

FEATURE SELECTION USING AUTOENCODERS IN NEURAL
NETWORK BASED INTRUSION DETECTION SYSTEM

submitted by

SUKANYA ROY

Class Roll No: 002210503015

Registration No: 163607 of 2022-2023

Examination Roll No.: MCA244015

Master of Computer Application

Under the Supervision of

Prof. ANUPAM SINHA

in the

Department of Computer Science and Engineering

Faculty of Engineering and Technology

Jadavpur University

2024

FACULTY OF ENGINEERING AND TECHNOLOGY,
JADAVPUR UNIVERSITY

CERTIFICATE OF RECOMMENDATION

To Whom It May Concern

I hereby recommend that the project report entitled “Feature Selection Using Autoencoders in Neural Network based Intrusion Detection System” has been carried out by SUKANYA ROY (Reg. No.: 163607 of 2022-2023, Roll No.: 002210503015, Exam Roll No: MCA244015) under my guidance and supervision and be accepted in partial fulfilment of the requirement for the degree of Master of COMPUTER APPLICATION in DEPARTMENT of COMPUTER SCIENCE AND ENGINEERING, JADAVPUR UNIVERSITY during the academic year 2023-24.

Prof. Anupam Sinha
Project Supervisor
Dept. of Computer Science and Engineering
Jadavpur University

Prof. (Dr.) Nandini Mukhopadhyay
Head of the Department
Dept. of Computer Science & Engineering
Jadavpur University

Prof. Dipak Laha
Dean, Faculty Council
of Engineering & Technology
Jadavpur University, Kolkata-032

FACULTY OF ENGINEERING AND TECHNOLOGY,
JADAVPUR UNIVERSITY

CERTIFICATE OF APPROVAL

This is to certify that the project report entitled “Feature Selection Using Autoencoders in Neural Network based Intrusion Detection System” is a bonafide record of work carried out by SUKANYA ROY in fulfilment of the requirements for the award of the Degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University. It is understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for that it has been submitted.

Signature of Examiner 1

Date:

Signature of Examiner 2

Date:

DECLARATION OF COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis entitled “Feature Selection Using Autoencoders in Neural Network based Intrusion Detection System” is an outcome of my genuine and independent work, as part of my Master of Computer Application program.

All information has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, all materials and results that are not original to this work have been fully cited and referenced.

Sukanya Roy

Registration No: 163607 of 2022-2023

Examination Roll No.: MCA244015

Thesis Title: Feature Extraction Using Autoencoders in Neural Network based Intrusion Detection System

Signature with Date

ACKNOWLEDGEMENT

With my utmost sincerity and gratitude, I would like to thank Prof. Anupam Sinha, Department of Computer Science & Engineering, my supervisor, for his overwhelming support throughout the duration of the project. His motivation always gave me the required inputs and momentum to continue with my work, without which the project work would not have taken its current shape. His valuable suggestion and numerous discussions have always inspired new ways of thinking. I feel deeply honored that I got this opportunity to work under him.

I would like to express my sincere thanks to all my teachers for providing sound knowledge base and cooperation.

I would like to thank all the faculty members of the Department of Computer Science & Engineering of Jadavpur University for their continuous support.

Last, but not the least, I would like to thank my batch mates for staying by side when I need them.

Sukanya Roy

Registration No: 163607 of 2022-2023

Examination Roll No.: MCA244015

Department of Computer Science & Engineering,

Jadavpur University

ABSTRACT

An Intrusion Detection System (IDS) can identify attacks by analysing data traffic patterns within a network. Given the substantial volume of data handled by the IDS, there is a necessity to conduct feature extraction to reduce the computational burden of processing raw data. Feature extraction involves converting features into a lower dimension to expedite the learning process and enhance accuracy. This study focuses on automatic feature extraction utilizing a simple autoencoder and Artificial Neural Network for classifying (Both Binary classification and Multiclass Classification) attacks detected by the IDS. Various activation functions and loss functions are employed to assess the extent to which feature extraction can enhance accuracy. The effectiveness of the detection mechanisms post feature extraction is evaluated using datasets such as KDD Cup '99. In the proposed model, the ReLU activation function and cross-entropy loss function as hyperparameters for the autoencoder yield the highest accuracy compared to other functions.

CONTENTS

	Page No.
Certificate Of Recommendation	ii
Certificate Of Approval	iii
Declaration Of Originality And Compliance Of Academic Ethic	iv
Acknowledgement	v
Abstract	vi
Contents	vii
List of Tables	x
List of Figures	xi
Chapter-1 INTRODUCTION	1
1.1 Overview	1
1.2 Motivation	3
1.3 Problem Statement	4
Chapter-2 BACKGROUND	5
2.1 Importance of Network Security	5
2.2 Intrusion Detection System (IDS)	6
2.3 Machine Learning	8
2.4 Artificial Neural Network	9
2.4.1 Artificial Neural Networks Architecture	9
2.4.2 How does Neural Network works	11
2.4.3 Types of Neural Network	11
2.5 Perceptron	12
2.5.1 Artificial neurons Vs Biological neurons	12
2.5.2 Components of Perceptron	14
2.5.3 How does Perceptron work?	15

2.6	Process of ANN Model Making	16
2.7	Importance of Feature Extraction	18
2.8	Autoencoders	19
2.8.1	Architecture of Autoencoder	20
2.8.2	Types of Autoencoders	21
2.8.3	Advantages of Autoencoders	22
2.9	Definitions of Important Term	23
2.9.1	StandardScaler	23
2.9.2	OneHotEncoder	23
2.9.3	Dropout Layer	24
2.9.4	Optimizer (Adam)	24
2.9.5	Loss Function	25
2.9.6	EarlyStopping	25
2.10	Evaluation Metrics	26
Chapter-3	Literary Review	27
3.1	Literary Review on Attacks	27
3.2	Literary Review on Performance metrics /variables	28
3.3	Literary Review on Intrusion detection approaches	29
3.4	Literary Review on Machine learning techniques	30
3.5	Discussion	32
Chapter-4	Methodology	34
4.1	Schematic of the System	34
4.1.1	Data Flow of Feature Selection using Autoencoder	34
4.1.2	End To End Data Flow	35
4.2	IDS – Datasets	37

4.3	Data preprocessing	41
4.4	Exploratory Data Analysis (EDA)	41
4.5	Modelling a Neural Network	45
4.6	Applying Feature selection in selected Model	46
Chapter-5	Results	48
5.1	Results of Binary Classification	48
5.1.1	Performance results	48
5.1.2	Graphs	53
5.1.3	Confusion Matrices	55
5.2	Results of Multiclass Classification	57
5.2.1	Performance results	57
5.2.2	Graphs	59
5.2.3	Confusion Matrices	60
Chapter-6	Algorithm And Code	62
6.1	Algorithm	62
6.2	Code	64
Chapter-7	Conclusion and Future work	
7.1	Conclusion	80
7.2	Future work	82
7.3	References	83

List of Tables

Table No.	Title	Page No.
4.1	KDD 99 Data Description	38
5.1	Binary Classification Accuracy Scores – Without Autoencoder	48
5.2	Binary Classification Accuracy Scores – Simple Autoencoder	48
5.3	Binary Classification Accuracy Scores – Stacked Autoencoder	49
5.4	Binary Classification Mean Values – Without Autoencoder	50
5.5	Binary Classification Mean Values – Simple Autoencoder	50
5.6	Binary Classification Mean Values – Stacked Autoencoder	51
5.7	Binary Classification Dataset Comparison	51
5.8	Multiclass Classification Accuracy Scores – Without Autoencoder	57
5.9	Multiclass Classification Accuracy Scores – Simple Autoencoder	57
5.10	Multiclass Classification Accuracy Scores – Stacked Autoencoder	57
5.11	Multiclass Classification Mean Values – Without Autoencoder	58
5.12	Multiclass Classification Mean Values – Simple Autoencoder	58

List of Figures

Figure No.	Title	Page No.
1.1	Intrusion Detection System	1
2.1	Types of IDS based on responsiveness.	6
2.2	Types of IDS based on approach	7
2.3	Types of IDS based on attack type.	8
2.4	Artificial Neural Networks Architecture	10
2.5	Neuron Vs Perceptron	13
2.6	Architecture of Perceptron	14
2.7	Process of Ann Model Making	17
2.8	Architecture of autoencoder	20
2.9	Components of autoencoder	21
2.10	Confusion Matrix	26
4.1	Data Flow of Feature Selection	35
4.2	End to End Data Flow	37
4.3	Correlation between features(heat map)	43
4.4	Attack type Count Plot	44
4.5	Protocol type Count Plot	44
5.1	Binary Classification–Model Accuracy (without Autoencoder, KDD Cup’ 99)	53
5.2	Binary Classification –Model Loss (without Autoencoder, KDD Cup’ 99)	53
5.3	Binary Classification–Model Accuracy (Stacked Autoencoder, KDD Cup’ 99)	53
5.4	Binary Classification –Model Loss (Stacked Autoencoder, KDD Cup’ 99)	53
5.5	Binary Classification–Model Accuracy (without Autoencoder, NSL KDD)	54
5.6	Binary Classification –Model Loss (without Autoencoder, NSL KDD)	54

5.7	Binary Classification–Model Accuracy (Stacked Autoencoder, NSL KDD)	54
5.8	Binary Classification –Model Loss (Stacked Autoencoder, NSL KDD)	54
5.9	Binary Classification–Confusion Matrix (without Autoencoder, KDD Cup’ 99)	55
5.10	Binary Classification–Confusion Matrix (Stacked Autoencoder, KDD Cup’ 99)	55
5.11	Binary Classification–Confusion Matrix (without Autoencoder, NSL KDD)	56
5.12	Binary Classification–Confusion Matrix (stacked Autoencoder, NSL KDD)	56
5.13	Multiclass Classification–Model Accuracy (without Autoencoder)	59
5.14	Multiclass Classification –Model Loss (without Autoencoder)	59
5.15	Multiclass Classification–Model Accuracy (Simple Autoencoder)	59
5.16	Multiclass Classification –Model Loss (Simple Autoencoder)	59
5.17	Multiclass Classification–Model Accuracy (Stacked Autoencoder)	60
5.18	Multiclass Classification –Model Loss (Stacked Autoencoder)	60
5.19	Multiclass Classification–Confusion Matrix (without Autoencoder)	60
5.20	Multiclass Classification–Confusion Matrix (Simple Autoencoder)	61
5.21	Multiclass Classification–Confusion Matrix (Stacked Autoencoder)	61

Introduction

1.3 Overview

In the realm of network security, Intrusion Detection Systems (IDS) play a crucial role in swiftly pinpointing and addressing potential threats as they arise. A prominent and extensively utilized resource for training and assessing IDS models is the KDD Cup 1999 dataset. This dataset encompasses an extensive array of network traffic data, comprising both regular and malevolent traffic instances. This comprehensive dataset enables researchers to devise and validate models capable of effectively discerning intrusions amidst network traffic.

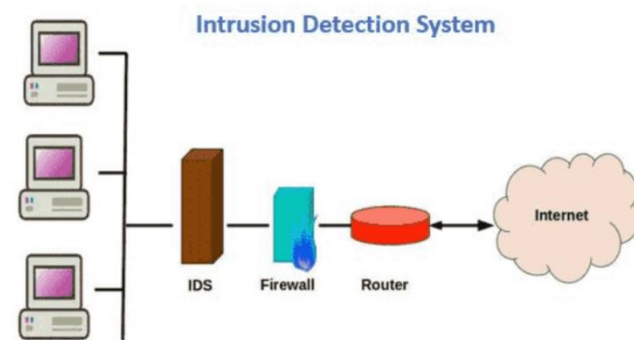


Figure 1.1: Intrusion Detection System

An anomaly, often termed as an outlier, represents data that diverges from the typical behaviour of a system, as learned by the detection system through a supplied set of accepted network behaviour data. Anomalies serve as indicators

that either a fault exists within the system or that a malicious event has compromised it. Despite its susceptibility to elevated false alarm rates, wherein specific activities are flagged as threats even when they are not, anomaly-based detection systems can be favoured over signature-based methods. This preference arises from their capability to identify unseen attacks, for which the system lacks predefined signatures.

Artificial Neural Networks (ANNs) stand as a cornerstone in the realm of artificial intelligence, drawing inspiration from the intricate workings of the human brain to simulate learning and decision-making processes. These networks consist of interconnected nodes, or "neurons," organized in layers, each layer contributing to the transformation of input data into meaningful output. ANNs have garnered immense popularity owing to their ability to tackle complex problems across diverse domains, ranging from image and speech recognition to natural language processing and financial forecasting.

Autoencoders represent a fascinating facet of artificial neural networks, offering a unique approach to data representation and feature extraction. Unlike traditional neural networks aimed at predicting outputs based on inputs, autoencoders are designed to reconstruct inputs themselves. This architectural nuance makes them particularly adept at capturing underlying patterns and structures inherent in the data, thereby facilitating tasks such as denoising, dimensionality reduction, and anomaly detection.

Autoencoders serve as efficient feature selection tools by automatically extracting and representing essential features from raw data. Their advantages include unsupervised learning, enabling identification of intricate patterns without

labelled data, and dimensionality reduction, reducing computational complexity while preserving critical information. Additionally, autoencoders excel in capturing nonlinear relationships within data, enhancing model performance, and generalization. Their ability to handle diverse data types and adapt to various tasks makes them invaluable for streamlined feature selection in a wide range of applications.

1.2 Motivation

The rapid evolution of cyber threats presents an ever-growing challenge for network security professionals. As a final year project, delving into "Feature Extraction Using Autoencoders in Neural Network-based Intrusion Detection Systems" offers a compelling opportunity to address this pressing issue.

Autoencoders, with their innate ability to learn compact representations of data, present a promising avenue for enhancing the efficacy of intrusion detection systems (IDS). By leveraging autoencoders within neural network architectures, we aim to revolutionize the process of feature extraction, enabling the IDS to discern subtle patterns indicative of potential threats amidst the vast sea of network traffic.

Through this project, we aspire to make significant contributions to the field of cybersecurity. By exploring the intricacies of autoencoder-based techniques and their integration into neural network architectures, we aim to provide network

defenders with powerful tools to stay ahead of emerging threats and safeguard critical infrastructures.

1.3 Problem Statement:

This project addresses the need for improved feature extraction methods by exploring the utilization of autoencoders within neural networks. This investigation seeks to overcome the limitations of existing feature extraction approaches and pave the way for more robust and adaptive intrusion detection systems capable of detecting sophisticated cyber threats.

1. Investigate the efficacy of autoencoders in extracting relevant features from network traffic data while reducing dimensionality and preserving critical information.

2. Design and implement a neural network-based IDS architecture augmented with autoencoder-based feature extraction modules.

3. Evaluate the performance of the proposed framework in terms of detection accuracy, false positive rate, and computational efficiency using real-world datasets.

Chapter 2

Basic Concepts

This chapter provides information on the background involved in developing this research. It introduces the concepts and technologies of Intrusion Detection System (IDS). Further, this chapter addresses the characteristics to be considered while developing Security solutions. Later, explanation of the network architecture is provided for the Artificial Neural Network (ANN) and Autoencoders. The final sections of the chapter explain the importance of Machine Learning and Deep Learning for security by quoting their relevant applications. The main intention of this chapter is to provide a complete overview of the concepts and algorithms for the reader with minimal knowledge in this field of research.

2.1 Importance of Network Security

Network security is paramount in today's interconnected world where digital technologies pervade every aspect of our lives. With the exponential growth of online activities, ranging from financial transactions and communication to critical infrastructure management, ensuring the security of networks has become imperative. Network security safeguards sensitive data, confidential information, and critical systems from unauthorized access, data breaches, and cyberattacks. It preserves privacy, maintains trust in online interactions, and protects individuals, organizations, and nations from financial losses, reputational damage, and disruption of essential services. In essence, network security is

indispensable for fostering a safe and secure digital environment conducive to innovation, collaboration, and economic growth in the modern era.

2.2 Intrusion Detection System (IDS)

Network security can be achieved with the help of a software application called an Intrusion Detection System (IDS), which monitors the malicious activities of the system or the networks. IDS can be classified into different types. Based on their responsive nature, IDS is categorized into Active IDS and Passive IDS.

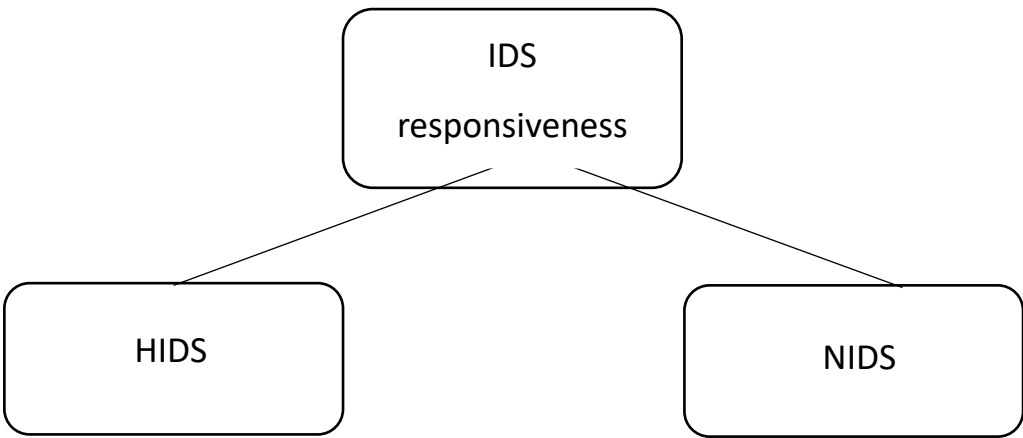


Figure 2.1: Types of IDS based on responsiveness.

An Active IDS is designed to block the malware attacks automatically, without any human intervention, whereas a passive IDS only monitors the network traffic and alerts the users.

Another categorization of IDS is Signature-Based IDS and Anomaly based IDS. In the signature-based approach, the IDS access a database of known signatures and vulnerabilities.

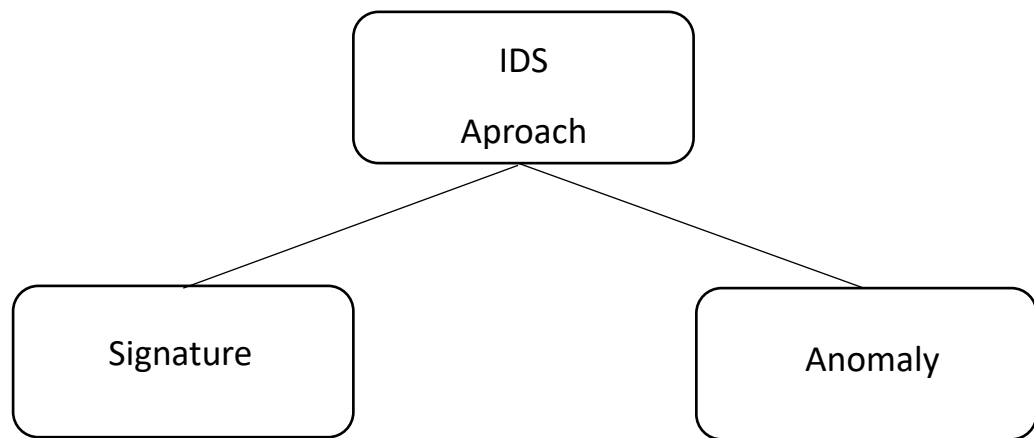


Figure 2.2: Types of IDS based on approach

Each intrusion attack contains the complete details of the attack called a signature and that is used to detect and prevent future attacks. The main drawback of this method is that the database needs to be updated frequently, whereas the anomaly-based IDS (behavior-based) learns from the baseline patterns to detect new intrusion attacks. Any deviation from the existing baseline patterns are identified as attacks and alarms are triggered.

Another division of the IDS is based on the place where it is mounted. When an IDS is placed on the network segment it is said to be a Network Intrusion Detection System, whereas, when an IDS is deployed in workstations, they are said to be Host-based Intrusion Detection Systems.

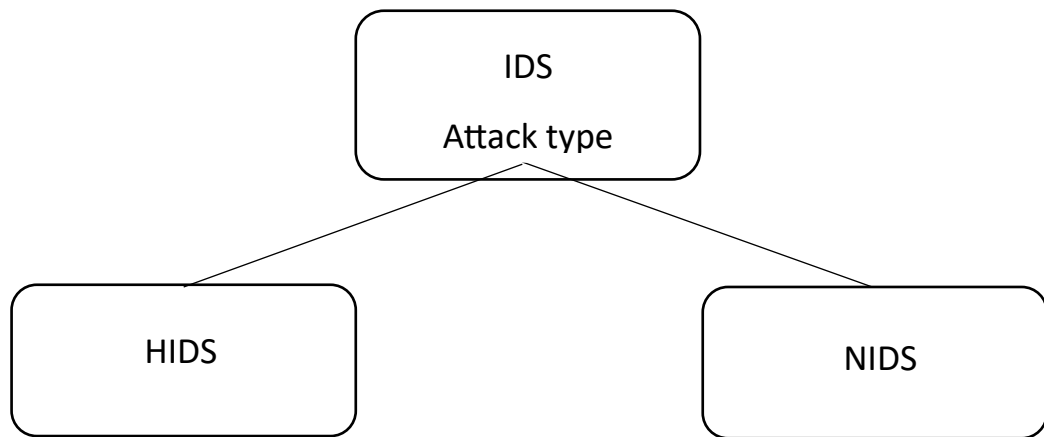


Figure 2.3. Types of IDS based on attack type.

Host-based Intrusion Detection Systems have a significant number of drawbacks and may not be suitable for research purposes.

2.3 Machine Learning

Machine Learning falls under the umbrella of Artificial Intelligence, wherein machines are equipped to acquire the ability to perform tasks autonomously, without explicit programming. This involves the machine's capacity to learn from data and experiences, employing statistical methodologies that allow for enhancement over time. The primary focus of Machine Learning revolves around classification and prediction based on insights gleaned from previously assimilated data during training. This approach to Machine Learning encompasses two distinct phases: training and testing.

Machine Learning is further categorized into three main types:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

2.4 Artificial Neural Network

Artificial Neural Networks contain artificial neurons which are called units. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset.

In majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.

2.4.1 Artificial Neural Networks Architecture

There are three layers in the network architecture: the input layer, the hidden layer (more than one), and the output layer.

Input Layer:

The input layer receives the initial data or features to be processed by the network. Each node in this layer represents a feature or input variable.

Hidden Layer(s):

Hidden layers are intermediary layers between the input and output layers where the neural network processes and transforms the input data.

Each node in a hidden layer performs a computation using weighted inputs from the previous layer, applying an activation function to produce an output.

ANNs can have one or multiple hidden layers, and the number of nodes in each hidden layer and the choice of activation function are design parameters.

Output Layer:

The output layer produces the final output of the neural network based on the processed information from the hidden layers.

The number of nodes in the output layer depends on the nature of the task; for example, it may correspond to the number of classes in a classification task or a single node for regression tasks.

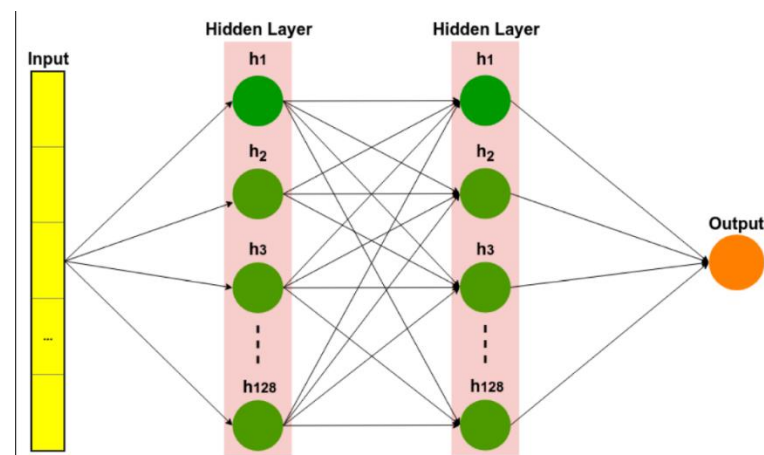


Figure 2.4: Artificial Neural Networks Architecture

2.4.2 How does Neural Network works

Artificial neural networks are trained using a training set. For example, suppose you want to teach an ANN to recognize a cat. Then it is shown thousands of different images

of cats so that the network can learn to identify a cat. Once the neural network has been trained enough using images of cats, then you need to check if it can identify cat images correctly. This is done by making the ANN classify the images it is provided by deciding whether they are cat images or not. The output obtained by the ANN is corroborated by a human-provided description of whether the image is a cat image or not. If the ANN identifies incorrectly then back-propagation is used to adjust whatever it has learned during training. Backpropagation is done by fine-tuning the weights of the connections in ANN units based on the error rate obtained. This process continues until the artificial neural network can correctly recognize a cat in an image with minimal possible error rates.

2.4.3 Types of Neural Network

Common types of neural networks include:

Feedforward Neural Networks: Information flows in one direction, from input to output.

Convolutional Neural Networks (CNNs): Designed for image processing, utilizing convolutional layers for feature extraction.

Recurrent Neural Networks (RNNs): Suited for sequential data, with connections between units forming directed cycles.

Long Short-Term Memory (LSTM) Networks: A type of RNN with memory cells, capable of learning and remembering over long sequences.

Generative Adversarial Networks (GANs): Comprising a generator and a discriminator, trained adversarial to generate realistic data.

Autoencoder: Learns to encode input data into a lower-dimensional representation and decode it back to the original form, useful for dimensionality reduction and unsupervised learning.

2.5 Perceptron

The Perceptron stands as one of the most basic forms of artificial neural network architecture, pioneered by Frank Rosenblatt in the 1950s. It represents the simplest variant of a feedforward neural network, comprising a sole layer of input nodes fully connected to an output layer. Proficient in learning linearly separable patterns, it employs a distinct type of artificial neurons called threshold logic units (TLU), initially conceptualized by McCulloch and Walter Pitts in the 1940s.

2.5.1 Artificial neurons Vs Biological neurons

The concept of artificial neural networks comes from biological neurons found in human brains. So they share a lot of similarities in structure and function wise.

Structure:

The structure of artificial neural networks is inspired by biological neurons. A biological neuron has a cell body or soma to process the impulses, dendrites to receive them, and an axon that transfers them to other neurons. The input nodes of artificial neural networks receive input signals, the hidden layer nodes compute these input signals, and the output layer nodes compute the final output by processing the hidden layer's results using activation functions.

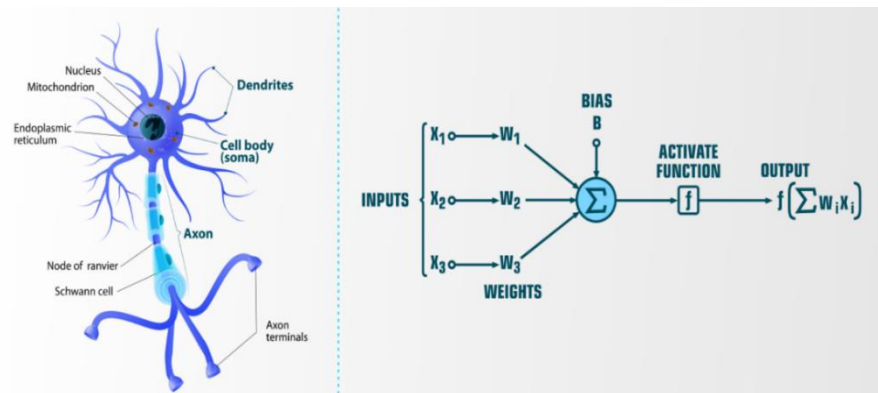


Figure 2.5: Neuron Vs Perceptron

Synapses:

Synapses are the links between biological neurons that enable the transmission of impulses from dendrites to the cell body. Synapses are the weights that join the one-layer nodes to the next-layer nodes in artificial neurons. The strength of the links is determined by the weight value.

Learning:

In biological neurons, learning happens in the cell body nucleus or soma, which has a nucleus that helps to process the impulses. An action potential is produced and travels through the axons if the impulses are powerful enough to reach the threshold. This becomes possible by synaptic plasticity, which represents the ability of synapses to become stronger or weaker over time in reaction to changes in their activity. In artificial neural networks, backpropagation is a technique used for learning, which adjusts the weights between nodes according to the error or differences between predicted and actual outcomes.

2.5.2 Components of Perceptron

It consists of the following components:

Input Layer: This layer receives input values, each representing a feature of the data being processed. Each input is assigned a weight, which determines its relative importance in the model.

Weights: Weights are assigned to each input in the input layer. These weights are adjustable parameters that the perceptron learns during the training process. They represent the strength of the connection between the inputs and the neuron.

Bias: The perceptron also has a bias term, which is similar to the intercept in a linear equation. It allows the perceptron to better fit the data by providing flexibility in the output.

Aggregation Function: The weighted sum of the inputs along with the bias term is calculated. This aggregation function is a linear combination of the inputs and weights.

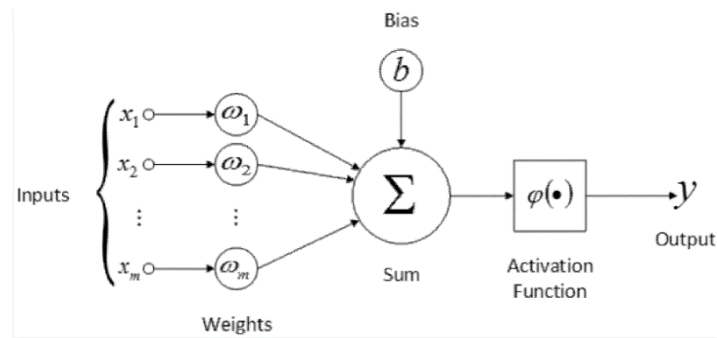


Figure 2.6: Architecture of Perceptron

Activation Function: The result of the aggregation function is passed through an activation function, which introduces non-linearity to the model. The most common activation function for perceptron is the step function, which outputs 1 if the aggregated

sum is above a certain threshold and 0 otherwise. Other activation functions such as the sigmoid or ReLU functions can also be used for more complex tasks.

Output: The output of the activation function represents the predicted class label for the input data.

Learning Algorithm (Weight Update Rule): During training, the perceptron learns by adjusting its weights and bias based on a learning algorithm. A common approach is the perceptron learning algorithm, which updates weights based on the difference between the predicted output and the true output.

2.5.3 How does Perceptron work?

Each input node of a perceptron is assigned a weight, denoting its importance in determining the output. The perceptron's output is computed by summing the weighted inputs, which are then passed through an activation function to determine whether the perceptron should fire. This computation can be expressed as

$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n = X^TW$$

The step function, comparing this sum to a threshold, is the most commonly used activation function for perceptrons. The Heaviside step function is a typical example

$$h(z) = \begin{cases} 0 & \text{if } z < \text{Threshold} \\ 1 & \text{if } z \geq \text{Threshold} \end{cases}$$

In a perceptron, a single layer comprises threshold logic units (TLUs), each connected to all inputs. This setup forms a fully connected layer, where each neuron is linked to every neuron in the preceding layer. The output of such a layer can be represented as

$$f_{W,b}(X) = h(XW + b)$$

where X represents the input, W signifies the weights for each input neuron, b denotes the bias, and h is the step function.

During training, the perceptron's weights are adjusted to minimize the disparity between predicted and actual outputs. This process typically involves supervised learning algorithms such as the delta rule or the perceptron learning rule. An adjustment formula for the weights is given by $w_{i,j} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$, where $w_{i,j}$ represents the weight between the i th input and j th output neuron, x_i is the i th input value, y_j and $\hat{y}_j$ are the j th actual and predicted values, and η denotes the learning rate.

2.6 Process of ANN Model Making

Creating an Artificial Neural Network (ANN) model involves several steps. Here is a generalized process:

Define the Problem: Clearly articulate what problem you want the ANN to solve. Whether it is classification, regression, or another task, understanding the problem is crucial.

Data Collection and Preprocessing: Gather relevant data for your problem. Preprocess the data, including handling missing values, scaling features, and encoding categorical variables.

Feature Extraction/Selection: This step involves identifying the most relevant features from the dataset. In some cases, feature extraction techniques like Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) or other dimensionality reduction techniques are used to transform the data into a lower-dimensional space while preserving important information.

Split Data: Divide your dataset into training, validation, and testing sets. The training set is used to train the model, the validation set to tune hyperparameters, and the testing set to evaluate the final performance.

Choose Network Architecture: Decide on the type and structure of your neural network (e.g., feedforward, convolutional, recurrent).

Determine the number of layers and neurons in each layer.

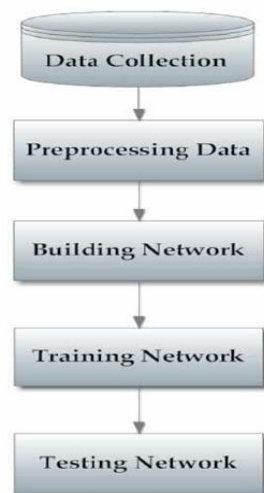


Figure 2.7: Process of Ann Model Making

Choose activation functions for each layer.

Initialize the Model: Initialize the weights and biases of the neural network.

Compile the Model: Specify the loss function, optimizer, and evaluation metrics for training the model. The loss function depends on the task (e.g., mean squared error for regression, categorical cross-entropy for classification). The optimizer determines how the weights are updated during training (e.g., stochastic gradient descent, Adam).

Choose evaluation metrics to monitor during training (e.g., accuracy, precision, recall).

Training: Feed the training data into the model.

The model iteratively adjusts its weights and biases to minimize the loss function. Monitor the model's performance on the validation set to prevent overfitting.

Experiment with different hyperparameters (e.g., learning rate, batch size) to optimize performance.

Evaluate Performance: Once training is complete, evaluate the model's performance on the testing set. Analyze metrics such as accuracy, precision, recall, and F1-score to assess performance.

Tune Hyperparameters (optional): Fine-tune hyperparameters based on the model's performance on the validation set. Hyperparameters include learning rate, number of neurons, number of layers, etc.

2.7 Importance of Feature Extraction

Feature extraction is crucial in the context of machine learning and data analysis for several reasons:

Dimensionality Reduction: Feature extraction reduces the number of features, making data analysis and model training more efficient and less prone to overfitting.

Noise Reduction: By identifying and removing irrelevant features, feature extraction improves model accuracy and generalization.

Improved Model Performance: Focusing on informative features enhances model accuracy and robustness.

Interpretability: Extracting meaningful features makes models easier to interpret, aiding in understanding data patterns.

Computational Efficiency: Feature extraction reduces computational complexity, making models more scalable for large datasets.

Domain-Specific Knowledge: Incorporating domain knowledge improves model alignment with specific requirements.

Handling Multicollinearity: Feature extraction mitigates multicollinearity issues by transforming correlated features.

In summary, feature extraction is vital for enhancing model efficiency, accuracy, interpretability, and scalability while incorporating domain knowledge and addressing multicollinearity.

2.8 Autoencoders

Autoencoders represent a specialized category of algorithms capable of learning efficient data representations without the need for labeled data. They belong to the realm of artificial neural networks primarily tailored for unsupervised learning. The fundamental concept behind autoencoders revolves around mastering the compression and effective representation of input data in the absence of explicit labels. This is achieved through a dual-component architecture comprising an encoder and a decoder. The encoder is responsible for transforming the input data into a condensed, lower-dimensional representation commonly referred to as the "latent space" or "encoding." Subsequently, the decoder reconstructs the original input from this representation. Through the iterative process of encoding and decoding, the network discerns meaningful patterns within the data, thereby delineating essential features.

2.8.1 Architecture of Autoencoder

The general architecture of an autoencoder includes an encoder, decoder, and bottleneck layer.

Encoder:

Input layer: take raw input data.

The hidden layers progressively reduce the dimensionality of the input, capturing important features and patterns. These layers compose the encoder.

The bottleneck layer (latent space): is the final hidden layer, where the dimensionality is significantly reduced. This layer represents the compressed encoding of the input data.

Decoder:

The bottleneck layer takes the encoded representation and expands it back to the dimensionality of the original input.

The hidden layers progressively increase the dimensionality and aim to reconstruct the original input.

The output layer: produces the reconstructed output, which ideally should be as close as possible to the input data.

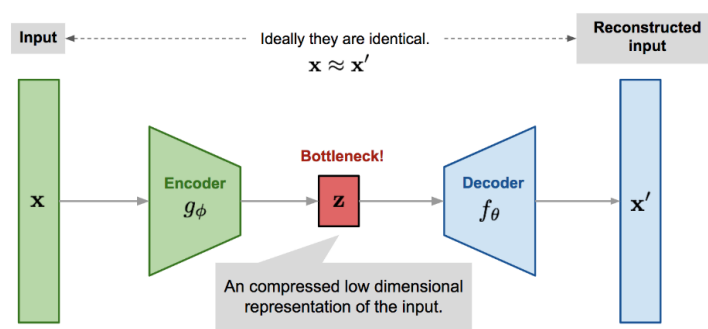


Figure 2.8: Architecture of autoencoder

The loss function used during training is typically a reconstruction loss, measuring the difference between the input and the reconstructed output. Common choices include mean squared error (MSE) for continuous data or binary cross-entropy for binary data.

During training, the autoencoder learns to minimize the reconstruction loss, forcing the network to capture the most important features of the input data in the bottleneck layer.

After the training process, only the encoder part of the autoencoder is retained to encode a similar type of data used in the training process.

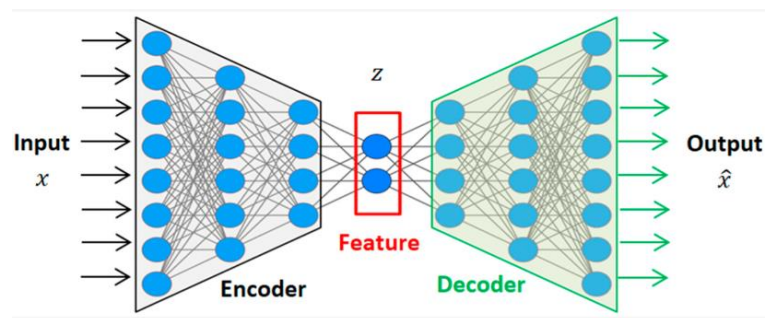


Figure 2.9: Components of autoencoder

2.8.2 Types of Autoencoders

Vanilla Autoencoder: it consists of an encoder and a decoder with symmetrical architectures. The objective is to reconstruct the input as accurately as possible.

Sparse Autoencoder: Sparse autoencoders introduce sparsity constraints during training. This can lead to more robust and interpretable feature representations.

Denoising Autoencoder: Denoising autoencoders are trained to reconstruct clean data from noisy inputs. During training, the model is presented with corrupted versions of the input data and learns to denoise them.

Variational Autoencoder (VAE): learn to generate new data samples from a learned probability distribution in the latent space. VAEs are often used for generative modeling tasks.

Sequence-to-Sequence Autoencoder: Sequence-to-sequence autoencoders are designed for sequential data, such as text or time series.

Stacked Autoencoder: Stacked autoencoders consist of multiple layers of autoencoders, where the output of one layer serves as the input to the next layer. Stacked autoencoders are often used for deep learning tasks and can learn more powerful representations compared to single-layer autoencoders.

2.8.3 Advantages of Autoencoders

Certainly! Here are the condensed advantages of using autoencoders, including stacked autoencoders:

Unsupervised Learning: Autonomous feature learning without labels.

Feature Learning: Capturing salient patterns from input data.

Dimensionality Reduction: Efficiently reducing data complexity.

Data Denoising: Filtering noise for robust representations.

Generative Modeling: Creating realistic data samples.

Hierarchical Representation: Learning complex patterns at different levels.

Transfer Learning: Pre-trained models for accelerated training.

Regularization: Implicitly preventing overfitting.

Robustness to Variations: Stability to input perturbations.

Scalability and Efficiency: Efficient training on large datasets.

These succinct points highlight the versatility, efficiency, and robustness of autoencoders in various machine learning tasks.

2.9 Definitions of Important Term

2.9.1 StandardScaler

StandardScaler is a preprocessing technique used in machine learning and data analysis to standardize the features by removing the mean and scaling to unit variance. This transformation ensures that each feature has a mean of 0 and a standard deviation of 1. StandardScaler is particularly useful when dealing with features that have different scales or units, as it helps algorithms to converge faster and can improve the performance of models.

The standard score of a sample x is calculated as:

$$z = (x - u) / s$$

where u is the mean of the training samples, and s is the standard deviation of the training samples.

2.9.2 OneHotEncoder:

One hot encoding is a technique used in machine learning and data preprocessing to convert categorical variables into a binary format, where each category is represented by a binary vector.

It is useful because it allows machine learning algorithms to work with categorical data, which many algorithms cannot directly handle. It's commonly used in scenarios where you have categorical variables with no inherent ordinal relationship, such as "color" or "type," and you want to avoid creating an artificial ordinal relationship by assigning numerical labels.

2.9.3 Dropout Layer

A Dropout layer is a regularization technique used in neural networks during training to prevent overfitting. It works by randomly "dropping out" (i.e., setting to zero) a proportion of the input units of the layer during each training iteration. This means that during forward and backward passes, some neurons are randomly ignored with a probability defined by the dropout rate.

By dropping out neurons, Dropout prevents units from co-adapting too much, forcing the network to learn more robust features. It also acts as a form of ensemble learning, as the network must learn redundant representations of the data.

During inference (i.e., when making predictions), Dropout is typically turned off, and the full network is used.

2.9.4 Optimizer (Adam)

Adam (Adaptive Moment Estimation) is an optimization algorithm commonly used in training neural networks. It combines the benefits of two other popular optimization algorithms: AdaGrad and RMSProp. Adam maintains two moving averages of gradients: the first moment (the mean) and the second moment (the uncentered variance). These moving averages are used to adaptively adjust the learning rates for each parameter during training.

Adam computes adaptive learning rates for each parameter, allowing it to converge faster and more reliably than traditional stochastic gradient descent (SGD). It's particularly effective in training deep neural networks on large datasets, as it automatically adjusts learning rates based on the gradients' magnitudes and directions

2.9.5 Loss Function

A loss function, also known as a cost function or objective function, is a measure of how well a machine learning model's predictions match the actual target values in a dataset. It quantifies the error or mismatch between the predicted and actual values, providing feedback to the model during training.

Different types of machine learning tasks (e.g., regression, classification, sequence generation) require different loss functions tailored to the specific problem and the output format of the model.

2.9.6 EarlyStopping

Early stopping is a technique used in machine learning to prevent overfitting and improve model generalization. During training, the performance of the model is monitored on a separate validation dataset. If the performance stops improving for a certain number of epochs, typically defined by a patience parameter, training is stopped early. This prevents the model from continuing to learn from noise or memorizing the training data excessively. Early stopping helps find the optimal balance between model complexity and generalization, reducing the risk of overfitting while saving computational resources and training time. It's a simple yet effective regularization technique widely used in training neural networks and other machine learning models.

2.10 Evaluation Metrics

To evaluate the performance of the classification model the following metrics are used in machine learning research. In general, the confusion matrix visualizes the performance of the algorithm in a tabular form as shown in the figure below:

	Predicted as Normal	Predicted as Attack
Actually as Normal	TP	FP
Actually as Attack	FN	TN

Figure 2.10: Confusion Metrics

Where,

True Positive (TP) is the total number of samples predicted as “normal” while they were “normal”.

False Negative (FN) is the total number of samples predicted “normal” while they were “attack”.

False Positive (FP) is the total number of samples predicted “attack” while they were “normal”.

True Negative (TN) is the total number of samples predicted “attack” while they were “attack”.

All other important metrics such as Precision, Accuracy, Recall, False Alarm Rate (FAR) and Area under ROC curve (AUC) can be calculated using these 4 measures taken from the confusion matrix.

Literature Review

In this segment, we initially provide an overview of the primary forms of assaults on an Intrusion Detection System (IDS). Subsequently, we elaborate on the methodologies and tactics employed in the development of such systems, alongside the metrics utilized to assess their efficacy.

3.1 Literary Review on Attacks

An assault denotes a security menace encompassing endeavors to pilfer, manipulate, or access information without authorization, or to infiltrate a network without consent. It encompasses actions possessing traits capable of compromising information or network integrity. The predominant assault categories include Denial of Service (DoS), User to Root (U2R), Remote to User (R2L), and Probing (Ahmed, Naser Mahmood, Hu 2016; Agrawal, Soni, Agrawal 2017). These assault categories are further delineated in subsequent sections.

Denial of Service Attack: Denial of Service is a type of attack whereby the attacker aims to prevent the authorised user from using the computer services. DoS can be done by flooding the network and disrupting a connection or service. Examples of DoS attacks include Smurf, Land and Ping of Death (Pod) (Dias, Cerqueira, Assis, Almeida 2017).

User to Root Attack: The user to root attack is a type of attack whereby the attacker gets access to a normal user account and then exploits system vulnerabilities to gain root

privileges. Examples of U2R attacks are Loadmodule, Rootkit and Buffer_overflow (Dias, Cerqueira, Assis, Almeida 2017).

Remote to User Attack: The remote to User attack is a type of attack whereby the attacker aims to get unauthorised access to a local machine to send packets over the network. An R2L attack allows the attacker to have privileges which a local user normally have when using that computer. Examples of U2L attacks are Ftp_write, Warezclient and Imap (Dias, Cerqueira, Assis, Almeida 2017).

Probing: A probe is a program that can be used to automatically scan and monitor the network activities or collect data from the network. The attacker collects information about the network and also finds vulnerabilities which can be used to attack the network. Examples of probing attacks are Ipsweep, Nmap and PortswEEP (Dias, Cerqueira, Assis, Almeida 2017).

3.2 Literary Review on Performance metrics/variables

Various metrics can be employed to evaluate and gauge the effectiveness of diverse techniques. Below, we outline some of the metrics commonly utilized in numerous studies (Kumar 2014; Agrawal, Soni, Agrawal 2017).

Accuracy: Accuracy denotes the capability of the IDS to detect intrusions and generate genuine alarms when an intrusion is genuinely present and identified. It is quantified by the ratio of correctly classified instances to the total number of samples within the dataset (Agrawal, Soni, Agrawal 2017).

Efficiency: Efficiency pertains to the utilization of resources allocated to the system for conducting intrusion detections. The system is deemed efficient if it utilizes resources to detect intrusions promptly (Kumar 2014).

Effectiveness: Effectiveness signifies the system's ability to differentiate between intrusion activities and non-intrusion activities (Kumar 2014). An IDS is effective if it exhibits a low false alarm rate, defined as the ratio between incorrect instances and the total number of normal instances (Agrawal, Soni, Agrawal 2017).

Reliability: Reliability denotes how well a detection approach fulfills its designated functions within a specified time frame and under specified conditions or in the event of a failure (Kaur, Kumar, Bhandari 2017).

Implementation Cost: Implementation cost refers to the total expenditure required for implementing a specific detection technique on the source-end, victim end, or core-end network (Kaur, Kumar, Bhandari 2017).

3.3 Literary Review on Intrusion detection approaches

An intrusion detection system, whether in the form of a device or software application, functions to monitor networks or systems for malicious activities or violations of policies (Agrawal, Soni, Agrawal 2017). Various approaches are employed in the development of such systems, encompassing Statistical-Based Anomaly Detection, Pattern Matching, Data Mining, and Machine Learning.

Statistical-Based Anomaly Detection: This approach utilizes statistical analysis to evaluate user or system behavior, examining values across different variables such as login session parameters (Jose, Malathi, Reddy, Jayaseeli 2018). Statistical properties are leveraged during anomaly detection to distinguish between intrusive and normal actions within the system.

Pattern Matching: Pattern matching involves detecting intrusions by comparing incoming traffic patterns with existing patterns (Agrawal, Soni, Agrawal 2017). Intrusions are identified by comparing current patterns with known attack signatures, which are regularly updated to enable the system to recognize known attacks based on saved signatures.

Data Mining: Data mining is utilized for extracting insights from large datasets and also for detecting intrusions in scenarios where data processing is challenging due to the dataset's size (Agrawal, Soni, Agrawal 2017).

Machine Learning: Machine learning entails systems that learn and refine their learning capabilities over time. It is particularly valuable for recognizing new attacks promptly (Agrawal, Soni, Agrawal 2017). Machine learning techniques encompass Neural Networks, Fuzzy Logic, and Support Vector Machine techniques.

4.4 Literary Review on Machine learning techniques

Support Vector Machine (SVM): SVM is a machine learning technique adept at diverse classification tasks, data analysis, and pattern recognition (Chowdhury, Ferens, Ferens 2016). SVMs are harnessed in IDSs for real-time intrusion detection capabilities (Shah, Hayat, Awan 2015).

Artificial Neural Network (ANN): ANN, a mathematical model, serves classification purposes by assessing if input data aligns with trained recognition characteristics (Jha, Ragha 2013). The neural network approach in intrusion detection aims to grasp the behavior of different system actors (Patel, Jhaveri 2015). Deep Learning, a subfield of machine learning, employed in IDSs, utilizes intricate architectures to achieve high-level abstractions in data (Ponkarthika, Dr, Saraswathy 2018).

Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs): CNNs and RNNs, two architectures within Deep Learning, are pivotal in IDS construction. CNNs enhance intrusion detection accuracy through improved behavior feature utilization (LIU, LIU, ZHAO 2017). RNNs offer robust modeling capabilities, particularly excelling in threat classification and false positive reduction (Mohammadpour, Ling, Liew, Chong 2018; Lin, Lin, Wang, Wu, Tsai 2018).

Decision Trees: Decision Trees, characterized by their tree-like structures, are potent tools for classification problems, thus utilized in IDSs for intrusion classification and detection (Shah, Hayat, Awan 2015; Rai, Devi, Guleria 2016).

Naïve Bayesian Networks: Naïve Bayesian Networks, prominent for handling uncertain information, demonstrate competitive intrusion detection results owing to their simplicity and accuracy (Amor, Benferhat, Elouedi 2003). The Naïve Bayesian Network's characteristics align with the desired traits of an effective IDS (Amor, Benferhat, Elouedi 2003).

3.5 Discussion

Intrusion detection systems (IDS) are constructed using various methodologies, each with its strengths and limitations. Ideally, an IDS should proficiently identify intrusions, particularly those falling within the four major attack categories (DoS, U2R, R2L, and Probing). However, different intrusion detection approaches exhibit distinct characteristics. The machine learning approach, for instance, operates autonomously with minimal human involvement.

The Statistical-Based Anomaly approach and Data Mining approach often yield numerous false alarms, indicating lower accuracy, a critical metric for effective IDS operation. Specifically, the Statistical-Based Anomaly approach is not recommended for accuracy-sensitive anomaly detection systems due to its subpar performance in providing high accuracy. Additionally, the Data Mining Approach is unsuitable for real-time detection environments, posing a limitation as timely intrusion detection is imperative.

The capability of an IDS to function continuously with minimal human oversight is highly desirable, aligning well with the automation provided by the machine learning approach. Hence, the machine learning approach is favored for IDS implementation requiring uninterrupted operation and minimal human intervention.

Different machine learning techniques utilized for intrusion detection possess unique attributes and drawbacks:

Bayesian Networks can leverage prior knowledge for intrusion detection but may suffer from reduced accuracy in the absence of robust classifiers.

Decision Trees perform well in detecting intrusions from large datasets and in real-time scenarios, although classification accuracy may diminish with numerous categories.

Support Vector Machine offers real-time detection and dynamic training pattern updates but may experience reduced accuracy due to complex feature requirements.

Neural Networks, especially Recurrent Neural Networks, swiftly make decisions and detect intrusions in real time with high accuracy, requiring minimal human intervention and thus ideal for building effective IDS.

Chapter 4

Methodology

4.1 Schematic of the System

This chapter provides a brief information to the design approach and the datasets used in building an intelligent model of an IDS. The key features required for solutions are adaptive, lightweight, multi-layer, distributive and ability to learn from the past. We designed an innovative architecture IDS that would reduce the size of the datasets for the classifier. We have tested the classifier with KDD Cup 1999 and NSL KDD Intrusion Detection Datasets for the experiments and proposed an intelligent solution which satisfies the key requirements of the solutions. We have performed the feature engineering using a Autoencoder and a Stacked Autoencoder and selected those features with high importance. We performed a rigorous data analysis and prepared the data in the required format before it was used as an input to the model.

4.2.1 Data Flow of Feature Selection using

Autoencoder

Data Splitting: The dataset is initially divided into training and test sets to ensure proper evaluation and validation of the models.

Autoencoder for Feature Selection: The training data is passed through an Autoencoder, a neural network architecture, which compresses the input data into a lower-dimensional representation.

Anomaly Detection by Neural Network: The compressed representations obtained from the Autoencoder serve as input features for an anomaly detection task. A separate

neural network model, trained on the compressed representations, is employed for anomaly detection.

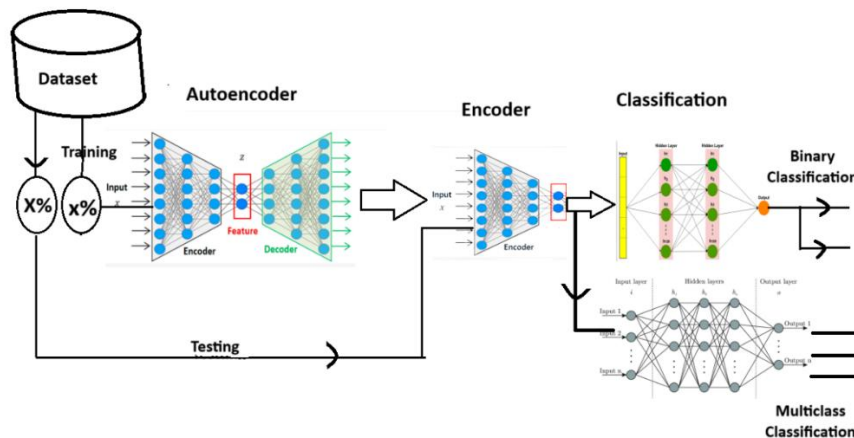


Figure 4.1: Data Flow of Feature Selection

4.1.2 End to End Data Flow

Data Preprocessing: Initial preprocessing steps include feature mapping, which involves transforming raw data into a format suitable for analysis.

One-hot encoding is applied to categorical features, resulting in a total of 113 features after encoding.

Feature scaling is performed to standardize the range of numeric features, ensuring uniformity in feature magnitudes.

Feature Selection Using Autoencoder: The pre-processed data is then subjected to feature selection utilizing an Autoencoder.

The Autoencoder compresses the input features into a lower-dimensional representation, effectively reducing the dimensionality of the dataset.

After feature selection, the dataset is transformed into 32 selected features, capturing the most relevant information while discarding noise and redundant features.

Data Splitting: The dataset, now reduced to 32 selected features, is split into training and test sets to facilitate model training and evaluation.

Neural Network Model for Binary Classification: For binary classification tasks, a neural network model is trained on the training dataset.

The neural network utilizes the selected features to learn patterns and relationships within the data, aiming to classify instances into one of two classes.

Model performance is assessed on the test dataset to evaluate its effectiveness in binary classification tasks.

Neural Network Model for Multiclass Classification: Similarly, for multiclass classification tasks, another neural network model is trained on the training dataset.

This model is configured to classify instances into multiple classes based on the learned features.

Evaluation of the multiclass classification model is conducted on the test dataset to gauge its performance across multiple class labels.

This end-to-end data flow encompasses the entire process from preprocessing and feature selection to model training and evaluation for both binary and multiclass classification tasks.

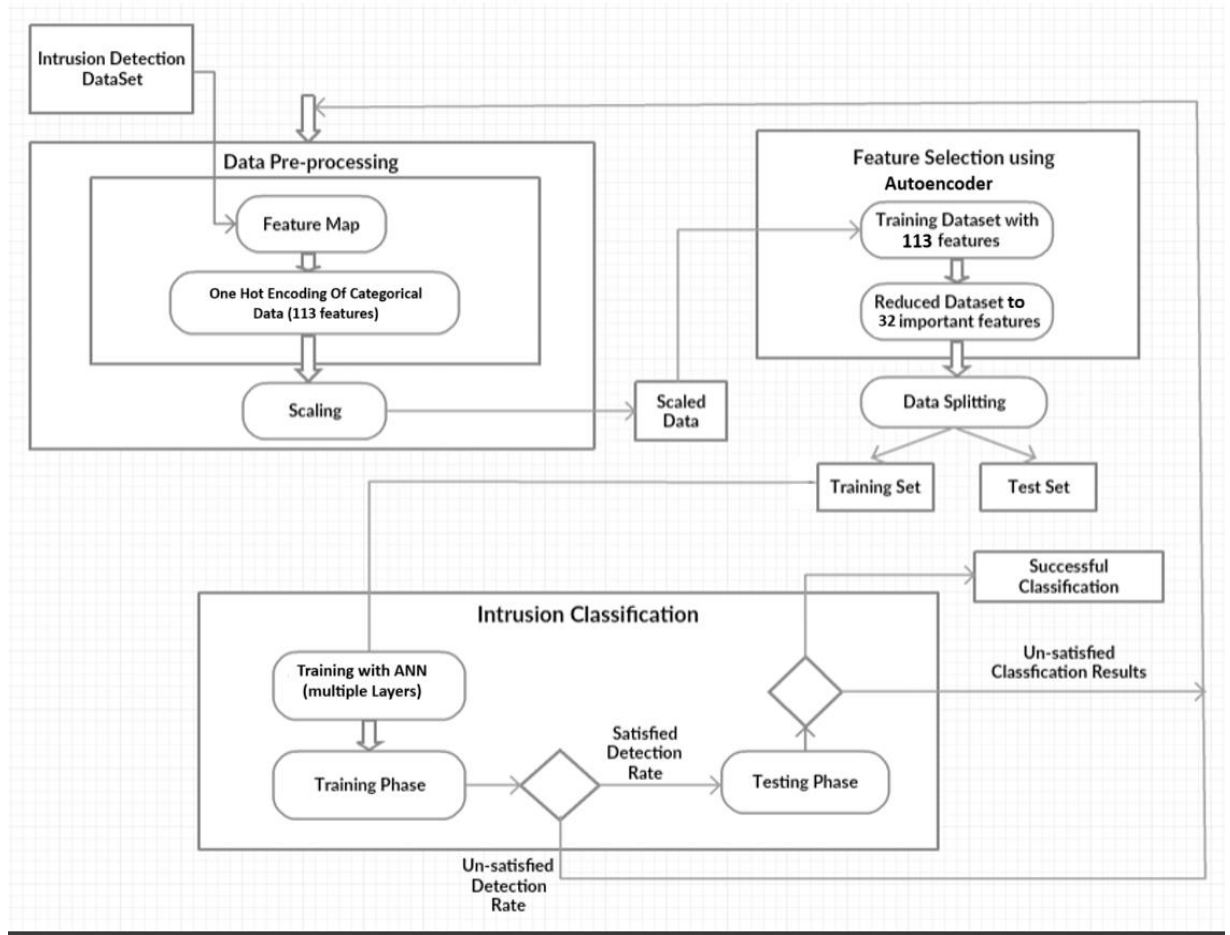


Figure 4.2: End to End Data Flow

4.2 IDS – Datasets

Intrusion detection data for training machine learning algorithms are limited in the literature. The mainly used dataset is and KDD Cup'99 /DARPA datasets and the update version NSL KDD Dataset. This way, the results obtained in this research can be compared with both data sets. The DARPA KDD Cup '99 datasets were generated by the Defense Advanced Research Projects Agency (DARPA ITO) on a simulated air force model. The training data was collected for seven weeks and the testing data were collected for two

weeks. The whole dataset contains 39 network-based attack types and has more than 200 instances of background traffic compared to an air force base model. The complete network traffic is either classified as one of the attack types or "normal". The datasets can be found on the UCI website where repository links to the three different versions of data set exist. The three versions of the KDD 99 Cup IDS datasets are – full KDD data set, corrected KDD, 10% KDD. Among these three, 10% KDD data set is used in most literature. The 10% KDD dataset contains 24 attack types, which are mainly categorized into four classes – **Probe**, **Denial of Service (DoS)**, **User to Root (U2R)** and **Remote to Local (R2L)**. The training and testing samples are represented with 41 features and a label with either "normal" or "attack type". The features can be divided into three types: the first group describes the features that are used for providing information on the command that is used for connections, the second group of features describes the specifications of the commands, and the third group describes the features that convey information about the connections having the same destination with the same service.

<i>Feature Name</i>	<i>Description</i>	<i>Type</i>
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous

ag	ormal or error status of the onnection	iscrete
and	if connection is from/to the same ost/port; 0 otherwise	iscrete
wrong_fragment	umber of ``wrong" fragments	ontinuous
urgent	umber of urgent packets	ontinuous
ot	umber of ``hot" indicators	ontinuous
um_failed_logins	umber of failed login attempts	ontinuous
ogged_in	if successfully logged in; 0 otherwise	iscrete
um_compromised	umber of ``compromised" conditions	ontinuous
ot_shell	if root shell is obtained; 0 otherwise	iscrete
u_attempted	if ``su root" command attempted; 0 therwise	iscrete
um_root	umber of ``root" accesses	ontinuous
um_file_creations	umber of file creation operations	ontinuous
um_shells	umber of shell prompts	ontinuous
um_access_files	umber of operations on access control les	ontinuous
um_outbound_cmds	umber of outbound commands in an p session	ontinuous
e_hot_login	if the login belongs to the ``hot" list; 0 therwise	iscrete
e_guest_login	if the login is a ``guest"login; 0 therwise	iscrete

count	number of connections to the same host as the current connection in the past two seconds	continuous
error_rate	% of connections that have ``SYN'' errors	continuous
error_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
srv_error_rate	% of connections that have ``SYN'' errors	continuous
srv_error_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Table 4.1: KDD 99 Data Description

4.3 Data preprocessing

This documentation outlines the various preprocessing step undertaken to prepare the dataset for further analysis and modeling. Each step contributes to enhancing the quality, usability, and efficiency of the data for subsequent tasks.

Column Names: The column names are extracted from the kddcup.names file, which provides a description of each attribute in the dataset. Assigning meaningful column names enhances the interpretability of the dataset and facilitates further analysis.

Categorization of Attacks: Attacks within the dataset are categorized into five distinct types: Denial of Service (DoS), Probe, Remote-to-Local (R2L), User-to-Root (U2R), and Normal. Categorizing attacks enables targeted analysis and facilitates the development of specialized detection algorithms for each attack type.

Memory Reduction: To optimize memory usage and computational efficiency, data type conversion technique is applied for memory reduction

4.4 Exploratory Data Analysis:

Count of Attack types:

dos	391458
normal	97278
probe	4107
r2l	1126
u2r	52

Percentage of Attack types:

dos	79.239142
normal	19.691066

probe 0.83134

1

r2l 0.227926

u2r 0.010526

Others:

Shape : (494021, 43)

Number of features : 43

Number of unique services : 66

Number of labels : 23

missing values : 0

Categorical features list:

'protocol_type',

'service',

'flag'

Graphs:

Correlation between features(heatmap):

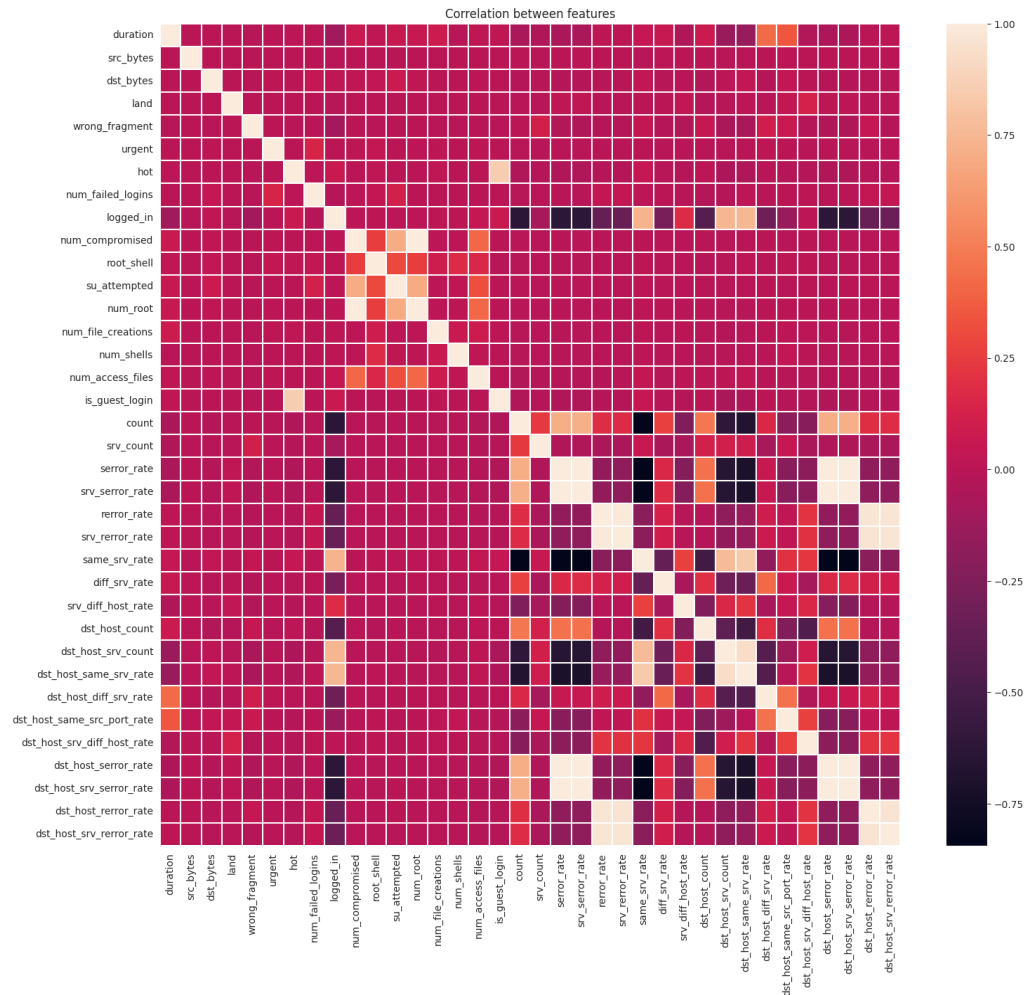


Figure 4.3: Correlation between features (heat map)

Attack Type (count plot)

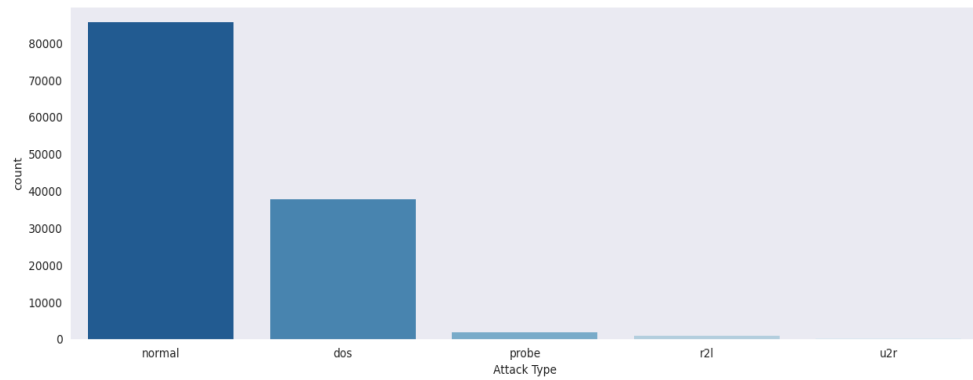


Figure 4.4: Attack type Count Plot

Protocol type (count plot)

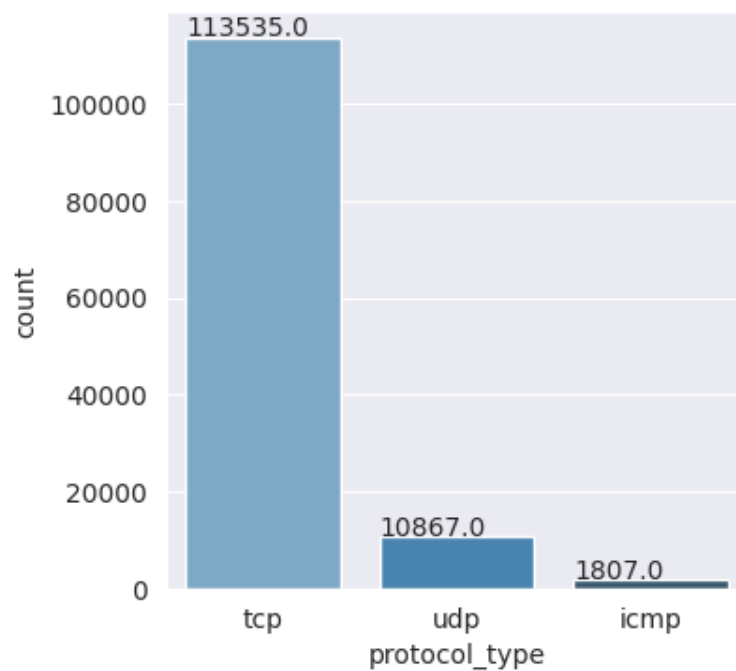


Figure 4.5: Protocol type Count Plot

4.5 Modelling a Neural Network :

Model Selection: Five different neural network architectures are tested for both binary and multiclass classification tasks, varying in the number of hidden layers from 1 to 5.

The models are designed with dropout layers and early stopping mechanisms to prevent overfitting and improve generalization performance.

Model Architecture: Each neural network model follows a similar architecture, consisting of input, hidden, and output layers.

Hidden layers utilize Rectified Linear Unit (ReLU) activation functions to introduce non-linearity and improve model expressiveness.

Output layers utilize either Sigmoid activation function for binary classification or Softmax activation function for multiclass classification to produce class probabilities.

Training Configuration: Models are trained using the Adam optimizer, an adaptive learning rate optimization algorithm.

Training is performed over 40 epochs with a batch size of 128 samples per batch.

Early stopping is applied to monitor validation loss and halt training when performance stops improving, preventing overfitting.

Dropout Regularization: Dropout layers are incorporated into the model architecture to randomly deactivate a fraction of neurons during training, reducing the likelihood of overfitting.

Dropout rates are tuned to strike a balance between regularization and model capacity, ensuring optimal performance.

Model Evaluation: Trained models are evaluated on separate validation and test datasets to assess their performance.

Evaluation metrics such as accuracy, precision, recall, F1-score, are computed to gauge model effectiveness for both binary and multiclass classification tasks.

Selection of the Most Efficient Model: The model with four hidden layers demonstrates the most efficient performance based on validation and test set results.

4.6 Applying Feature selection in selected Model

Model Selection: A neural network model with a predetermined architecture, such as the preferred model with four hidden layers, is chosen as the baseline.

This model serves as the foundation for integrating feature selection techniques using Autoencoders.

Autoencoder Integration: Two different types of Autoencoders are tested: a simple Autoencoder and a stacked Autoencoder with two layers.

The Autoencoders are trained to reconstruct the input data with minimal mean squared error (MSE), utilizing the Adam optimizer and Rectified Linear Unit (ReLU) activation functions for hidden layers.

Feature Selection Process: The selected neural network model is modified to accept the compressed representations generated by the trained Autoencoders as input features.

These compressed representations serve as a reduced-dimensional feature set, capturing the most salient information while discarding irrelevant noise and redundancies.

Training and Evaluation: The modified neural network model, incorporating feature selection through Autoencoders, is trained, and evaluated using the same configuration as the baseline model.

Training is conducted using the Adam optimizer, with MSE as the loss function, and ReLU activation functions for hidden layers.

Evaluation metrics such as accuracy, precision, recall, F1-score are computed to assess the model's performance on validation and test datasets.

Comparison and Selection of Optimal Approach: Performance metrics obtained from models with and without feature selection are compared to determine the effectiveness of integrating Autoencoders for feature selection.

Chapter 5

Results And Evaluation

5.1 Results of Binary Classification

5.1.1 Performance result

Accuracy Scores:

Without Autoencoder

	dnn1	dnn2	dnn3	dnn4	dnn5
accuracy	0.99726	0.99746	0.99793	0.99807	0.99829
F1 Score	0.99814	0.99814	0.99849	0.99881	0.99875
Precision Score	0.99704	0.99704	0.99837	0.99860	0.99866
Recall Score	0.99924	0.99924	0.99860	0.99901	0.99883

Table 5.1: Binary classification Accuracy scores – Without Autoencoder

Simple Autoencoder

	dnn1	dnn2	dnn3	dnn4	dnn5
accuracy	0.99730	0.99690	0.99687	0.99710	0.99738
F1 Score	0.99770	0.99770	0.99768	0.99785	0.99806
Precision Score	0.99829	0.99829	0.99730	0.99829	0.99800
Recall Score	0.99712	0.99712	0.99806	0.99741	0.99812

Table 5.2: Binary classification Accuracy scores – Simple Autoencoder

Stacked Autoencoder

	dnn1	dnn2	dnn3	dnn4	dnn5
accuracy	0.99726	0.99730	0.99718	0.99805	0.99706
F1 Score	0.99801	0.99801	0.99793	0.99824	0.99871
Precision Score	0.99761	0.99761	0.99697	0.99812	0.99847
Recall Score	0.99842	0.99842	0.99889	0.99836	0.99894

Table 5.3: Binary classification Accuracy scores – Stacked Autoencoder

Note:

Accuracy= how many predictions we got right = (True Positive + False Positive) / (Total number of data)

Precision= True Positive / (True Positive + False Positive) Recall= True positive / (True positive + False negative)

F1 - score = Harmonic mean of precision and recall = $2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$

Without Feature Selection : (mean values)

	dnn1	dnn2	dnn3	dnn4	dnn5
accuracy	0.99795	0.99810	0.99791	0.99820	0.99819
Validation accuracy	0.99744	0.99753	0.99766	0.99759	0.99767
loss	0.00789	0.00702	0.00768	0.00653	0.00667
Validation loss	0.01297	0.02066	0.01698	0.01648	0.01785

Table 5.4: Binary classification mean values – Without Autoencoder

With Feature Selection with Simple Autoencoder : (mean values)

	dnn1	dnn2	dnn3	dnn4	dnn5
accuracy	0.99333	0.99555	0.99352	0.99605	0.99603
Validation accuracy	0.99495	0.99655	0.99646	0.99739	0.99754
loss	0.02665	0.01668	0.02191	0.01426	0.01439
Validation loss	0.01997	0.01274	0.01165	0.01005	0.00970

Table 5.5: Binary classification mean values – Simple Autoencoder

With Feature Selection with Stacked Autoencoder : (mean values)

	nn1	nn2	nn3	nn4	nn5
accuracy	0.99457	0.99565	0.99407	0.99495	0.99539
validation accuracy	0.99490	0.99620	0.99601	0.99602	0.99641
loss	0.01934	0.01563	0.01890	0.01641	0.01515
validation loss	0.02620	0.01357	0.01296	0.01751	0.01185

Table 5.6: Binary classification mean values – Stacked Autoencoder

Comparing model with NSL KDD And KDD Cup 1999

dataset:

Selected model – with 4 hidden layers

	KDD cup 99		NSL KDD	
	Accuracy	F1 Score	Accuracy	F1 Score
Without Autoencoder	0.99807	0.99807	0.99067	0.99127
Simple Autoencoder	0.99710	0.99710	0.98968	0.99039
Stacked Autoencoder	0.99805	0.99805	0.99067	0.99128

Table 5.7: Binary classification Dataset Comparison

Observation:

The observed decrease in overfitting when employing Autoencoder in the model, on both the KDD Cup 99 dataset and NSL KDD dataset implies a significant improvement in the models' ability to generalize. Overfitting, where a model learns noise or irrelevant details from the training data, can severely impair its performance on new, unseen data.

By reducing overfitting, our models demonstrate an enhanced capacity to extract essential features and patterns from the datasets while filtering out irrelevant noise. This means that they are better equipped to classify data accurately beyond the training samples, making them more reliable tools for real-world applications.

In summary, the decrease in overfitting signifies the effectiveness of our approach in building models that generalize well to new data, thus improving their practical utility for classification tasks.

5.1.2 Graphs:

KDD Cup 99 –

Without Autoencoder:

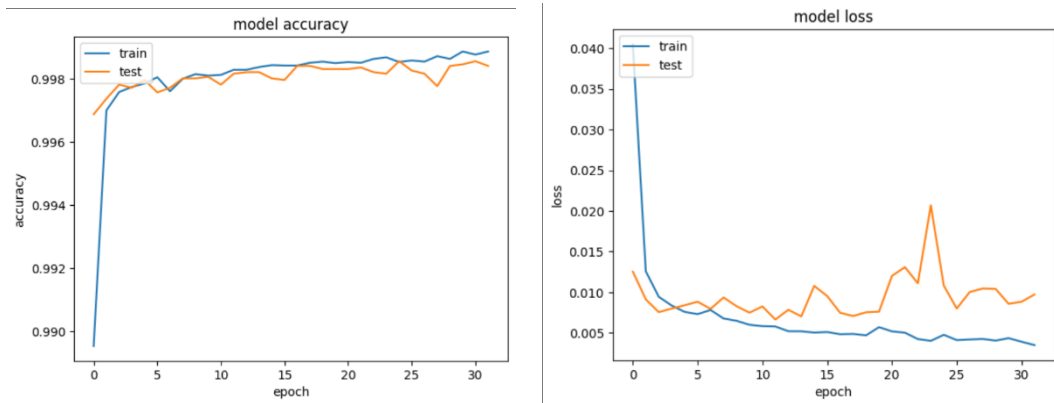


Figure 5.1: Binary Classification–Model Accuracy (without Autoencoder, KDD Cup’ 99)

Figure 5.2: Binary Classification –Model Loss (without Autoencoder, KDD Cup’ 99)

With Autoencoder(stacked):

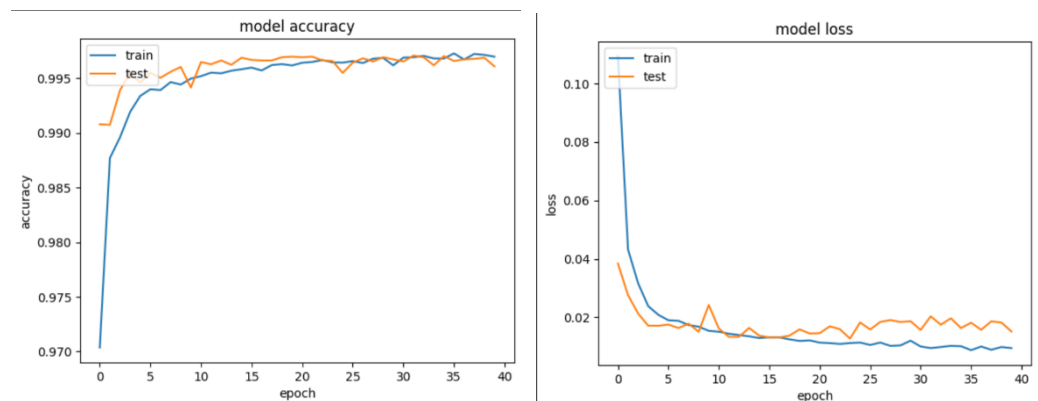


Figure 5.3: Binary Classification–Model Accuracy (Stacked Autoencoder, KDD Cup’ 99)

Figure 5.4: Binary Classification –Model Loss (Stacked Autoencoder, KDD Cup’ 99)

NSL KDD –

Without Autoencoder:

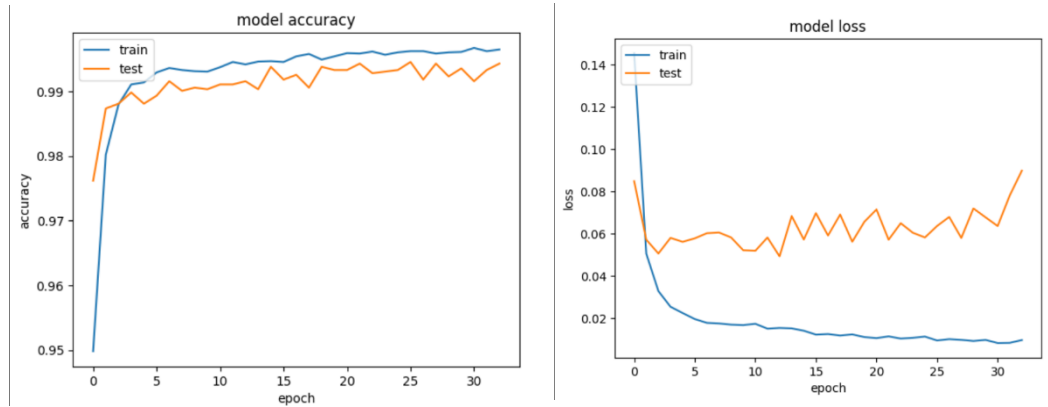


Figure 5.5: Binary Classification–Model Accuracy (without Autoencoder, NSL KDD)

Figure 5.6: Binary Classification –Model Loss (without Autoencoder, NSL KDD)

With Autoencoder(stacked):

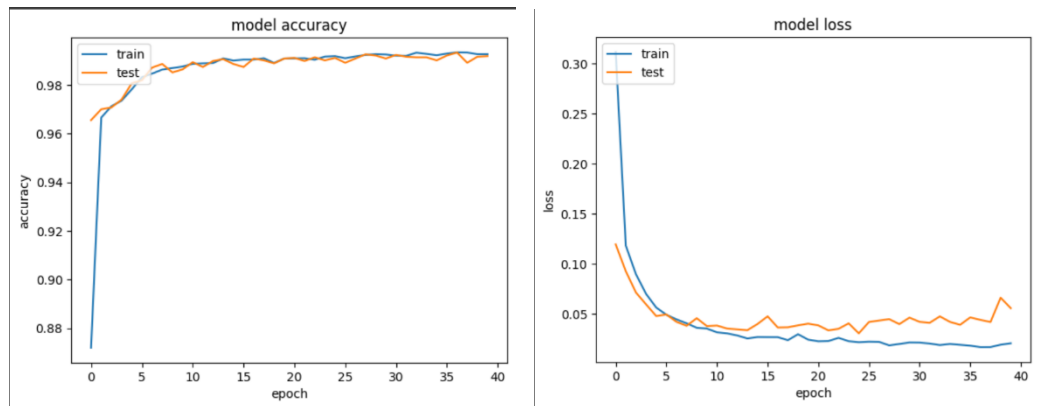


Figure 5.7: Binary Classification–Model Accuracy (Stacked Autoencoder, NSL KDD)

Figure 5.8: Binary Classification –Model Loss (Stacked Autoencoder, NSL KDD)

5.1.3 Confusion Matrices

KDD Cup 99 –

Without Autoencoder:

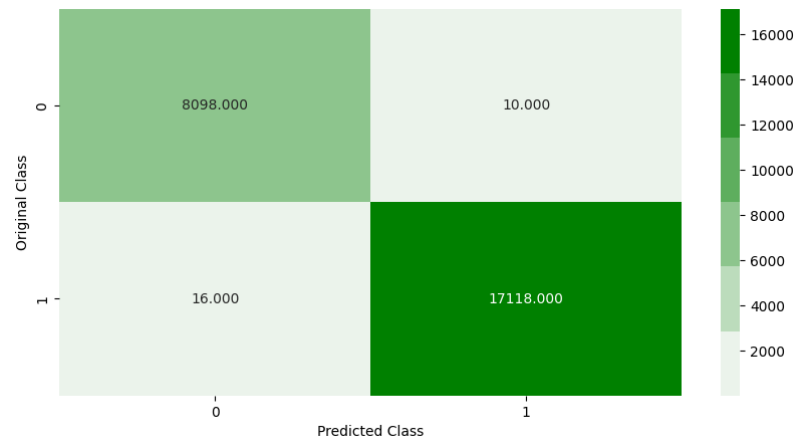


Figure 5.9: Binary Classification–Confusion Matrix (without Autoencoder, KDD Cup’ 99)

With Autoencoder(stacked):

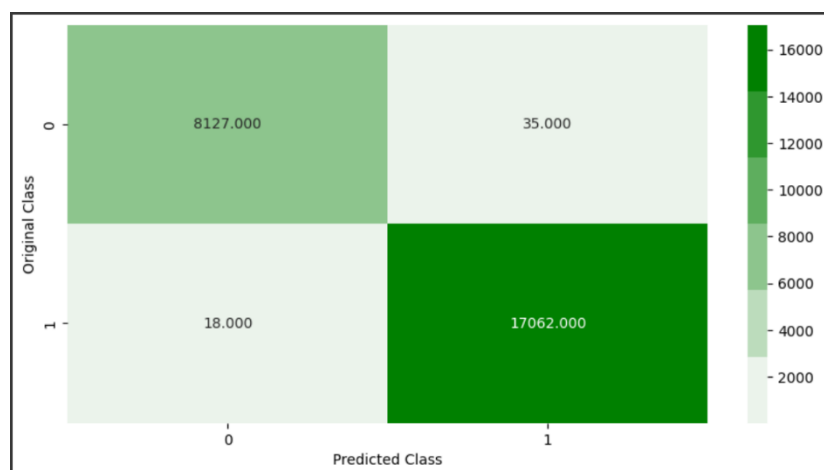


Figure 5.10: Binary Classification–Confusion Matrix (Stacked Autoencoder, KDD Cup’ 99)

NSL KDD –

Without Autoencoder:

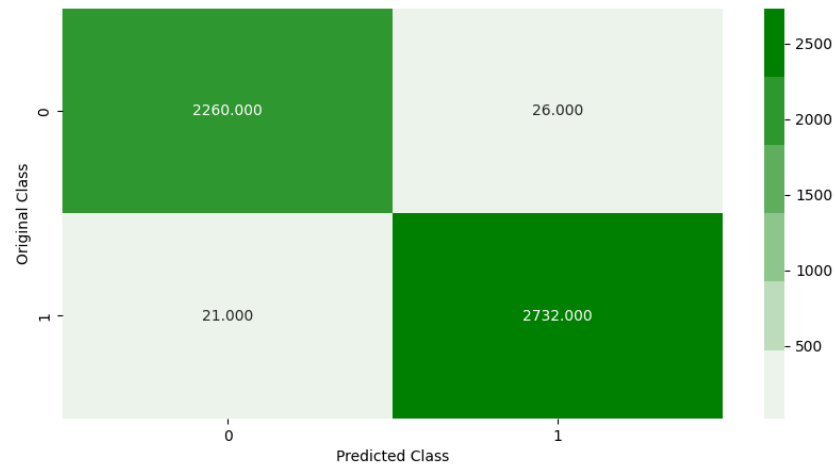


Figure 5.11: Binary Classification–Confusion Matrix (without Autoencoder, NSL KDD)

With Autoencoder(stacked):

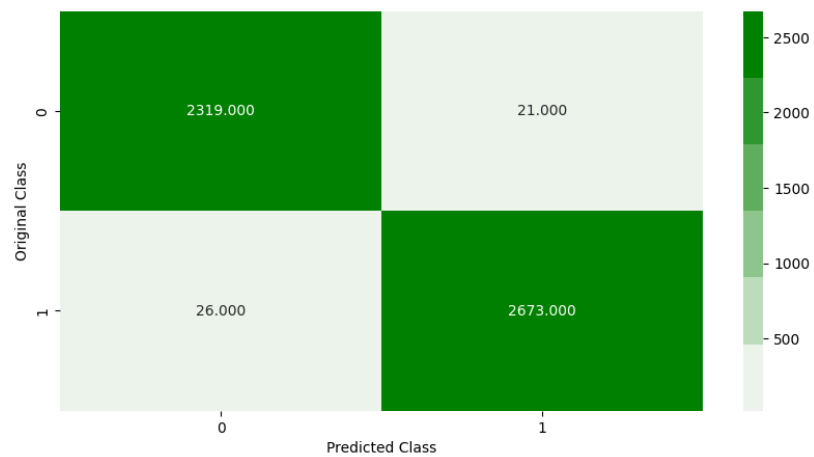


Figure 5.12: Binary Classification–Confusion Matrix (stacked Autoencoder, NSL KDD)

5.2 Results of Multiclass Classification

5.2.1 Performance results

Accuracy Scores:

Without Autoencoder

	dnn1	dnn2	dnn3	dnn4	dnn5
accuracy	0.99821	0.99821	0.99825	0.99833	0.99790

Table 5.8: Multiclass classification Accuracy scores – without Autoencoder

Simple Autoencoder

	dnn1	dnn2	dnn3	dnn4	dnn5
accuracy	0.99746	0.99746	0.99726	0.99726	0.99730

Table 5.9: Multiclass classification Accuracy scores – Simple Autoencoder

Stacked Autoencoder

	dnn1	dnn2	dnn3	dnn4	dnn5
accuracy	0.99714	0.99714	0.99718	0.99754	0.99730

Table 5.10: Multiclass classification Accuracy scores – Stacked Autoencoder

Without Feature Selection : (mean values)

	nn1	nn2	nn3	nn4	nn5
accuracy	99685	99688	99605	99649	99647
validation accuracy	99715	99722	99732	99730	99735
loss	01470	01374	01624	01476	01466
validation loss	06605	06458	03661	04232	03640

Table 5.11: Multiclass classification mean values – Without Autoencoder

With Feature Selection with Simple Autoencoder : (mean values)

	nn1	nn2	nn3	nn4	nn5
accuracy	99528	99584	99206	99443	99350
validation accuracy	99600	99661	99581	99594	99526
loss	01926	01550	02998	02097	02411
validation loss	01838	01654	02052	02203	02275

Table 5.12: Multiclass classification mean values – Simple Autoencoder

5.2.2 Graphs

Without Autoencoder:

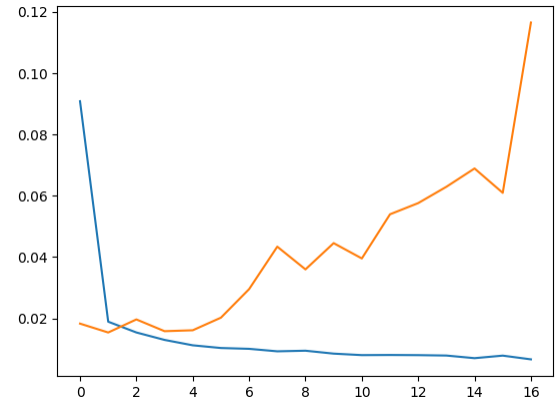
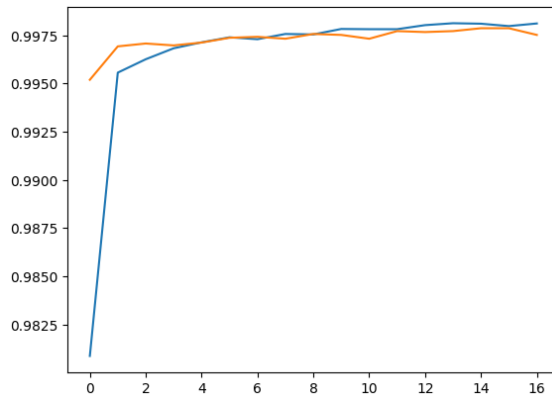


Figure 5.13: Multiclass Classification-Model Accuracy (without Autoencoder)

Figure 5.14: Multiclass Classification -Model Loss (without Autoencoder)

With Simple Autoencoder:

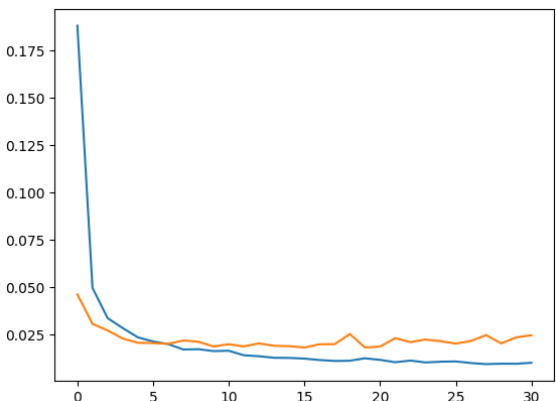
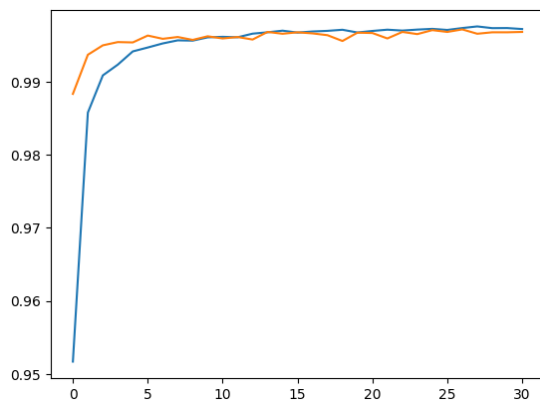


Figure 5.15: Multiclass Classification-Model Accuracy (Simple Autoencoder)

Figure 5.16: Multiclass Classification -Model Loss (Simple Autoencoder)

With Stacked Autoencoder:

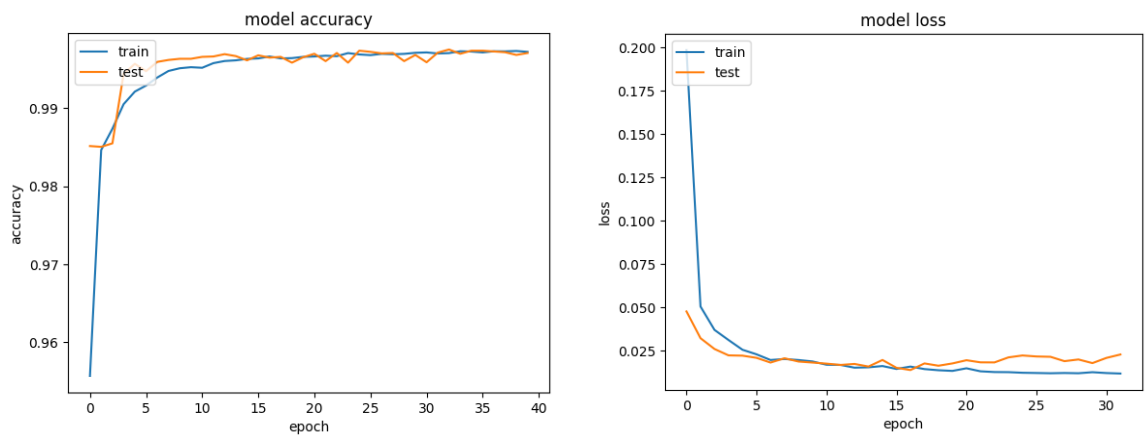


Figure 5.17: Multiclass Classification–Model Accuracy (Stacked Autoencoder)

Figure 5.18: Multiclass Classification –Model Loss (Stacked Autoencoder)

5.2.3 Confusion Matrices:

Without Autoencoder:



Figure 5.19: Multiclass Classification–Confusion Matrix (without Autoencoder)

With Simple Autoencoder:

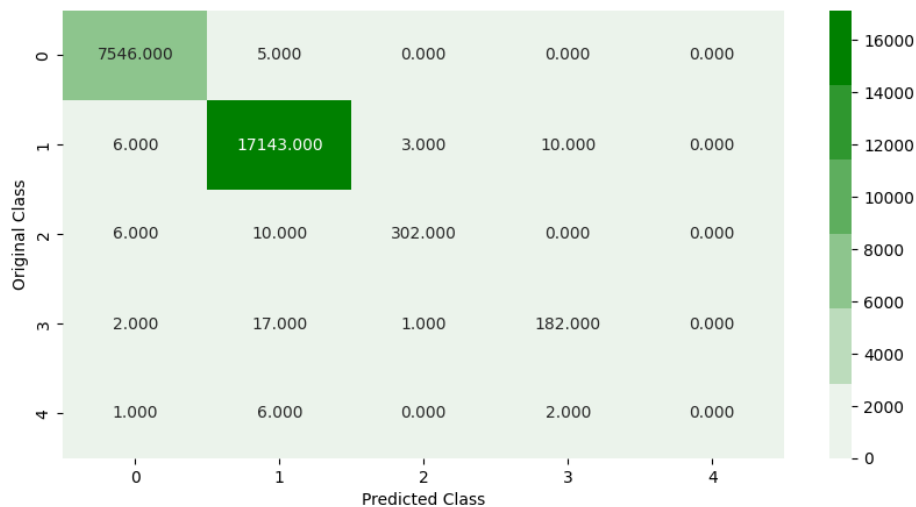


Figure 5.20: Multiclass Classification–Confusion Matrix
(Simple Autoencoder)

With Stacked Autoencoder:

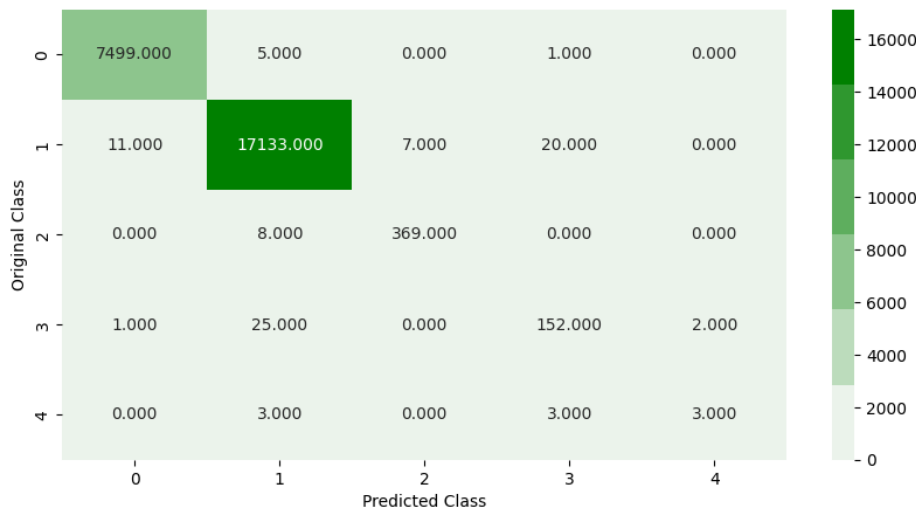


Figure 5.21: Multiclass Classification–Confusion Matrix(Stacked
Autoencoder)

Algorithm And Code

6.1 Algorithm

Step 1: Import Libraries:

- Import necessary libraries for data manipulation, visualization, preprocessing, and model building.

Step 2: Load Dataset:

- Load the dataset using pandas.

Step 3: Data Preprocessing:

- Create a new column 'class' and set default value to 'anomaly'.
- Update 'class' to 'normal' where 'Attack Type' is 'normal'.
- Drop unnecessary columns.
- Encode the target variable using LabelEncoder.
- Convert categorical features to dummy variables using `pd.get_dummies`.
- Split data into training and testing sets using `train_test_split`.

Step 4: Feature Scaling:

- Scale the features using StandardScaler.

Step 5: AutoEncoder:

- Define an AutoEncoder model with an input layer, encoder layer, bottleneck, decoder layer, and output layer.
- Compile and train the AutoEncoder model.
- Extract the encoder part of the AutoEncoder to transform the data into a lower-dimensional space.

Step 6: Evaluation Function:

- Define a function `plot_confusion_matrix` to plot the confusion matrix and calculate misclassification rate.

Step 7: Binary Classification using DNN (1-layer,2-layer,3-layer,4-layer):

- Define and compile a DNN model with different number of hidden layers.
- Train the model and evaluate its performance using accuracy, F1 score, precision, and recall.
- Plot the training history and confusion matrix.

Step 8: Multiclass Classification (1-layer,2-layer,3-layer,4-layer):

- Reload the dataset and preprocess it for multiclass classification.
- Encode the target variable.
- Define and compile a DNN model for multiclass classification with different hidden layers.
- Train the model and evaluate its performance using accuracy.
- Plot the training history and confusion matrix.

6.2 Code

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

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import
accuracy_score, f1_score, precision_score, recall_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix

import keras

from keras.models import Sequential, Model
from keras.utils import plot_model
from keras.layers import Dense, Input
from keras.layers import Dropout
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

df = pd.read_csv('/content/cleaned_kdd.csv')
# Create a new column 'class' and set default value to 'anomaly'
df['class'] = 'anomaly'
# Update 'class' to 'normal' where 'label' is 'normal'
df.loc[df['Attack Type'] == 'normal', 'class'] = 'normal'
# Print a sample of the DataFrame to verify the changes
```

```

df.sample(5)

X = df.drop(
    ['class', 'label', 'Attack Type'], axis=1
) # independent variable
y = df['class'] # dependent variable
le = LabelEncoder()
y = le.fit_transform(y)

X = pd.get_dummies(
X, columns=['protocol_type', 'flag', 'service'], drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size = 0.2)

# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test) #113

# Feature Selection

# INPUT Layer
input_layer = Input(shape=(113,))

# Encoder Layer - 64
encoded_layer = Dense(64, activation='relu')(input_layer)

# BottleNeck
bottleneck = Dense(32, activation='relu')(encoded_layer)

# Decoder Layer - 64
decoded_layer = Dense(64, activation='relu')(bottleneck)

```

```

# OutPut Layer
output_layer = Dense(113, activation='relu')(decoded_layer)

AutoEncoder = Model(
    inputs = input_layer, outputs = output_layer)
AutoEncoder.summary()

AutoEncoder.compile(optimizer='rmsprop', loss='mse')
AutoEncoder_History = AutoEncoder.fit(
    X_train, X_train, batch_size=64, epochs=30,
    validation_data=(X_test, X_test))

encoder = Sequential()
encoder.add(AutoEncoder.layers[0])
encoder.add(AutoEncoder.layers[1])
encoder.add(AutoEncoder.layers[2])
encoder.summary()

X_train = encoder.predict(X_train)
X_test = encoder.predict(X_test)
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)

#For confusion matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ", (len(test_y)-
np.trace(C))/len(test_y)*100)

```

```

A = (((C.T)/(C.sum(axis=1))).T)
B = (C/C.sum(axis=0))
labels = [0,1]
cmap=sns.light_palette("green")
#Confusion matrix
print("-"*50, "Confusion matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(C,      annot=True,      cmap=cmap,      fmt=".3f",
xticklabels=labels, yticklabels=labels)

plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()


epochs = 40
batch_size = 128


"""Binary Classification"""


"""## **1 Layer DNN**"""


model = Sequential()
model.add(Dense(X_train.shape[1],activation = 'relu',input_dim
= X_train.shape[1]))
model.add(Dense(1,activation = 'sigmoid'))
model.summary()
model.compile(optimizer='Adam',loss
=
'binary_crossentropy',metrics=['accuracy'])

```

```

callback = EarlyStopping(
    monitor="val_loss",
    min_delta=0.00001,patience=20,
    verbose=1,mode="auto",baseline=None,
    restore_best_weights=False)

history = model.fit(X_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_split=0.2,callbacks=callback)

y_log=model.predict(X_test)
y_pred = np.where(y_log>0.5,1,0)
y_pred = y_pred.flatten()

print('accuracy score : ',accuracy_score(y_test,y_pred))
print('f1 score : ',f1_score(y_test,y_pred))
print('precision score : ',precision_score(y_test,y_pred))
print('recall score : ',recall_score(y_test,y_pred))

print("accuracy: ",np.mean(history.history['accuracy']))
print("val_accuracy: ",np.mean(history.history['val_accuracy']))
print("loss: ",np.mean(history.history['loss']))
print("val_loss: ",np.mean(history.history['val_loss']))

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.plot(history.history['accuracy'])

```

```

plt.plot(history.history['val_accuracy'])

plot_confusion_matrix(y_test, y_pred)

"""## **2 Layer DNN**"""

model_l2 = Sequential()
model_l2.add(Dense(X_train.shape[1],activation =
'relu',input_dim = X_train.shape[1]))
model_l2.add(Dense(X_train.shape[1],activation = 'relu',))
model_l2.add(Dropout(0.2))
model_l2.add(Dense(1,activation = 'sigmoid'))
model_l2.summary()

model_l2.compile(optimizer='Adam',loss =
'binary_crossentropy',metrics=['accuracy'])

history_l2 = model_l2.fit(X_train,y_train,batch_size =
batch_size,epochs =
epochs,verbose=1,validation_split=0.2,callbacks=callback)

y_log_l2=model.predict(X_test)
y_pred_l2 = np.where(y_log_l2>0.5,1,0)
y_pred_l2 = y_pred_l2.flatten()

print('accuracy score : ',accuracy_score(y_test,y_pred_l2))
print('f1 score : ',f1_score(y_test,y_pred_l2))
print('precision score : ',precision_score(y_test,y_pred_l2))
print('recall score : ',recall_score(y_test,y_pred_l2))

```

```

print("accuracy: ",np.mean(history_l2.history['accuracy']))
print("val_accuracy:
",np.mean(history_l2.history['val_accuracy']))
print("loss: ",np.mean(history_l2.history['loss']))
print("val_loss: ",np.mean(history_l2.history['val_loss']))

plt.plot(history_l2.history['loss'])
plt.plot(history_l2.history['val_loss'])

plt.plot(history_l2.history['accuracy'])
plt.plot(history_l2.history['val_accuracy'])

plot_confusion_matrix(y_test, y_pred_l2)

"""## **3 Layer DNN**"""

model_l3 = Sequential()
model_l3.add(Dense(X_train.shape[1],activation =
'relu',input_dim = X_train.shape[1]))
model_l3.add(Dropout(0.2))
model_l3.add(Dense(X_train.shape[1],activation = 'relu'))
model_l3.add(Dropout(0.2))
model_l3.add(Dense(X_train.shape[1],activation = 'relu'))
model_l3.add(Dropout(0.2))

model_l3.add(Dense(1,activation = 'sigmoid'))
model_l3.summary()

```

```

model_l3.compile(optimizer='Adam',loss                                     =
'binary_crossentropy',metrics=['accuracy'])

history_l3      =      model_l3.fit(X_train,y_train,batch_size      =
batch_size,epochs                                          =
epochs,verbose=1,validation_split=0.2,callbacks=callback)

y_log_l3=model_l3.predict(X_test)
y_pred_l3 = np.where(y_log_l3>0.5,1,0)
y_pred_l3 = y_pred_l3.flatten()

print('accuracy score : ',accuracy_score(y_test,y_pred_l3))
print('f1 score : ',f1_score(y_test,y_pred_l3))
print('precision score : ',precision_score(y_test,y_pred_l3))
print('recall score : ',recall_score(y_test,y_pred_l3))

print("accuracy: ",np.mean(history_l3.history['accuracy']))
print("val_accuracy:
",np.mean(history_l3.history['val_accuracy']))
print("loss: ",np.mean(history_l3.history['loss']))
print("val_loss: ",np.mean(history_l3.history['val_loss']))

plt.plot(history_l3.history['loss'])
plt.plot(history_l3.history['val_loss'])
plt.plot(history_l3.history['accuracy'])
plt.plot(history_l3.history['val_accuracy'])

plot_confusion_matrix(y_test, y_pred_l3)

```

```

"""4 layer DNN"""

model_l4 = Sequential()
model_l4.add(Dense(X_train.shape[1],activation = 'relu',input_dim = X_train.shape[1]))
model_l4.add(Dense(X_train.shape[1],activation = 'relu'))
model_l4.add(Dropout(0.2))
model_l4.add(Dense(X_train.shape[1],activation = 'relu'))
model_l4.add(Dropout(0.2))
model_l4.add(Dense(X_train.shape[1],activation = 'relu'))
model_l4.add(Dropout(0.2))
model_l4.add(Dense(1,activation = 'sigmoid'))
model_l4.summary()

model_l4.compile(optimizer='Adam',loss = 'binary_crossentropy',metrics=['accuracy'])

history_l4 = model_l4.fit(X_train,y_train,batch_size = batch_size,epochs = epochs,verbose=1,validation_split=0.2,callbacks=callback)

y_log_l4=model_l4.predict(X_test)
y_pred_l4 = np.where(y_log_l4>0.5,1,0)
y_pred_l4 = y_pred_l4.flatten()

print('accuracy score : ',accuracy_score(y_test,y_pred_l4))
print('f1 score : ',f1_score(y_test,y_pred_l4))
print('precision score : ',precision_score(y_test,y_pred_l4))
print('recall score : ',recall_score(y_test,y_pred_l4))

```

```

print("accuracy: ",np.mean(history_l4.history['accuracy']))
print("val_accuracy:
",np.mean(history_l4.history['val_accuracy']))
print("loss: ",np.mean(history_l4.history['loss']))
print("val_loss: ",np.mean(history_l4.history['val_loss']))

# summarize history for accuracy
plt.plot(history_l4.history['accuracy'])
plt.plot(history_l4.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history_l4.history['loss'])
plt.plot(history_l4.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plot_confusion_matrix(y_test, y_pred_l4)

"""Multiclass Classiication"""

df = pd.read_csv('/content/cleaned_kdd.csv')

```

```

y = df['Attack Type']          # dependent variable
le = LabelEncoder()
y = le.fit_transform(y)

"""## **1 Layer DNN**"""
model = Sequential()
model.add(Dense(X_train.shape[1],activation = 'relu',input_dim
= X_train.shape[1]))
model.add(Dense(5,activation = 'softmax'))
model.summary()
model.compile(optimizer='Adam',loss =
'sparse_categorical_crossentropy',metrics=['accuracy'])
callback = EarlyStopping(
    monitor="val_loss",min_delta=0.00001,
    patience=15,verbose=1,
    mode="auto", baseline=None,
    restore_best_weights=False)
history = model.fit(X_train,y_train,batch_size =
batch_size,epochs =
epochs,verbose=1,validation_split=0.2,callbacks=callback)
y_prob=model.predict(X_test)
y_pred = y_prob.argmax(axis = 1)
accuracy_score(y_test,y_pred)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

```

```

print("accuracy: ",np.mean(history.history['accuracy']))
print("val_accuracy:
",np.mean(history.history['val_accuracy']))
print("loss: ",np.mean(history.history['loss']))
print("val_loss: ",np.mean(history.history['val_loss']))
plot_confusion_matrix(y_test, y_pred)

"""## **2 Layer DNN**"""

model_l2 = Sequential()
model_l2.add(Dense(X_train.shape[1],activation =
'relu',input_dim = X_train.shape[1]))
model_l2.add(Dense(X_train.shape[1],activation = 'relu'))
model_l2.add(Dropout(0.2))
model_l2.add(Dense(5,activation = 'softmax'))
model_l2.summary()
model_l2.compile(optimizer='Adam',loss =
'sparse_categorical_crossentropy',metrics=['accuracy'])
history_l2 = model_l2.fit(X_train,y_train,batch_size =
batch_size,epochs =
epochs,verbose=1,validation_split=0.2,callbacks=callback)

y_log_l2=model.predict(X_test)
y_pred_l2 = y_log_l2.argmax(axis=1)
print('accuracy score : ',accuracy_score(y_test,y_pred_l2))

plt.plot(history_l2.history['loss'])
plt.plot(history_l2.history['val_loss'])

```

```

plt.plot(history_l2.history['accuracy'])
plt.plot(history_l2.history['val_accuracy'])
print("accuracy: ",np.mean(history_l2.history['accuracy']))
print("val_accuracy:
",np.mean(history_l2.history['val_accuracy']))
print("loss: ",np.mean(history_l2.history['loss']))
print("val_loss: ",np.mean(history_l2.history['val_loss']))
plot_confusion_matrix(y_test, y_pred_l2)

"""## **3 Layer DNN**"""
model_l3 = Sequential()
model_l3.add(Dense(X_train.shape[1],activation =
'relu',input_dim = X_train.shape[1]))
model_l3.add(Dropout(0.2))
model_l3.add(Dense(X_train.shape[1],activation = 'relu'))
model_l3.add(Dropout(0.2))
model_l3.add(Dense(X_train.shape[1],activation = 'relu'))
model_l3.add(Dropout(0.2))
model_l3.add(Dense(5,activation = 'softmax'))
model_l3.summary()
model_l3.compile(optimizer='Adam',loss =
'sparse_categorical_crossentropy',metrics=['accuracy'])
history_l3 = model_l3.fit(X_train,y_train,batch_size =
batch_size,epochs =
epochs,verbose=1,validation_split=0.2,callbacks=callback)
y_log_l3=model_l3.predict(X_test)
y_pred_l3 = y_log_l3.argmax(axis=1)
print('accuracy score : ',accuracy_score(y_test,y_pred_l3))

```

```

plt.plot(history_l3.history['loss'])
plt.plot(history_l3.history['val_loss'])
plt.plot(history_l3.history['accuracy'])
plt.plot(history_l3.history['val_accuracy'])
print("accuracy: ",np.mean(history_l3.history['accuracy']))
print("val_accuracy:
",np.mean(history_l3.history['val_accuracy']))
print("loss: ",np.mean(history_l3.history['loss']))
print("val_loss: ",np.mean(history_l3.history['val_loss']))
plot_confusion_matrix(y_test, y_pred_l3)

"""***4 Layer DNN***"""
model_l4 = Sequential()
model_l4.add(Dense(X_train.shape[1],activation =
'relu',input_dim = X_train.shape[1]))
model_l4.add(Dense(X_train.shape[1],activation = 'relu'))
model_l4.add(Dropout(0.2))
model_l4.add(Dense(X_train.shape[1],activation = 'relu'))
model_l4.add(Dropout(0.2))
model_l4.add(Dense(X_train.shape[1],activation = 'relu'))
model_l4.add(Dropout(0.2))
model_l4.add(Dense(5,activation = 'softmax'))
model_l4.summary()

model_l4.compile(
optimizer='Adam',loss = 'sparse_categorical_crossentropy',
metrics=['accuracy'])

```

```
history_l4 = model_l4.fit(X_train,y_train,
batch_size = batch_size,epochs = epochs,
verbose=1,validation_split=0.2,callbacks=callback)
y_log_l4=model_l4.predict(X_test)
y_pred_l4 = y_log_l4.argmax(axis=1)
print('accuracy score : ',accuracy_score(y_test,y_pred_l4))
plt.plot(history_l4.history['loss'])
plt.plot(history_l4.history['val_loss'])
plt.plot(history_l4.history['accuracy'])
plt.plot(history_l4.history['val_accuracy'])
plot_confusion_matrix(y_test, y_pred_l4)
```

Conclusion And Future Work

7.1 Conclusion

In this study, a comprehensive survey was conducted on Intrusion Detection Systems (IDS), encompassing both binary and multiclass classification tasks. The aim was to evaluate the performance of neural network models with and without feature selection using Autoencoders across various architectures.

For each classification task, three types of neural network models were tested: without Autoencoder, with a simple Autoencoder, and with a stacked Autoencoder for feature selection. Additionally, models with different numbers of hidden layers ranging from 1 to 5 were evaluated to determine the optimal architecture.

From the results analysis, it was observed that the neural network model with 4 hidden layers consistently outperformed other architectures for binary classification tasks. Furthermore, the stacked Autoencoder exhibited superior performance in feature selection compared to the simple Autoencoder and no feature selection model. While the accuracy of models without Autoencoder and stacked Autoencoder was comparable (99.807%), the difference in loss between them was significantly lower in the stacked Autoencoder model, indicating better generalization.

Similar trends were observed in the NSL KDD dataset, validating the effectiveness of the selected models across different datasets. However, for multiclass classification tasks, the accuracy slightly decreased when using

Autoencoders for feature selection. Nonetheless, the reduction in overfitting was evident, highlighting the importance of feature selection in improving model generalization.

Overall, this study underscores the significance of feature selection using Autoencoders in enhancing the performance and robustness of neural network models for IDS. The findings provide valuable insights into selecting optimal architectures and feature selection techniques, contributing to the advancement of intrusion detection methodologies in cybersecurity applications.

7.2 Future Work

Moving forward, there are several exciting avenues for further exploration and refinement:

Hyperparameter Optimization:

Delve deeper into hyperparameter tuning to fine-tune model parameters such as learning rates, batch sizes, and dropout rates. This optimization process can significantly enhance model performance and convergence.

Exploration of Diverse Autoencoder Types:

Explore a diverse range of Autoencoder architectures, including Variational Autoencoders (VAEs) and Convolutional Autoencoders. Investigating these alternatives can provide valuable insights into their efficacy for capturing intricate data patterns in intrusion detection scenarios.

Evaluation of Different Neural Network Architectures:

Assess the performance of various neural network architectures beyond traditional feedforward networks. This exploration could include Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), which might offer distinct advantages for IDS tasks.

Deeper Analysis of Multiclass Classification:

Conduct in-depth analyses to better understand the behavior of models in multiclass classification scenarios. Further investigations can focus on strategies to mitigate the observed decrease in accuracy when employing Autoencoders for feature selection in multiclass settings.

By pursuing these avenues, we can refine our approaches, uncover new insights, and ultimately advance the effectiveness of intrusion detection systems for enhanced cybersecurity.

7.3 References

- [1] Agrawal, Gaurav, Soni, Shivank Kumar And Agrawal, Chetan, 2017. A SURVEY ON ATTACKS AND APPROACHES OF INTRUSION DETECTION SYSTEMS. International Journal of Advanced Research in Computer Science. 30 August 2017. Vol. 8, no. 8, p. 499-504. DOI 10.26483/ijarcs.v8i8.4771
- [2] Ahmed, Mohiuddin, Naser Mahmood, Abdun And Hu, Jiankun, 2016. A survey of network anomaly detection techniques. 1 January 2016. Academic Press.
- [3] Amor, Nahla Ben, Benferhat, Salem And Elouedi, Zied, 2003. Naive Bayesian Networks in Intrusion Detection Systems. In: The Fourteenth European Conference on Machine Learning. 2003.
- [4] Chollet, François, 2017. Deep Learning with Python. Manning Publications.
- [5] Choudhary, Amit Kumar And Swarup, Akhilesh, 2009. Neural network approach for intrusion detection. In: ACM International Conference Proceeding Series. 2009. p. 1297-1301. ISBN 9781605587103.
- [6] Chowdhury, Md. Nasimuzzaman, Ferens, Ken And Ferens, Mike, 2016. Network Intrusion Detection Using Machine Learning. In: International Conference on Security and Management. 2016.
- [7] Dias, L. P., Cerqueira, J. J.F., Assis, K. D.R. And Almeida, R. C., 2017. Using artificial neural network in intrusion detection systems to computer networks. In: 2017 9th Computer Science and Electronic Engineering Conference, CEEC 2017 - Proceedings. Institute of Electrical and Electronics Engineers Inc. 8 November 2017. p. 145-150. ISBN 9781538630075.
- [8] García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G. And Vázquez, E., 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers and Security. February 2009. Vol. 28, no. 1-2, p. 18-28. DOI 10.1016/j.cose.2008.08.003.

- [9] Jha, J. And Ragha, L, 2013. Intrusion detection system using support vector machine. In: International Conference and workshop on Advanced Computing. 2013. p. 25-30.
- [10] Jose, Shijoe, Malathi, D., Reddy, Bharath And Jayaseeli, Dorathi, 2018. A Survey on Anomaly Based Host Intrusion Detection System. In: Journal of Physics: Conference Series. Institute of Physics Publishing. 26 April 2018.
- [11] Kadam, Priya U And Deshmukh, Profmanjusha, 2007. Various Approaches for Intrusion Detection System: An Overview. International Journal of Innovative Research in Computer and Communication Engineering(An ISO [online]. 2007. Vol. 3297, no. 11. [Accessed 2 December 2019]. Available from: Available from: <http://www.ijircce.com>
- [12] Kaur, Parneet, Kumar, Manish And Bhandari, Abhinav, 2017. A review of detection approaches for distributed denial of service attacks. Systems Science & Control Engineering [online]. 20 January 2017. Vol. 5, no. 1, p. 301-320. [Accessed 2 December 2019]. DOI 10.1080/21642583.2017.1331768. Available from: Available from: <https://www.tandfonline.com/doi/full/10.1080/21642583.2017.1331768>
- [13] Khandagale, Vaishali V And Kalshetty, Yoginath, 2013. Review and Discussion on different techniques of Anomaly Detection Based and Recent Work. International Journal of Engineering Research & Technology (IJERT) [online]. 2013. Vol. 2, no. 10, p. 5. [Accessed 2 December 2019]. Available from: Available from: <http://www.ijert.org>
- [14] Kumar, Gulshan, 2014. Evaluation Metrics for Intrusion Detection Systems - A Study. 2014.
- [15] Lin, Wen Hui, Lin, Hsiao Chung, Wang, Ping, Wu, Bao Hua And Tsai, Jeng Ying, 2018. Using convolutional neural networks to network intrusion detection for cyber threats. In: Proceedings of 4th IEEE International Conference on Applied System Innovation 2018, ICASI 2018. Institute of

- Electrical and Electronics Engineers Inc. 22 June 2018. p. 1107-1110. ISBN 9781538643426.
- [16] Liu, Yuchen, Liu, Shengli And Zhao, Xing, 2017. Intrusion Detection Algorithm Based on Convolutional Neural Network. In: 4th International Conference on Engineering Technology and Application (ICETA 2017). DEStech Publications. 23 March 2017.
- [17] Marcelino, Maria José, Pessoa, Teresa, Vieira, Celeste, Salvador, Tatiana And Mendes, António José, 2018. Learning Computational Thinking and scratch at distance. Computers in Human Behavior. 2018. Vol. 80, p. 470-477. DOI 10.1016/j.chb.2017.09.025.
- [18] Mohammadpour, L, Ling, T.C., Liew, C. S. And Chong, C. Y., 2018. A Convolutional Neural Network for Network Intrusion Detection System. In: Proceedings of the Asia-Pacific Advanced Network [online]. 2018. p. 6. [Accessed 3 December 2019]. Available from: <http://journals.sfu.ca/apan/index.php/apan/article/view/239>
- [19] Okoli, Chitu And Schabram, Kira, 2012. A Guide to Conducting a Systematic Literature Review of Information Systems Research. SSRN Electronic Journal. 5 January 2012. DOI 10.2139/ssrn.1954824.
- [20] Patel, Nirav J. And Jhaveri, Rutvij H., 2015. Detecting packet dropping nodes using machine learning techniques in Mobile ad-hoc network: A survey. In: International Conference on Signal Processing and Communication Engineering Systems - Proceedings of SPACES 2015, in Association with IEEE. Institute of Electrical and Electronics Engineers Inc. 10 March 2015. p. 468-472. ISBN 9781479961085.
- [21] Ponkarthika, M, Dr And Saraswathy, V R, 2018. Network Intrusion Detection Using Deep Neural Networks. Asian Journal of Applied Science and Technology (AJAST) (Open Access Quarterly International Journal [online]. 2018. Vol. 2,

- no. 2, p. 665-673. [Accessed 2 December 2019]. Available from: Available from: <http://www.ajast.net>
- [22] Rai, Kajal, Devi, S. And Guleria, Ajay, 2016. Decision Tree Based Algorithm for Intrusion Detection. International Journal Advanced Network and Applications. 2016. Vol. 7, no. 4, p. 2828-2834.
- [23] Richariya, Vineet, Singh, Uday Pratap And Mishra, Renu, 2012. Distributed Approach of Intrusion Detection System: Survey.
- [24] Sarmah, Abhijit, 2019. Intrusion Detection Systems: Definition, Need and Challenges [online]. [Accessed 2 December 2019]. Available from: Available from: <https://www.sans.org/reading-room/whitepapers/detection/intrusion-detection-systems-definition-challenges-343>
- [25] Schell, Bernadette H. And Martin, Clemens., 2006. Webster's new world hacker dictionary. Wiley Pub. ISBN 0470047526.
- [26] Shah, Asghar Ali, Hayat, Malik Sikander And Awan, Muhammad Daud, 2015. Analysis of Machine Learning Techniques for Intrusion Detection System: A Review. International Journal of Computer Applications. 2015. Vol. 119, no. 3, p. 19-29.
- [27] Sharma, Vikas And Nema, Aditi, 2013. Innovative genetic approach for intrusion detection by using decision tree. In: Proceedings - 2013 International Conference on Communication Systems and Network Technologies, CSNT 2013. 2013. p. 418-422.
- [28] Singh, J And Nene, Mj, 2013. A survey on machine learning techniques for intrusion detection systems. International Journal of Advanced Research in Computer and Communication Engineering [online]. 2013. Vol. 2, no. 11, p. 4349-55. [Accessed 2 December 2019]. Available from: Available from: <http://www.sciepub.com/reference/212012>
- [29] Vani, R, 2007. Towards Efficient Intrusion Detection using Deep Learning Techniques: A Review. International Journal of Advanced Research in

- Computer and Communication Engineering ISO. 2007. Vol. 3297. DOI 10.17148/IJARCCE.2017.61066.
- [30] Vinayakumar, R., Soman, K. P. And Poornachandran, Prabakaran, 2017. Evaluation of recurrent neural network and its variants for intrusion detection system (IDS). International Journal of Information System Modeling and Design. 1 July 2017. Vol. 8, no. 3, p. 43-63. DOI 10.4018/IJISMD.2017070103.
- [31] Wang, Juan, Yang, Qiren And Ren, Dasen, 2009. An intrusion detection algorithm based on decision tree technology. In: Proceedings - 2009 Asia-Pacific Conference on Information Processing, APCIP 2009. 2009. p. 333-335. ISBN 9780769536996.