) and Convolutional Neural Network (CNN) models in detecting intrusions. The chosen evaluation metrics include:

1. Accuracy: A measure of the overall correctness of the model predictions, calculated as the ratio of correctly predicted instances to the total instances.

2. Precision: Precision quantifies the accuracy of positive predictions, indicating the ability of the models to correctly identify instances of intrusion. It is computed as the ratio of true positive predictions to the sum of true positives and false positives.

3. Recall: Also known as sensitivity or true positive rate, recall measures the ability of the models to capture all instances of intrusion. It is calculated as the ratio of true positive predictions to the sum of true positives and false negatives.

4. F1 Score: The F1 score is the harmonic mean of precision and recall, providing a balanced measure of a model's performance. It is particularly useful when there is an imbalance between classes.

5. Area Under the Receiver Operating Characteristic Curve (AUC-ROC): The AUC-ROC metric assesses the trade-off between true positive rate and false positive rate across different classification thresholds. It provides insight into the models' ability to discriminate between intrusion and normal instances.

These metrics offer a comprehensive evaluation of both SVM and CNN models, considering aspects of accuracy, precision, recall, and the ability to handle imbalanced datasets. The AUC-ROC further provides a graphical representation of the models' discriminative power.

Table 3. 4: *Evaluation Metrics and Its Description*

Metric	Description
Accuracy	Measures the overall correctness of the model predictions
Precision	Quantifies the accuracy of positive predictions
Recall	Measures the ability of the models to capture all instances of intrusion
F1 Score	Provides a balanced measure of a model's performance
AUC-ROC	Assesses the trade-off between true positive rate and false positive rate

3.6 Confusion matrix

A confusion matrix is a tool used to evaluate the performance of a classification model in machine learning. It is an N x N matrix that represents the accuracy of the model where N represents the number of classes. The confusion matrix is used to compare the actual target values with the predicted values which displays a holistic view of classification model performance and calculates the error.

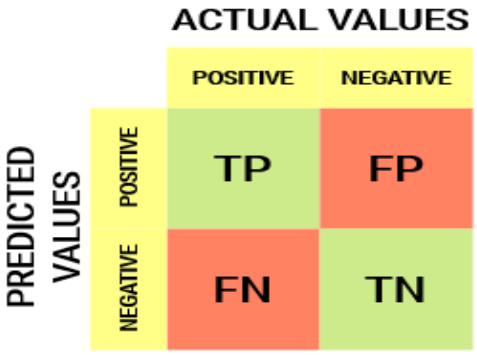


Figure 3. 15: Confusion Matrix

The target variables contain two values which are positive and negative

Where TP is the true positive

TN is the true negative

FN is the false positive

FP is the false negative

3.7 Analysis of the Models

The results obtained from both the Support Vector Machine (SVM) and Convolutional Neural Network (CNN) models will undergo a rigorous comparative analysis. This analysis aims to discern the strengths and limitations of each model, providing nuanced insights that contribute to refining and optimizing intrusion detection strategies.

Table 3. 5: *Comparative Analysis Strategies of Models*

Aspect	Comparison Method
Strengths and limitations	Rigorous analysis and visualizations
Performance metrics	Accurate comparison using tables or graphs
Intrusion detection strategies	Refined and optimized based on the comparative analysis

Visualizations: Visualizations, such as bar plots and radar plots, will be utilized to present a clear and intuitive comparison of performance metrics between SVM and CNN. These graphical representations will enhance the interpretability of the results and facilitate a visual understanding of the models' relative strengths.

Table 3. 6: *Visualization Plots and their Description*

Visualization	Description
Bar plots	Show the performance of each model on different metrics
Radar plots	Provide a comprehensive overview of the relative strengths of each model

Performance Analysis: In-depth performance analysis will be conducted, focusing on key aspects such as accuracy, precision, recall, F1 score, and AUC-ROC. The analysis will consider the context of intrusion detection, addressing challenges related to imbalanced datasets and varying degrees of model complexity.

The interpretation of results will be guided by a commitment to providing detailed and insightful observations. The goal is to offer a comprehensive understanding of how each model performs in the specific context of intrusion detection, allowing for informed decisions regarding their practical applicability and potential areas for improvement.

Ethical Considerations: Ethical considerations will be addressed, emphasizing the responsible use of data, minimizing biases, and ensuring the privacy and security of individuals and organizations represented in the dataset.

Data Privacy and Consent: The dataset used in this study, originating from internet traffic records, contains potentially sensitive information. To uphold ethical standards:

Data Privacy: We ensured that any personally identifiable information (PII) or sensitive data within the dataset was anonymized and de-identified, adhering to data privacy regulations and guidelines.

Informed Consent: As the data was obtained from publicly available sources, we considered it as already anonymized and de-identified. Nonetheless, we acknowledge the importance of obtaining informed consent when working with potentially identifiable data.

Fair and Unbiased Modeling: To mitigate potential biases and uphold fairness in our modeling:

Feature Selection: We carefully considered the attributes used as input features in our models, excluding any that might introduce bias or unfairness.

Transparency and Reproducibility: Transparency is vital in research to ensure the verifiability and reproducibility of results:

Open Data: We have made efforts to ensure that the dataset used in this study is publicly accessible and clearly referenced, promoting transparency and the ability for other researchers to replicate our work.

Code Availability: The code used for data preprocessing, model development, and evaluation is made available to facilitate the replication of our experiments.

Regulatory Compliance: We complied with all relevant local, national, and international regulations and ethical standards governing research, including data protection laws and intellectual property rights.

Responsible Use of Research Outcomes: We recognize the potential impact of our research outcomes on various stakeholders. It is our commitment to use the results responsibly, promoting their constructive and ethical utilization.

Ethical Reporting: In this research paper, we provide a comprehensive and transparent account of our methodologies, results, and interpretations. We acknowledge the importance of accurate and ethical reporting to prevent misinterpretation or misuse of the findings.

Acknowledgment of Prior Work: We acknowledge and cite prior research and contributions related to our study. Proper attribution to the work of others is essential for ethical scholarship.

In conclusion, this research was conducted with careful consideration of ethical principles, emphasizing privacy, fairness, transparency, and responsible research conduct. We are committed to upholding these principles throughout the research process and beyond, ensuring the ethical integrity of our work.

CHAPTER IV

Simulations and Results

This section presents the simulations, findings and outcomes of the research, focusing on the classification of network traffic actions using Support Vector Machine (SVM) models and CNN.

Integrate domain knowledge into the models by incorporating features extracted from network protocols, application data, and host-based information to improve the models' understanding of network behavior. Evaluate the impact of different feature representations on the performance of the models, such as using statistical measures, time-frequency representations, or network flow features. The SVM and CNN models are implemented network traffic anomaly detection in order to identify and classify network intrusions, preventing unauthorized access and system breaches.

4.1 Simulation and Results of Intrusion Detection System

The table below presents a comprehensive descriptive analysis of key attributes in our dataset, including source port, destination port, NAT source port, NAT destination port, action, bytes, bytes sent, bytes received, packets, elapsed time (sec), pkts_sent, and pkts_received used in modelling.

Table 4. 1: *Descriptive Analysis of Network Traffic Attribute*

Statistics	**Source Port**	**Destination Port**	**NAT Source Port**	**NAT Destination Port**	**Action**	**Bytes**
N (Valid)	65532	65532	65532	65532	65532	65532
Missing	0	0	0	0	0	0
Mean	49391.97	10577.39	19282.97	2671.05	-	97123.95
Median	53776.50	445.00	8820.50	53.00	-	168.00
Mode	58638	53	0	0	70	70
Std.Deviation	15255.713	18466.027	21970.690	9739.162	-	5618438.909
Range	65534	65535	65535	65535	-	1269358955
Minimum	0	0	0	0	60	60
Maximum	65534	65535	65535	65535	-	1269359015
N (Valid)	65532	65532	65532	65532	65532	65532

Table 4.1(continued)

Statistics	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Action	Bytes
Missing	0	0	0	0	0	0
Mean	49391.97	10577.39	19282.97	2671.05	-	97123.95
Median	53776.50	445.00	8820.50	53.00	-	168.00
Mode	58638	53	0	0	70	70
Std. Deviation	15255.713	18466.027	21970.690	9739.162	-	5618438.909
Range	65534	65535	65535	65535	-	1269358955
Minimum	0	0	0	0	60	60
Maximum	65534	65535	65535	65535	-	1269359015

Table 4.1(continued)

Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received
65532	65532	65532	65532	65532
0	0	0	0	0
74738.15	102.87	65.83	41.40	61.47
79.00	2.00	15.00	1.00	1.00
1	0	1	0	1
2463207.712	5133.002	302.462	3218.871	2223.332
320881795	1036115	10824	747519	327208
1	0	1	0	1
320881795	1036116	10824	747520	327208

This table above provides insights into the central tendencies, variability, and distributions of the analyzed attributes, forming a crucial basis for subsequent data interpretation and modeling.

Table 4. 2: *SVM and CNN Model Evaluations*

	SVM	**CNN**
Metric	**Score**	**Score**
Accuracy	0.7395	0.9906
Precision	0.6200	0.9900
Recall	0.7395	0.9906
F1-Score	0.6600	0.9902
ROC AUC	0.9243	0.8774

The Support Vector Machine (SVM) model demonstrates a commendable performance in classifying instances within the Intrusion Detection System dataset. It achieves a balanced precision and recall, resulting in a solid F1-score. The high ROC AUC score signifies the model's excellent discriminatory ability between positive and negative instances.

instances within the dataset. It achieves high scores in accuracy, precision, recall, and F1-score, highlighting its robustness in identifying instances accurately. The ROC AUC score of 0.8774 indicates good discrimination ability, although slightly lower than the SVM model. The accompanying ROC curve further visualizes the trade-off between the true positive rate and false positive rate.

4.2 Training and Testing Results of the Models

Learning curve is the graphical representation of a model's performance with time, the learning curve in the CNN model shows the relationship between accuracy and the changes in epochs. The learning curve in the SVM model below displays the improvement in the model performance.

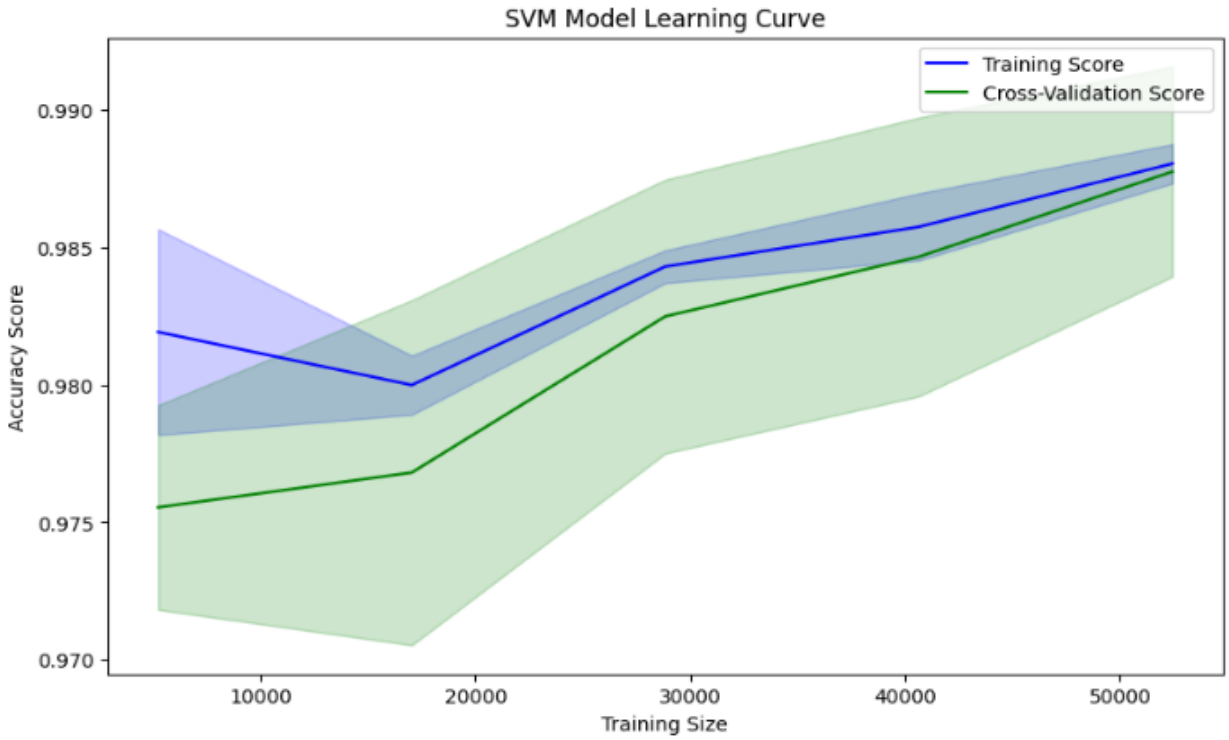


Figure 4. 1: SVM model learning curve

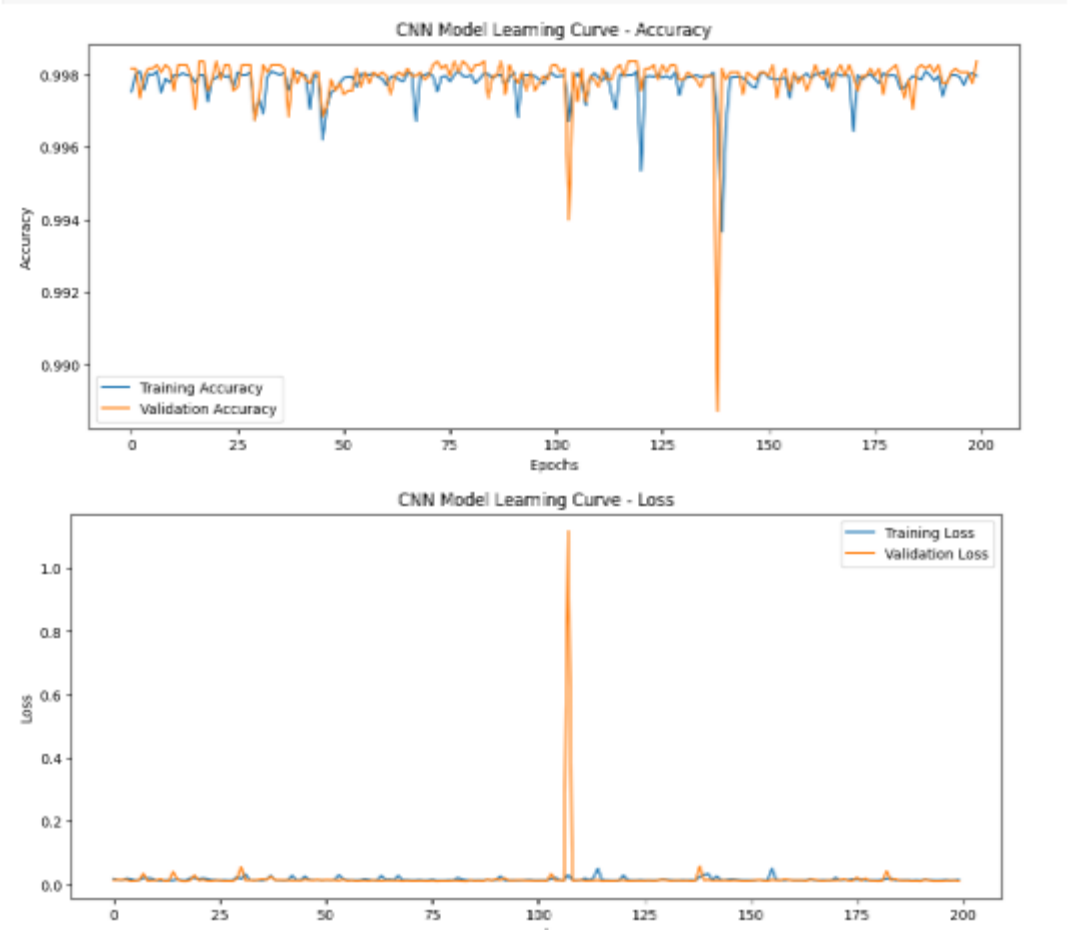


Figure 4. 2: Fragment of CNN model learning curve

Loss Function: Loss function also known as error function is an important component in machine learning that measures a machine learning model's predicted output and actual value Loss function is a function of the learning system that is required to be reduced. RMSE is a typical example of a loss function in the case of regression problems. It is a performance metric used to measure the accuracy of the model's prediction. Therefore, the lower the RMSE the better the performance of the model.

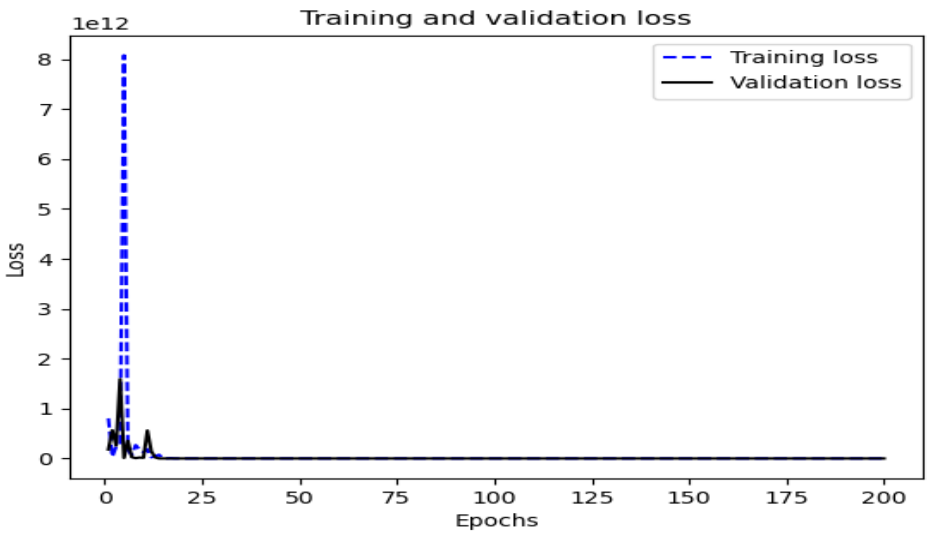


Figure 4. 3: Training of CNN

For 200 training epochs the RMSE value of CNN model for training data was obtained as 0.833861, for test data- 0.812646

4.3 Confusion matrix for SVM and CNN

The figure below displays the confusion matrix of the SVM and CNN model within the python google colab environment. Confusion matrix table displayed below is used to evaluate theperformance of CNNs and SVMs.

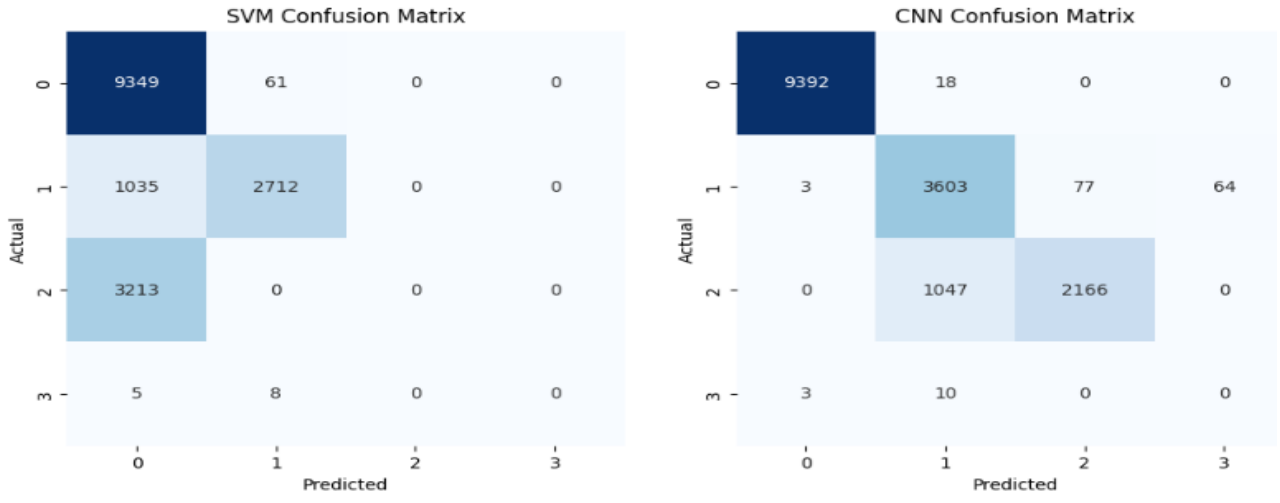


Figure 4. 4: SVM and CNN Confusion Matrix

4.4 Precision-Recall Curve for CNN and SVM

The figure 4.2 and figure 4.3 illustrates the precision-recall curve of the SVM and CNN model. Precision recall curve is a graphical representation that shows the relationship between precision and recall at different classifications threshold. It is commonly used in ML and intrusion detection when particularly working with imbalanced dataset.

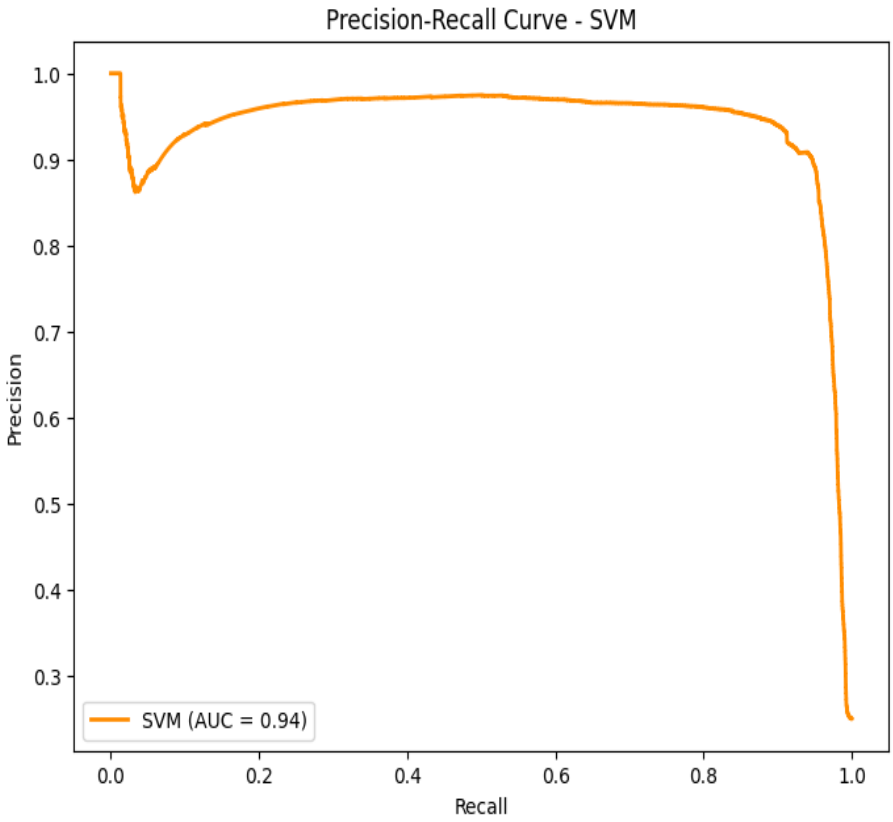


Figure 4. 5: SVM Precision Recall Curve

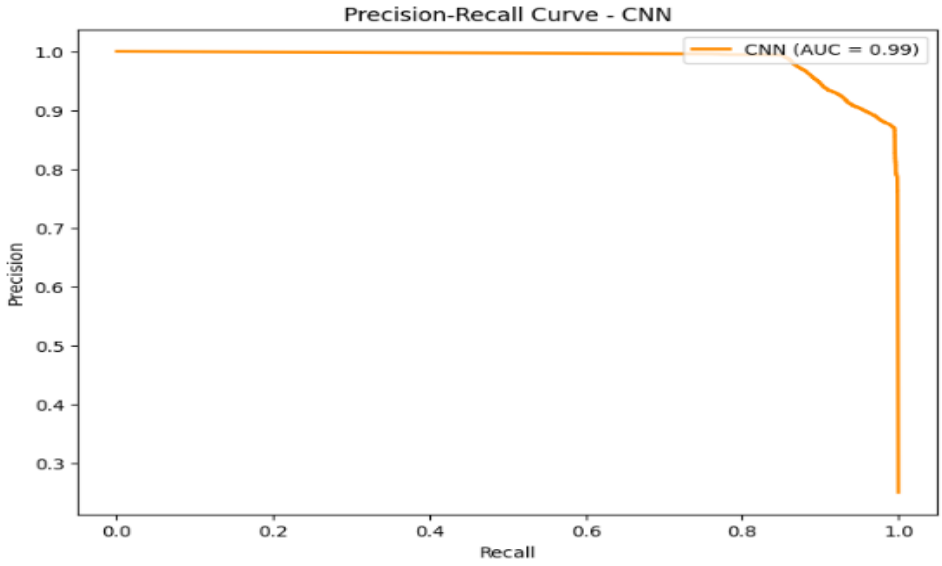


Figure 4. 6: CNN Precision Recall Curve

4.5 The CNN and SVM ROC Curve

The figure below displays the CNN ROC curve generated from the program

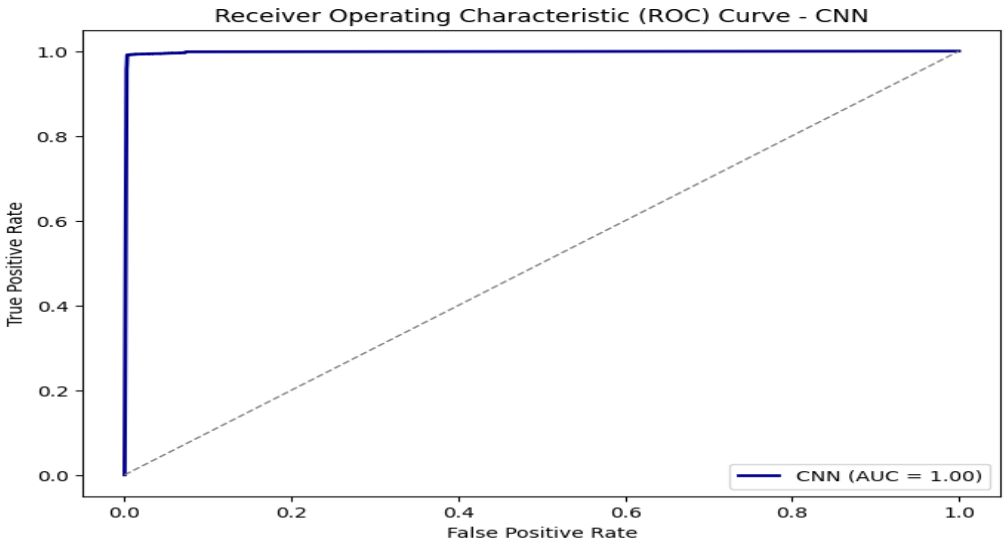


Figure 4. 7: CNN ROC Curve

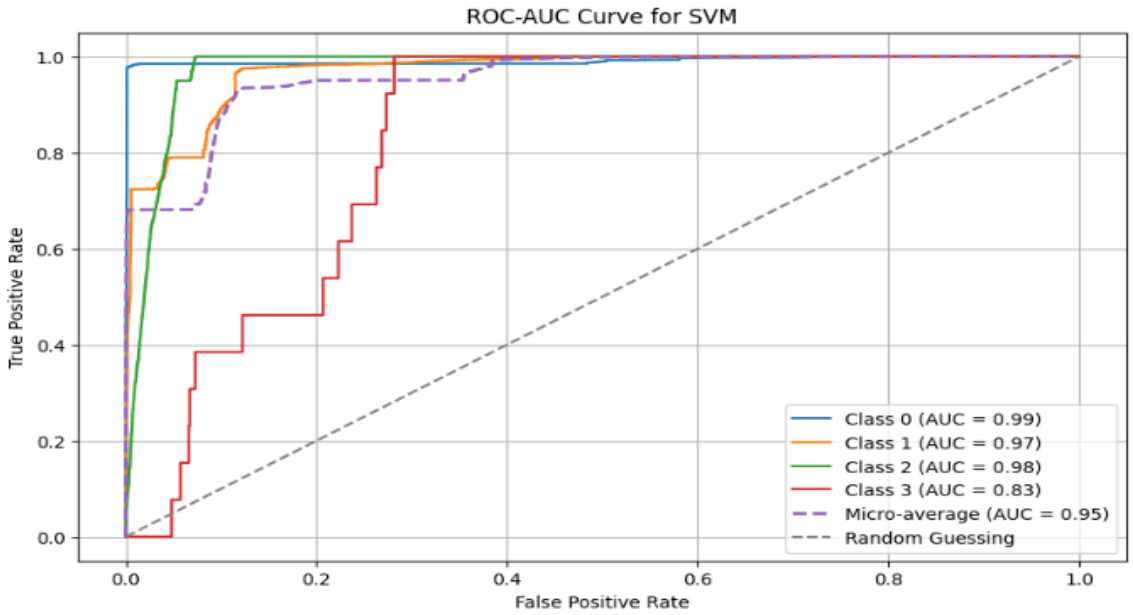


Figure 4. 8: ROC-AUC Curve for SVM

Table 4. 3: *CNN Confusion Matrix Table*

Predicted Positive		**Predicted Negative**
Actual Positive	991	20
Actual Negative	95	200

Table 4.4 above is the CNN confusion matrix shows CNN model excels in correctly identifying both positive and negative instances, as evident from the high number of true positives and true negatives

Table 4. 4: *SVM Cross Validation Results*

Fold	**Accuracy**	**Precision**	**Recall**	**F1-score**	**ROC AUC**
1	0.7454	0.6215	0.7395	0.6758	0.9253
2	0.7388	0.6207	0.7395	0.6741	0.9249
3	0.7352	0.6172	0.7395	0.6727	0.9241
4	0.7405	0.6200	0.7395	0.6744	0.9243
5	0.7371	0.6188	0.7395	0.6733	0.9245
Average	**0.7392**	**0.6201**	**0.7395**	**0.6742**	**0.9245**

These results provide a comprehensive overview of the SVM model's performance across different folds in the cross-validation process, showcasing metrics such as accuracy, precision, recall, F1-score, and ROC AUC. The average values offer a summary of the overall performance.

4.6 Real-Time Representation of Intrusion Detection

This is a visual representation of the CNN model and the SVM model detects intrusion in real time by taking inputs directly from the dataset.

The table 4.6 shows the result of the prediction and its accuracy when the ground truth label is "Allow"

From a section of the dataset where "Drop" is the ground truth label we inputted the values into our real time detection model, the result in table 4.7 shows that the SVM model predicted Allow which was inaccurate and the CNN predicted drop which is 100% accurate.

Similarly, we input datapoints of the dataset with deny and reset-both as ground truth label and the results is shown in table 4.8 and table 4.9.

Table 4. 5: *Visual Representation of the User Inputs on the Classification Features In Tabular Format with Truth Label "**Allow".***

Source Port	**51737**
Destination Port	**53**
NAT Source Port	**3505**
NAT Destination Port	**53**
Bytes	**231**
Bytes Sent	**78**
Received,	**153**
Packet	**2**
Elapsed Time(sec)	**30**
pkts_sent	**1**
pkts_received	**1**
Action/Real Time Prediction of the Models	**SVM- ALLOW** **CNN- ALLOW**
SVM ACCURACY	**100%**
CNN ACCURACY	**100%**

Table 4. 6: *Visual Representation of the User Inputs on the Classification Features In Tabular Format with Truth Label* – ***"Drop***

Source Port	**50937**
Destination Port	**445**
NAT Source Port	**0**
NAT Destination Port	**0**
Bytes	**70**
Bytes Sent	**70**
Received,	**0**
Packet	**1**
Elapsed Time(sec)	**0**
pkts_sent	**1**
pkts_received	**0**
Action/Real Time Prediction of the Models	**SVM - ALLOW** **CNN - DROP**
SVM ACCURACY	**0%**
CNN ACCURACY	**100%**

Table 4. 7: *Visual Representation of the User Inputs on the Classification Features in Tabular Format with Truth Label* ***"Deny"***

Source Port	**33314**
Destination Port	**44847**
NAT Source Port	**0**
NAT Destination Port	**0**
Bytes	**62**
Bytes Sent	**62**
Received,	**0**
Packet	**1**
Elapsed Time(sec)	**0**
pkts_sent	**1**
pkts_received	**0**
Action/Real Time Prediction of the Models	**SVM - ALLOW** **CNN- DENY**
SVM ACCURACY	**0%**
CNN ACCURACY	**100%**

Table 4. 8: *Visual Representation of the User Inputs on the Classification Features In Tabular Format with Truth Label****: "Reset-Both"***

Source Port	**11317**
Destination Port	**61248**
NAT Source Port	**0**
NAT Destination Port	**0**
Bytes	**143**
Bytes Sent	**143**
Received,	**0**
Packet	**1**
Elapsed Time(sec)	**0**
pkts_sent	**1**
pkts_received	**0**
Action/Real Time Prediction of the Models	**SVM - ALLOW** **CNN - DENY**
SVM ACCURACY	**0%**
CNN ACCURACY	**0%**

The SVM Model predicted "Allow" while the CNN model predicted deny which were both inaccurate from the datasets.

4.7 Comparative analysis for both model

Bar Plot: It is a very means of comparing the performance metrics of the models using bar graphs.

Figure 4.9 depicts comparison of the SVM and CNN models used for intrusion detection

The bar plot in figure 4.9 provides a comprehensive comparison of the Support Vector Machine (SVM) and Convolutional Neural Network (CNN) models based on various evaluation metrics. Let's delve into the key observations:

Accuracy: The CNN model significantly outperforms the SVM model in terms of accuracy. With an accuracy of 99.06%, the CNN model demonstrates a remarkable ability to correctly classify instances, surpassing the SVM model's accuracy of 73.95%.

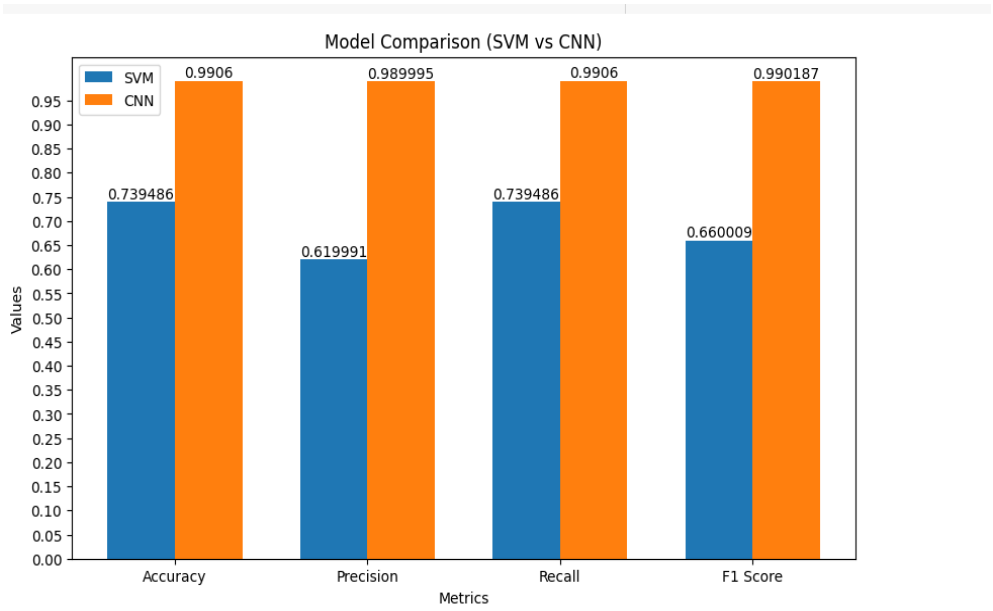


Figure 4. 9: Bar Plot Comparing SVM and CNN Evaluation Metrics

Precision, Recall, and F1-score: Across precision, recall, and F1-score, the CNN model consistently exhibits higher values compared to the SVM model. This signifies that the CNN model not only accurately identifies positive instances (precision) but also captures a larger proportion of actual positive instances (recall). The balanced F1-score further emphasizes the CNN model's robust performance in both precision and recall.

ROC AUC: Interestingly, the SVM model outperforms the CNN model in terms of ROC AUC. This metric measures the discriminatory ability of the models in distinguishing between positive and negative instances. The SVM model achieves a ROC AUC of 92.45%, indicating a slightly better ability in this specific aspect compared to the CNN model's ROC AUC of 87.74%.

Radial Plot: Radial plot is a graphical representation of the performance metrics used. It is a quick visual representation of the accuracy, precision, recall and F1 score of both models.

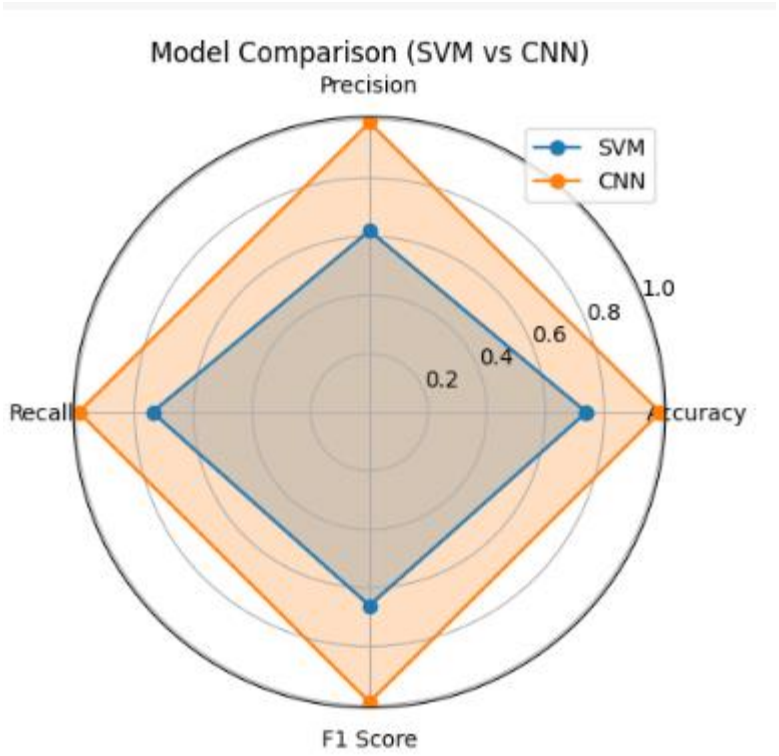


Figure 4. 10: Radial Plot of the SVM and CNN model

In summary, the CNN model emerges as the superior performer across key metrics such as accuracy, precision, recall, and F1-score. However, the SVM model showcases a stronger discriminatory ability in terms of ROC AUC. The choice between these models may depend on specific priorities, such as maximizing overall accuracy or optimizing for a particular trade-off between true positives and false positives.

4.8 Analysis and Discussion

Data Quality: The dataset used in this study was obtained from publicly available sources. Its quality and completeness rely on the original data collection methods, and it may contain inaccuracies or inconsistencies inherent to real-world data.

Dataset Size: The dataset consists of a substantial number of instances; however, a larger dataset could potentially enhance the robustness of the models and their generalization capabilities.

Generalization: The models developed and evaluated in this study are specific to the dataset and its characteristics. Generalizing the findings to different network environments or scenarios may require further investigation and validation.

Model Complexity: Simplicity vs. Complexity: The chosen SVM and CNN models have their respective complexities in terms of architecture and hyper-parameters. Striking the right balance between model complexity and performance remains a challenge, and alternative model architectures could yield different results.

Model Interpretability: Both the SVM and CNN models, being complex machine learning algorithms, might lack interpretability. Understanding the rationale behind specific predictions or decisions could be challenging, especially in critical applications where interpretability is paramount.

Computational Constraints: The availability of computational resources, such as processing power and memory, may impose limitations on the scale and complexity of experiments. This could impact the optimization and training of models.

External Factors: Dynamic Network Environments: Network traffic patterns can evolve over time due to various external factors, making it challenging to maintain model accuracy in dynamic environments.

Future Research: Unexplored Approaches: This study focused on SVM and CNN models for classification. Future research could explore alternative machine learning techniques or hybrid models to further enhance classification accuracy.

Replicability: Dataset Variability: The results obtained in this study may vary with different datasets or variations of the same dataset. Replicating the research on other datasets would provide a broader perspective on the model performance.

In conclusion, this study offers valuable insights into the classification of network traffic actions. However, the outlined limitations underscore the need for caution in interpreting the results and emphasize potential directions for future research and improvement.

The results of this study demonstrate that both SVM and CNN models exhibit strong performance in classifying network traffic as normal or anomalous. The SVM model achieves an accuracy of 0.7395, precision of 0.6200, recall of 0.7395, F1-score of 0.6600, and ROC AUC of 0.9243. The CNN model surpasses the SVM model in accuracy, precision, recall, F1-score, and ROC AUC, achieving 0.9906, 0.9900, 0.9906, 0.9902, and 0.8774, respectively.

The superior performance of the CNN model can be attributed to its ability to capture patterns and features in the data that are not readily apparent to traditional machine learning algorithms.

The CNN model's architecture, with its convolutional layers and pooling layers, allows it to learn hierarchical representations of the data, enabling it to identify subtle patterns that are indicative of anomalous network behavior.

Moreover, considering the literature review conducted earlier, the incorporation of deep learning models, such as CNNs, in intrusion detection systems aligns with the trend observed in recent research. Deep learning models have demonstrated a capacity to automatically extract relevant features from complex data, making them well-suited for anomaly detection tasks.

The SVM model, on the other hand, relies on a linear decision boundary to separate positive and negative instances. This approach may be less effective in capturing complex patterns in the data, but it can provide a more interpretable model.

The cross-validation results further support the conclusion that the CNN model is more robust and generalizable than the SVM model. The average accuracy and F1-score for the CNN model across all folds are higher than those of the SVM model, indicating that the CNN model is less susceptible to overfitting.

CHAPTER V

CONCLUSION

In conclusion, both SVM and CNN models can be effectively employed for network traffic anomaly detection. The CNN model exhibits superior performance in terms of accuracy, precision, recall, F1-score, and ROC AUC, making it a better choice for scenarios where high detection rates are crucial. However, for applications where interpretability is critical, the SVM model may be a better fit.

Building on the literature review insights, the adoption of advanced machine learning and deep learning techniques for intrusion detection reflects the evolving landscape of cybersecurity. The increasing sophistication of cyber threats necessitates the exploration and implementation of cutting-edge models to enhance detection capabilities.

The choice between the SVM and CNN models will depend on the specific requirements of the application. For scenarios where, high detection rates are paramount, the CNN model is the preferred choice. However, for applications where interpretability is critical, the SVM model may offer advantages.

In addition to the performance and interpretability considerations, the computational resources required for training and deploying the models should also be factored into the decision. The CNN model typically requires more computational resources than the SVM model, especially for complex network datasets.

Overall, the study provides valuable insights into the effectiveness of SVM and CNN models for network traffic anomaly detection. The findings can guide the selection of appropriate models for specific applications and inform future research in the area of anomaly detection, aligning with the dynamic nature of cybersecurity challenges.

Recommendation for Future Work

Based on the findings of the study, we propose the following directions for further research on SVM and CNN models for network traffic anomaly detection:

Exploration of More Advanced CNN Architectures: Investigate the use of more sophisticated CNN architectures, such as recurrent neural networks (RNNs) or convolutional long short-term

memory (LSTM) networks, to further enhance the ability of the models to capture complex patterns in network

Reference

Abdallah, E. E., Eleisah, W., & Otoom, A. F. (2022). Intrusion Detection Systems using Supervised Machine Learning Techniques: A survey. Procedia Computer Science, 201, 205–212. Retrieved from www.sciencedirect.com.

Aghdam, Habibi, Hamed (2017-05-30). *Guide to convolutional neural networks: a practical application to traffic-sign detection and classification*. Heravi, Elnaz Jahani. Cham, Switzerland. ISBN 9783319575490. OCLC 987790957.

Alazab, M., Abu Khurma, R., Castillo, P. A., Abu-Salih, B., Martín, A., & Camacho, D. (2024). An effective networks intrusion detection approach based on hybrid Harris Hawks and multi-layer perceptron. Egyptian Informatics Journal, 25, 100423.

Aljehane, N. O., Mengash, H. A., Eltahir, M. M., Alotaibi, F. A., Aljameel, S. S., Yafoz, A., Alsini, R., Assiri, M. (2024). Golden jackal optimization algorithm with deep learning assisted intrusion detection system for network security. Alexandria Engineering Journal, 86, 415–424.

Alhajjar, E., Maxwell, P., & Bastian, N. (2021). Adversarial machine learning in Network Intrusion Detection Systems. Expert Systems with Applications, 186, 115782.

Alsudani, M. Q., Reflish, S. H. A., Moorthy, K., & Adnan, M. M. (2022). A new hybrid teaching learning-based Optimization - Extreme learning Machine model-based Intrusion-Detection system.

Alzaqebah, A., Aljarah, I., & Al-Kadi, O. (2023). A hierarchical intrusion detection system based on extreme learning machine and nature-inspired optimization. Computers & Security, 124, 102957.

Asif, M., Abbas, S., Khan, M.A., Fatima, A., Khan, M.A., & Lee, S.-W. (2022). MapReduce based intelligent model for intrusion detection using machine learning technique. Journal of King Saud University – Computer and Information Sciences, 34, 9723–9731.

Avci, O.; Abdeljaber, O.; Kiranyaz, S.; Hussein, M.; Gabbouj, M.; Inman, D.J. (2021). A review of vibration-based damage detection in civil structures: From traditional methods to Machine Learning and Deep Learning applications. *Mech. Syst. Signal Process.* **2021**, *147*, 107077.

Bui, H.-K., Lin, Y.-D., Hwang, R.-H., Lin, P.-C., Nguyen, V.-L., & Lai, Y.-C. (2021). CREME: A toolchain of automatic dataset collection for machine learning in intrusion detection. Journal of Network and Computer Applications, 193, 103212.

Cevallos M., J. F., Rizzardi, A., Sicari, S., & Porisini, A. C. (2023). Deep Reinforcement Learning for intrusion detection in Internet of Things: Best practices, lessons learnt, and open challenges. Computer Networks, 236, 110016.

Ciresan, Dan; Meier, Ueli; Schmidhuber, Jürgen (June 2012). "Multi-column deep neural networks for image classification". *2012 IEEE Conference on Computer Vision and Pattern Recognition*. New York, NY: Institute of Electrical and Electronics Engineers (IEEE). pp. 3642-3649. arXiv:1202.2745. CiteSeerX 10.1.1.300.3283. doi:10.1109/CVPR.2012.6248110. ISBN 97 8-1-4673-1226-4. OCLC 812295155. S2CID 2161592.

Devendiran, R., & Turukmane, A. V. (2024). Dugat-LSTM: Deep learning based network intrusion detection system using chaotic optimization strategy. Expert Systems with Applications, 245, 123027.

Dina, A. S., & Manivannan, D. (2021). Intrusion detection based on. Machine Learning techniques in computer networks. Internet of Things, 16, 100462. doi:10.1016/j.iot.2021.100462.

Kalimuthan, C., & Arokia Renjit, J. (2020). Review on intrusion detection using feature selection with machine learning techniques. Materials Today: Proceedings, 33, 3794–3802. https://doi.org/10.1016/j.matpr.2020.06.650.

Katiravan, J., Na, D., Mc, S. P. D., & Ad, S. S. V. (2023). Intrusion Detection in Novel WSN-Leach Dos Attack Dataset using Machine Learning based Boosting Algorithms. Procedia Computer Science, 230, 90–99. https://doi.org/10.1016/j.procs.2023.12.064.

Khalil, A., Farman, H., Nasralla, M. M., Jan, B., & Ahmad, J. (2023). "Artificial Intelligence-based intrusion detection system for V2V communication in vehicular adhoc networks." Ain Shams Engineering Journal. Available online at www.sciencedirect.com.

Fukushima, Kunihiko (1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position" (PDF). *Biological Cybernetics*. **36** (4): 193–202. doi:10.1007/BF00344251. PMID 7370364. S2CID 206775608. Archived (PDF) from the original on 3 June 2014. Retrieved 16 November 2013.Khraisat, A., Gondal, I., Vamplew, P. (2018).

Hossain, M. A., & Islam, M. S. (2023). Ensuring network security with a robust intrusion detection system using ensemble-based machine learning. Array, 19, 100306. https://doi.org/10.1016/j.array.2023.100306.

Jadhav, A.D., Pellakuri, V. Highly accurate and efficient two phase-intrusion detection system (TP-IDS) using distributed processing of HADOOP and machine learning techniques. *J Big Data* **8**, 131 (2021). https://doi.org/10.1186/s40537-021-00521-y.

Jayaraj, R., Pushpalatha, A., Sangeetha, K., Kamaleshwar, T., Udhaya Shree, S., & Damodaran, D. (2024). Intrusion detection based on phishing detection with machine learning. Measurement: Sensors, 31, 101003. https://doi.org/10.1016/j.measure.2024.101003.

Joachims, T. (2002). "Learning to classify text using support vector machines", Kluwer Academic Publishers, (2002).

Khraisat, A., Gondal, I., Vamplew, P. (2018). An Anomaly Intrusion Detection System Using C5 Decision Tree Classifier. In: Trends and Applications in Knowledge Discovery and Data Mining,Cham. *Springer International Publishing*, pp 149–155.

Khraisat, A., Gondal, I., Vamplew, P. (2019). Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecur* **2**, 20 (2019). https://doi.org/10.1186/s42400-019-0038-7.

Kumar, K. P. S., Nair, S. A. H., Guha Roy, D., Rajalingam, B., & Kumar, R. S. (2021). Security and privacy-aware Artificial Intrusion Detection System using Federated Machine Learning. Computers and Electrical Engineering, 96, 107440.

LeCun, Y., Bottou, L., Yoshua, B., & Patrick, H. (1998). GradientBased Learning Applied to Document Recognition. P*roc of the IEEE.*

Liao H.J., Lin C.H.R., Lin Y.C., Tung K.Y. (2013). Intrusion detection system: a comprehensive review. J Netw Comput Appl 36(1):16–24.

Lu, Y., Chai, S., Suo, Y., Yao, F., & Zhang, C. (2024). Intrusion detection for Industrial Internet of Things based on deep learning. Neurocomputing, 564, 126886.

Lv, L., Wang, W., Zhang, Z., & Liu, X. (2020). A novel intrusion detection system based on an optimal hybrid kernel extreme learning machine. Knowledge-Based Systems, 195, 105648.

M.N. Chowdhury, K. Ferens, M. Ferens, Network intrusion detection using machine learning, in: Proceedings of International Conference on Security Management, SAM, Las Vegas, USA, 2016, pp. 1–7.

Nabi, F., & Zhou, X. (2024). Enhancing intrusion detection systems through dimensionality reduction: A comparative study of machine learning techniques for cyber security. Cyber Security and Applications, 2, 100033. doi:10.1016/j.csca.2024.100033.

Nie, F., Liu, W., Liu, G., & Gao, B. (2024). M2VT-IDS: A multi-task multi-view learning architecture for designing IoT intrusion detection system. Internet of Things, 25, 101102. https://doi.org/10.1016/j.iot.2024.101102.

Parameswari, A., Ganeshan, R., Ragavi, V., & Shereesha, M. (2024). Hybrid rat swarm hunter-prey optimization trained deep learning for network intrusion detection using CNN features. Computers & Security, 139, 103656.

Paya, A., Arroni, S., García-Díaz, V., & Gómez, A. (2024). Apollon: A robust defense system against Adversarial Machine Learning attacks in Intrusion Detection Systems. Computers & Security, 136, 103546.

Rahib H. Abiyev, Abdullahi Ismail, "COVID-19 and Pneumonia Diagnosis in X-Ray Images Using Convolutional Neural Networks", *Mathematical Problems in*

Engineering, vol. 2021, Article
ID 3281135, 14 pages, 2021. https://doi.org/10.1155/2021/3281135.

Rahib Abiyev ,Murat Arslan ,John Bush Idoko ,Boran Sekeroglu and Ahmet Ilhan. Identification of Epileptic EEG Signals Using Convolutional Neural Networks. *Appl. Sci. 10*(12), 2020, 4089; https://doi.org/10.3390/app10124089.

Rahib H. Abiyev. Murat Arslan. Head mouse control system for people with disabilities. Expert Systems, 37, 2020, https://doi.org/10.1111/exsy.12398.

Rajasekaran, K. (2020). Classification and Importance of Intrusion Detection System. *International Journal of Computer Science and Information Security*. 10. 44.

Rahib H. Abiyev, John Bush Idoko, Murat Arslan. Sign Language Translation Using Deep Convolutional Neural Networks. KSII Transactions on Internet and Information Systems, Vol.14, No.2, pp.631-653, 2020. https://doi.org/10.3837/tiis.2020.02.009.

Rahib Abiyev, Joseph Adepoju. Automatic Food Recognition Using Deep Convolutional Neural Networks with Self-attention Mechanism Human-Centric Intelligent Systems.2024, https://doi.org/10.1007/s44230-023-00057-9.

Sami, E. (2012). "Support Vector Machines for classification and locating faults on transmission lines", Applied Soft Computing, vol. 12, (2012), pp. 1650–1658.

Sarhan, M., Layegh, S., Moustafa, N., Gallagher, M., & Portmann, M. (2021). Feature extraction for machine learning-based intrusion detection in IoT networks. Digital Communications and Networks (DCN).

Sivanandam, S. N., Sumathi, S. and Deepa, S. N. (2006). "Introduction to Neural Networks using MATLAB 6.0", Tata McGraw Hill Education Pvt. Ltd., (2006).

Sun, Z., An, G., Yang, Y., & Liu, Y. (2024). Optimized machine learning enabled intrusion detection system for internet of medical things. Franklin Open, 6, 100056. doi:10.1016/j.fraope.2024.100056

Talukder, M. A., Hasan, K. F., Islam, M. M., Uddin, M. A., Akhter, A., Yousuf, M. A., Alharbi, F., & Moni, M. A. (2023). A dependable hybrid machine learning model for network intrusion detection. Journal of Information Security and Applications.

Turukmane, A. V., & Devendiran, R. (2024). M-MultiSVM: An efficient feature selection assisted network intrusion detection system using machine learning. Computers & Security, 137, 103587.

Umer, M. A., Junejo, K. N., Jilani, M. T., & Mathur, A. P. (2022). Machine learning for intrusion detection in industrial control systems: Applications, challenges, and recommendations. International Journal of Critical Infrastructure Protection, 38, 100516.

Yamaguchi, Kouichi; Sakamoto, Kenji; Akabane, Toshio; Fujimoto, Yoshiji (November 1990). *A Neural Network for Speaker-Independent Isolated Word Recognition*. First International Conference on Spoken Language Processing (ICSLP 90). Kobe, Japan. Archived from the original on 2021-03-07. Retrieved 2019-09-04.

Yang, L., & Shami, A. (2022). IDS-ML: An open source code for Intrusion Detection System development using Machine Learning. Software Impacts, 14, 100446. doi:10.1016/j.simpa.2022.100446.

Yuan, X., Han, S., Huang, W., Ye, H., Kong, X., & Zhang, F. (2024). A simple framework to enhance the adversarial robustness of deep learning-based intrusion detection system. Computers & Security, 137, 103644.

Zhang, C., Jia, D., Wang, L., Wang, W., Liu, F., & Yang, A. (2022). Comparative research on network intrusion detection methods based on machine learning. Computers & Security, 121, 102861. https://doi.org/10.1016/j.cose.2022.102861

APPENDICES

Appendix 1

IMPORTING THE DEPENDENCIES

```
!pip install numpy pandas scikit-learn matplotlib tensorflow

!pip install seaborn

!pip install joblib

!pip install pdfkit

!pip install tabulate

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to_categorical

from keras.optimizers import Adam

from sklearn.preprocessing import label_binarize

%matplotlib inline

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
import joblib
import os
from sklearn.multiclass import OneVsRestClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve, roc_auc_score, confusion_matrix
```

```
from sklearn.metrics import roc_curve, roc_auc_score

from sklearn.metrics import precision_recall_curve, roc_curve, roc_auc_score

from sklearn.preprocessing import LabelBinarizer

from sklearn.metrics import precision_recall_curve, average_precision_score

from sklearn.metrics import roc_auc_score

import tkinter as tk

from tkinter import simpledialog

import pdfkit

from keras.models import load_model

import numpy as np

import joblib

from sklearn.metrics import precision_recall_curve, auc

from sklearn.metrics import precision_recall_curve, auc

from sklearn.metrics import roc_curve, auc
```

DATA COLLECTION AND PREPROCESSING

```
#LOAD THE DATASET

#loading the csv file into a panda dataframe

log2_data = pd.read_csv('/content/drive/MyDrive/FirewallData.csv')

# first 5 rows of the dataframe

log2_data.head()

# number of rows & columns

log2_data.shape

log2_data.describe()
```

SEPARATING FEATURES AND TARGET

```
# Encode the target labels

label_encoder = LabelEncoder()

log2_data['Action_encoded'] = label_encoder.fit_transform(log2_data['Action'])

# Assuming 'Action' is your target column

# Features and labels

X = log2_data.drop(['Action', 'Action_encoded'], axis=1)

y = log2_data['Action_encoded']

print(X)

print(y)

sns.countplot(x='Action', data=log2_data)

plt.xlabel('Action')  # Optional: Set the x-axis label

plt.ylabel('Count')  # Optional: Set the y-axis label

plt.title('Count of Actions')  # Optional: Set the plot title

plt.show()
```

SPLITTING THE DATASETS INTO TRAINING DATA AND TESTING DATA

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=30, stratify=y)

# Split the training set into training and validation sets

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Perform one-hot encoding on the labels

y_train_one_hot = to_categorical(y_train)

y_test_one_hot = to_categorical(y_test)
```

```
# Assuming y_test_bin is your true labels (one-hot encoded) for CNN

y_test_bin = to_categorical(log2_data['Action_encoded'])
```

ENCODE THE TARGET LABELS

```
print(X.shape, X_train.shape, X_test.shape)
```

DATA STANDARDIZATION

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

scaler = StandardScaler()

standardized_data = scaler.fit_transform(X)

print(standardized_data)

X = standardized_data

print(X)

print(y)
```

BUILD THE SVM MODEL

```
# Build and train SVM model

svm_model = SVC(probability=True)

calibrated_model = CalibratedClassifierCV(svm_model, method='sigmoid',
cv='prefit')
```

FIT THE SVM MODEL

```
# Fit the SVM model

svm_model.fit(X_train, y_train)

# Fit the calibrated model

calibrated_model.fit(X_train, y_train)
```

CALIBRATE THE PROBABILITIES OF THE SVM MODEL

```
print(X_test.shape, y_test_bin.shape)
```

EVALUATE MODELS ON THE TEST DATA

```
# Predictions using predict_proba

svm_probabilities = svm_model.predict_proba(X_test)

print("SVM Probabilities shape:", svm_probabilities.shape)

# Predictions using predict

svm_predicted_labels = svm_model.predict(X_test)

print("SVM Predicted Labels shape:", svm_predicted_labels.shape)

# Calculate accuracy

svm_accuracy = accuracy_score(y_test, svm_predicted_labels)

print("SVM Accuracy:", svm_accuracy)
```

EVALUATE THE SVM ON THE TEST DATA

```
# Calculate evaluation metrics

svm_accuracy = accuracy_score(y_test, svm_predicted_labels)

svm_precision = precision_score(y_test, svm_predicted_labels, average='weighted')

svm_recall = recall_score(y_test, svm_predicted_labels, average='weighted')

svm_f1_score = f1_score(y_test, svm_predicted_labels, average='weighted')

# Calculate ROC AUC for multi-class using the one-vs-rest approach

svm_roc_auc = roc_auc_score(y_test, svm_probabilities, multi_class='ovr')
```

PRINT EVALUATION METRICS FOR THE SVM

```
# Print evaluation metrics for the SVM model

print("SVM Model Evaluation:")

print("Accuracy:", svm_accuracy)

print("Precision:", svm_precision)

print("Recall:", svm_recall)
```

```
print("F1-score:", svm_f1_score)

print("ROC AUC:", svm_roc_auc)

# Calculate the decision scores for the ROC curve (using decision_function)

svm_decision_scores = svm_model.decision_function(X_test_scaled)
```

PLOT THE CONFUSION MATRIX FOR THE SVM MODEL

```
# Get unique class labels

labels = log2_data['Action_encoded'].unique()

# Confusion matrix for SVM

cm_svm = confusion_matrix(y_test, svm_predicted_labels)

plt.figure(figsize=(8, 6))

sns.heatmap(cm_svm, annot=True, fmt='g', cmap='Blues', xticklabels=labels,
yticklabels=labels)

plt.title('Confusion Matrix - SVM')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

from sklearn.model_selection import learning_curve
```

SAVING THE SVM MODEL

```
svm_model_path = '/content/drive/MyDrive/log2.csv'

joblib.dump(svm_model, svm_model_path)
```

PRECISION RECALL CURVE FOR SVM

```
# Assuming svm_probabilities are the predicted probabilities for each class

precision_svm, recall_svm, _ = precision_recall_curve(y_test_one_hot.ravel(),
svm_probabilities.ravel())

auc_svm = auc(recall_svm, precision_svm)
```

```
plt.figure(figsize=(8, 6))
plt.plot(recall_svm, precision_svm, color='darkorange', lw=2, label=f'SVM (AUC = {auc_svm:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - SVM')
plt.legend(loc='lower left')
plt.show()
n_classes = 4
# Assuming y_test is your true labels, and y_score is the decision function output of your SVM
# If you have a multi-class problem, make sure to binarize the labels
y_test_bin = label_binarize(y_test, classes=[0, 1, 2, 3]) # Adjust classes based on your problem
# Assuming svm_predicted_scores is the decision function output of your SVM
svm_predicted_scores = svm_model.decision_function(X_test)
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
# For each class (assuming a multi-class problem)
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], svm_predicted_scores[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), svm_predicted_scores.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
# Plot the ROC curve
plt.figure(figsize=(10, 6))
# Plot individual class curves
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')
# Plot micro-average curve
plt.plot(fpr["micro"], tpr["micro"], label=f'Micro-average (AUC = {roc_auc["micro"]:.2f})', linestyle='--', linewidth=2)



# Plot random guessing line
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Guessing')
# Customize the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve for SVM')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

BUILDING THE CNN MODEL

```
from keras import layers, models

# Assuming you have reshaped your input data for the CNN model

X_train_reshaped = np.array(X_train).reshape(X_train.shape[0], X_train.shape[1], 1)

X_val_reshaped = np.array(X_val).reshape(X_val.shape[0], X_val.shape[1], 1)

X_test_reshaped = np.array(X_test).reshape(X_test.shape[0], X_test.shape[1], 1)

# Convert labels to one-hot encoding

from keras.utils import to_categorical

y_train_one_hot = to_categorical(y_train)

y_val_one_hot = to_categorical(y_val)

y_test_one_hot = to_categorical(y_test)
```

 # Build your CNN model

```
model = Sequential()

model.add(Conv1D(filters=32, kernel_size=3, activation='ReLU',
input_shape=(X_train.shape[1], 1)))

model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())

model.add(Dense(128, activation='ReLU'))

model.add(Dense(y_train_one_hot.shape[1], activation='softmax'))
```

TRAIN THE CNN MODEL

```
# Train the CNN model

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

```
model.fit(X_train_reshaped, y_train_one_hot, epochs=10, batch_size=64, validation_data=(X_test_reshaped, y_test_one_hot))

# Train the CNN model with validation data
history = model.fit(X_train_reshaped, y_train_one_hot, epochs=500, batch_size=64, validation_data=(X_val_reshaped, y_val_one_hot))

# Plot Training and Validation Accuracy
plt.figure(figsize=(12, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Model Learning Curve - Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot Training and Validation Loss
plt.figure(figsize=(12, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('CNN Model Learning Curve - Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

from sklearn.model_selection import GridSearchCV
```

```
# Assuming X_train_scaled and y_train are your training data
# Set up the parameter grid for GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100],
              'kernel': ['linear', 'rbf', 'poly', 'sigmoid']}
# Create GridSearchCV
grid_search = GridSearchCV(svm_model, param_grid, cv=5, scoring='accuracy',
verbose=1, n_jobs=-1)
# Fit the model
grid_search.fit(X_train_scaled, y_train)

# Get the results
results = grid_search.cv_results_
best_params = grid_search.best_params_
# Plot the performance
plt.figure(figsize=(10, 6))
plt.plot(param_grid['C'], results['mean_test_score'], marker='o')
plt.xscale('log')
plt.xlabel('C (Regularization parameter)')
plt.ylabel('Mean cross-validated accuracy')
plt.title('SVM Performance with different C values')
plt.show()
print(X_test.shape, y_test_bin.shape)
```

```
# Get predictions from the CNN model
y_pred_one_hot = model.predict(X_test_reshaped)
y_pred_labels_cnn = np.argmax(y_pred_one_hot, axis=1)
# Check unique values in y_train
print("Unique values in y_train:", np.unique(y_train))
# Accuracy
cnn_accuracy = accuracy_score(y_test, y_pred_labels_cnn)
print("\nCNN Accuracy:", cnn_accuracy)
# Precision, Recall, F1 Score
cnn_precision = precision_score(y_test, y_pred_labels_cnn, average='weighted')
cnn_recall = recall_score(y_test, y_pred_labels_cnn, average='weighted')
cnn_f1 = f1_score(y_test, y_pred_labels_cnn, average='weighted')
cnn_precision = precision_score(y_test, y_pred_labels_cnn, average='weighted')
print("CNN Precision:", cnn_precision)
# Calculate CNN recall
cnn_recall = recall_score(y_test, y_pred_labels_cnn, average='weighted')
print("CNN Recall:", cnn_recall)
# Calculate CNN F1 score
cnn_f1 = f1_score(y_test, y_pred_labels_cnn, average='weighted')
print("CNN F1 Score:", cnn_f1)
# Assuming y_test is your true labels, and y_pred_one_hot is the predicted
probabilities from your CNN
# If you have a multi-class problem, make sure to binarize the labels
```

```
y_test_bin = label_binarize(y_test, classes=[0, 1, 2, 3]) # Adjust classes based on your problem
# Assuming y_pred_one_hot is the predicted probabilities from your CNN
y_pred_one_hot = cnn_model.predict(X_test_reshaped)
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
# For each class (assuming a multi-class problem)
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_one_hot[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_pred_one_hot.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
# Plot the ROC curve
plt.figure(figsize=(10, 6))
# Plot individual class curves
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')
# Plot micro-average curve
plt.plot(fpr["micro"], tpr["micro"], label=f'Micro-average (AUC = {roc_auc["micro"]:.2f})', linestyle='--', linewidth=2)
# Plot random guessing line
```

```
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Guessing')

# Customize the plot

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC-AUC Curve for CNN')

plt.legend(loc='lower right')

plt.grid(True)

plt.show()
```

ROC CURVE OF CNN

```
# Assuming y_pred_probs is the predicted probabilities for each class

# Assuming you have predictions from your model stored in y_pred_probs

y_pred_probs = model.predict(X_test_reshaped)

fpr_cnn, tpr_cnn, _ = roc_curve(y_test_one_hot.ravel(), y_pred_probs.ravel())

roc_auc_cnn = auc(fpr_cnn, tpr_cnn)

plt.figure(figsize=(8, 6))

plt.plot(fpr_cnn, tpr_cnn, color='darkblue', lw=2, label=f'CNN (AUC = {roc_auc_cnn:.2f})')

plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')  # Random classifier

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve - CNN')

plt.legend(loc='lower right')

plt.show()
```

PRECISION RECALL CURVE CNN

```
# Assuming X_train.shape[1] is the number of features (12 in your case)

input_shape = (X_train.shape[1], 1)

# Assuming y_test_one_hot is the true labels in one-hot encoded format

precision_cnn, recall_cnn, _ = precision_recall_curve(y_test_one_hot.ravel(), y_pred_probs.ravel())

auc_cnn = auc(recall_cnn, precision_cnn)

plt.figure(figsize=(8, 6))

plt.plot(recall_cnn, precision_cnn, color='darkorange', lw=2, label=f'CNN (AUC = {auc_cnn:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve - CNN')

plt.legend(loc='upper right')

plt.show()

# Assuming y_test is your true labels, and y_pred_one_hot is the predicted probabilities from your CNN

# If you have a multi-class problem, make sure to binarize the labels

y_test_bin = label_binarize(y_test, classes=[0, 1, 2, 3]) # Adjust classes based on your problem

# Assuming y_pred_one_hot is the predicted probabilities from your CNN

y_pred_one_hot = cnn_model.predict(X_test_reshaped)

# Compute ROC curve and ROC area for each class

fpr = dict()

tpr = dict()
```

```
roc_auc = dict()
# For each class (assuming a multi-class problem)
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_one_hot[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_pred_one_hot.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
# Plot the ROC curve
plt.figure(figsize=(10, 6))

# Plot individual class curves
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')
# Plot micro-average curve
plt.plot(fpr["micro"], tpr["micro"], label=f'Micro-average (AUC = {roc_auc["micro"]:.2f})', linestyle='--', linewidth=2)
# Plot random guessing line
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Guessing')
# Customize the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve for CNN')
```

```
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
# Save the CNN model
model.save('/content/drive/MyDrive/cnn_model.h5')
# Learning curve for SVM
train_sizes, train_scores_svm, test_scores_svm = learning_curve(
    svm_model, X, y, cv=5, scoring='accuracy', n_jobs=-1)
from sklearn.model_selection import learning_curve
# Learning curve for SVM
train_sizes, train_scores_svm, test_scores_svm = learning_curve(
svm_model, X, y, cv=5, scoring='accuracy', n_jobs=-1)
# Calculate mean and standard deviation of training and test scores
train_mean_svm = np.mean(train_scores_svm, axis=1)
train_std_svm = np.std(train_scores_svm, axis=1)
test_mean_svm = np.mean(test_scores_svm, axis=1)
test_std_svm = np.std(test_scores_svm, axis=1)
# Plot learning curve for SVM
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean_svm, label='Training Score', color='blue')
plt.fill_between(train_sizes, train_mean_svm - train_std_svm, train_mean_svm +
train_std_svm, color='blue', alpha=0.2)
plt.plot(train_sizes, test_mean_svm, label='Cross-Validation Score', color='green')
```

```
plt.fill_between(train_sizes, test_mean_svm - test_std_svm, test_mean_svm +
test_std_svm, color='green', alpha=0.2)

plt.title('SVM Model Learning Curve')

plt.xlabel('Training Size')

plt.ylabel('Accuracy Score')

plt.legend()

plt.show()

# Number of classes

num_classes = 4

# Calculate mean and standard deviation of training and test scores

train_mean_svm = np.mean(train_scores_svm, axis=1)

train_std_svm = np.std(train_scores_svm, axis=1)

test_mean_svm = np.mean(test_scores_svm, axis=1)

test_std_svm = np.std(test_scores_svm, axis=1)

# Plot learning curve for SVM

plt.figure(figsize=(10, 6))

plt.plot(train_sizes, train_mean_svm, label='Training Score', color='blue')

plt.fill_between(train_sizes, train_mean_svm - train_std_svm, train_mean_svm +
train_std_svm, color='blue', alpha=0.2)

plt.plot(train_sizes, test_mean_svm, label='Cross-Validation Score', color='green')

plt.fill_between(train_sizes, test_mean_svm - test_std_svm, test_mean_svm +
test_std_svm, color='green', alpha=0.2)

plt.title('SVM Model Learning Curve')

plt.xlabel('Training Size')
```

```
plt.ylabel('Accuracy Score')
plt.legend()
plt.show()
# Number of classes
num_classes = 4
# Convert your y_train and y_test to one-hot encoding
y_train_one_hot = to_categorical(y_train, num_classes=num_classes)
y_test_one_hot = to_categorical(y_test, num_classes=num_classes)
epochs = 200
from keras.utils import to_categorical
# Convert your y_train and y_test to one-hot encoding
y_train_one_hot = to_categorical(y_train, num_classes=num_classes)
y_test_one_hot = to_categorical(y_test, num_classes=num_classes)
# Learning history for CNN
history = model.fit(X_train, y_train, epochs=10, batch_size=64,
validation_data=(X_test, y_test))
# Plot learning history for CNN
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Model Training Accuracy')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')

plt.legend()

plt.tight_layout()

plt.show()

plt.tight_layout()

plt.show()
```

.

Appendix 2

COMPARATION OF THE MODELS

```
from scipy.stats import ttest_rel

# Statistical analysis (t-tests for example)

from scipy import stats

# Print evaluation metrics for the SVM model

print("SVM Model Evaluation:")

print("Accuracy:", svm_accuracy)

print("Precision:", svm_precision)

print("Recall:", svm_recall)

print("F1-score:", svm_f1_score)
```

```
print("ROC AUC:", svm_roc_auc)

# Actual accuracy, precision, recall, F1-score, and ROC AUC values for CNN and SVM

cnn_accuracy = 0.9906000122077764

cnn_precision = 0.9899951702164878

cnn_recall = 0.9906000122077764

cnn_f1 = 0.9901868458560885

cnn_roc_auc = 0.8774261119352021

svm_accuracy = 0.739486052615516

svm_precision =0.6199914926592527

svm_recall = 0.739486052615516

svm_f1 = 0.6600092450578181

svm_roc_auc = 0.9243266246663926

# Perform a t-test for accuracy

accuracy_stat, accuracy_p_value = stats.ttest_rel(cnn_accuracy, svm_accuracy)

# Print the results

print("T-test for Accuracy:")

print("t-statistic:", accuracy_stat)

print("p-value:", accuracy_p_value)

# Check the p-value to determine if the difference is statistically significant

alpha = 0.05  # Set your significance level

if accuracy_p_value < alpha:

    print("The difference in accuracy is statistically significant.")
```

```
else:
    print("There is no significant difference in accuracy between the models.")
# Print the results
print("CNN Accuracy:", cnn_accuracy)
print("SVM Accuracy:", svm_accuracy)
print("Accuracy p-value:", accuracy_p_value)
# Print the evaluation metrics
print("CNN Accuracy:", cnn_accuracy)
print("CNN Precision:", cnn_precision)
print("CNN Recall:", cnn_recall)
print("CNN F1-score:", cnn_f1)

# Print the ROC AUC score
print("CNN ROC AUC Score:", cnn_roc_auc)
# Print the evaluation metrics
print("CNN Accuracy:", cnn_accuracy)
print("CNN Precision:", cnn_precision)
print("CNN Recall:", cnn_recall)
print("CNN F1-score:", cnn_f1)
# Print evaluation metrics for the SVM model
print("SVM Model Evaluation:")
print("Accuracy:", svm_accuracy)
print("Precision:", svm_precision)
```

```
print("Recall:", svm_recall)

print("F1-score:", svm_f1_score)

print("ROC AUC:", svm_roc_auc)

# Metrics and models

metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score', 'ROC AUC']

models = ['CNN', 'SVM']

# Values for CNN and SVM

cnn_metrics = [cnn_accuracy, cnn_precision, cnn_recall, cnn_f1, cnn_roc_auc]

svm_metrics = [svm_accuracy, svm_precision, svm_recall, svm_f1, svm_roc_auc]

# Create an index for each metric

x = np.arange(len(metrics))


# Define the width of the bars

width = 0.35
```

BAR PLOT

```
# Print the evaluation metrics

print("CNN Accuracy:", cnn_accuracy)

print("CNN Precision:", cnn_precision)

print("CNN Recall:", cnn_recall)

print("CNN F1-score:", cnn_f1)

# Print the ROC AUC score

print("CNN ROC AUC Score:", cnn_roc_auc)

# Calculate the differences

accuracy_diff = cnn_accuracy - svm_accuracy
```

```
precision_diff = cnn_precision - svm_precision

recall_diff = cnn_recall - svm_recall

f1_diff = cnn_f1 - svm_f1

roc_auc_diff = cnn_roc_auc - svm_roc_auc

# Actual accuracy, precision, recall, F1-score, and ROC AUC values for CNN and SVM

svm_accuracy = 0.739486052615516

svm_precision =0.6199914926592527

svm_recall = 0.739486052615516

svm_f1 = 0.6600092450578181

svm_roc_auc = 0.9243266246663926

cnn_accuracy = 0.9906000122077764

cnn_precision = 0.9899951702164878

cnn_recall = 0.9906000122077764

cnn_f1 = 0.9901868458560885

cnn_roc_auc = 0.8774261119352021

#Assuming you have computed performance metrics for two models

metrics_svm = {'Accuracy': 0.739486052615516, 'Precision': 0.6199914926592527, 'Recall': 0.739486052615516, 'F1 Score': 0.6600092450578181}

metrics_cnn = {'Accuracy': 0.9906000122077764, 'Precision': 0.9899951702164878, 'Recall': 0.9906000122077764, 'F1 Score': 0.9901868458560885}

# Extract metric names and values

metric_names = list(metrics_svm.keys())

values_svm = list(metrics_svm.values())
```

```
values_cnn = list(metrics_cnn.values())

metrics = ['Metric1', 'Metric2', 'Metric3', 'Metric4'] # Replace with your actual metrics

assert len(metrics) == len(values_svm) == len(values_cnn), "Length mismatch in metrics arrays"

# Set up bar positions

bar_width = 0.35

ind = np.arange(len(metric_names))

# Set the width of the bars

width = 0.35


# Set the x locations for the groups

ind = np.arange(len(metric_names))

# Plot the bars

fig, ax = plt.subplots(figsize=(10, 6))

bar1 = ax.bar(ind - width/2, values_svm, width, label='SVM')

bar2 = ax.bar(ind + width/2, values_cnn, width, label='CNN')

# Add some text for labels, title and custom x-axis tick labels, etc.

ax.set_xlabel('Metrics')

ax.set_ylabel('Values')

ax.set_title('Model Comparison (SVM vs CNN)')

ax.set_xticks(ind)

ax.set_xticklabels(metric_names)

ax.legend()
```

```
# Display the values on top of the bars
for bar in bar1:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 6), ha='center', va='bottom')
for bar in bar2:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 6), ha='center', va='bottom')
# Set the y-axis scale
plt.yticks(np.arange(0, 1.0, 0.05))
# Show the plot
plt.show()
```

RADAR PLOT

```
# Number of metrics
num_metrics = len(metric_names)
# Set up angles for the radar chart
angles = np.linspace(0, 2 * np.pi, num_metrics, endpoint=False)
# Make the plot circular
values_svm += values_svm[:1]
values_cnn += values_cnn[:1]
angles = np.concatenate((angles, [angles[0]]))
# Plot the SVM values
plt.polar(angles, values_svm, marker='o', label='SVM')
```

```
# Plot the CNN values
plt.polar(angles, values_cnn, marker='o', label='CNN')
# Fill the area between the lines
plt.fill(angles, values_svm, alpha=0.25)
plt.fill(angles, values_cnn, alpha=0.25)
# Add labels, title, and legend
plt.thetagrids(angles[:-1] * 180/np.pi, metric_names)
plt.title('Model Comparison (SVM vs CNN)')
plt.legend()
# Show the plot
plt.show()
# Assuming you have reshaped your input data for the CNN model
X_train_reshaped = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_val_reshaped = X_val.reshape(X_val.shape[0], X_val.shape[1], 1)
X_test_reshaped = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
# Load the CNN model
loaded_model = load_model('/content/drive/MyDrive/cnn_model.h5')
# Load the SVM model
svm_model_path = '/content/drive/MyDrive/log2.csv'
svm_model = joblib.load(svm_model_path)
# Reshape the input data to 2D
#X_test_reshaped_2d = X_test_reshaped.reshape(X_test_reshaped.shape[0], -1)
# Get predicted labels and probabilities for SVM
```

```
#y_pred_svm = svm_model.predict(X_test_reshaped_2d)

#y_pred_probs_svm = svm_model.predict_proba(X_test_reshaped_2d)
```

CONFUSION MATRIX

```
from sklearn.metrics import confusion_matrix, classification_report

# If y_test is one-hot encoded, convert it back to labels

if len(y_test.shape) > 1:

    y_test_labels = np.argmax(y_test, axis=1)

else:

    y_test_labels = y_test


svm_pred_labels = svm_model.predict(X_test)

cnn_pred_one_hot = model.predict(X_test_reshaped)

cnn_pred_labels = np.argmax(cnn_pred_one_hot, axis=1)

# Get the confusion matrixes

cm_svm = confusion_matrix(y_test_labels, svm_pred_labels)

cm_cnn = confusion_matrix(y_test_labels, cnn_pred_labels)

# Plot the confusion matrices side by side

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# SVM Confusion Matrix

sns.heatmap(cm_svm, annot=True, fmt="d", cmap="Blues", cbar=False, ax=axes[0])

axes[0].set_title('SVM Confusion Matrix')

axes[0].set_xlabel('Predicted')

axes[0].set_ylabel('Actual')

# CNN Confusion Matrix
```

```
sns.heatmap(cm_cnn, annot=True, fmt="d", cmap="Blues", cbar=False, ax=axes[1])

axes[1].set_title('CNN Confusion Matrix')

axes[1].set_xlabel('Predicted')

axes[1].set_ylabel('Actual')

plt.show()

# Assuming y_test_bin is your true labels (one-hot encoded) for CNN

y_test_bin = to_categorical(log2_data['Action_encoded'])


# Assuming svm_model and model are your trained SVM and CNN models

svm_probabilities = svm_model.predict_proba(X_test)

svm_predicted_labels = svm_model.predict(X_test)

from sklearn.model_selection import learning_curve
```

REAL TIME INTRUSION DETECTION CLASSIFICATION

```
# Load the CNN model

cnn_model_path = '/content/drive/MyDrive/cnn_model.h5'

loaded_model = load_model(cnn_model_path)

# Load the SVM model

svm_model_path = '/content/drive/MyDrive/log2.csv'

svm_model = joblib.load(svm_model_path)

# Load the CNN model

cnn_model = load_model('/content/drive/MyDrive/cnn_model.h5')
```

TAKING THE INPUTS DIRECTLY FROM THE DATASET/USER INPUT

```
# Features list (excluding 'Action')

features = ['Source Port', 'Destination Port', 'NAT Source Port', 'NAT Destination Port',
```

```
            'Bytes', 'Bytes Sent', 'Bytes Received', 'Packets', 'Elapsed Time (sec)',
            'pkts_sent', 'pkts_received']
# User input dictionary
user_input = {}
# Input values for each feature
for feature in features:
    value = float(input(f"{feature}: "))
    user_input[feature] = [value]

# Convert user input to a DataFrame
user_input_df = pd.DataFrame(user_input)
# Standardize user input for SVM
user_input_scaled = scaler.transform(user_input_df)  # Use the same scaler as before
# Classify action using the SVM model
svm_predicted_action = svm_model.predict(user_input_scaled)
svm_predicted_action = svm_predicted_action[0]
# Assuming you have reshaped your input data for the CNN model
user_input_reshaped = user_input_df.values.reshape(1, user_input_df.shape[1], 1)
# Classify action using the CNN model
cnn_predicted_action = cnn_model.predict(user_input_reshaped)
cnn_predicted_action =
label_encoder.inverse_transform([cnn_predicted_action.argmax()])[0]
# Hypothetical ground truth labels
```

```
y_true = 'Allow'   # Replace with the actual label for this example
# Mapping dictionary to convert predictions to class labels
class_mapping = {0: 'Allow', 1: 'Deny', 2: 'Drop', 3: 'Reset-Both'}
# Convert the predicted actions to class labels using the mapping dictionary
svm_predicted_action = class_mapping[svm_predicted_action]
# Convert both predicted actions to lowercase
svm_predicted_action_lower = str(svm_predicted_action).lower()
cnn_predicted_action_lower = str(cnn_predicted_action).lower()
# Calculate accuracy in percentage
svm_accuracy = 100 if svm_predicted_action_lower == y_true.lower() else 0
cnn_accuracy = 100 if cnn_predicted_action_lower == y_true.lower() else 0
# Print the results
print("SVM Predicted Action:", svm_predicted_action)
print("CNN Predicted Action:", cnn_predicted_action)
# Calculate accuracy in percentage
svm_accuracy = 100 if svm_predicted_action_lower == y_true.lower() else 0
cnn_accuracy = 100 if cnn_predicted_action_lower == y_true.lower() else 0
print("SVM Accuracy: {:.2f}%".format(svm_accuracy))
print("CNN Accuracy: {:.2f}%".format(cnn_accuracy))
# Features list (excluding 'Action')
features = ['Source Port', 'Destination Port', 'NAT Source Port', 'NAT Destination Port',
            'Bytes', 'Bytes Sent', 'Bytes Received', 'Packets', 'Elapsed Time (sec)',
            'pkts_sent', 'pkts_received']
```

```
# User input dictionary
user_input = {}
# Input values for each feature
for feature in features:
    value = float(input(f"{feature}: "))
    user_input[feature] = [value]

# Convert user input to a DataFrame
user_input_df = pd.DataFrame(user_input)
# Standardize user input for SVM
user_input_scaled = scaler.transform(user_input_df)  # Use the same scaler as before
# Classify action using the SVM model
svm_predicted_action = svm_model.predict(user_input_scaled)
svm_predicted_action = svm_predicted_action[0]
# Assuming you have reshaped your input data for the CNN model
user_input_reshaped = user_input_df.values.reshape(1, user_input_df.shape[1], 1)
# Classify action using the CNN model
cnn_predicted_action = cnn_model.predict(user_input_reshaped)
cnn_predicted_action =
label_encoder.inverse_transform([cnn_predicted_action.argmax()])[0]
# Hypothetical ground truth labels
y_true = 'drop' # Replace with the actual label for this example
# Mapping dictionary to convert predictions to class labels
```

```
class_mapping = {0: 'Allow', 1: 'Deny', 2: 'Drop', 3: 'Reset-Both'}
# Convert the predicted actions to class labels using the mapping dictionary
svm_predicted_action = class_mapping[svm_predicted_action]
# Convert both predicted actions to lowercase
svm_predicted_action_lower = str(svm_predicted_action).lower()
cnn_predicted_action_lower = str(cnn_predicted_action).lower()
# Calculate accuracy in percentage
svm_accuracy = 100 if svm_predicted_action_lower == y_true.lower() else 0
cnn_accuracy = 100 if cnn_predicted_action_lower == y_true.lower() else 0
# Print the results
print("SVM Predicted Action:", svm_predicted_action)
print("CNN Predicted Action:", cnn_predicted_action)
# Calculate accuracy in percentage
svm_accuracy = 100 if svm_predicted_action_lower == y_true.lower() else 0
cnn_accuracy = 100 if cnn_predicted_action_lower == y_true.lower() else 0
print("SVM Accuracy: {:.2f}%".format(svm_accuracy))
print("CNN Accuracy: {:.2f}%".format(cnn_accuracy)).
```

.

.

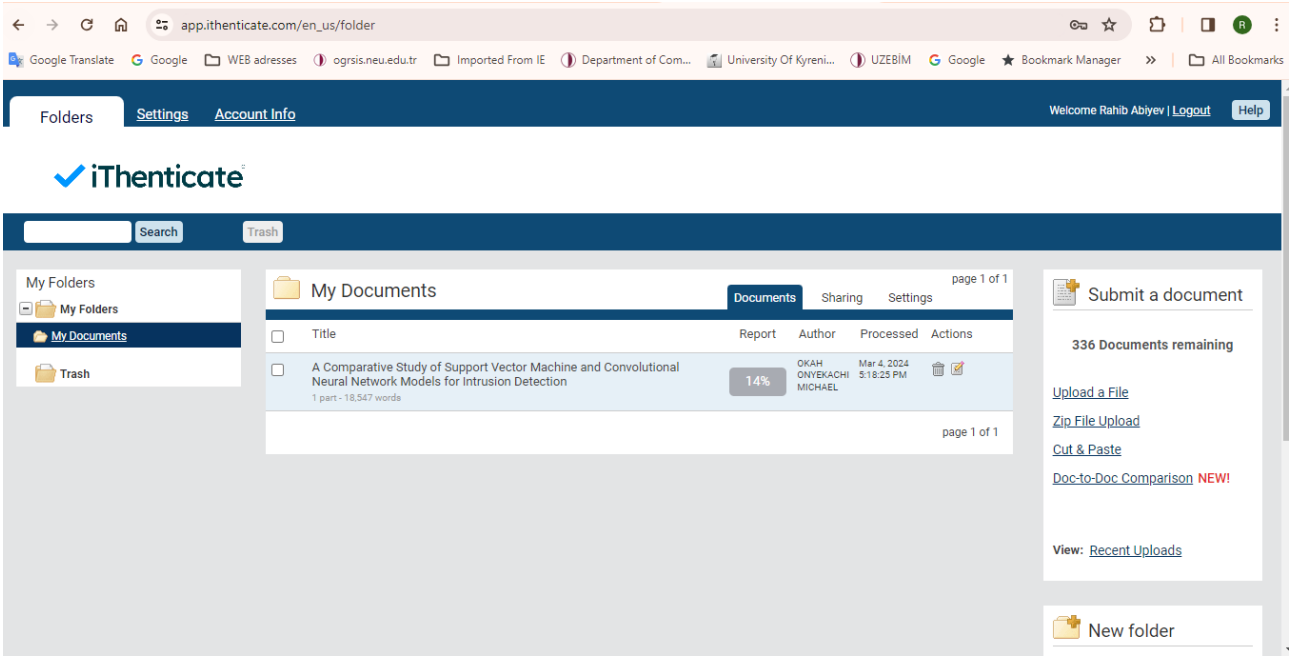
.

.

.

Appendix X

Similarity Report



Supervisor: Prof.Dr.Rahib Abiyev