

*A project report on*

# **Anomaly Detection in Network Traffic: Using Hybrid Techniques**

*Submitted in partial fulfillment for the award of the degree of*

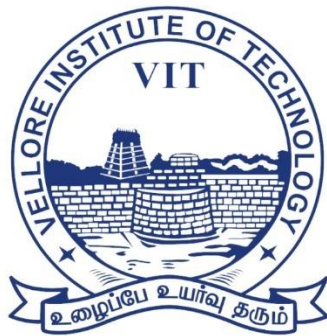
## **Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning**

*by*

**DOGIPARTHI AASRITH (21BAI1702)**

**KALLA BHARATH VARDHAN (21BCE5846)**

**VARANASI SAI CHARAN (21BCE5870)**



**VIT<sup>®</sup>**  
**CHENNAI**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

April, 2025

# **Anomaly Detection in Network Traffic: Using Hybrid Techniques**

*Submitted in partial fulfillment for the award of the degree of*

## **Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning**

*by*

**DOGIPARTHI AASRITH (21BAI1702)**

**KALLA BHARATH VARDHAN (21BCE5846)**

**VARANASI SAI CHARAN (21BCE5870)**



**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

April, 2025



**VIT<sup>®</sup>**  
**CHENNAI**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

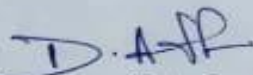
### DECLARATION

I hereby declare that the thesis entitled "Anomaly detection in Network traffic: Using Hybrid Techniques" submitted by Dogiparthi Aasrith(21BAI1702), for the award of the degree of Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning, Vellore Institute of Technology, Chennai is a record of bonafide work carried out by me under the supervision of Dr Bhavadharini R M.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: 03-04-2025

  
Signature of the Candidate



**VIT<sup>®</sup>**  
**CHENNAI**  
Vellore Institute of Technology  
(Deemed to be University under section 3 of UGC Act, 1956)

**School of Computer Science and Engineering**

## CERTIFICATE

This is to certify that the report entitled "Anomaly detection in Network traffic: Using Hybrid Techniques" is prepared and submitted by **Dogiparthi Aasrith(21BAI1702)** to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning** is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:

Name: Dr. Bhavadharini R M

Date: 03-04-2025

Signature of the Examiner

Name: 14/4/25

Date: Dr. K. L. EKSHMI

Signature of the Examiner

Name: RAJMOHAN SAIKARAN

Date: 17-4-25

Approved by the Head of Department,

**B.Tech. CSE with Specialization in Artificial Intelligence and Machine Learning**

Name: Dr. Sweetlin Hemalatha

Date: 03-04-2025

(Seal of SCHOOL YOU BELONG TO)



D

## **ABSTRACT**

Network traffic deals with a lot of issues regarding data interception to ensuring confidentiality, integrity, and authenticity of communication. However, it possesses severe threats from malicious ones, such as data exfiltration, Distributed Denial-of-Service (DDoS) attacks, and command-and-control operations by cybercriminals. The inability to directly inspect payloads complicates anomaly detection, leaving security systems to rely on indirect features like packet metadata and flow patterns, which are often insufficient to comprehensively identify sophisticated threats. Traditional anomaly detection methods, such as deep packet inspection and rule-based systems, are increasingly rendered ineffective in encrypted environments. Deep packet inspection requires decryption, which is computationally expensive, violates privacy regulations, and introduces latency into the network. Similarly, rule-based systems rely heavily on predefined signatures and heuristics, which fail to adapt to emerging threats or detect zero-day anomalies. These approaches create significant gaps in security, especially as network traffic continues to grow in volume and complexity across various industries. The proposed hybrid model of Artificial Neural Network (ANN) and Convolutional Neural Network (CNN) model achieved 98.34%, with the precision, recall and F1-score of 92.5%, 96.5%, and 95.5% respectively. While ANN model resulted with 96.82% accuracy, and CNN model showed 94.36%. Comparatively, the hybrid system results in more accuracy and is more efficient than ANN and CNN models.



## **ACKNOWLEDGEMENT**

It's my pleasure to express with deep sense of gratitude to Dr. Bhavadharini R M, Assistant Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Deep Learning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor i/c & Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. Sweetlin Hemalatha C, Head of the Department, B.Tech. CSE with Specialization in Artificial Intelligence and Machine Learning and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: 03-04-2025

**Dogiparthi Aasrith**

# **CONTENTS**

<b>CONTENTS .....</b>	<b>iii</b>
<b>LIST OF FIGURES .....</b>	<b>v</b>
<b>LIST OF TABLES .....</b>	<b>v</b>
<b>LIST OF ACRONYMS .....</b>	<b>vi</b>
<b>LIST OF EQUATIONS .....</b>	<b>vii</b>

## **CHAPTER 1**

### **INTRODUCTION**

1.1 INTRODUCTION .....	1
1.2 OVERVIEW OF PROJECT .....	2
1.3 CHALLENGES PRESENT IN PROJECT .....	2
1.4 PROJECT STATEMENT .....	3
1.5 OBJECTIVES AND SCOPE OF THE PROJECT .....	3
1.6 CONTRIBUTION TO THE PROJECT .....	4

## **CHAPTER 2**

### **RELATED STUDY**

2.1 LITERATURE SURVEY .....	5
2.2 BACKGROUND STUDY TABLE .....	8

## **CHAPTER 3**

### **PROPOSED ARCHITECHTURE**

3.1 DATASET .....	9
3.2 DATA AUGMENTATION AND DATA PRE-PROCESSING .....	12
3.3 CNN ARCHITECTURE .....	13
3.4 ADVANTAGES AND DISADVANTAGES OF CNN .....	16
3.5 ANN ARCHITECTURE .....	17
3.6 ADVANTAGES AND DISADVANTAGES OF ANN....	21

**CHAPTER 4**  
**PROPOSED METHODOLOGY**

4.1 ALGORITHM .....	22
4.2 PSEUDOCODE.....	23
4.3 DEEP LEARNING ALGORITHMS .....	25

**CHAPTER 5**  
**RESULTS AND DISCUSSION**

5.1 PERFORMANCE METRICS EXPLANATION .....	30
5.2 RESULTS .....	34

**CHAPTER 6**  
**CONCLUSION**

6.1 CONCLUSION .....	39
----------------------	----



## **LIST OF FIGURES**

3.1 Class Distribution	9
3.1 Features in the dataset	10
3.1 Various protocols in dataset	10
3.1 Services in dataset	11
3.1 Proposed Architecture	11
3.2 Balanced data using SMOTE technique	12
3.2 Dataset after using label encoder	12
3.3 CNN Architecture	15
3.5 ANN Architecture	20
4.2 CNN Model Summary	26
4.2 ANN Model Summary	28
4.2 Hybrid Model Summary	29
5.1 Confusion matrix	31
5.2 Comparison of Results	35
5.2 Confusion Matrix	36
5.2 Performance of CNN	36
5.2 Performance of ANN	37
5.2 Performance of Hybrid	37
5.2 Comparison of Proposed work with existing studies	38

## **LIST OF TABLES**

2.2 Literature Survey	8
3.1 Dataset Description	10
5.2 Results of the proposed system for 40 epochs	34
5.2 Performance metrics of the Proposed system	35

## **LIST OF ACRONYMS**

- 1.1 CONVOLUTIONAL NEURAL NETWORKS (CNN)
- 1.1 ARTIFICIAL NEURAL NETWORKS (CNN)
- 1.1 INTERNET CONTROL MESSAGE PROTOCOL (ICMP)
- 1.1 USER DATAGRAM PROTOCOL (UDP)
- 1.1 TRANSMISSION CONTROL PROTOCOL (TCP)
- 1.2 DISTRIBUTED DENIAL OF SERVICES (DDoS)
- 1.4 TRANSPORT LAYER SECURITY(TLS)
- 1.4 SECURE SOCKET LAYER (SSL)
- 2.1 AUTOENCODER (AEs)
- 2.1 POISON-RESISTANT ANOMALY DETECTION (PRAD)
- 2.1 QUALITY OF SERVICE (QoS)
- 2.1 LONG SHORT- TERM MEMORY (LSTM)
- 2.1 GATED RECURRENT UNITS (GRUs)
- 2.1 DEEP ENCRYPTED TRAFFIC DETECTION (DETD)
- 2.1 INTERNET OF THINGS (IoT)
- 2.1 SUPPORT VECTOR MACHINE (SVM)
- 2.1 STOCHASTIC GRADIENT DESCENT (SGD)
- 3.2 RECTIFIED LINEAR UNIT (RELU)
- 3.5 Unit GAUSSIAN ERROR LINEAR UNIT(GELU)
- 3.5 MEAN SQUARED ERROR (MSE)
- 5.1 TRUE POSITIVE (TP)
- 5.1 TRUE NEGATIVE (TN)
- 5.1 FALSE POSITIVE (FP)
- 5.1 FALSE NEGATIVE (FN)
- 5.1 RECEIVER OPERATING CHARACTERISTIC (ROC)
- 5.1 AREA UNDER THE CURVE (AUC)
- 5.1 NATURAL LANGUAGE PROCESSING (NLP)

## **LIST OF EQUATIONS**

4.2 CONVOLUTIONAL LAYER

4.2 RELU ACTIVATION FUNCTION

4.2 DROPOUT LAYER

4.2 LOSS FUNCTION (BINARY CROSSENTROPY)

4.2 ADAM OPTIMIZER

4.2 FLATTEN LAYER

4.2 SWISH ACTIVATION FUNCTION

4.2 GELU ACTIVATION FUNCTION

## Chapter 1

# Introduction

### 1.1 INTRODUCTION

The widespread adoption of communication protocols, such as Internet Control Message Protocol (ICMP), User Datagram Protocol (UDP) and Transmission Control Protocol (TCP), has significantly enhanced data security and privacy in digital networks. These protocols ensure that sensitive information remains confidential, making them integral to modern communication systems. Advanced machine learning techniques have emerged as promising solutions, leveraging metadata and statistical features to detect deviations indicative of malicious activities. However, existing models often face limitations such as high computational costs, overfitting on imbalanced datasets, and difficulty adapting to evolving threats [1]. These limitations underscore the need for robust, scalable, and privacy-preserving solutions capable of maintaining high detection accuracy without decrypting the traffic. The challenge is further exacerbated by the need to balance privacy preservation with effective threat detection. While network traffic protects user's sensitive data, it also hinders the ability to monitor network activity, creating a blind spot for security solutions [4]. This trade-off necessitates the development of innovative approaches that can accurately detect anomalies without decrypting traffic. Achieving this balance requires a paradigm shift from traditional techniques to advanced machine learning-based models capable of leveraging network traffic metadata to identify malicious behaviors with minimal computational overhead and high accuracy.

This research proposes a hybrid anomaly detection system that integrates Artificial Neural Networks (ANNs) and Convolutional Neural Networks (CNNs) with custom enhancements to address these challenges. By incorporating advanced activation functions, such as Swish and GELU, and regularization techniques tailored to mitigate overfitting, the model achieves superior generalization and performance. The hybrid architecture, designed to optimize feature learning and classification accuracy, demonstrates significant potential for anomaly detection in network traffic. This approach enhances detection capabilities, making it a viable solution for modern cybersecurity landscapes.

## 1.2 OVERVIEW OF PROJECT

The research project aims to develop a novel anomaly detection framework for encrypted network traffic applications, as current security solutions struggle with privacy, execution costs, and changing cyber threat characteristics. Traditional security techniques have drawbacks in deep packet inspection and rule-based systems because decryption methods reduce privacy while increasing latency, and signature-based detection cannot detect unknown threats. The project combats current cybersecurity obstacles by developing a neural network architecture which merges Artificial Neural Networks (ANN) with Convolutional Neural Networks (CNN) utilizing packet-size data and flow-pattern data and statistical values extracted from network metadata while leaving payloads unexamined. The system ensures data privacy and successfully detects Distributed Denial-of-Service (DDoS) attacks together with data exfiltration and command-and-control operations.

- **Hybrid Architecture:** Combining the strengths of ANN (excellent for learning complex relationships) and CNN (effective in extracting spatial patterns from structured data).
- **Advanced Activation Functions:** Utilizing Swish and GELU functions to improve learning capability and convergence.
- **Regularization Techniques:** Employing strategies like dropout and batch normalization to reduce overfitting, especially on imbalanced datasets.

## 1.3 CHALLENGES PRESENT IN PROJECT

- **Obfuscation of Payload Content:** Encryption obscures the payload content, making it impossible to evaluate the actual data being delivered, which is critical for detecting unwanted activity such as data exfiltration and malware spreading.
- **Dynamic Nature of Encrypted Communication Patterns:** The encryption methods are constantly evolving, making it difficult for anomaly detection systems to track and recognize new patterns associated with valid and malicious communications.
- **Computational Costs:** Traditional anomaly detection techniques are computationally expensive, particularly when dealing with massive amounts of encrypted communication in real time. This high overhead hampers the deployment of scalable detection systems greatly.
- **Imbalanced Datasets:** Encrypted traffic datasets are usually imbalanced, with considerably more genuine traffic than malicious traffic, causing overfitting in machine learning models and restricting their ability to generalize successfully to unexpected assaults.



- **Adaptability to Evolving Threats:** Current detection systems frequently struggle to adapt to new or emerging attack strategies, especially when dealing with encrypted traffic, making it even more challenging to maintain a high detection accuracy rate.

## 1.4 PROJECT STATEMENT

The widespread use of encrypted communication protocols such as Transport Layer Security (TLS) and Secure Sockets Layer (SSL) has significantly improved data security and privacy over digital networks. Anomaly detection systems face additional challenges because of these excellent security measures for sensitive information. These security protocols force traditional anomaly detection systems to fail because they encrypt both regular traffic and suspicious network activity. SKU and other dangerous activities including data exfiltration and Distributed Denial-of-Service assaults and malware dissemination remain undetectable by current security systems. Standard evaluation models are incapable of analyzing encrypted packets effectively while encrypted traffic continues to grow thus creating an immediate necessity for unsecured detection methods.

## 1.5 OBJECTIVES AND SCOPE OF THE PROJECT

The primary research goal involves creating a resilient hybrid anomaly detection system by merging Artificial Neural Networks (ANNs) with customized improvements which solve encryption traffic-specific problems. The specific goals are:

- **Integration of Advanced Activation Functions:** Incorporating advanced activation functions such as Swish or GELU to improve the learning process and increase model efficiency in detecting anomalies.
- **Regularization Techniques to Mitigate Overfitting:** Implementing tailored regularization methods (such as L1, L2 regularization) to combat overfitting, especially in scenarios where the dataset is imbalanced.
- **Efficient Feature Learning:** Optimizing feature extraction and selection to improve detection capabilities without the need for decrypting traffic, thereby maintaining privacy and security.
- **Real-time Anomaly Detection:** The proposed model needs to detect malicious activities in encrypted traffic in real-time mode but with minimal computational expenses and privacy protection features.
- **Scalable and Privacy-Preserving Solution:** Establishing a workable answer which functions in extensive network systems while protecting privacy standards by not demanding access to the encrypted content.

This research stands to improve encrypted traffic anomaly detection by maintaining adaptability to emerging attack patterns as well as operating efficiently and precisely in actual deployment scenarios.

## 1.6 CONTRIBUTION TO THE PROJECT

- **Designing a Hybrid Detection Model:** Developed a novel hybrid model combining Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN) specifically tailored for encrypted network traffic anomaly detection. This hybrid architecture leverages the strengths of both models to enhance feature extraction and classification accuracy.
- **Enhancing Detection Accuracy:** Achieved a significant improvement in anomaly detection performance, with the hybrid model reaching an accuracy of 98.34%, and high precision, recall, and F1-scores. This advancement demonstrates the model's capability to identify sophisticated threats more effectively than standalone ANN or CNN models.
- **Ensuring Privacy Preservation:** Focused on using metadata (packet sizes, flow patterns, statistical features) rather than decrypting network payloads, ensuring compliance with privacy regulations while still detecting malicious activities such as DDoS attacks and data exfiltration.
- **Mitigating Overfitting and Improving Generalization:** Incorporated advanced activation functions (Swish and GELU) and applied regularization techniques like dropout and batch normalization, leading to better generalization and minimizing the risk of overfitting, especially on imbalanced datasets.
- **Optimizing Computational Efficiency:** Designed the model architecture to balance high detection performance with computational efficiency, making it suitable for real-time deployment in large-scale network environments without introducing significant latency.
- **Comparative Evaluation of Models:** Conducted a detailed comparative analysis of the individual ANN, CNN, and hybrid models, showcasing how the hybrid system outperforms the standalone models in terms of accuracy, efficiency, and adaptability to evolving threats.

## Chapter 2

### Related Study

#### 2.1 LITERATURE SURVEY

Anomaly detection methods that rely on inspecting packet contents become less effective since encrypted network traffic usage steadily increases. Multiple methods employing semi-supervised techniques and deep learning approaches along with unsupervised methods have been used for detecting anomalies within encrypted data streams. The paper presents an extensive analysis of advanced methodologies which solve the problems of encrypted traffic anomaly detection. The semi-supervised anomaly detection framework Poison-Resistant Anomaly Detection (PRAD) operates to defend against poisoning attacks as well as reduce human labeling mistakes. This framework includes a feature encoding module built from an autoencoder architecture which utilizes Amsgrad gradient descent algorithm together with warm-up strategy for improving both feature extraction and generalization capabilities [1]. This module performs an analysis of both inter-class distance and reconstruction error distribution patterns under poisoning attack conditions. An online clustering approach forms the basis of anomaly detection to increase detection reliability. Experimental testing demonstrates that PRAD delivers superior resistance against poisoning attacks when compared to standard semi-supervised systems. The deep dictionary learning with LSTM-based autoencoder forms the basis of D2LAD which operates as a completely unsupervised online anomaly detection system. Through dictionary learning the system detects hidden normal patterns from raw encrypted streams of data while extracting sequences of features. The deep dictionary controls how the anomaly score calculation determines relevance. D2LAD shows high performance across real-world benchmarks through experiments which prove its ability to successfully detect anomalies with low resource requirements while surpassing current state-of-the-art techniques [2,3]. An anomaly detection framework based on deep learning analytics utilizes wavelet transforms together with byte-to-image transformation to capture features from IPsec-secured encrypted network traffic. The detection method produced 93% successful results in abnormal traffic identification which demonstrates its usefulness for early anomaly detection systems in power grid communication networks. The combination of methods strengthens critical infrastructure cybersecurity because attacks get discovered at dawn. An encryption traffic detection method relies on multiple autoencoders (AEs). Through sandbox-based process data collection users can identify standard traffics along with their malicious counterparts. Labeling features run sequentially through multiple AE layers that creates expanded dimensional vectors to enhance classification quality. Experimental results demonstrate that the multi-AE system outperforms traditional models

by reaching better accuracy rates with lower loss numbers [4,5]. The study also investigates dataset imbalance effects which show that unbalanced conditions reduce positive detection success. A new classification system uses a two-step randomized forest procedural approach which features modified random forest elements. The evaluation with ISCX VPN-NonVPN data demonstrated that multi-AE achieves better classification results than standard approaches by showing 5% increased accuracy since decision trees and KNN and general random forest algorithms were tested. By integrating this approach the precision of traffic classification gets better during the time encrypted traffic prevails over the Internet. [6,7] Different detection methods exist for network traffic anomalies according to this investigation. The analytical method proposed an unsupervised procedure to detect zero-day DDoS attacks against IoT networks through random projection and ensemble models of K-means, GMM, and one-class SVM which reached 94.55% accuracy. The authors applied deep convolutional autoencoder (MR-DCAE) using manifold regularization to detect unauthorized broadcasting through their model for feature compression autoencoders in research [8,9]. In terms of machine learning framework, the paper proposed a generic FL-based deep anomaly detection framework for energy theft detection, where each user trained local deep autoencoder-based detectors using their private electricity usage data [10,11]. Regarding identity resolution system traffic analysis, this research is the first to analyze the identity resolution system from the perspective of traffic analysis, designing a machine learning-based framework to automatically extract critical information of the identity resolution system from network traffic. The research reports numerous academic investigations that took place in malware attack detection together with encrypted traffic anomaly detection and related areas. An anomaly detection research team developed a reconstruction error-based system utilizing an autoencoder (AE) for packet metadata analysis without particular node information and demonstrated superior performance over conventional approaches with improved accuracy and AUC score and precision rates and F1 scores [12,13]. Another study provided a comprehensive overview of machine learning-based methods for encrypted malicious traffic detection, proposing a framework for systematic discussion and analysis of these models. Multiple machine learning algorithms including decision trees, random forests and gradient boosting and SVM and neural networks operate in identity resolution to enhance data quality and automatic resolution processes and dynamic data management. The field of encrypted traffic detection and sophisticated cyber-attack modeling depends on advanced detection methods alongside machine learning models according to two studies discussing this issue [14,15]. Network security research on machine learning techniques highlights the growing concern about adversarial attacks, especially poisoning attempts during model training, which can alter training data to impair model performance. Researchers have examined the effects of these attacks on various machine learning models and classified them according to their goals [16]. Strong training mechanisms, anomaly detection in training data, and safe aggregation methods for federated learning are some of the countermeasures that have

been suggested. Researchers are now using data-driven methods that examine network flow metadata since the growing usage of encrypted communication protocols has reduced the efficacy of conventional pattern-matching techniques in threat identification. In certain situations, supervised machine learning models that have been trained on huge datasets have demonstrated promise in reaching 0% false discovery rates [17, 18]. Feature selection is now being automated using machine learning models to identify subtle patterns indicative of malicious activity. Feature-based tactics are more reliable and adaptable than payload-dependent ones, according to comparative research. For network security and performance management, traffic classification is essential, and session start information and encryption protocol design offer helpful insights. Payload-based and feature-based categorization algorithms are distinguished in surveys; the latter is becoming more popular because it is independent of protocols. The trade-offs between accuracy, processing cost, and adaptability to shifting network environments are highlighted in thorough studies [19, 20]. Traffic classification has developed over the course of two decades and has been instrumental in applications including Quality of Service (QoS) provisioning and billing, as well as security enforcement in firewalls and intrusion detection systems. The traditional approaches like port-based analysis, packet inspection, and traditional machine learning techniques have found it difficult to deal with the growing dominance of encrypted traffic, paving the way for moving toward deep learning-based methods [21,22]. Recent work emphasizes deep learning models such as Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRUs) as the potential solutions to anomaly detection from encrypted traffic due to their potential to learn complex patterns without human intervention in selecting features [23]. Experiments have also been performed with hybrid architectures, and CNN-GRU has proved superior to other hybrids by combining effective feature extraction and improved learning of temporal patterns. Models such as Deep Encrypted Traffic Detection (DETD) and autoencoder-based methods have also shown significant improvement in the accuracy of anomaly detection and effectiveness of feature extraction, particularly in resource-constrained environments like the Internet of Things (IoT). Moreover, incremental machine learning methods like Support Vector Machines (SVM) with Stochastic Gradient Descent (SGD) have been introduced for encrypted malware detection, with adaptability via periodic updates. These developments underscore the increasing role of deep learning and adaptive machine learning techniques in confronting the difficulties created by encrypted traffic in today's cybersecurity and network management [24,25]. Table 1 shows the comparison of the referenced results.



## 2.2 BACKGROUND STUDY TABLE

S.NO	Authors	Models	Results
1	Wu, Z., et al (2024) [1]	Variational Recurrent Neural Network	Accuracy-85.2%
2	Xing, J., et al (2020) [2]	LSTM-based Autoencoder	Accuracy-94.5%
3	Behdadnia, T., et al (2023) [3]	CNN	Accuracy-93%
4	Yu, T., et al (2019) [4]	Autoencoder	Accuracy-96.3%
5	Lv, G., et al. (2020) [5]	Decision tree, KNN, Random Forest	Accuracy Decision tree-71%, KNN-67.47%, Random Forset-73.46%
6	Kopp, M., et al (2018) [6]	Clustering	Threshold-95%
7	Han, S., et al (2022) [7]	K Means, SVM, XGBoost, LSTM	Accuracy K Means-93%, SVM-88%, XGBoost-88%, LSTM-88%
8	Shen, M., et al (2022) [8]	Decision Tree, Random Forest, LSTM	Accuracy Decision Tree-97%, Random Forest-96%, LSTM-98%
9	Meghdouri, F., et al (2020) [9]	Random Forest	Accuracy CICIDS2017-99%, UNSW-NB15-99.4%, ISCX-Bot-2014-99.7%
10	Wang, Z., et al (2022) [10]	Convolutional Autoencoders	Accuracy-87%
11	Zhu, Z., et al (2022) [11]	Random Forest, CNN, LeNet-5	Recall Random Forest-85%, CNN-100%, LeNet-5- 99.49%
12	Pratiwi, M. et al (2024) [16]	LSTM	Accuracy-99.37%
13	Wingarz, T., et al (2024) [23]	ANN, SVM	Accuracy ANN-82%, SVM-91%
14	Wang, J., et al (2022) [22]	FS-Net, Single Node method,BERT	Accuracy FS-Net-84.2%, Single Node method- 71.4%,BERT-96%
15	Anderson, B., et al (2016) [18]	11-logistic regression	Accuracy-93.978%
16	Shekhawat, A. S., et al (2019) [19]	SVM, Random Forest, XGBoost	Accuracy SVM-92%, Random Forest- 96.80%, XGBoost- 97%
17	Bakhshi, T., et al (2021) [24]	Hybrid: CNN+GRU	Accuracy NSL-KDD-93.10% NB15-91.21%, CIC-17-90.17%
18	Long, G., et al (2023) [15]	Conventional Stacked Autoencoder	Accuracy-96.98
19	Kim, M. G., et al (2024) [13]	IsolationForest, OneClassSVM,Auto Encoder	Accuracy IsolationForest-85.4%, OneClassSVM- 73.9%, Auto Encoder-85.6%
20	Lee, I., Roh, H., et al (2021) [25]	Stochastic Gradient Descent, Passive Aggressive, Gaussian Naive Bayes	Accuracy Stochastic Gradient Descent-94.4%, Passive Aggressive-86.6%, Gaussian Naive Bayes- 86%

*Table 1: Literature Survey*

## Chapter 3

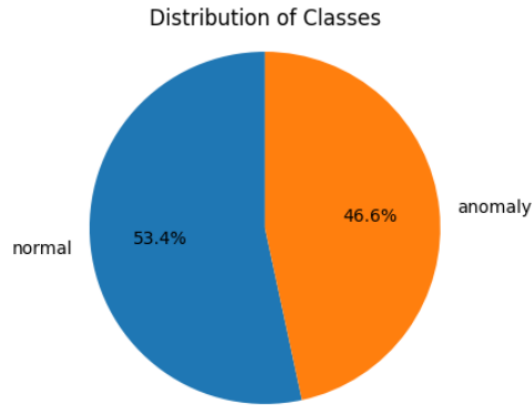
# Proposed Architecture

### 3.1 DATASET

The dataset, *Network Intrusion Detection* [26] contains various intrusions simulated in a military network environment. It simulated an average US Air Force LAN and established an environment to obtain raw TCP/IP dump data for a network. The LAN was targeted like a real-world scenario and bombarded with different attacks. A connection is a sequence of TCP packets that begin and finish at a specific time interval, during which data travels to and from a source IP address to a target IP address using a well-defined protocol. In addition, each link is labeled as either normal or an attack, with just one unique attack kind. Each connection record is around 100 bytes in size. Table 2 shows total no of records in the dataset i.e 25212, the data provided was split into 80 training and 20 testing.

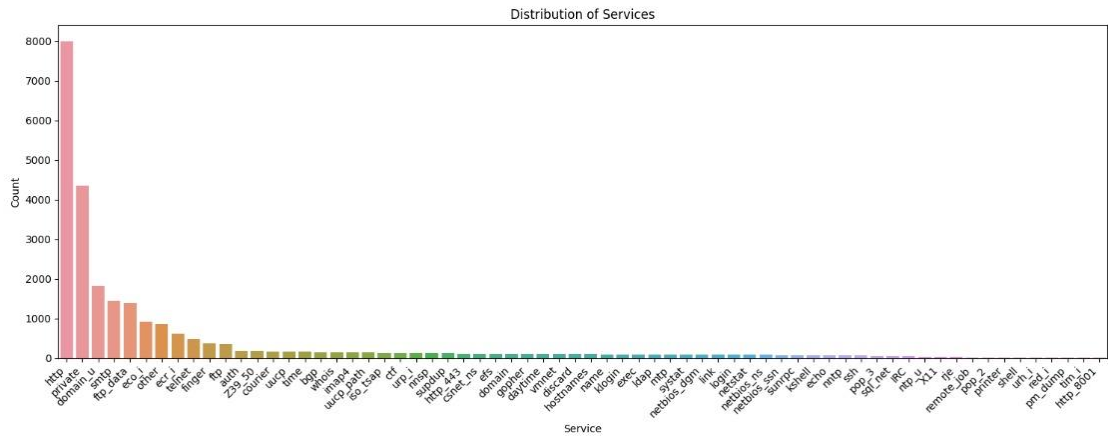
Normal and attack data include 41 quantitative and qualitative parameters for each TCP/IP connection (3 qualitative and 38 quantitative). The class variable has two categories as shown in Fig.1:

- Normal [53.4%]
- Anomaly [46.6%]

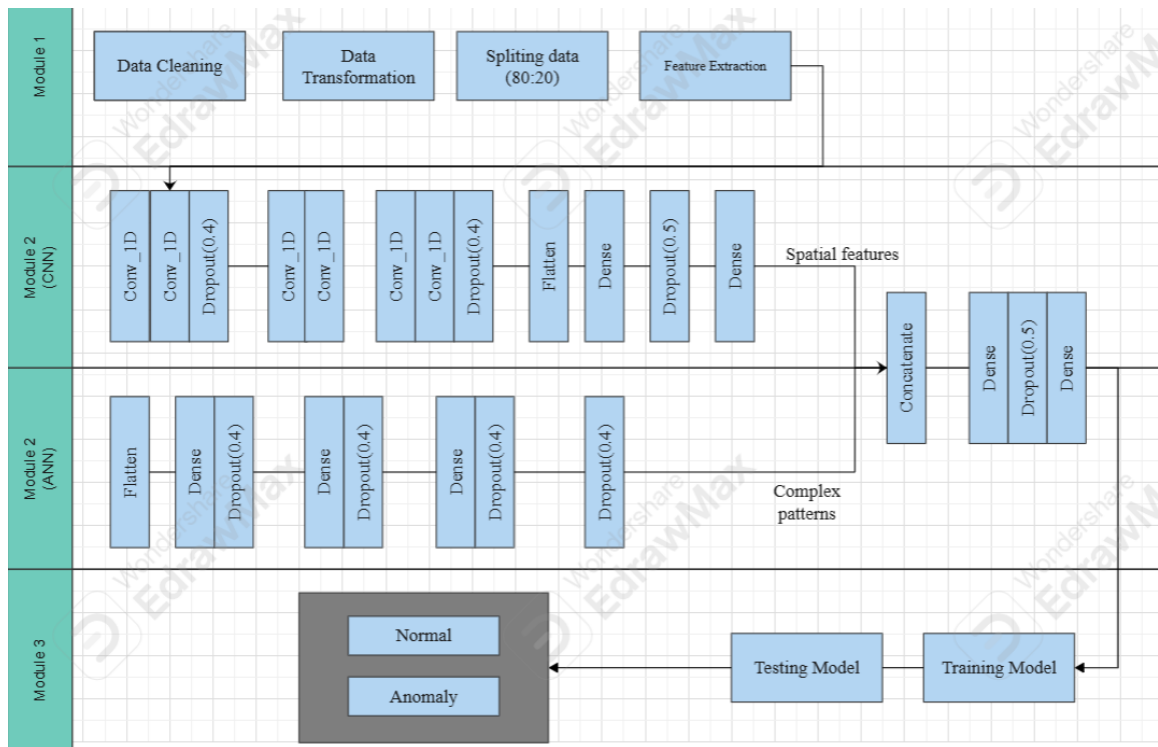


**Figure 1: Class Distribution**





**Figure 4: Services in dataset**



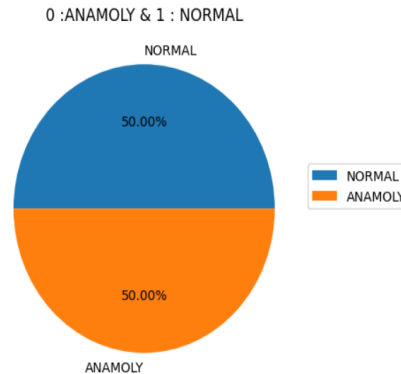
**Figure 5: Proposed Architecture**

The architecture (Fig.5). shows how hybrid deep learning model employs Convolutional Neural Networks (CNNs) together with Artificial Neural Networks (ANNs) for classification purposes. Module-1 (Data Preprocessing) starts the process by performing data cleaning then splitting and extracting features to make the dataset ready. The preprocessed dataset enters Module-2 for training of CNN and ANN alongside each other as part of the proposed hybrid system. The model evaluation takes place during Module-3 after its training phase as part of

the testing process. The proposed system produces outputs to identify instances either as "Anomaly" or "Normal."

### 3.2 Data Augmentation and Data preprocessing

This module is responsible for cleaning and preparing raw network traffic data from dataset: *Network Intrusion Detection* [26] for effective model training. It involves handling values which are missing, and performing feature extraction to ensure that the dataset is optimized for analysis. Performed label-encoding, for protocol\_type, service, flag and class columns to change categorical values to numerical values, and smote technique for class imbalance as shown in Fig.6. The actual data count was normal:13449 and anomaly:11743, now we made both of them equal as 13449. These steps enhance the quality and relevance of the input data, which is critical for training a robust ANN. Techniques like data splitting further enable the separation of training and testing datasets, in 80:20 ratio as shown in Table 2, to ensure unbiased evaluation.



**Figure 6: Balanced data using SMOTE technique**

[illegible]

**Figure 7: Dataset after using label encoder**



### 3.3 CNN ARCHITECTURE

Convolutional Neural Networks (CNNs) are a class of deep learning models that are particularly effective for analyzing visual data. They are inspired by the structure of the animal visual cortex and are designed to automatically and adaptively learn spatial hierarchies of features from input images. CNNs are widely used for image and video recognition, image classification, medical image analysis, and many other applications.

#### Key Components of CNNs

##### Convolutional Layers:

- **Filters (Kernels):** Small, learnable matrices (e.g., 3x3 or 5x5) that slide over the input image, performing convolution operations to extract features. Each filter detects a specific feature like edges, textures, or patterns.
- **Feature Maps:** The result of applying filters to the input image. Each feature map represents the presence of a specific feature at various spatial locations in the image.
- **Stride:** The step size with which the filter moves across the image. A larger stride reduces the spatial dimensions of the output.
- **Padding:** Adding extra pixels around the border of the input image to control the spatial dimensions of the output. Common padding types are 'valid' (no padding) and 'same' (padding to maintain input size).

##### Activation Functions:

###### ▪ Sigmoid Function

The sigmoid function is a non-linear activation function that maps any input to a value between 0 and 1.

###### Advantages:

- **Output range:** Maps inputs to the range (0, 1), making it useful for binary classification.
- **Smooth gradient:** Provides smooth gradients that can be used to update weights during backpropagation.

###### Disadvantages:

- **Vanishing gradient problem:** For very high or very low inputs, the gradient approaches zero, which can slow down the learning process.
- **Outputs not zero-centered:** Can cause the gradient updates to zigzag, slowing down convergence.

### ▪ **ReLU Function (Rectified Linear Unit)**

The ReLU function is a piecewise linear activation function that outputs zero for negative inputs and the input value itself for positive inputs.

#### **Advantages:**

- **Computational efficiency:** Simple to compute and often speeds up the convergence of the training process.
- **Sparse activation:** Activates only a portion of the neurons, which makes the network more efficient and helps mitigate the vanishing gradient problem.

#### **Disadvantages:**

- **Dead neurons:** Neurons can die during training if they consistently output zero, leading to no gradient updates.
- **Unbounded output:** Can produce very high values, which might make the model sensitive to outliers.

### ▪ **Softmax Function**

The softmax function is typically used in the output layer for multi-class classification problems. It converts logits into probabilities that sum to one.

#### **Advantages:**

- **Probabilistic interpretation:** Outputs can be interpreted as probabilities of each class, which is useful for classification tasks.
- **Sum-to-one constraint:** Ensures that the sum of the outputs is equal to one, making it easier to interpret the results.

#### **Disadvantages:**

- **Computationally expensive:** Involves exponentials and summations over all classes, which can be computationally intensive for large numbers of classes.
- **Gradient issues:** Like other exponential functions, can lead to very small gradients for certain inputs, affecting the learning process.

## Pooling Layers:

- **Max Pooling:** Reduces the spatial dimensions by taking the maximum value in a defined window (e.g., 2x2) and down-sampling the feature maps. This operation helps in reducing the computational load and controlling overfitting.
- **Average Pooling:** Similar to max pooling but takes the average value within the window. Less common than max pooling.

## Fully Connected Layers:

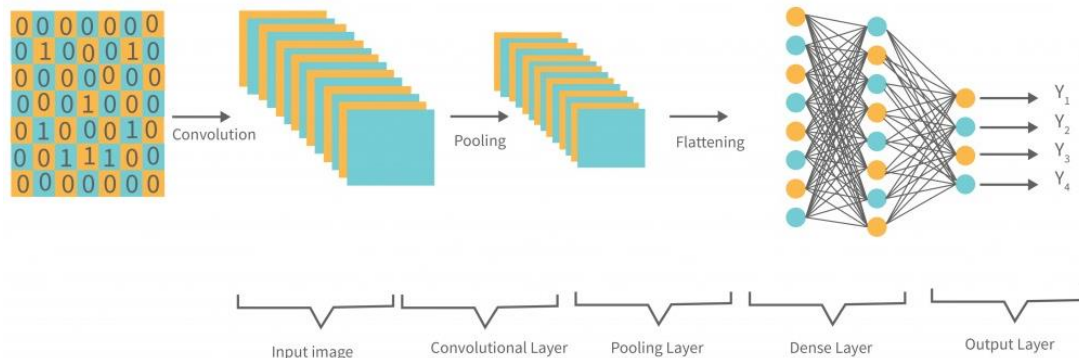
After several convolutional and pooling layers, the high-level reasoning in the neural network is done via fully connected layers. These layers take the flattened output from the convolutional layers and produce the final output, such as class scores in classification tasks.

## Dropout:

A regularization technique where a fraction of neurons is randomly set to zero during training to prevent overfitting. This forces the network to learn more robust features.

## Output Layer:

For classification tasks, the output layer typically uses a softmax activation function to produce a probability distribution over class labels.



**Figure 8: CNN Architecture**

### 3.4 ADVANTAGES AND DISADVANTAGES OF CNN

#### Advantages of CNNs

- **Automatic Feature Extraction:**  
CNNs can automatically learn relevant features directly from the input data without the need for manual feature engineering.
- **Parameter Sharing:**  
The same filter is applied across different parts of the input image, which reduces the number of parameters and computations, making the model more efficient.
- **Translation Invariance:**  
CNNs can recognize patterns regardless of their position in the input image due to the convolution and pooling operations.
- **Hierarchical Feature Learning:**  
CNNs can learn hierarchical representations of the input data, with lower layers capturing simple features like edges and higher layers capturing more complex features like objects.

#### Disadvantages of CNNs

- **Large Data Requirements:**  
CNNs typically require a large amount of labeled training data to achieve good performance, which can be a limitation in applications with limited data availability.
- **Computationally Intensive:**  
Training CNNs can be computationally expensive and time-consuming, especially with large datasets and deep architectures. They often require powerful hardware, such as GPUs, for efficient training.
- **Sensitivity to Hyperparameters:**  
The performance of CNNs can be sensitive to the choice of hyperparameters, such as the number of filters, filter size, stride, and learning rate. Tuning these hyperparameters can be challenging and time-consuming.
- **Overfitting:**  
CNNs with a large number of parameters can overfit to the training data, especially if the dataset is small. Regularization techniques like dropout and data augmentation are often necessary to mitigate overfitting.

## 3.5 ANN ARCHITECTURE

The Artificial Neural Network (ANN) remains a fundamental deep learning model which takes inspiration from human brain neural networks. The modeling architecture contains multiple layers of neural connections which build understanding of complex data pattern relationships. The system performs its operation by accepting data inputs which it processes through various hidden layers activated through activation functions until it produces an output. During training each neural connection learns an associated weight value to reduce the prediction discrepancies. ANNs demonstrate flexible application because they serve all kinds of purposes such as classification, regression, anomaly detection, natural language processing and time series forecasting. Among their key machine learning applications is their ability to process raw data while understanding non-linear connections which provides them powerful capabilities.

### **Key Components of ANNs**

#### **Input Layer:**

- The input layer is the first layer of an ANN that receives the raw data.
- Each neuron in this layer represents a feature of the input data.
- It passes the input values to the next layer without performing any computation.

#### **Hidden Layers:**

- One or more layers between the input and output layers.
- Each hidden layer consists of neurons that process the input using weighted connections.
- These layers apply activation functions to introduce non-linearity, allowing the network to learn complex patterns.
- Deeper networks with multiple hidden layers are referred to as Deep Neural Networks (DNNs).

#### **Neurons (Nodes):**

- Fundamental processing units of the network.
- Each neuron performs a weighted sum of its inputs and applies an activation function.
- Outputs from neurons are passed to subsequent layers.

## Weights and Biases:

- **Weights:** Parameters that determine the importance of each input connection to a neuron.
- **Bias:** An additional parameter added to the weighted sum to improve model flexibility.
- Both weights and biases are updated during training using optimization algorithms.

## Activation Functions:

### ▪ GELU Function

The GELU (Gaussian Error Linear Unit) function is a smooth, non-linear activation function that combines properties of ReLU and sigmoid. It approximates the input multiplied by the cumulative distribution function of a Gaussian distribution.

#### Advantages:

- **Smooth activation:** GELU is smoother than ReLU, which can lead to better gradient flow and improved learning.
- **Performance boost:** Often results in better model performance, especially in deep networks, compared to ReLU and tanh.

#### Disadvantages:

- **Computational cost:** Slightly more computationally expensive due to the Gaussian function approximation.
- **Not always better:** In some cases, simpler functions like ReLU may perform similarly, especially in smaller models.

### ▪ Swish Function

The Swish function is defined as  $f(x) = x * \text{sigmoid}(x)$ . It is a self-gated activation function that is smooth and non-monotonic.

#### Advantages:

- **Non-monotonicity:** This helps in better information flow and can outperform ReLU in deeper networks.
- **Smooth and differentiable:** Leads to better optimization and improved accuracy in many tasks.

#### Disadvantages:

- **Higher computational cost:** Requires computation of the sigmoid function, making it slightly more expensive than ReLU.

- **May not generalize better:** In certain cases, simpler activation functions may yield comparable results.

- **Sigmoid Function**

The sigmoid function is a non-linear activation function that maps any input to a value between 0 and 1.

**Advantages:**

- **Output range:** Maps inputs to the range (0, 1), making it useful for binary classification.
- **Smooth gradient:** Provides smooth gradients that can be used to update weights during backpropagation.

**Disadvantages:**

- **Vanishing gradient problem:** For very high or very low inputs, the gradient approaches zero, which can slow down the learning process.
- **Outputs not zero-centered:** Can cause the gradient updates to zigzag, slowing down convergence.

**Output Layer:**

- The final layer of the ANN.
- Produces the output of the network based on the problem:
  - For classification tasks, softmax or sigmoid functions are commonly used.
  - For regression tasks, a linear activation may be used.

**Loss Function:**

- Measures the difference between the predicted output and the actual target values.
- Common loss functions:
  - **Mean Squared Error (MSE)** for regression tasks.
  - **Categorical Cross-Entropy** for classification tasks.
- Guides the model during training by providing feedback.

### Optimization Algorithm:

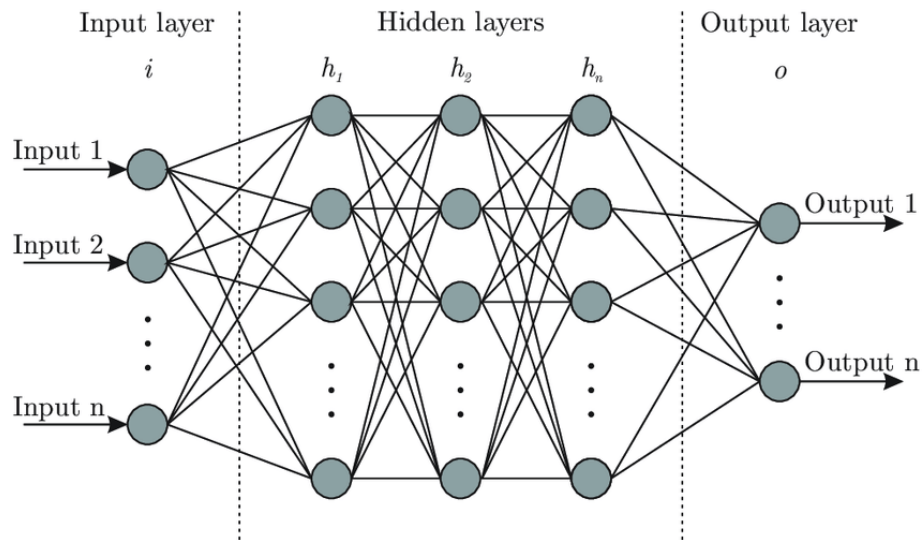
- Used to minimize the loss function by updating weights and biases.
- Common algorithms:
  - Gradient Descent
  - Adam Optimizer
  - RMSprop
- These algorithms help the ANN learn efficiently.

### Backpropagation:

- A key algorithm for training ANNs.
- Calculates gradients of the loss function with respect to weights and biases.
- Propagates errors backward from the output layer to adjust parameters.

### Regularization Techniques:

- Methods like Dropout and L2 Regularization are used to prevent overfitting.
- They ensure the model generalizes well to unseen data.



**Figure 9: ANN Architecture**



## 3.6 ADVANTAGES AND DISADVANTAGES OF ANN

### Advantages of ANNs

- **Versatility:** Can be applied to various problems like classification, regression, anomaly detection, etc.
- **Non-linearity:** Due to the use of non-linear activation functions, ANNs can model complex, non-linear relationships.
- **Automatic Feature Learning:** Capable of learning feature representations from raw data without manual feature extraction.
- **Parallel Processing:** Due to their structure, ANNs can process multiple inputs simultaneously, making them efficient with the right hardware (like GPUs).

### Disadvantages of ANNs

- **Computational Complexity:** Training large ANNs can be computationally expensive, requiring high-performance hardware.
- **Black Box Nature:** ANNs often lack interpretability, making it difficult to understand how decisions are made.
- **Overfitting:** ANNs with large numbers of neurons and layers may overfit to the training data if proper regularization techniques like dropout aren't applied.
- **Large Data Requirement:** Require large amounts of labeled data for effective training.
- **Hyperparameter Sensitivity:** Performance is highly dependent on the choice of hyperparameters such as learning rate, number of layers, number of neurons, and activation functions.

## Chapter 4

### Proposed Methodology

#### 4.1 ALGORITHM

---

Algorithm: Hybrid Model

---

- 1: Hybrid CNN\_ANN ( $X_{train}, Y_{train}, X_{val}, Y_{val}$ )
  - 2: Input: Dataset ( $X_{train}, Y_{train}, X_{val}, Y_{val}$ )
  - 3: Output: Performance metrics
  - 4: Define Hybrid CNN-ANN architecture:
  - 5: CNN branch: set Conv1D layers, flatten, dense, and dropout layers
  - 6: ANN branch: set Flatten, dense layers, and dropout layers
  - 7: Concatenate CNN and ANN branches
  - 8: Add fully connected layers with dense, and dropout layers
  - 9: Compile the Hybrid CNN-ANN model
  - 10: Set the batch size, optimizer, learning rate, epoch count (n), and early stopping criteria (esc) to their initial values.
  - 11: for i in 1 to n do
  - 12: while (esc)
  - 13: Train Hybrid CNN-ANN model using  $X_{train}, Y_{train}$
  - 14: end while
  - 15: Evaluate the Hybrid CNN-ANN model using  $X_{val}, Y_{val}$  and calculate evaluation metrics
  - 16: Return the calculated metrics
-

## 4.2 PSEUDOCODE

- Import necessary libraries and modules:
  - Import TensorFlow, Keras layers, models, and optimizers.\
  -
- Load dataset:
  - Load data from CSV file.
  - Normalize and reshape features for model input.
  - Split data into training and validation sets.
- Define CNN branch:
  - Create CNN input layer with shape (41, 1).
  - Add Conv1D layer with 32 filters, kernel size 3, and ReLU activation.
  - Add another Conv1D layer with the same parameters.
  - Apply MaxPooling1D layer with pool size 2.
  - Apply Dropout layer (rate = 0.4).
  - Add another set of Conv1D layers with 64 filters.
  - Apply MaxPooling1D and Dropout layers again.
  - Flatten the CNN branch output.
- Define ANN branch:
  - Create ANN input layer with shape (41, 1).
  - Flatten input data.
  - Add Dense layer with 256 neurons and ReLU activation.
  - Apply Dropout layer (rate = 0.4).
  - Add Dense layers with 128 and 64 neurons, each followed by Dropout.
- Merge CNN and ANN branches:
  - Concatenate outputs from both branches.
- Add fully connected layers after merging:
  - Add Dense layer with 256 neurons and ReLU activation.

- Apply Dropout layer (rate = 0.5).
  - Add output Dense layer with 2 neurons and sigmoid activation for binary classification.
- Compile the model:
  - Use Adam optimizer with learning rate 0.0002.
  - Set loss function as binary cross-entropy.
  - Set accuracy as the evaluation metric.
- Train the model:
  - Fit the model on the training data for a specified number of epochs.
  - Validate the model on the validation dataset.
- Define a function to get true labels from the dataset:
  - Iterate through batches in the dataset and collect true labels into an array.
- Get predictions for validation data:
  - Use the trained model to predict classes on the validation data.
  - Extract predicted class labels from prediction probabilities.
- Calculate evaluation metrics:
  - Precision: Calculate weighted precision score.
  - Recall: Calculate weighted recall score.
  - F1 Score: Calculate weighted F1 score.
  - ROC AUC Score: Calculate multi-class ROC AUC score.
- Compute confusion matrix:
  - Generate a confusion matrix based on true and predicted labels.
- Plot confusion matrix:
  - Use Matplotlib and Seaborn to visualize the confusion matrix with a heatmap.

## 4.3 DEEP LEARNING ALGORITHMS

### Convolutional Neural Networks

The proposed Convolutional Neural Network (CNN) model is designed to efficiently detect anomalies in network traffic data by leveraging the powerful feature extraction capabilities of convolutional layers. The model begins with multiple 1D convolutional layers, (1), each using 32 filters with a kernel size of 3 and RELU activation function, (2), which helps in capturing local patterns and relationships within the sequential input features of shape (41, 1). Dropout layers, (3) with a rate of 40% are introduced after every two convolutional layers to prevent overfitting and improve model generalization. Furthermore, additional convolutional layers with 64 filters are incorporated to capture more complex patterns. After feature extraction, a Flatten layer is used to convert the multi-dimensional output into a one-dimensional vector, which is then passed to a dense layer with 256 neurons and RELU activation for high-level feature learning. A final dropout layer with a higher rate of 50% further enhances robustness. The output layer consists of two neurons with softmax activation, effectively classifying the input data into two categories: normal or malicious. This refined CNN architecture combines convolutional feature learning with regularization techniques, ensuring accurate and efficient anomaly detection. The model summary is shown in Fig.10.

#### Convolutional Layer (Conv1D, 32 Filters, ReLU)

$$Z_1^{(i)} = \sum_{j=1}^k W_1^{(i)} X^{(i+j)} + b_1 \dots\dots\dots (1)$$

where:

- X is the input vector of shape (41,1).
- $W_1$  is the kernel (weight) matrix of shape (3,1,32).
- k=3 is the kernel size.
- $b_1$  is the bias vector
- $Z_1^{(i)}$  is the **pre-activation output** of the convolution.
- $X^{(i+j)}$  are the input values covered by the filter.
- The filter slides along the **time/feature dimension**.

The **ReLU activation** is applied:

$$A_1^{(i)} = \max(0, Z_1^{(i)}) \dots\dots\dots (2)$$

Where  $A_1^{(i)}$  is the activated output.

### Dropout Layer (40%)

$$A'_2 = D_1 \odot A_2 \dots\dots\dots (3)$$

where:

- $D_1$  is a binary dropout mask (with probability 0.4 for dropping neurons)
- $\odot$  represents element-wise multiplication.

### Loss Function (Binary Crossentropy)

$$L = -\sum_{i=1}^m Y_i \log(\hat{Y}_i) \dots\dots\dots (4)$$

where:

- $Y_i$  is the **true label** (one-hot encoded)
- $\hat{Y}_{yl}$  is the **predicted probability**.

### Adam Optimization Update Rule

$$W_{t+1} = W_t - \eta \cdot \frac{m_t}{\sqrt{v_t + \epsilon}} \dots\dots\dots (5)$$

where:

- $m_t$  and  $n_t$  are the first and second moment estimates,
- $\eta$  is the learning rate (0.001),
- $\epsilon$  prevents division by zero.

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 39, 32)	128
conv1d_1 (Conv1D)	(None, 37, 32)	3,104
dropout_3 (Dropout)	(None, 37, 32)	0
conv1d_2 (Conv1D)	(None, 35, 32)	3,104
conv1d_3 (Conv1D)	(None, 33, 32)	3,104
dropout_4 (Dropout)	(None, 33, 32)	0
conv1d_4 (Conv1D)	(None, 31, 64)	6,208
conv1d_5 (Conv1D)	(None, 29, 64)	12,352
dropout_5 (Dropout)	(None, 29, 64)	0
flatten_1 (Flatten)	(None, 1856)	0
dense_4 (Dense)	(None, 256)	475,392
dropout_6 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 2)	514

**Figure 10: CNN Model Summary**

## Artificial Neural Networks

The proposed Artificial Neural Network (ANN) model is intended exclusively for binary classification tasks, with a focus on anomaly detection in network traffic data. The model employs a sequential architecture with numerous dense layers and innovative activation functions like Swish, (8) and GELU,(9), which serve to increase non-linear learning capabilities and convergence. The input layer,(6) accepts a feature vector of shape (41, 1), representing extracted network features, and flattens it for processing. To reduce overfitting and ensure robust generalization, three hidden layers of 256, 128, and 64 neurons each are utilized, followed by a 40% dropout layer. The output layer is made up of two neurons that employ a softmax activation function to categorize network data as normal or malignant. Furthermore, the model is created with the Adam optimizer (5), a learning rate of 0.001, and a binary cross-entropy loss function (4), which enables fast weight updates and accurate classification. The combination of advanced activation functions, dropout regularization, and optimized architecture contributes to the model's high performance, achieving strong accuracy while maintaining computational efficiency. The model summary is shown in Fig.11.

### Input Layer Transformation:

$$x' = Flatten(x)..... (6)$$

Since the input shape is (41,1), flattening converts it into a vector of size 41.

### Hidden Layer (256 Neurons, Swish Activation):

$$Z_1 = W_1 X' + b_1..... (7)$$

$$A_1 = Swish(Z_1) = Z_1 \cdot \sigma(Z_1) = Z_1 \cdot \frac{1}{1+e^{-Z_1}}..... (8)$$

where:

- $x$  is the input data.
- $x'$  is the input vector
- $W_1$  is the weight
- $b_1$  is the bias vector
- $\sigma(Z_1)$  is the sigmoid function.

### Hidden Layer (128 Neurons, GELU Activation):

$$A_2 = GELU(Z_2) = \frac{1}{2} Z_2 (1 + \tanh(\sqrt{\frac{2}{\pi}} (Z_2 + 0.044715 Z_2^3)))..... (9)$$

GELU (Gaussian Error Linear Unit) smooths activation with a probabilistic approach.

where:

- $W_2$  is the weight
- $b_2$  is the bias vector

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 41)	0
dense (Dense)	(None, 256)	10,752
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 2)	130

*Figure 11: ANN Model Summary*

## Hybrid

The proposed hybrid model integrates both Convolutional Neural Network (CNN) and Artificial Neural Network (ANN) architectures to enhance the accuracy and efficiency of anomaly detection in network traffic data. The model consists of two parallel branches: the CNN branch extracts local and spatial features using multiple Conv1D layers with GELU activation, followed by MaxPooling1D and dropout layers to reduce overfitting. The ANN branch, on the other hand, flattens the input features and passes them through several dense layers with GELU activations and dropout layers, focusing on learning complex patterns and relationships in the data. After independent feature extraction, both branches are concatenated, allowing the model to leverage the strengths of both architectures—CNN's ability to capture spatial dependencies and ANN's capacity for high-level feature abstraction. The combined output is processed through fully connected layers with dropout for regularization and ends with a sigmoid-activated output layer for binary classification. The model is compiled using the Adam optimizer and binary cross-entropy loss function, ensuring effective learning. By combining CNN and ANN branches, this hybrid model achieves superior performance in identifying anomalies while maintaining robustness against overfitting. The hybrid model summary is shown in Fig.12.



Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 41, 1)	0
conv1d_10 (Conv1D)	(None, 39, 32)	128
conv1d_11 (Conv1D)	(None, 37, 32)	3,104
input_layer_5 (InputLayer)	(None, 41, 1)	0
max_pooling1d_2 (MaxPooling1D)	(None, 18, 32)	0
flatten_5 (Flatten)	(None, 41)	0
dropout_14 (Dropout)	(None, 18, 32)	0
dense_12 (Dense)	(None, 256)	10,752
conv1d_12 (Conv1D)	(None, 16, 64)	6,208
dropout_16 (Dropout)	(None, 256)	0
conv1d_13 (Conv1D)	(None, 14, 64)	12,352
dense_13 (Dense)	(None, 128)	32,896
max_pooling1d_3 (MaxPooling1D)	(None, 7, 64)	0
dropout_17 (Dropout)	(None, 128)	0
dropout_15 (Dropout)	(None, 7, 64)	0
dense_14 (Dense)	(None, 64)	8,256
flatten_4 (Flatten)	(None, 448)	0
dropout_18 (Dropout)	(None, 64)	0
concatenate_1 (Concatenate)	(None, 512)	0
dense_15 (Dense)	(None, 256)	131,328
dropout_19 (Dropout)	(None, 256)	0
dense_16 (Dense)	(None, 2)	514

*Figure 12: Hybrid Model Summary*

## Chapter 5

# Results and Discussion

### 5.1 PERFORMANCE METRICS EXPLANATION

#### Confusion matrix

A confusion matrix is a performance evaluation tool used in machine learning that summarizes the performance of a classification model by tabulating true positive, true negative, false positive, and false negative predictions. It helps assess the accuracy and effectiveness of the model's predictions. The matrix is based on the concepts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). It provides a granular view of a model's performance across different classes.

#### Structure

- **True Positives (TP):** Instances where the model correctly predicts a positive class when it is indeed positive. Consider a cancer diagnostic model: a true positive would occur when the model correctly identifies a patient with cancer as having the disease. TP is a vital measure of the model's ability to recognize positive instances accurately.
- **True Negatives (TN):** Instances where the model correctly predicts a negative class when it is indeed negative. Continuing the medical analogy, a true negative would be when the model correctly identifies a healthy patient as not having the disease. TN reflects the model's proficiency in recognizing negative instances.
- **False Positives (FP):** Instances where the model incorrectly predicts a positive class when it should have been negative. In the medical scenario, a false positive would mean the model wrongly indicates a patient has the disease when they are, in fact, healthy. FP illustrates instances where the model exhibits overconfidence in predicting positive outcomes.
- **False Negatives (FN):** Instances where the model incorrectly predicts a negative class when it should have been positive. In the medical context, a false negative would be when the model fails to detect a disease in a patient who actually has it. FN highlights situations where the model fails to capture actual positive instances.

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) <b>Type II Error</b>	<b>Sensitivity</b> $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) <b>Type I Error</b>	True Negative (TN)	<b>Specificity</b> $\frac{TN}{(TN + FP)}$
		<b>Precision</b> $\frac{TP}{(TP + FP)}$	<b>Negative Predictive Value</b> $\frac{TN}{(TN + FN)}$	<b>Accuracy</b> $\frac{TP + TN}{(TP + TN + FP + FN)}$

**Figure 13: Confusion matrix**

The confusion matrix serves as the basis for calculating essential evaluation metrics that offer nuanced insights into a model's performance:

- **Accuracy:** Accuracy quantifies the ratio of correct predictions (TP and TN) to the total number of predictions. While informative, this metric can be misleading when classes are imbalanced.
- **Precision:** Precision evaluates the proportion of true positive predictions among all positive predictions ( $TP / (TP + FP)$ ). This metric is crucial when the cost of false positives is high.
- **Recall (Sensitivity or True Positive Rate):** Recall measures the ratio of true positive predictions to the actual number of positive instances ( $TP / (TP + FN)$ ). This metric is significant when missing positive instances is costly.
- **Specificity (True Negative Rate):** Specificity calculates the ratio of true negative predictions to the actual number of negative instances ( $TN / (TN + FP)$ ). This metric is vital when the emphasis is on accurately identifying negative instances.
- **F1-Score:** The F1-Score strikes a balance between precision and recall, making it useful when both false positives and false negatives carry similar importance.
- **Support:** Support is a representation of how many actual examples of each class there are in the dataset.

- **ROC Curve:** The Receiver Operating Characteristic (ROC) curve plots the true positive rate (TPR) against the false positive rate (FPR) for different threshold levels. It makes it easier to see how the classifier balances sensitivity and specificity at various threshold settings.
- **AUC (Area Under the ROC Curve):** AUC, or area under the ROC curve, provides an overall performance measure across all possible categorization levels and is used to quantify the area under the ROC curve. Better classifier performance is indicated by a higher number, which runs from 0 to 1.

## Applications of the Confusion Matrix

The confusion matrix has applications across various fields:

- **Model Evaluation:** The primary application of the confusion matrix is to evaluate the performance of a classification model. It provides insights into the model's accuracy, precision, recall, and F1-score.
- **Medical Diagnosis:** The confusion matrix finds extensive use in medical fields for diagnosing diseases based on tests or images. It aids in quantifying the accuracy of diagnostic tests and identifying the balance between false positives and false negatives.
- **Fraud Detection:** Banks and financial institutions use confusion matrices to detect fraudulent transactions by showcasing how AI algorithms help identify patterns of fraudulent activities.
- **Natural Language Processing (NLP):** NLP models use confusion matrices to evaluate sentiment analysis, text classification, and named entity recognition.
- **Customer Churn Prediction:** Confusion matrices play a pivotal role in predicting customer churn and show how AI-driven models use historical data to anticipate and mitigate customer attrition.
- **Image and Object Recognition:** Confusion matrices assist in training models to identify objects in images, enabling technologies like self-driving cars and facial recognition systems.
- **A/B Testing:** A/B testing is crucial for optimizing user experiences. Confusion matrices help analyze the results of A/B tests, enabling data-driven decisions in user engagement strategies.

## **Multiclass Classification with Example:**

For multi-class classification, the confusion matrix extends to an  $N \times N$  matrix, where  $N$  is the number of classes. Each cell in the matrix represents the count of instances for a specific pair of actual and predicted classes.

## **Detailed Explanation of Terms**

### **True Positives (TP)**

- Definition: True Positives are instances where the model correctly predicts the positive class.  
Example: In a medical diagnosis scenario for detecting a particular disease, if a patient has the disease (actual positive) and the model correctly predicts that the patient has the disease (predicted positive), this instance is a True Positive.
- Significance: High TP values indicate that the model is effective in correctly identifying positive instances.

### **True Negatives (TN)**

- Definition: True Negatives are instances where the model correctly predicts the negative class.  
Example: In the same medical diagnosis scenario, if a patient does not have the disease (actual negative) and the model correctly predicts that the patient does not have the disease (predicted negative), this instance is a True Negative.
- Significance: High TN values indicate that the model is effective in correctly identifying negative instances. However, in multi-class settings, TN is less commonly used due to the focus on individual class performance.

### **False Positives (FP)**

- Definition: False Positives are instances where the model incorrectly predicts the positive class.  
Example: If a patient does not have the disease (actual negative) but the model predicts that the patient has the disease (predicted positive), this instance is a False Positive.
- Significance: High FP values indicate a problem with the model's specificity, meaning it is incorrectly identifying negative instances as positive. In critical applications like medical diagnosis, a high FP rate can lead to unnecessary treatments and anxiety.

### False Negatives (FN)

- Definition: False Negatives are instances where the model incorrectly predicts the negative class.

Example: If a patient has the disease (actual positive) but the model predicts that the patient does not have the disease (predicted negative), this instance is a False Negative.

- Significance: High FN values indicate a problem with the model's sensitivity or recall, meaning it is incorrectly identifying positive instances as negative. In critical applications, a high FN rate can lead to missed diagnoses and lack of necessary treatment.

## 5.2 RESULTS

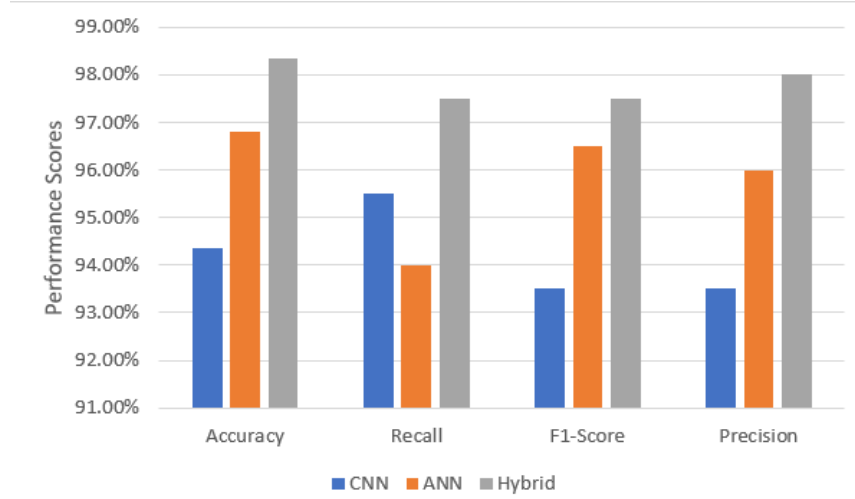
Model	Accuracy	Loss	Val-accuracy
CNN	94.36%	2.009%	95.53%
ANN	96.82%	2.2037%	94.63%
Hybrid	97.65%	2.6534%	98.34%

*Table 3: Results of the proposed system for 40 epochs*

For all our models we tried different epoch sizes i.e. 20, 40 and 60, and among those, 40 epochs showed better fit. Table-3 shows the accuracy, loss, validation accuracy, and validation loss evaluation of the proposed models for 40 epochs from the dataset: Network Intrusion Detection [26]. We ran all our models on same device having Intel core i5 processor with 8 cores and NVIDIA GeForce GTX 6090. The CNN model demonstrated 94.36% accuracy with 95.53% validation accuracy although it resulted in higher validation loss. The ANN delivered marginal higher accuracy of 96.82% but showed a 94.63% validation accuracy which indicates an overfitting problem. The hybrid model delivered superior performance with 98.34% accuracy and 97.65% validation accuracy compared to standalone models due to its better learning ability. The higher model complexity in this system seems to lead to a 4.58% training loss quantity.

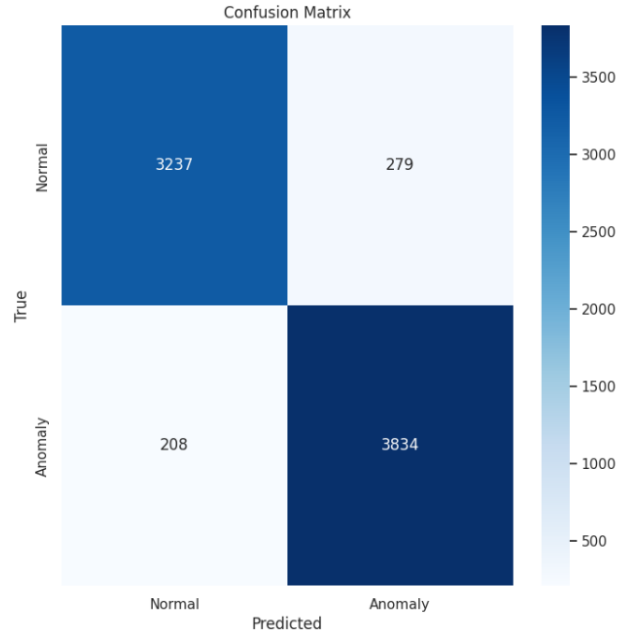
Model	Class type	Precision	Recall	F1-score
CNN	Normal	92%	94%	93%
	Anomaly	95%	93%	94%
ANN	Normal	98%	93%	96%
	Anomaly	94%	95%	97%
Hybrid	Normal	100%	95%	97%
	Anomaly	96%	100%	98%

**Table 4: Performance metrics of the Proposed system**



**Figure 14: Comparison of Results**

Table-4 and Fig.14 shows the performance analysis of CNN, ANN, and Hybrid models through precision, recall and F1-score measurements for Normal and Anomaly identification. The CNN model shows 92%, 94% and 93% of precision, recall and F1-score respectively for normal class and 95%, 93% and 94% for anomaly class. ANN demonstrates 98%, 93% and 94% of precision, recall and F1-score respectively for normal class and 94%, 95% and 97% for anomaly class. The Hybrid model surpasses both previous models with its precision reaching 100%, 95% recall and 97% F1-score for normal instances while achieving 96% precision, 100% recall and 98% F1-score for anomalies.



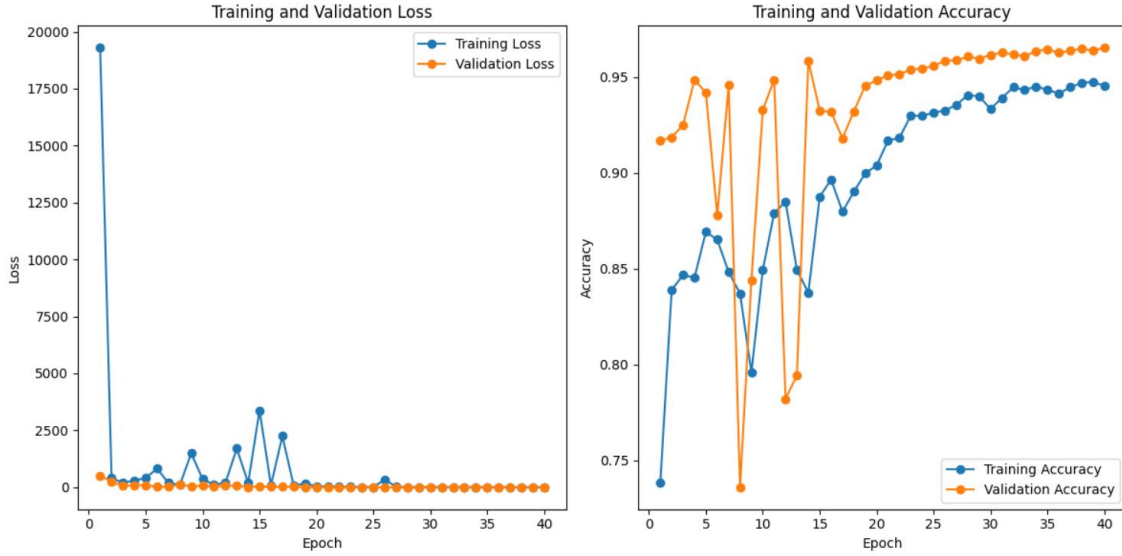
**Figure 15: Confusion Matrix**

The confusion matrix evaluates how well a classification model diagnoses Normal and Anomaly cases. Where the model successfully detected Normal cases among a total of 6661 instances it obtained 3,237 True Positive results while achieving 3,834 True Negative results. Two hundred seventy-nine legitimate Normal cases were mistakenly identified as Anomalies while two hundred eight actual cases of Anomalies were mistakenly classified as Normal. The model presents good overall performance through its numerous correct classifications yet displays several misclassifications shown through its false positive and false negative values.



**Figure 16: Performance of CNN**





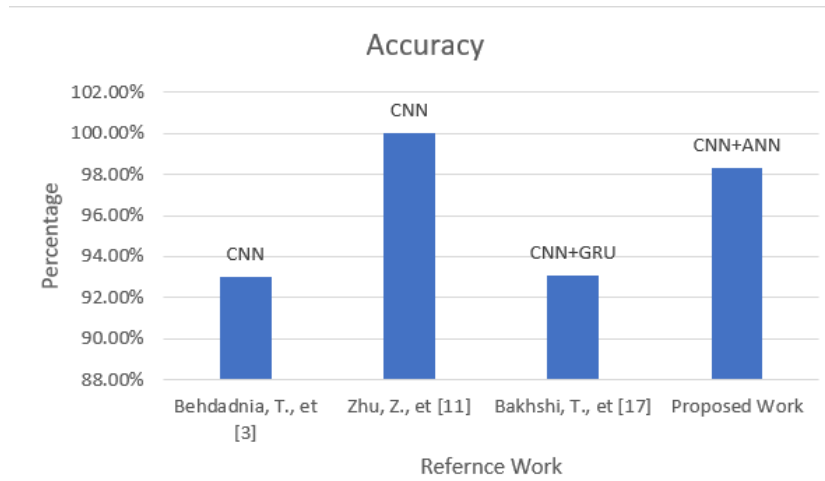
**Figure 17: Performance of ANN**



**Figure 18: Performance of Hybrid**

Fig.18 graph shows accuracy graph of hybrid model with validation accuracy as red which is 98.34% and training accuracy as blue which is 97.65% across 40 epochs. Fig.16 graph shows loss graph of hybrid model with validation loss as red which is 98.34% and training loss as blue which is 2.534% across 40 epochs. The authors Behdadnia, T., et al in [3] used CNN model which achieved 93% accuracy, and the authors in Zhu, Z., et al [11] also used CNN model which shows signs of overfitting as it is achieving 100% accuracy, and the authors in Bakhshi, T., et al [17] used a hybrid of CNN+GRU(Gated Recurrent Unit) which achieved 93.10% accuracy, comparative with all these researches our proposed hybrid model of CNN+ANN shows more efficiency, reduces overfitting and increased

performance as shown in Fig.19. The training accuracy shows rapid growth starting from zero up to approximately 98.34% before experiencing both small improvements and small declines. The validation accuracy levels stay stable at 97.65% after exceeding 95% with negligible ups and downs. The model demonstrates strong performance in test data although training accuracy fails to match it which might result from regularization techniques or the model achieving widespread generalization as opposed to overfitting.



***Figure 19: Comparison of Proposed work with existing studies***

## Chapter 6

# Conclusion

### 6.1 CONCLUSION

The proposed hybrid model of ANN and CNN system achieve higher accuracy and shows much more promising results for anomaly detection in encryption traffic. The data preprocessing module assures that no null values existed and we performed label encoding to change categorical data to numerical data. To remove class imbalance and balance the dataset we used SMOTE technique. The ANN and CNN architectures were refined with added hidden layers to increase the efficiency, and we used ReLU and GeLU activation functions to update weights for better performance. We also used Adam and RMSprop optimizers for the model to learn more effectively. Sigmoid activation function is used to venture more accurate results for binary classification. Both of the models did give very good results with 96.82% and 94.36% accuracies respectively, but for more compelling and promising outputs we combined them both into a hybrid model, which gave us excellent results with 98.34% accuracy. Comparatively the hybrid model is more promising, efficient and effective than refined ANN and CNN architectures. The proposed hybrid model also achieved higher precision, recall and F1-score of 98%, 96.5% and 97.5% respectively.

## REFERENCES

- [1]. Wu, Z., Li, H., Qian, Y., Hua, Y., & Gan, H. (2024). Poison-resilient anomaly detection: Mitigating poisoning attacks in semi-supervised encrypted traffic anomaly detection. *IEEE Transactions on Network Science and Engineering*.
- [2]. Xing, J., & Wu, C. (2020, July). Detecting anomalies in encrypted traffic via deep dictionary learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 734-739). IEEE.
- [3]. Behdadnia, T., Deconinck, G., Ozkan, C., & Singelee, D. (2023, October). Encrypted Traffic Classification for Early-Stage Anomaly Detection in Power Grid Communication Network. In *2023 IEEE PES Innovative Smart Grid Technologies Europe (ISGT EUROPE)* (pp. 1-6). IEEE.
- [4]. Yu, T., Zou, F., Li, L., & Yi, P. (2019, October). An encrypted malicious traffic detection system based on neural network. In *2019 international conference on cyber-enabled distributed computing and knowledge discovery (CyberC)* (pp. 62-70). IEEE.
- [5]. Lv, G., Yang, R., Wang, Y., & Tang, Z. (2020, August). Network encrypted traffic classification based on secondary voting enhanced random forest. In *2020 IEEE 3rd International Conference on Computer and Communication Engineering Technology (CCET)* (pp. 60-66). IEEE.
- [6]. Kopp, M., Grill, M., & Kohout, J. (2018, December). Community-based anomaly detection. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)* (pp. 1-6). IEEE.
- [7]. Han, S., Wu, Q., Zhang, H., & Qin, B. (2022, July). Light-weight unsupervised anomaly detection for encrypted malware traffic. In *2022 7th IEEE International Conference on Data Science in Cyberspace (DSC)* (pp. 206-213). IEEE.
- [8]. Shen, M., Ye, K., Liu, X., Zhu, L., Kang, J., Yu, S., ... & Xu, K. (2022). Machine learning-powered encrypted network traffic analysis: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 25(1), 791-824.
- [9]. Meghdouri, F., Vázquez, F. I., & Zseby, T. (2020, October). Cross-layer profiling of encrypted network data for anomaly detection. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)* (pp. 469-478). IEEE.
- [10]. Wang, Z., Wang, P., & Sun, Z. (2022, December). SDN traffic anomaly detection method based on convolutional autoencoder and federated learning. In *GLOBECOM 2022-2022 IEEE Global Communications Conference* (pp. 4154-4160). IEEE.
- [11]. Zhu, Z., Zhou, H., Yang, Q., Wang, C., & Li, Z. (2022, December). Anomaly Detection in Encrypted Identity Resolution Traffic based on Machine Learning. In *2022*

IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS) (pp. 264-275). IEEE.

[12]. Ucci, D., Sobrero, F., Bisio, F., & Zorzino, M. (2021, December). Near-real-time anomaly detection in encrypted traffic using machine learning techniques. In 2021 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 01-08). IEEE.

[13]. Pratiwi, M., & Choi, Y. H. (2024, September). Context-Aware Anomaly-based Detection for Ransomware using Multivariate Feature. In 2024 IEEE Conference on Communications and Network Security (CNS) (pp. 1-9). IEEE.

[14]. Wingarz, T., See, A., Gondesen, F., & Fischer, M. (2024, September). Privacy-preserving Network Anomaly Detection on Homomorphically Encrypted Data. In 2024 IEEE Conference on Communications and Network Security (CNS) (pp. 1-9). IEEE.

[15]. Wang, J., He, T., He, G., Zhu, H., Xu, B., & Zhang, L. (2023, December). Cooperative Detection of Camouflaged Malicious TLS Traffic. In 2023 Eleventh International Conference on Advanced Cloud and Big Data (CBD) (pp. 116-121). IEEE.

[16]. Ashraf, J., Bakhshi, A. D., Moustafa, N., Khurshid, H., Javed, A., & Beheshti, A. (2020). Novel deep learning-enabled LSTM autoencoder architecture for discovering anomalous events from intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 22(7), 4507-4518.

[17]. Tian, Z., Cui, L., Liang, J., & Yu, S. (2022). A comprehensive survey on poisoning attacks and countermeasures in machine learning. *ACM Computing Surveys*, 55(8), 1-35.

[18]. Anderson, B., & McGrew, D. (2016, October). Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM workshop on artificial intelligence and security* (pp. 35-46).

[19]. Shekhawat, A. S., Di Troia, F., & Stamp, M. (2019). Feature analysis of encrypted malicious traffic. *Expert Systems with Applications*, 125, 130-141.

[20]. Velan, P., Čermák, M., Čeleda, P., & Drašar, M. (2015). A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 25(5), 355-374.

[21]. Rezaei, S., & Liu, X. (2019). Deep learning for encrypted traffic classification: An overview. *IEEE communications magazine*, 57(5), 76-81.

[22]. Bakhshi, T., & Ghita, B. (2021). Anomaly detection in encrypted internet traffic using hybrid deep learning. *Security and Communication Networks*, 2021(1), 5363750.

[23]. Long, G., & Zhang, Z. (2023). Deep encrypted traffic detection: An anomaly detection framework for encryption traffic based on parallel automatic feature extraction. *Computational Intelligence and Neuroscience*, 2023(1), 3316642.

[24]. Kim, M. G., & Kim, H. (2024). Anomaly Detection in Imbalanced Encrypted Traffic with Few Packet Metadata-Based Feature Extraction. *CMES-Computer Modeling in*

Engineering & Sciences, 141(1).

[25]. Lee, I., Roh, H., & Lee, W. (2020, July). Encrypted malware traffic detection using incremental learning. In IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 1348-1349). IEEE.

[26] <https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection>

## Appendices

### SOURCE CODE

#### Importing required libraries

```
✓ [1] import pandas as pd
1s #list of useful imports that I will use
%matplotlib inline
import os

import matplotlib.pyplot as plt
import pandas as pd
import cv2
import numpy as np
from glob import glob
import seaborn as sns
import random
import pickle

from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
```

#### Reading the csv file

```
✓ [2] data=pd.read_csv(r"/content/Anamoly_detection.csv")
0s
```

#### First five rows in the dataset

```
✓ [3] data.head()
0s
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_sam
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	0.03	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	0.60	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	0.05	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	0.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	0.00	

5 rows × 42 columns

```
✓ [3] data.head()
0s
```

st_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	class
0.03	0.17	0.00	0.00	0.00	0.05	0.00	normal
0.60	0.88	0.00	0.00	0.00	0.00	0.00	normal
0.05	0.00	0.00	1.00	1.00	0.00	0.00	anomaly
0.00	0.03	0.04	0.03	0.01	0.00	0.00	normal
0.00	0.00	0.00	0.00	0.00	0.00	0.00	normal

## Last five rows in the dataset

```
[69] data.tail()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_hos
25187	0	tcp	exec	RSTO	0	0	0	0	0	0	...	7	0.03		0.06
25188	0	tcp	ftp_data	SF	334	0	0	0	0	0	...	39	1.00		0.00
25189	0	tcp	private	REJ	0	0	0	0	0	0	...	13	0.05		0.07
25190	0	tcp	nnsnp	SO	0	0	0	0	0	0	...	20	0.08		0.06
25191	0	tcp	finger	SO	0	0	0	0	0	0	...	49	0.19		0.03

5 rows x 42 columns

```
[69] data.tail()
```

	st_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	class
	0.06	0.00	0.00	0.0	0.0	1.0	1.0	anomaly
	0.00	1.00	0.18	0.0	0.0	0.0	0.0	anomaly
	0.07	0.00	0.00	0.0	0.0	1.0	1.0	anomaly
	0.06	0.00	0.00	1.0	1.0	0.0	0.0	anomaly
	0.03	0.01	0.00	1.0	1.0	0.0	0.0	anomaly

Size of the dataset.

```
[70] data.shape
```

(25192, 42)

## Data Cleaning

Checking for null vaules

```
[71] data.isnull().sum()
```

duration	0
protocol_type	0
service	0
flag	0
src_bytes	0
dst_bytes	0
land	0
wrong_fragment	0
urgent	0
hot	0
num_failed_logins	0
logged_in	0
num_compromised	0
root_shell	0
su_attempted	0
num_root	0
num_file_creations	0
num_shells	0
num_access_files	0
num_outbound_cmds	0
is_host_login	0
is_guest_login	0
count	0
srv_count	0
serror_rate	0
srv_serror_rate	0
rerror_rate	0
srv_rerror_rate	0
same_srv_rate	0
diff_srv_rate	0
srv_diff_host_rate	0
dst_host_count	0
dst_host_srv_count	0
dst_host_same_srv_rate	0
dst_host_diff_srv_rate	0
dst_host_same_src_port_rate	0
dst_host_srv_diff_host_rate	0
dst_host_serror_rate	0
dst_host_srv_serror_rate	0
dst_host_rerror_rate	0
dst_host_srv_rerror_rate	0
class	0

dtype: int64



## Checking the data types

✓ [72] data.dtypes	
0s	
↕	0
duration	int64
protocol_type	object
service	object
flag	object
src_bytes	int64
dst_bytes	int64
land	int64
wrong_fragment	int64
urgent	int64
hot	int64
num_failed_logins	int64
logged_in	int64
num_compromised	int64
root_shell	int64
su_attempted	int64
num_root	int64
num_file_creations	int64
num_shells	int64
num_access_files	int64
num_outbound_cmds	int64
is_host_login	int64
is_guest_login	int64
count	int64
srv_count	int64
serror_rate	float64
srv_serror_rate	float64
rerror_rate	float64
srv_rerror_rate	float64
same_srv_rate	float64
diff_srv_rate	float64
srv_diff_host_rate	float64
dst_host_count	int64
dst_host_srv_count	int64
dst_host_same_srv_rate	float64
dst_host_diff_srv_rate	float64
dst_host_same_src_port_rate	float64
dst_host_srv_diff_host_rate	float64
dst_host_serror_rate	float64
dst_host_srv_serror_rate	float64
dst_host_rerror_rate	float64
dst_host_srv_rerror_rate	float64
class	object
	dtype: object

## Class Distribution

✓	▶ data['class'].value_counts()
0s	
↕	count
	class
	normal 13449
	anomaly 11743
	dtype: int64

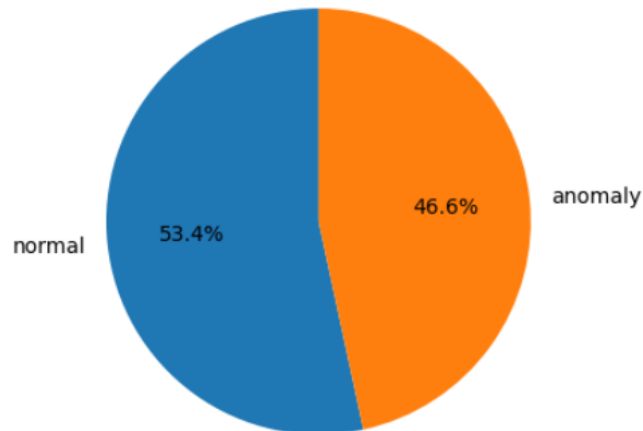
```

[74] import matplotlib.pyplot as plt
      class_counts = data['class'].value_counts()
      plt.figure(figsize=(4, 4))
      plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%', startangle=90)
      plt.title('Distribution of Classes')
      plt.axis('equal')
      plt.show()

```



Distribution of Classes



## Services

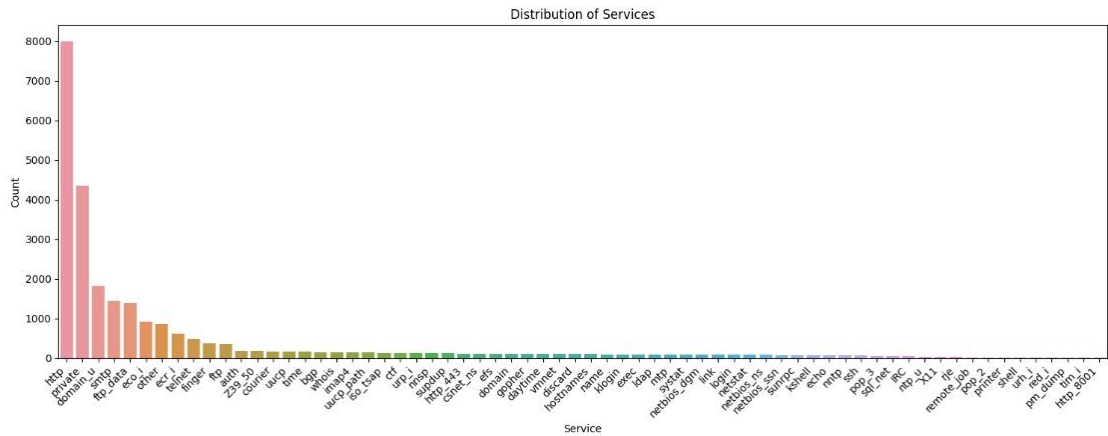
```

[75] import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns

      if 'service' in data.columns:
          service_counts = data['service'].value_counts()

          plt.figure(figsize=(15, 6))
          sns.barplot(x=service_counts.index, y=service_counts.values)
          plt.xlabel("Service")
          plt.ylabel("Count")
          plt.title("Distribution of Services")
          plt.xticks(rotation=45, ha="right")
          plt.tight_layout()
          plt.show()
      else:
          print("'service' column not found in the DataFrame. Please replace 'service' with the correct column name.")

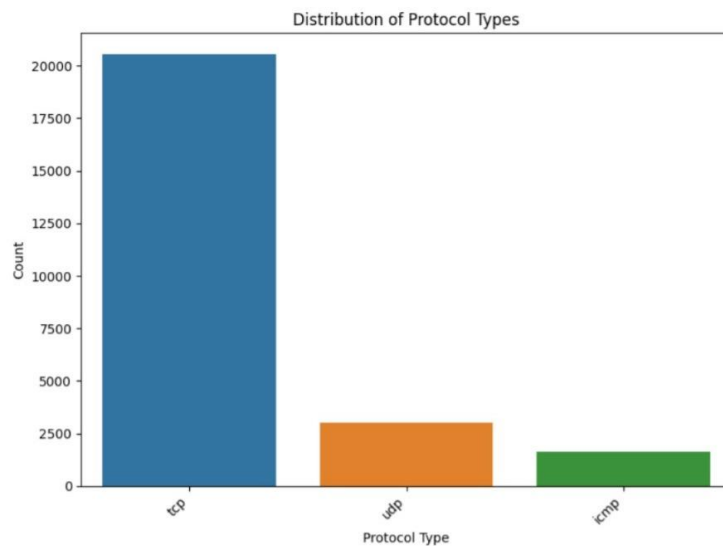
```



## Protocol

```
[76] if 'protocol_type' in data.columns:
      protocol_counts = data['protocol_type'].value_counts()

      plt.figure(figsize=(8, 6))
      sns.countplot(x='protocol_type', data=data)
      plt.xlabel("Protocol Type")
      plt.ylabel("Count")
      plt.title("Distribution of Protocol Types")
      plt.xticks(rotation=45, ha="right")
      plt.tight_layout()
      plt.show()
    else:
      print("'protocol_type' column not found in the DataFrame.")
```



Checking for duplicates

```
[77] data = data.drop_duplicates()
```

No duplicates found.

Converting categorical to numerical

```
✓ 0s ▶ from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for col in ['protocol_type', 'service', 'flag', 'class']:
    if col in data.columns:
        data[col] = le.fit_transform(data[col])
    else:
        print(f"Column '{col}' not found in the DataFrame.")

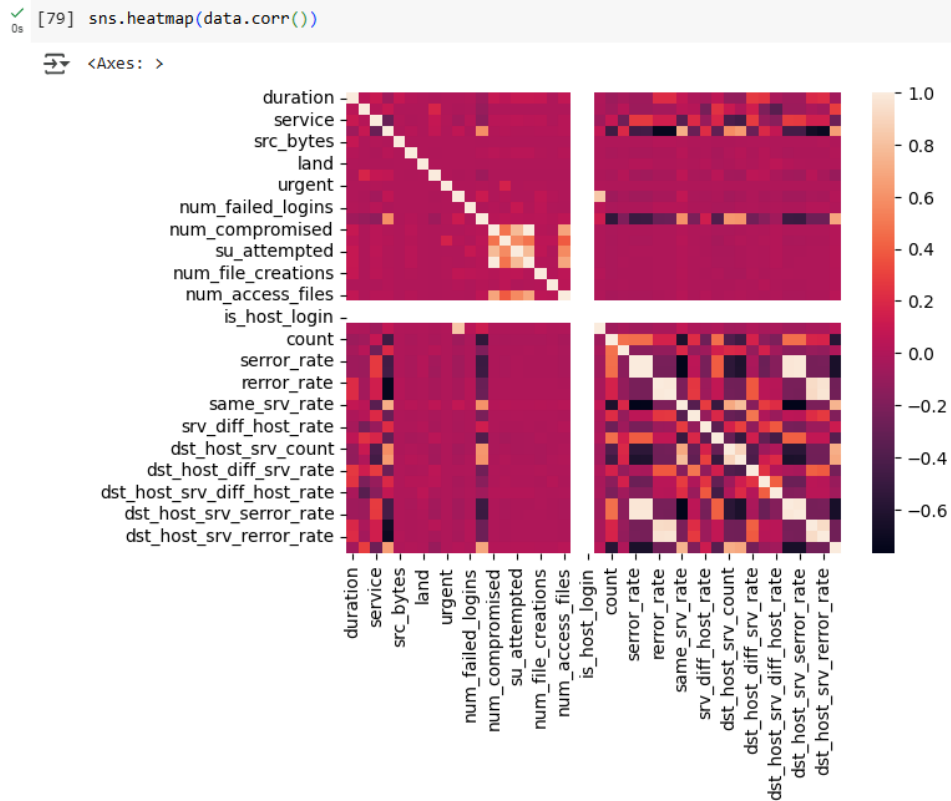
data.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same_src_port_rate	...
0	0	1	19	9	491	0	0	0	0	0	...	25	0.17	0.03	0.17	...
1	0	2	41	9	146	0	0	0	0	0	...	1	0.00	0.60	0.88	...
2	0	1	46	5	0	0	0	0	0	0	...	26	0.10	0.05	0.00	...
3	0	1	22	9	232	8153	0	0	0	0	...	255	1.00	0.00	0.03	...
4	0	1	22	9	199	420	0	0	0	0	...	255	1.00	0.00	0.00	...

5 rows x 42 columns

dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	class
0.03	0.17	0.00	0.00	0.00	0.05	0.00	1
0.60	0.88	0.00	0.00	0.00	0.00	0.00	1
0.05	0.00	0.00	1.00	1.00	0.00	0.00	0
0.00	0.03	0.04	0.03	0.01	0.00	0.01	1
0.00	0.00	0.00	0.00	0.00	0.00	0.00	1

## HeatMap



After converting all the values in the dataset are in int and float

[80] data.dtypes

duration	int64	is_host_login	int64
protocol_type	int64	is_guest_login	int64
service	int64	count	int64
flag	int64	srv_count	int64
src_bytes	int64	error_rate	float64
dst_bytes	int64	srv_error_rate	float64
land	int64	error_rate	float64
wrong_fragment	int64	srv_error_rate	float64
urgent	int64	same_srv_rate	float64
hot	int64	diff_srv_rate	float64
num_failed_logins	int64	srv_diff_host_rate	float64
logged_in	int64	dst_host_count	int64
num_compromised	int64	dst_host_srv_count	int64
root_shell	int64	dst_host_same_srv_rate	float64
su_attempted	int64	dst_host_diff_srv_rate	float64
num_root	int64	dst_host_same_src_port_rate	float64
num_file_creations	int64	dst_host_srv_diff_host_rate	float64
num_shells	int64	dst_host_error_rate	float64
num_access_files	int64	dst_host_srv_error_rate	float64
num_outbound_cmds	int64	dst_host_error_rate	float64
		dst_host_srv_error_rate	float64
		dst_host_error_rate	float64
		dst_host_srv_error_rate	float64
		class	int64

dtype: object

## Removing class imbalance using SMOTE

```
✓ [81] import pandas as pd
0s      from imblearn.over_sampling import SMOTE

X = data.drop('class', axis=1)
y = data['class']
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

data_resampled = pd.DataFrame(X_resampled, columns=X.columns)
data_resampled['class'] = y_resampled

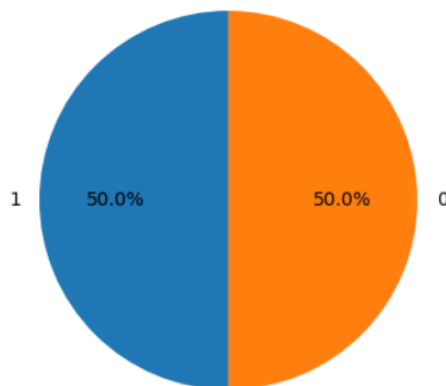
print(data_resampled['class'].value_counts())
```

```
↕ class
1    13449
0    13449
Name: count, dtype: int64
```

```
✓ [82] import matplotlib.pyplot as plt
0s      class_counts_resampled = data_resampled['class'].value_counts()

plt.figure(figsize=(4, 4))
plt.pie(class_counts_resampled, labels=class_counts_resampled.index, autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Classes after SMOTE')
plt.axis('equal')
plt.show()
```

↕ Distribution of Classes after SMOTE



After resampling

```
✓ [83] data_resampled['class'].value_counts()  
0s
```

```
↩  
count  
  
class  
1    13449  
0    13449  
  
dtype: int64
```

Feature extraction

```
✓ [84] X = data.drop('class', axis=1)  
      y = data['class']  
0s
```

```
[85] X.head()  
↩  
duration  protocol_type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_count  dst_host_srv_count  dst_host_same_srv_rate  dst_host_diff_srv_rate  
0         0             1       19     9         491         0     0             0         0     0  ...           150             25             0.17             0.03  
1         0             2       41     9         146         0     0             0         0     0  ...           255              1             0.00             0.60  
2         0             1       46     5          0          0     0             0         0     0  ...           255             26             0.10             0.05  
3         0             1       22     9         232       8153     0             0         0     0  ...            30            255             1.00             0.00  
4         0             1       22     9         199        420     0             0         0     0  ...           255            255             1.00             0.00  
5 rows x 41 columns  
  
dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  dst_host_serror_rate  dst_host_srv_serror_rate  dst_host_rerror_rate  dst_host_srv_rerror_rate  
0.17                        0.00                        0.00                        0.00                        0.05                        0.00  
0.88                        0.00                        0.00                        0.00                        0.00                        0.00  
0.00                        0.00                        1.00                        1.00                        0.00                        0.00  
0.03                        0.04                        0.03                        0.01                        0.00                        0.01  
0.00                        0.00                        0.00                        0.00                        0.00                        0.00
```

```
✓ [86] y.tail()  
0s  
↩  
class  
25187    0  
25188    0  
25189    0  
25190    0  
25191    0  
  
dtype: int64
```

## Train and test split ratio

```
[87] from sklearn.model_selection import train_test_split
      X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

[88] X\_train

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_count	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate
741	0	1	46	5	0	0	0	0	0	0	...	255	1	0.00	0.08
411	0	1	16	2	0	0	0	0	0	0	...	255	4	0.02	0.06
17841	0	1	3	1	0	0	0	0	0	0	...	255	14	0.05	0.07
20962	0	0	13	9	8	0	0	0	0	0	...	2	152	1.00	0.00
17790	0	1	46	5	0	0	0	0	0	0	...	94	9	0.10	0.12
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21575	0	2	11	9	46	71	0	0	0	0	...	255	254	1.00	0.01
5390	1	1	51	9	1601	326	0	0	0	0	...	92	52	0.57	0.05
860	0	1	22	1	0	0	0	0	0	0	...	255	6	0.02	0.07
15795	0	1	22	9	309	4281	0	0	0	0	...	21	255	1.00	0.00
23654	0	1	46	5	0	0	0	0	0	0	...	255	4	0.02	0.07

17634 rows x 41 columns

dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate
0.08	0.00	0.00	1.0	1.0	0.0	0.0
0.06	0.00	0.00	0.0	0.0	1.0	1.0
0.07	0.00	0.00	0.0	0.0	1.0	1.0
0.00	1.00	0.50	0.0	0.0	0.0	0.0
0.12	0.01	0.00	1.0	1.0	0.0	0.0
...	...	...	...	...	...	...
0.01	0.00	0.00	0.0	0.0	0.0	0.0
0.05	0.01	0.00	0.0	0.0	0.0	0.0
0.07	0.00	0.00	0.0	0.0	1.0	1.0
0.00	0.05	0.05	0.0	0.0	0.0	0.0
0.07	0.00	0.00	1.0	1.0	0.0	0.0

[ ] Y\_train

	class
741	0
411	0
17841	0
20962	0
17790	0
...	...
21575	1
5390	1
860	0
15795	1
23654	0

17634 rows x 1 columns

dtype: int64



✓ [91] Y\_train.shape

0s

⇒ (17634, 2)

✓ [92] Y\_test.shape

0s

⇒ (7558, 2)

✓ [93] Y\_train[741]

0s

⇒ array([1., 0.])

✓ [96] X\_test = X\_test.values.reshape((len(X\_test),41,1))  
X\_train = X\_train.values.reshape((len(X\_train),41,1))

0s

✓ [97] X\_test.shape

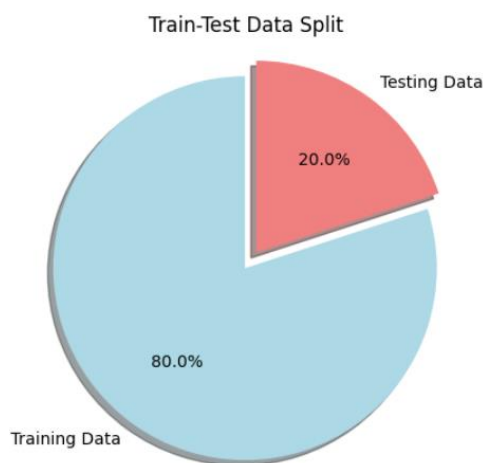
0s

⇒ (7558, 41, 1)

✓ [98] X\_train.shape

0s

⇒ (17634, 41, 1)



## CNN Model

### Importing the required libraries and layers for CNN

```
from tensorflow.keras.utils import to_categorical # for one-hot encoding
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv1D, MaxPooling1D, GlobalAvgPool1D, GlobalMaxPooling1D
from tensorflow.keras.optimizers import RMSprop, Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

### Splitting the dataset into train and test

```
✓ [87] from sklearn.model_selection import train_test_split
0s      X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

### Model Building

```
[ ] cnn_model = Sequential()

cnn_model.add(Conv1D(filters = 32, kernel_size = 3, activation = 'gelu', input_shape = (41,1)))
cnn_model.add(Conv1D(filters = 32, kernel_size = 3, activation = 'gelu'))
cnn_model.add(Dropout(0.4))

cnn_model.add(Conv1D(filters = 32, kernel_size = 3, activation = 'gelu'))
cnn_model.add(Conv1D(filters = 32, kernel_size = 3, activation = 'gelu'))
cnn_model.add(Dropout(0.4))

cnn_model.add(Conv1D(filters = 64, kernel_size = 3, activation = 'gelu'))
cnn_model.add(Conv1D(filters = 64, kernel_size = 3, activation = 'gelu'))
cnn_model.add(Dropout(0.4))

cnn_model.add(Flatten())
cnn_model.add(Dense(256, activation = "gelu"))
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(2, activation = "softmax"))
cnn_model.summary()
```

```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_8"

```

Layer (type)	Output Shape	Param #
conv1d_28 (Conv1D)	(None, 39, 32)	128
conv1d_29 (Conv1D)	(None, 37, 32)	3,104
dropout_52 (Dropout)	(None, 37, 32)	0
conv1d_30 (Conv1D)	(None, 35, 32)	3,104
conv1d_31 (Conv1D)	(None, 33, 32)	3,104
dropout_53 (Dropout)	(None, 33, 32)	0
conv1d_32 (Conv1D)	(None, 31, 64)	6,208
conv1d_33 (Conv1D)	(None, 29, 64)	12,352
dropout_54 (Dropout)	(None, 29, 64)	0
flatten_16 (Flatten)	(None, 1856)	0
dense_50 (Dense)	(None, 256)	475,392
dropout_55 (Dropout)	(None, 256)	0
dense_51 (Dense)	(None, 2)	514

**Total params:** 503,906 (1.92 MB)  
**Trainable params:** 503,906 (1.92 MB)  
**Non-trainable params:** 0 (0.00 B)

## Training the model

```

cnn_model.compile(optimizer = 'rmsprop' , loss = "binary_crossentropy", metrics=["accuracy"])

```

```

[ ] history2 = cnn_model.fit(X_train,Y_train, batch_size= 128,
                             epochs = 40, validation_data = (X_test,Y_test))

```

```

Epoch 1/40
158/158 — 13s 41ms/step - accuracy: 0.6344 - loss: 2978.9761 - val_accuracy: 0.8859 - val_loss: 174.2031
Epoch 2/40
158/158 — 1s 6ms/step - accuracy: 0.7555 - loss: 302.2926 - val_accuracy: 0.9030 - val_loss: 146.0607
Epoch 3/40
158/158 — 1s 7ms/step - accuracy: 0.7711 - loss: 214.8839 - val_accuracy: 0.8710 - val_loss: 77.1681
Epoch 4/40
158/158 — 1s 5ms/step - accuracy: 0.7697 - loss: 191.5165 - val_accuracy: 0.8742 - val_loss: 50.6633
Epoch 5/40
158/158 — 1s 5ms/step - accuracy: 0.8045 - loss: 663.5130 - val_accuracy: 0.7934 - val_loss: 96.8147
Epoch 6/40
158/158 — 1s 5ms/step - accuracy: 0.8033 - loss: 172.5798 - val_accuracy: 0.8579 - val_loss: 46.1696
Epoch 7/40
158/158 — 1s 5ms/step - accuracy: 0.8018 - loss: 1667.0009 - val_accuracy: 0.8597 - val_loss: 31.9553
Epoch 8/40
158/158 — 1s 5ms/step - accuracy: 0.8188 - loss: 204.3563 - val_accuracy: 0.8805 - val_loss: 87.9802
Epoch 9/40
158/158 — 1s 5ms/step - accuracy: 0.8009 - loss: 153.1257 - val_accuracy: 0.8883 - val_loss: 98.6739
Epoch 10/40
158/158 — 1s 5ms/step - accuracy: 0.8260 - loss: 307.6790 - val_accuracy: 0.8980 - val_loss: 177.2501
Epoch 11/40

```

```

Epoch 29/40
158/158 ————— 1s 5ms/step - accuracy: 0.8182 - loss: 202.8431 - val_accuracy: 0.9303 - val_loss: 95.5109
Epoch 30/40
158/158 ————— 1s 5ms/step - accuracy: 0.8173 - loss: 179.6674 - val_accuracy: 0.9270 - val_loss: 174.9730
Epoch 31/40
158/158 ————— 1s 5ms/step - accuracy: 0.8098 - loss: 696.3585 - val_accuracy: 0.9222 - val_loss: 284.1356
Epoch 32/40
158/158 ————— 1s 5ms/step - accuracy: 0.8083 - loss: 99.0774 - val_accuracy: 0.9331 - val_loss: 162.7282
Epoch 33/40
158/158 ————— 1s 5ms/step - accuracy: 0.7910 - loss: 106.3475 - val_accuracy: 0.9297 - val_loss: 112.3814
Epoch 34/40
158/158 ————— 1s 5ms/step - accuracy: 0.7699 - loss: 90.8548 - val_accuracy: 0.9353 - val_loss: 70.6163
Epoch 35/40
158/158 ————— 1s 5ms/step - accuracy: 0.7423 - loss: 506.7877 - val_accuracy: 0.9325 - val_loss: 122.0899
Epoch 36/40
158/158 ————— 2s 7ms/step - accuracy: 0.7355 - loss: 148.9017 - val_accuracy: 0.9139 - val_loss: 102.8252
Epoch 37/40
158/158 ————— 1s 6ms/step - accuracy: 0.7303 - loss: 112.8615 - val_accuracy: 0.9292 - val_loss: 83.9951
Epoch 38/40
158/158 ————— 1s 5ms/step - accuracy: 0.7406 - loss: 285.3622 - val_accuracy: 0.8355 - val_loss: 129.3563
Epoch 39/40
158/158 ————— 1s 5ms/step - accuracy: 0.7128 - loss: 389.4075 - val_accuracy: 0.9434 - val_loss: 143.5847
Epoch 40/40
158/158 ————— 1s 5ms/step - accuracy: 0.8144 - loss: 207.3708 - val_accuracy: 0.9369 - val_loss: 207.0464

```

## Performance Metrics

```

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Get model predictions
Y_pred = cnn_model.predict([X_test, X_test])

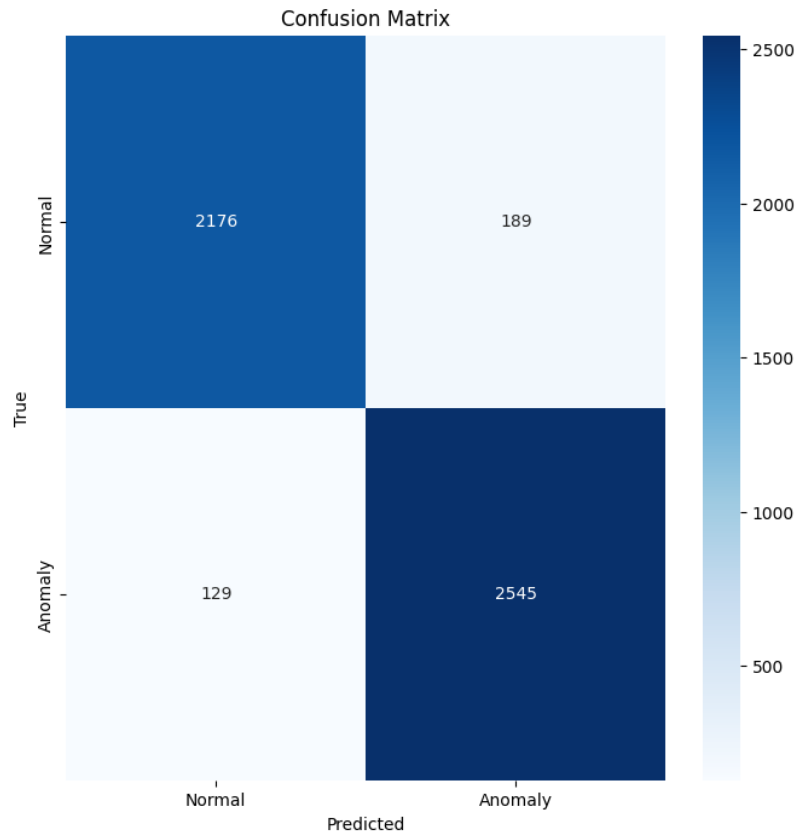
# Convert probabilities to class labels
y_pred = np.argmax(Y_pred, axis=1) # Get predicted class (0 or 1)
y_true = np.argmax(Y_test, axis=1) # Convert one-hot encoded labels to class labels

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Define class labels
class_labels = ['Normal', 'Anomaly']

# Plot confusion matrix
plt.figure(figsize=(8,8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

```



```

from sklearn.metrics import classification_report

# Get model predictions
Y_pred = cnn_model.predict([X_test, X_test])

# Convert probabilities to class labels
y_pred = np.argmax(Y_pred, axis=1) # Predicted class (0 or 1)
y_true = np.argmax(Y_test, axis=1) # True class labels

# Generate classification report
report = classification_report(y_true, y_pred, target_names=['Normal', 'Anomaly'])

# Print the classification report
print("Classification Report:\n", report)

```

158/158 ————— 0s 2ms/step

Classification Report:

	precision	recall	f1-score	support
Normal	0.94	0.92	0.93	2365
Anomaly	0.93	0.95	0.94	2674
accuracy			0.94	5039
macro avg	0.94	0.94	0.94	5039
weighted avg	0.94	0.94	0.94	5039

## ANN Model

Importing the required libraries and layers for ANN

```
▶ from tensorflow.keras.models import Sequential
  from tensorflow.keras.layers import Dense, Dropout, Flatten
  from tensorflow.keras.activations import swish, gelu
  from tensorflow.keras.optimizers import Adam
```

Splitting the dataset into train and test

```
✓ [87] from sklearn.model_selection import train_test_split
0s      X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Model Building

```
# Define the ANN model
nn_model = Sequential()

# Input layer
nn_model.add(Flatten(input_shape=(41, 1)))

# Hidden layers with Swish and GELU activations
nn_model.add(Dense(256, activation=swish))
nn_model.add(Dropout(0.4))

nn_model.add(Dense(128, activation=gelu))
nn_model.add(Dropout(0.4))

nn_model.add(Dense(64, activation=swish))
nn_model.add(Dropout(0.4))

# Output layer with softmax activation for binary classification
nn_model.add(Dense(2, activation='sigmoid'))

# Print model summary
nn_model.summary()
```


```
super().__init__(**kwargs)
Model: "sequential_7"
```

Layer (type)	Output Shape	Param #
flatten_15 (Flatten)	(None, 41)	0
dense_46 (Dense)	(None, 256)	10,752
dropout_49 (Dropout)	(None, 256)	0
dense_47 (Dense)	(None, 128)	32,896
dropout_50 (Dropout)	(None, 128)	0
dense_48 (Dense)	(None, 64)	8,256
dropout_51 (Dropout)	(None, 64)	0
dense_49 (Dense)	(None, 2)	130

Total params: 52,034 (203.26 KB)  
Trainable params: 52,034 (203.26 KB)  
Non-trainable params: 0 (0.00 B)

## Training the model

```
# Compile the model
nn_model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

 `history1 = nn_model.fit(X_train,Y_train, batch_size= 128,  
epochs = 40, validation_data = (X_test,Y_test))`

```
Epoch 1/40
158/158 — 6s 18ms/step - accuracy: 0.6859 - loss: 1894.9584 - val_accuracy: 0.9377 - val_loss: 460.5956
Epoch 2/40
158/158 — 1s 4ms/step - accuracy: 0.7907 - loss: 833.1293 - val_accuracy: 0.8615 - val_loss: 148.7936
Epoch 3/40
158/158 — 1s 4ms/step - accuracy: 0.7686 - loss: 354.8790 - val_accuracy: 0.8408 - val_loss: 74.8944
Epoch 4/40
158/158 — 1s 4ms/step - accuracy: 0.7368 - loss: 4766.1528 - val_accuracy: 0.8158 - val_loss: 257.9389
Epoch 5/40
158/158 — 1s 3ms/step - accuracy: 0.7567 - loss: 353.6205 - val_accuracy: 0.8010 - val_loss: 234.8032
Epoch 6/40
158/158 — 1s 3ms/step - accuracy: 0.7631 - loss: 384.7056 - val_accuracy: 0.8067 - val_loss: 58.7943
Epoch 7/40
158/158 — 1s 3ms/step - accuracy: 0.7562 - loss: 130.9056 - val_accuracy: 0.8081 - val_loss: 90.5001
Epoch 8/40
158/158 — 1s 4ms/step - accuracy: 0.7629 - loss: 371.8989 - val_accuracy: 0.7623 - val_loss: 26.0417
Epoch 9/40
158/158 — 1s 3ms/step - accuracy: 0.7989 - loss: 225.7092 - val_accuracy: 0.8254 - val_loss: 13.6458
Epoch 10/40
158/158 — 1s 4ms/step - accuracy: 0.8022 - loss: 104.0370 - val_accuracy: 0.9176 - val_loss: 3.8036
Epoch 11/40
158/158 — 1s 3ms/step - accuracy: 0.8406 - loss: 39.9831 - val_accuracy: 0.9153 - val_loss: 0.1854
Epoch 12/40
158/158 — 1s 3ms/step - accuracy: 0.8282 - loss: 70.5381 - val_accuracy: 0.9194 - val_loss: 0.4704
Epoch 13/40
158/158 — 1s 3ms/step - accuracy: 0.8381 - loss: 28.3711 - val_accuracy: 0.9155 - val_loss: 0.1970
Epoch 14/40
158/158 — 1s 3ms/step - accuracy: 0.8476 - loss: 55.4894 - val_accuracy: 0.9337 - val_loss: 0.1671
Epoch 15/40
158/158 — 1s 3ms/step - accuracy: 0.8641 - loss: 22.9827 - val_accuracy: 0.9454 - val_loss: 0.1626
Epoch 16/40
158/158 — 1s 3ms/step - accuracy: 0.8847 - loss: 181.8250 - val_accuracy: 0.9401 - val_loss: 0.1673
Epoch 17/40
158/158 — 1s 4ms/step - accuracy: 0.8955 - loss: 11.0490 - val_accuracy: 0.9337 - val_loss: 0.1837
Epoch 18/40
```

```

Epoch 26/40
158/158 — 1s 4ms/step - accuracy: 0.9279 - loss: 1.0241 - val_accuracy: 0.9605 - val_loss: 0.1394
Epoch 27/40
158/158 — 1s 4ms/step - accuracy: 0.9306 - loss: 3.1727 - val_accuracy: 0.9605 - val_loss: 0.1394
Epoch 28/40
158/158 — 1s 3ms/step - accuracy: 0.9271 - loss: 1.9444 - val_accuracy: 0.9583 - val_loss: 0.1403
Epoch 29/40
158/158 — 1s 3ms/step - accuracy: 0.9378 - loss: 7.8367 - val_accuracy: 0.9589 - val_loss: 0.1397
Epoch 30/40
158/158 — 1s 3ms/step - accuracy: 0.9392 - loss: 16.4023 - val_accuracy: 0.9603 - val_loss: 0.1365
Epoch 31/40
158/158 — 1s 3ms/step - accuracy: 0.9448 - loss: 0.7123 - val_accuracy: 0.9603 - val_loss: 0.1344
Epoch 32/40
158/158 — 1s 3ms/step - accuracy: 0.9398 - loss: 1.5379 - val_accuracy: 0.9607 - val_loss: 0.1343
Epoch 33/40
158/158 — 1s 3ms/step - accuracy: 0.9419 - loss: 2.7132 - val_accuracy: 0.9613 - val_loss: 0.1338
Epoch 34/40
158/158 — 1s 3ms/step - accuracy: 0.9453 - loss: 439.6749 - val_accuracy: 0.9613 - val_loss: 0.1344
Epoch 35/40
158/158 — 1s 4ms/step - accuracy: 0.9448 - loss: 3.7689 - val_accuracy: 0.9625 - val_loss: 0.1313
Epoch 36/40
158/158 — 1s 3ms/step - accuracy: 0.9462 - loss: 2.0286 - val_accuracy: 0.9597 - val_loss: 0.1353
Epoch 37/40
158/158 — 1s 3ms/step - accuracy: 0.9473 - loss: 2.9388 - val_accuracy: 0.9615 - val_loss: 0.1319
Epoch 38/40
158/158 — 1s 3ms/step - accuracy: 0.9469 - loss: 0.9226 - val_accuracy: 0.9615 - val_loss: 0.1308
Epoch 39/40
158/158 — 1s 4ms/step - accuracy: 0.9501 - loss: 0.3720 - val_accuracy: 0.9621 - val_loss: 0.1285
Epoch 40/40
158/158 — 1s 5ms/step - accuracy: 0.9488 - loss: 1.8078 - val_accuracy: 0.9629 - val_loss: 0.1299

```

## Performance Metrics

```

[ ] from sklearn.metrics import confusion_matrix
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Get model predictions
Y_pred = nn_model.predict(X_test)
y_pred = (Y_pred[:, 1] > 0.5).astype(int) # Assuming second column corresponds to "Anomaly"
y_true = np.argmax(Y_test, axis=1) # Convert one-hot encoded labels to class labels

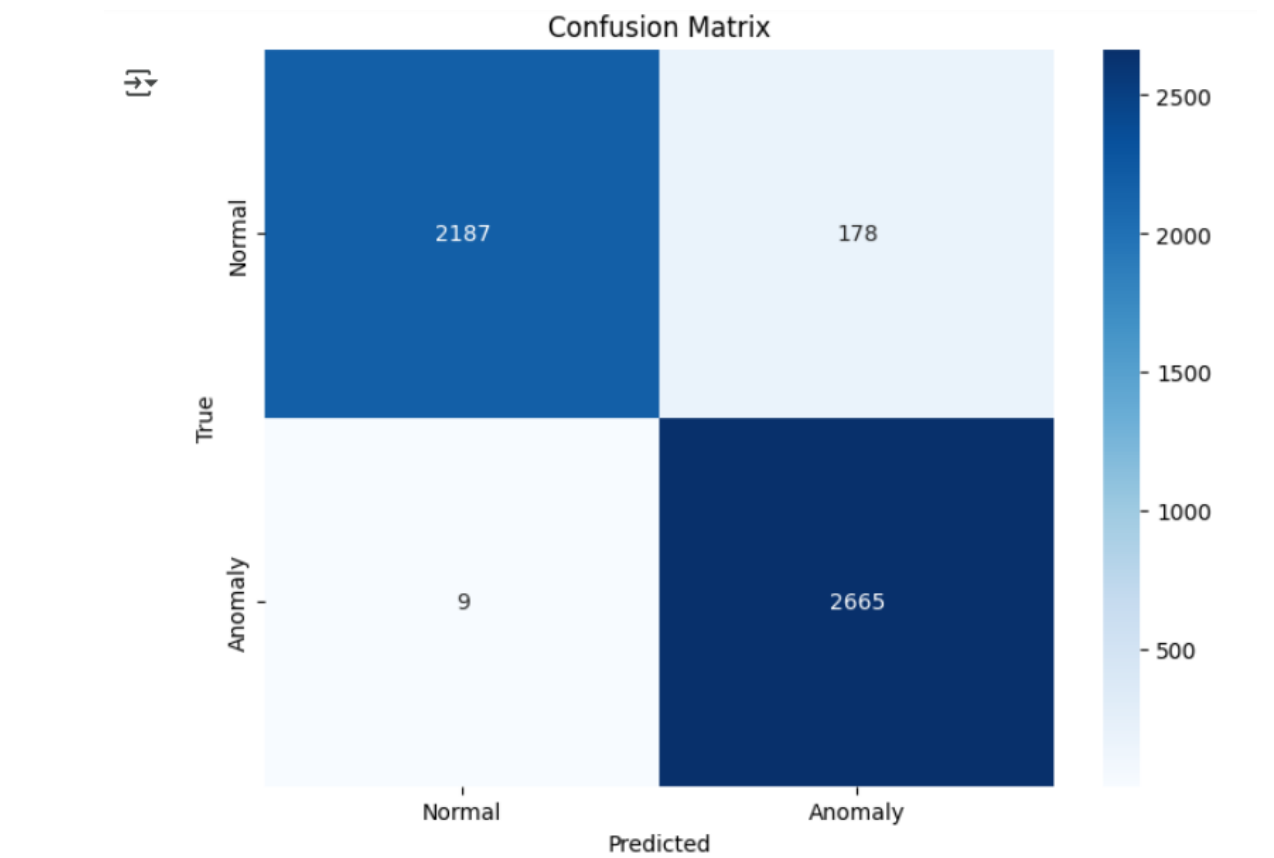
# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Define class labels
class_labels = ['Normal', 'Anomaly']

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

```





```
[ ] from sklearn.metrics import classification_report

# Get model predictions
Y_pred = nn_model.predict([X_test, X_test])

# Convert probabilities to class labels
y_pred = np.argmax(Y_pred, axis=1) # Predicted class (0 or 1)
y_true = np.argmax(Y_test, axis=1) # True class labels

# Generate classification report
report = classification_report(y_true, y_pred, target_names=['Normal', 'Anomaly'])

# Print the classification report
print("Classification Report:\n", report)
```

32/158 — 0s 2ms/step /usr/local/lib/python3.11/dist-packages/keras/src/models/functional.py:237: UserWarning: The structure of 'inputs' doesn't match the expected structure.  
Expected: keras\_tensor\_861  
Received: inputs=('Tensor(shape=(32, 41, 1))', 'Tensor(shape=(32, 41, 1))')  
warnings.warn(msg)

158/158 — 1s 3ms/step /usr/local/lib/python3.11/dist-packages/keras/src/models/functional.py:237: UserWarning: The structure of 'inputs' doesn't match the expected structure.  
Expected: keras\_tensor\_861  
Received: inputs=('Tensor(shape=(None, 41, 1))', 'Tensor(shape=(None, 41, 1))')  
warnings.warn(msg)

Classification Report:

	precision	recall	f1-score	support
Normal	1.00	0.93	0.96	2365
Anomaly	0.94	1.00	0.97	2674
accuracy			0.96	5039
macro avg	0.97	0.96	0.96	5039
weighted avg	0.96	0.96	0.96	5039

## Hybrid Model

Importing the required libraries and layers for ANN

```
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout, Input, Concatenate
from tensorflow.keras.optimizers import Adam
import tensorflow.keras as keras
```

Splitting the dataset into train and test

```
[87] from sklearn.model_selection import train_test_split
      X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Model Building

```
# Define CNN branch
cnn_input = Input(shape=(41, 1))
cnn_branch = Conv1D(filters=32, kernel_size=3, activation='gelu')(cnn_input)
cnn_branch = Conv1D(filters=32, kernel_size=3, activation='gelu')(cnn_branch)
cnn_branch = MaxPooling1D(pool_size=2)(cnn_branch)
cnn_branch = Dropout(0.4)(cnn_branch)

cnn_branch = Conv1D(filters=64, kernel_size=3, activation='gelu')(cnn_branch)
cnn_branch = Conv1D(filters=64, kernel_size=3, activation='gelu')(cnn_branch)
cnn_branch = MaxPooling1D(pool_size=2)(cnn_branch)
cnn_branch = Dropout(0.4)(cnn_branch)

cnn_branch = Flatten()(cnn_branch)

# Define ANN branch
ann_input = Input(shape=(41, 1))
ann_branch = Flatten()(ann_input)
ann_branch = Dense(256, activation='gelu')(ann_branch)
ann_branch = Dropout(0.4)(ann_branch)
ann_branch = Dense(128, activation='gelu')(ann_branch)
ann_branch = Dropout(0.4)(ann_branch)
ann_branch = Dense(64, activation='gelu')(ann_branch)
ann_branch = Dropout(0.4)(ann_branch)
```

```

# Combine both branches
combined = Concatenate()([cnn_branch, ann_branch])

# Fully connected layer after merging
fc = Dense(256, activation='gelu')(combined)
fc = Dropout(0.5)(fc)
fc = Dense(2, activation='swish')(fc) # Assuming binary classification

# Create model
eemodel = Model(inputs=[cnn_input, ann_input], outputs=fc)

# Compile the model
optimiser = keras.optimizers.Adam(learning_rate=0.0002)
eemodel.compile(optimizer=optimiser, loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])

# Print summary
eemodel.summary()

```

Layer (type)	Output Shape	Param #	Connected to
input_layer_4 (InputLayer)	(None, 41, 1)	0	-
conv1d_10 (Conv1D)	(None, 39, 32)	128	input_layer_4[0][0]
conv1d_11 (Conv1D)	(None, 37, 32)	3,104	conv1d_10[0][0]
input_layer_5 (InputLayer)	(None, 41, 1)	0	-
max_pooling1d_2 (MaxPooling1D)	(None, 18, 32)	0	conv1d_11[0][0]
flatten_5 (Flatten)	(None, 41)	0	input_layer_5[0][0]
dropout_14 (Dropout)	(None, 18, 32)	0	max_pooling1d_2[0][0]
dense_12 (Dense)	(None, 256)	10,752	flatten_5[0][0]
conv1d_12 (Conv1D)	(None, 16, 64)	6,208	dropout_14[0][0]
dropout_16 (Dropout)	(None, 256)	0	dense_12[0][0]
conv1d_13 (Conv1D)	(None, 14, 64)	12,352	conv1d_12[0][0]
dense_13 (Dense)	(None, 128)	32,896	dropout_16[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 7, 64)	0	conv1d_13[0][0]
dropout_17 (Dropout)	(None, 128)	0	dense_13[0][0]
dropout_15 (Dropout)	(None, 7, 64)	0	max_pooling1d_3[0][0]
dense_14 (Dense)	(None, 64)	8,256	dropout_17[0][0]
flatten_4 (Flatten)	(None, 448)	0	dropout_15[0][0]
dropout_18 (Dropout)	(None, 64)	0	dense_14[0][0]
concatenate_1 (Concatenate)	(None, 512)	0	flatten_4[0][0], dropout_18[0][0]
dense_15 (Dense)	(None, 256)	131,328	concatenate_1[0][0]
dropout_19 (Dropout)	(None, 256)	0	dense_15[0][0]
dense_16 (Dense)	(None, 2)	514	dropout_19[0][0]

## Training the model

```
# Train the model
history4 = eemodel.fit([X_train, X_train], Y_train, batch_size=128,
                       epochs=40, validation_data=([X_test, X_test], Y_test))
```

```
Epoch 1/40
158/158 — 13s 52ms/step - accuracy: 0.8386 - loss: 13.2344 - val_accuracy: 0.9415 - val_loss: 4.6252
Epoch 2/40
158/158 — 10s 5ms/step - accuracy: 0.9229 - loss: 4.9349 - val_accuracy: 0.9440 - val_loss: 2.2499
Epoch 3/40
158/158 — 1s 5ms/step - accuracy: 0.9344 - loss: 2.1000 - val_accuracy: 0.9502 - val_loss: 1.3999
Epoch 4/40
158/158 — 1s 5ms/step - accuracy: 0.9365 - loss: 2.2197 - val_accuracy: 0.9355 - val_loss: 0.8578
Epoch 5/40
158/158 — 1s 5ms/step - accuracy: 0.9433 - loss: 1.4021 - val_accuracy: 0.9538 - val_loss: 0.7791
Epoch 6/40
158/158 — 1s 4ms/step - accuracy: 0.9487 - loss: 4.1156 - val_accuracy: 0.9565 - val_loss: 0.3063
Epoch 7/40
158/158 — 1s 5ms/step - accuracy: 0.9518 - loss: 20.1825 - val_accuracy: 0.9581 - val_loss: 0.4569
Epoch 8/40
158/158 — 1s 5ms/step - accuracy: 0.9529 - loss: 1.2601 - val_accuracy: 0.9595 - val_loss: 0.3578
Epoch 9/40
158/158 — 1s 5ms/step - accuracy: 0.9573 - loss: 2.1133 - val_accuracy: 0.9631 - val_loss: 0.1366
Epoch 10/40
158/158 — 1s 6ms/step - accuracy: 0.9585 - loss: 0.4127 - val_accuracy: 0.9613 - val_loss: 0.3274
Epoch 11/40
158/158 — 1s 5ms/step - accuracy: 0.9619 - loss: 2.4308 - val_accuracy: 0.9565 - val_loss: 0.3982
Epoch 12/40
158/158 — 1s 4ms/step - accuracy: 0.9614 - loss: 0.7441 - val_accuracy: 0.9680 - val_loss: 0.3421
Epoch 13/40
158/158 — 1s 5ms/step - accuracy: 0.9637 - loss: 0.8408 - val_accuracy: 0.9671 - val_loss: 0.1632
Epoch 14/40
158/158 — 1s 5ms/step - accuracy: 0.9632 - loss: 1.1996 - val_accuracy: 0.9690 - val_loss: 0.1384
Epoch 15/40
158/158 — 1s 4ms/step - accuracy: 0.9666 - loss: 0.6250 - val_accuracy: 0.9748 - val_loss: 0.1455
Epoch 16/40
158/158 — 1s 5ms/step - accuracy: 0.9706 - loss: 0.4750 - val_accuracy: 0.9770 - val_loss: 0.2132
Epoch 17/40
158/158 — 1s 4ms/step - accuracy: 0.9684 - loss: 0.3728 - val_accuracy: 0.9758 - val_loss: 0.2838
Epoch 18/40
158/158 — 1s 5ms/step - accuracy: 0.9732 - loss: 1.9435 - val_accuracy: 0.9748 - val_loss: 0.2211
Epoch 19/40
158/158 — 1s 5ms/step - accuracy: 0.9777 - loss: 0.2422 - val_accuracy: 0.9792 - val_loss: 0.0753
Epoch 20/40
158/158 — 1s 5ms/step - accuracy: 0.9771 - loss: 0.3336 - val_accuracy: 0.9808 - val_loss: 0.0806
Epoch 21/40
158/158 — 1s 5ms/step - accuracy: 0.9793 - loss: 0.3229 - val_accuracy: 0.9817 - val_loss: 0.0587
Epoch 22/40
158/158 — 1s 5ms/step - accuracy: 0.9791 - loss: 0.2803 - val_accuracy: 0.9800 - val_loss: 0.0760
Epoch 23/40
158/158 — 1s 4ms/step - accuracy: 0.9821 - loss: 0.1299 - val_accuracy: 0.9788 - val_loss: 0.0699
Epoch 24/40
158/158 — 1s 5ms/step - accuracy: 0.9802 - loss: 0.2137 - val_accuracy: 0.9859 - val_loss: 0.2388
Epoch 25/40
158/158 — 2s 7ms/step - accuracy: 0.9814 - loss: 0.3833 - val_accuracy: 0.9849 - val_loss: 0.3058
Epoch 26/40
158/158 — 1s 6ms/step - accuracy: 0.9834 - loss: 10.9176 - val_accuracy: 0.9875 - val_loss: 0.0806
Epoch 27/40
158/158 — 1s 5ms/step - accuracy: 0.9832 - loss: 0.1224 - val_accuracy: 0.9879 - val_loss: 0.0868
Epoch 28/40
158/158 — 1s 5ms/step - accuracy: 0.9825 - loss: 0.1436 - val_accuracy: 0.9861 - val_loss: 0.0652
Epoch 29/40
158/158 — 1s 4ms/step - accuracy: 0.9811 - loss: 1.4869 - val_accuracy: 0.9835 - val_loss: 0.2777
Epoch 30/40
158/158 — 1s 4ms/step - accuracy: 0.9802 - loss: 0.3217 - val_accuracy: 0.9879 - val_loss: 0.2129
Epoch 31/40
158/158 — 1s 5ms/step - accuracy: 0.9775 - loss: 0.6147 - val_accuracy: 0.9863 - val_loss: 0.0577
Epoch 32/40
158/158 — 1s 5ms/step - accuracy: 0.9801 - loss: 0.5461 - val_accuracy: 0.9865 - val_loss: 0.0939
Epoch 33/40
158/158 — 1s 5ms/step - accuracy: 0.9824 - loss: 2.0820 - val_accuracy: 0.9851 - val_loss: 0.0556
Epoch 34/40
158/158 — 1s 4ms/step - accuracy: 0.9810 - loss: 0.4250 - val_accuracy: 0.9867 - val_loss: 0.0785
```

```

from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Get model predictions
y_pred = ee_model.predict(X_test)

# Convert predictions to class labels
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(Y_test, axis=1)

# Generate classification report
report = classification_report(y_true_classes, y_pred_classes)
print("Classification Report:\n", report)

# Generate confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)
print("Confusion Matrix:\n", cm)

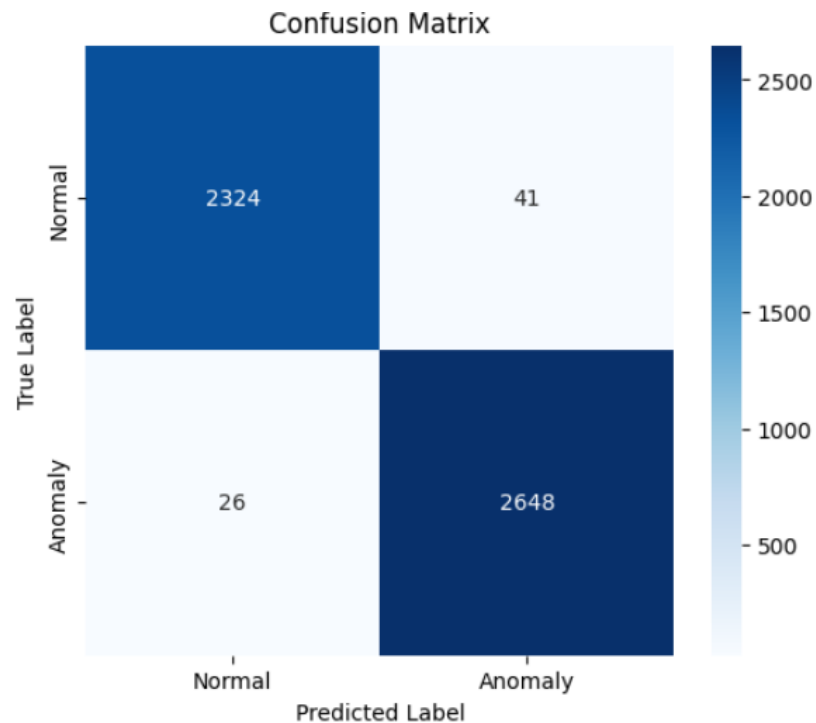
# Plot confusion matrix
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Normal", "Anomaly"], yticklabels=["Normal", "Anomaly"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

```

158/158 ————— 1s 6ms/step

↔ Classification Report:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	2365
1	0.98	0.99	0.99	2674
accuracy			0.99	5039
macro avg	0.99	0.99	0.99	5039
weighted avg	0.99	0.99	0.99	5039



## Prediction


```
[ ] import numpy as np
import pandas as pd
from tensorflow.keras.models import load_model
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Load the trained ANN model
ann_model = load_model("cnn_model.h5") # Ensure correct model path

# Sample input data (ensure exactly 41 features)
sample_data = pd.DataFrame([[
    0, 1, 19, 9, 491, 0, 0, 0, 0, 0, # First 10 features
    0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, # More features
    0, 0, 0, 1, 0, 0, 150, 25, # Last features
    0.17, 0.03, 0.17, 0, 0, 0, 0.05, 0 # Ensure total 41 features
]])

# Define 41 column names (EXCLUDING the class label)
column_names = ['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land',
                'wrong_fragment', 'urgent', 'hot', 'num_failed_logins', 'logged_in', 'num_compromised',
                'root_shell', 'su_attempted', 'num_root', 'num_file_creations', 'num_shells', 'num_access_files',
                'num_outbound_cmds', 'is_host_login', 'is_guest_login', 'count', 'srv_count', 'serror_rate',
                'srv_serror_rate', 'error_rate', 'srv_error_rate', 'same_srv_rate', 'diff_srv_rate',
                'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count', 'dst_host_same_srv_rate',
                'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
                'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_error_rate', 'dst_host_srv_error_rate']

sample_data.columns = column_names # Ensure correct feature alignment
```

```
 # Encode categorical features (Ensure consistency with training)
categorical_features = ['protocol_type', 'service', 'flag']
encoders = {col: LabelEncoder().fit(sample_data[col]) for col in categorical_features}

for col in categorical_features:
    sample_data[col] = encoders[col].transform(sample_data[col])

# Convert to numpy array
input_data = sample_data.values # Should have shape (1, 41)

# Standardize numerical features
scaler = StandardScaler().fit(input_data)
input_data = scaler.transform(input_data)


# Make a prediction
prediction = ann_model.predict(input_data)

# Get the class with the highest probability
predicted_class_index = np.argmax(prediction)

# Define class labels
class_labels = ['anomaly', 'normal']

# Get the predicted class
predicted_class = class_labels[predicted_class_index]

print("Predicted class:", predicted_class)
```

 WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile\_metrics' will be empty until you train or evaluate the model.  
1/1 1s 981ms/step  
Predicted class: normal

