# Autoencoders for Anomaly Detection in Network Intrusion Detection Systems

*A thesis is submitted to the department of*

***Computer Science & Engineering***

*of*

***International Institute of Information Technology Bhubaneswar***

*in partial fulfilment of the requirements*

*for the degree of*

***Bachelor of Technology***

*by*

***Penthaa Patel***

*(Roll - B116033)*

*under the supervision of*

***Prof. Jay Bhuyan & Prof. Tushar Ranjan Sahoo***


**Department of Computer Science and Engineering**

**International Institute of Information Technology Bhubaneswar**

**Bhubaneswar Odisha - 751003, India**

**2020**


**International Institute of Information Technology Bhubaneswar**

Bhubaneswar Odisha -751 003, India. www.iiit-bh.ac.in

May 24, 2020

# Undertaking

I declare that the work presented in this thesis titled "Autoencoders for Anomaly Detection in Network Intrusion Detection Systems", submitted to the Department of Computer Science and Engineering, International Institute of Information Technology, Bhubaneswar, for the award of the Bachelors of Technology degree in Computer Science and Engineering, is my original work. I have not plagiarised or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, I accept that my degree may be unconditionally withdrawn.

***Penthaa Patel***

***B116033***

**International Institute of Information Technology Bhubaneswar**

Bhubaneswar Odisha -751 003, India. www.iiit-bh.ac.in

May 24, 2020

# Certificate

This is to certify that the work in the thesis entitled *"Autoencoders for Anomaly Detection in Network Intrusion Detection Systems"* by *Penthaa Patel* is a record of an original research work carried out by her under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Bachelor of Technology* in *Computer Science & Engineering*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Jay Bhuyan

Digitally signed by Jay Bhuyan
Date: 2020.05.27 10:27:44 -05'00'

***Jay Bhuyan***

***Professor, Dept. of Computer Science***

***Tuskegee University***


**International Institute of Information Technology Bhubaneswar**
Bhubaneswar Odisha -751 003, India. www.iiit-bh.ac.in

May 24, 2020

# Certificate

This is to certify that the work in the thesis entitled *"Autoencoders for Anomaly Detection in Network Intrusion Detection Systems"* by *Penthaa Patel* is a record of an original research work carried out by her under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Bachelor of Technology* in *Computer Science & Engineering*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

***Tushar Ranjan Sahoo***
***Professor, Dept. of Computer Science***
***IIIT, Bhubaneswar***

# Acknowledgment

I would like to express my deep and sincere gratitude to my research guide, Professor Jay Bhuyan, Dept. of Computer Science, Tuskegee University, for giving me the opportunity to pursue this research work and providing invaluable guidance throughout this research.

I would like to express my deepest appreciation to my research co-guide, Professor Tushar Ranjan Sahoo, Dept. of Computer Science, IIIT-Bhubaneswar, for his sincere contributions and many fruitful discussions in the course of preparing this report.

Penthaa Patel
B116033, CSE

# Abstract

Emerging technologies are now critical infrastructure that are prone to attacks by unauthorized users. Successful exploitation of vulnerabilities in modern systems can lead an attacker to access critical information stored in them, hence the need arises to flag the attempts to compromise the system. Interconnected networks have attracted the attention of attackers. They may compromise systems remotely by sending in anomalous internet traffic. This led to the development of Network Intrusion Detection Systems (NIDS). A NIDS helps security analysts detect network security breaches. NIDS flag attempts made to exploit the systems. This can save critical data from potential misuse by attackers. In this report we implement Autoencoders, which are artificial neural network systems, to detect network-based attacks as anomalies. We compare the performance of nine different models of autoencoders, each with a different set of optimizers and activation functions. Compared performance metrics include accuracy, precision, recall, and F1 score values.


***Keywords*:** Network Intrusion Detection Systems, Artificial Neural Networks, Autoencoders, Cyber Security.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Network Intrusion Detection System (NIDS)

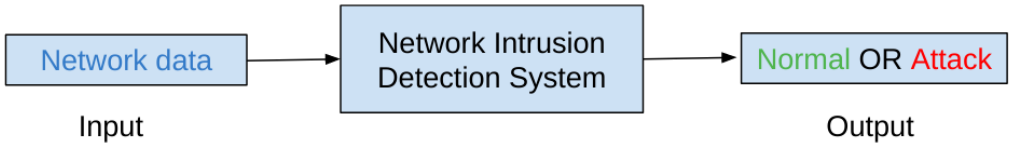


Figure 1.1: Block diagram for a NIDS.

NIDS is a network based defensive monitoring tool used to identify potential malicious activity on computer networks. It monitors network traffic and server applications. It can flag an event as an attack if certain conditions are met. Traditional NIDS used signature based methods to distinguish normal network behaviour from possible network intrusions. These had a high false alarm rate and a low accuracy rate. Such NIDS were incapable in flagging new patterns of attacks since they relied on a set of

static rules. Attackers can evade such static rules. Modern NIDS use methods like behavior modelling, data mining, and machine learning algorithms that are dynamic in nature. Such NIDS can evolve with the evolving patterns of network attacks thus protecting the computer systems from new forms of attacks. NIDS are different from intrusion prevention systems. NIDS only flag potential malicious activity, they do not take further action to mitigate a data breach.

# Chapter 2

# Problem statement

## 2.1 Anomaly detection

Our aim is to build a robust predictive model using autoencoders that can distinguish between normal and malicious network behaviour and flag potential attempts to attack the system. Any deviation from normal behaviour will be flagged as an anomaly. Our aim is to build a model with high accuracy and low false alarm rate.

# Chapter 3

# Dataset

The dataset used to build the predictive model is the NSL-KDD dataset [9]. The NSL KDD dataset consists of internet traffic records. It is a benchmark dataset which has been widely used by researchers for developing predictive models for NIDS. The NSL-KDD is a subset of KDD'99 dataset. The KDD'99 dataset was prepared using the network traffic captured by 1998 DARPA IDS evaluation program. This newer version of the dataset is free from duplicate and redundant records. KDDTrain+ and KDDTest+ contain the full NSL-KDD train and test set respectively.

## 3.1 Data Exploration

There are 125973 instances and 43 features for each instance in KDDTrain+ dataset. It contains 0 duplicate instances.

| # | Feature Name | Description | Data Type | Type |
|---|---|---|---|---|
| 1 | duration | Length of time duration of the connection | int64 | Continuous |
| 2 | protocol_type | Protocol used in the connection | object | Categorical |
| 3 | service | Destination network service used | object | Categorical |
| 4 | flag | Status of the connection – Normal or Error | object | Categorical |
| 5 | src_bytes | Number of data bytes transferred from source to destination in single connection | int64 | Continuous |
| 6 | dst_bytes | Number of data bytes transferred from destination to source in single connection | int64 | Continuous |
| 7 | land | If source and destination IP addresses and port numbers are equal then, this variable takes value 1 else 0 | int64 | Binary |
| 8 | wrong_fragment | Total number of wrong fragments in this connection | int64 | Discrete |
| 9 | urgent | Number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated | int64 | Discrete |
| 10 | hot | Number of “hot” indicators in the content such as: entering a system directory, creating programs and executing programs | int64 | Continuous |
| 11 | num_failed_logins | Count of failed login attempts | int64 | Continuous |
| 12 | logged_in | Login Status : 1 if successfully logged in; 0 otherwise | int64 | Binary |
| 13 | num_compromised | Number of "compromised" conditions | int64 | Continuous |
| 14 | root_shell | 1 if root shell is obtained; 0 otherwise | int64 | Binary |
| 15 | su_attempted | 1 if "su root" command attempted or used; 0 otherwise | int64 | Discrete |
| 16 | num_root | Number of "root" accesses or number of operations performed as a root in the connection | int64 | Continuous |
| 17 | num_file_creations | Number of file creation operations in the connection | int64 | Continuous |
| 18 | num_shells | Number of shell prompts | int64 | Continuous |
| 19 | num_access_files | Number of operations on access control files | int64 | Continuous |
| 20 | num_outbound_cmds | Number of outbound commands in an ftp session | int64 | Continuous |
| 21 | is_host_login | 1 if the login belongs to the "hot" list i.e., root or admin; else 0 | int64 | Binary |
| 22 | is_guest_login | 1 if the login is a "guest" login; 0 otherwise | int64 | Binary |
| 23 | count | Number of connections to the same destination host as the current connection in the past two seconds | int64 | Discrete |
| 24 | srv_count | Number of connections to the same service (port number) as the current connection in the past two seconds | int64 | Discrete |
| 25 | serror_rate | The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23) | float64 | Discrete |
| 26 | srv_serror_rate | The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in srv_count (24) | float64 | Discrete |
| 27 | rerror_rate | The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23) | float64 | Discrete |
| 28 | srv_rerror_rate | The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24) | float64 | Discrete |
| 29 | same_srv_rate | The percentage of connections that were to the same service, among the connections aggregated in count (23) | float64 | Discrete |
| 30 | diff_srv_rate | The percentage of connections that were to different services, among the connections aggregated in count (23) | float64 | Discrete |
| 31 | srv_diff_host_rate | The percentage of connections that were to different destination machines among the connections aggregated in srv_count (24) | float64 | Discrete |
| 32 | dst_host_count | Number of connections having the same destination host IP address | int64 | Discrete |
| 33 | dst_host_srv_count | Number of connections having the same port number | int64 | Discrete |
| 34 | dst_host_same_srv_rate | The percentage of connections that were to different services, among the connections aggregated in dst_host_count (32) | float64 | Discrete |
| 35 | dst_host_diff_srv_rate | The percentage of connections that were to different services, among the connections aggregated in dst_host_count (32) | float64 | Discrete |
| 36 | dst_host_same_src_port_rate | The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (33) | float64 | Discrete |
| 37 | dst_host_srv_diff_host_rate | The percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (33) | float64 | Discrete |
| 38 | dst_host_serror_rate | The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (32) | float64 | Discrete |
| 39 | dst_host_srv_serror_rate | The percent of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (33) | float64 | Discrete |
| 40 | dst_host_rerror_rate | The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32) | float64 | Discrete |
| 41 | dst_host_srv_rerror_rate | The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_count (33) | float64 | Discrete |
| 42 | labels | Classification of the traffic input | object | Categorical |
| 43 | score | Difficulty level | int64 | Discrete |

Figure 3.1: List of features in NSL-KDD dataset.

## 3.2 Data Visualization

### 3.2.1 Categorical Features

Feature ‘protocol_type’ has 3 categories.

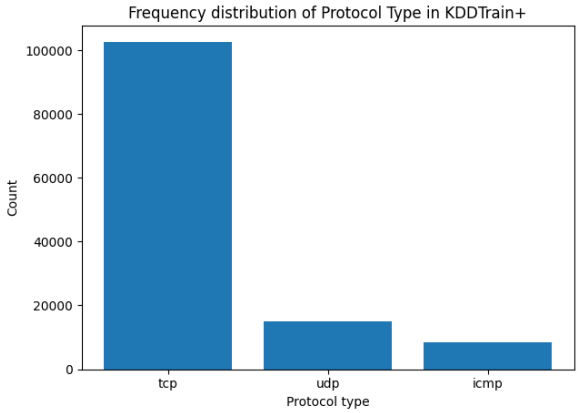


Figure 3.2: Protocol Type

Feature ‘service’ has 70 categories.

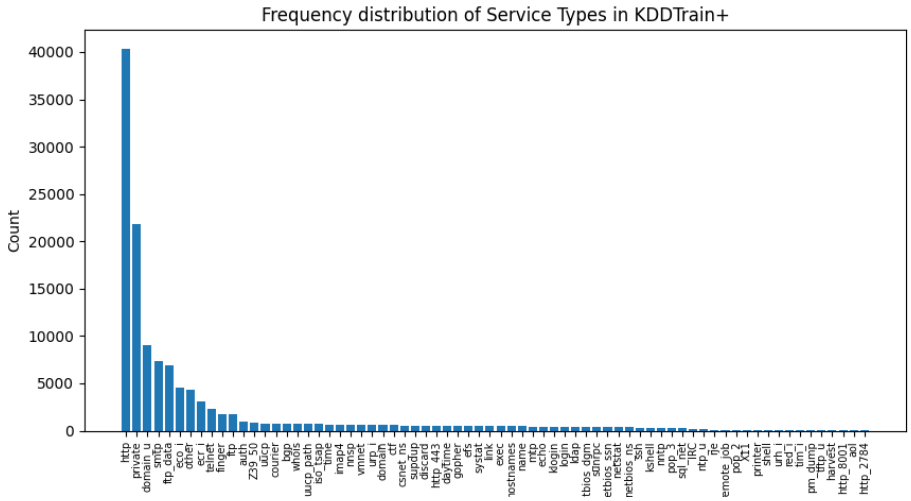


Figure 3.3: Service Type

Feature ‘flag’ has 11 categories.

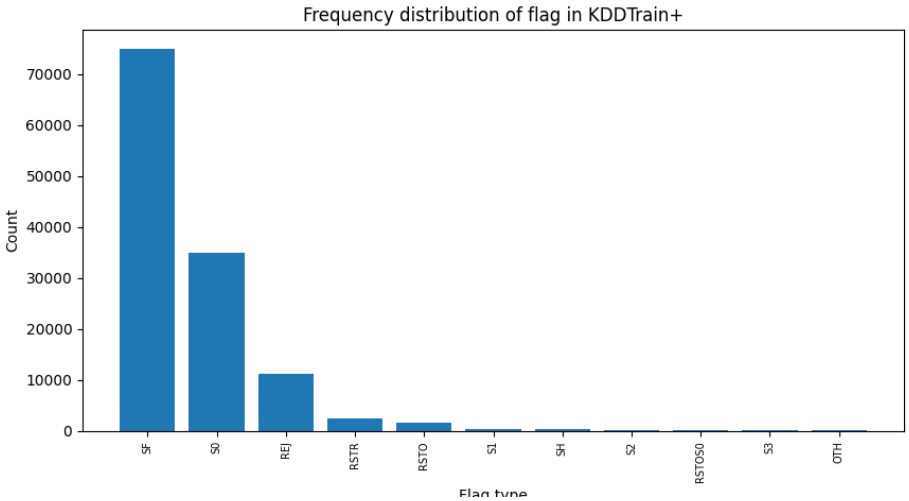


Figure 3.4: Flag Type

Feature 'labels' has 23 categories.

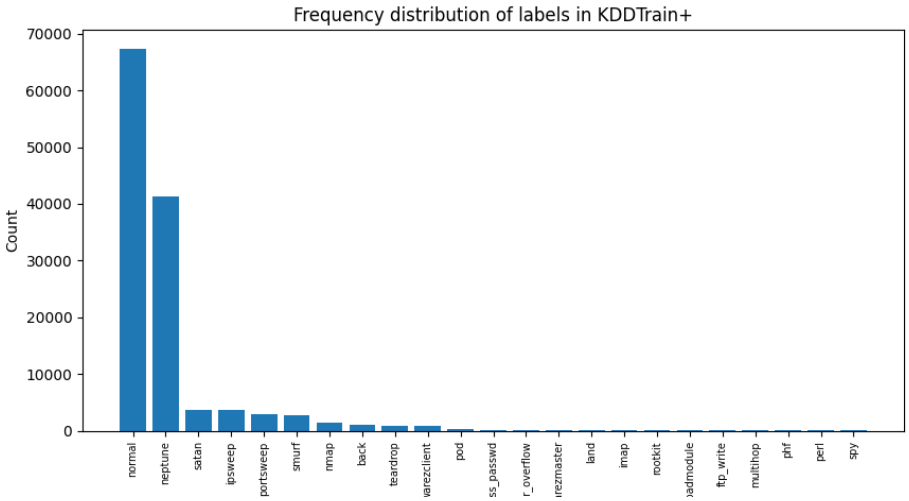


Figure 3.5: Labels Type

## 3.3 Data Preprocessing

### 3.3.1 Categorizing Labels Into Categories of Attack

There are four main categories of attacks, namely: denial-of-service (DOS), Remote-to-Local attacks - (R2L) unauthorized access from a remote machine, User-to-Root attacks(U2R) - unauthorized access to local superuser (root) privileges, and probing. [10]

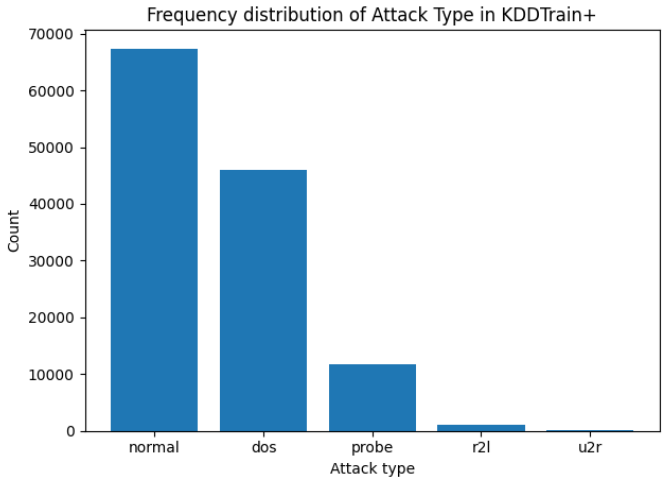


Figure 3.6: Attack Type

### 3.3.2 Encoding Categorical Features

#### Encoding 'labels' into binary values

Since the feature 'labels' contains non-numeric data, it is encoded into binary values where 0 represents a 'normal' connection and 1 represents an 'attack'. After encoding, we observe for KD-

DTrain+ set there are:
67343 instances representing ‘normal’.
58630 instances representing ‘attack’.

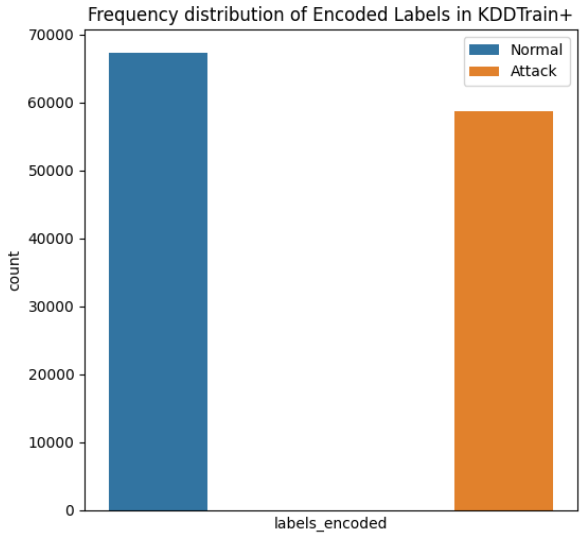


Figure 3.7: Labels Encoded

**One-Hot-Encoding**

Most machine learning algorithms require numerical data as input to the model, hence the categorical features in our data set must be first converted into some numeric form for the machine learning algorithms to be able to process it. It can be done using label encoding or one-hot-encoding. Label encoding however has a disadvantage, that is, it gives a false sense of natural ordering and the algorithm may assume a false ordering based on assigned orders. This results in poor performance or unexpected results. One-hot-encoding is a widely used technique for creating categor-

ical features. This method creates a new feature per level of the original feature. Hence, we implement the encoding step using the one-hot-encoding method. After replacing all the columns by the derived one-hot encoded columns and adding dummy columns, we obtain 124 columns in the dataset.

### 3.3.3 Feature Scaling

The NSL KDD dataset includes features that highly vary in magnitudes, units, and range. Feature scaling is an effective means to improve predictive results. It refers to the standardization of independent features in a dataset. This method rescales the values of a feature to fall within a certain range.

#### Min-Max Scaling

It rescales data to have values between 0 and 1.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

where:
$x'$ = Normalized Value

### 3.3.4 Feature Elimination

#### Eliminating features based on variance

If the variance is low or close to zero, then it can be removed from the dataset. Features with low or zero variance are approximately constant and will not improve the performance of the predictive

model. In the NSL KDD dataset, we observe the feature 'num outbound cmds' has a variance of zero, hence it can be dropped from the dataset.

### 3.3.5 Summary

y_train and y_test contain encoded labels. X_train and X_test contain 122 features as listed in Figure3.8.

| | X_train | y_train | X_test | y_test |
|---|---|---|---|---|
| Number of rows | 125973 | 125973 | 22544 | 22544 |
| Number of columns | 122 | 1 | 122 | 1 |
| Data Type | float64 | float64 | float64 | float64 |

Table 3.1: Summary of datasets after preprocessing.

**List of features after one-hot encoding**

| # | Features | # | Features | # | Features | # | Features | # | Features | # | Features | # | Features |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | duration | 16 | num_access_files | 32 | dst_host_same_src_port_rate | 48 | flag_S1 | 64 | service_domain_u | 80 | service_kshell | 96 | service_pop_3 | 112 | service_urp_i |
| 2 | src_bytes | 17 | is_host_login | 33 | dst_host_srv_diff_host_rate | 49 | flag_S2 | 65 | service_echo | 81 | service_ldap | 97 | service_printer | 113 | service_uucp |
| 3 | dst_bytes | 18 | is_guest_login | 34 | dst_host_serror_rate | 50 | flag_S3 | 66 | service_eco_i | 82 | service_link | 98 | service_private | 114 | service_uucp_path |
| 4 | land | 19 | count | 35 | dst_host_srv_serror_rate | 51 | flag_SF | 67 | service_ecr_i | 83 | service_login | 99 | service_remote_job | 115 | service_vmnet |
| 5 | wrong_fragment | 20 | srv_count | 36 | dst_host_rerror_rate | 52 | flag_SH | 68 | service_efs | 84 | service_mtp | 100 | service_rje | 116 | service_whois |
| 6 | urgent | 21 | serror_rate | 37 | dst_host_srv_rerror_rate | 53 | service_IRC | 69 | service_exec | 85 | service_name | 101 | service_shell | 117 | service_urh_i |
| 7 | hot | 22 | srv_serror_rate | 38 | score | 54 | service_X11 | 70 | service_finger | 86 | service_netbios_dgm | 102 | service_smtp | 118 | service_aol |
| 8 | num_failed_logins | 23 | rerror_rate | 39 | protocol_type_icmp | 55 | service_Z39_50 | 71 | service_ftp | 87 | service_netbios_ns | 103 | service_sql_net | 119 | service_http_2784 |
| 9 | logged_in | 24 | srv_rerror_rate | 40 | protocol_type_tcp | 56 | service_auth | 72 | service_ftp_data | 88 | service_netbios_ssn | 104 | service_ssh | 120 | service_red_i |
| 10 | num_compromised | 25 | same_srv_rate | 41 | protocol_type_udp | 57 | service_bgp | 73 | service_gopher | 89 | service_netstat | 105 | service_sunrpc | 121 | service_http_8001 |
| 11 | root_shell | 26 | diff_srv_rate | 42 | flag_OTH | 58 | service_courier | 74 | service_hostnames | 90 | service_nnsp | 106 | service_supdup | 122 | service_harvest |
| 12 | su_attempted | 27 | srv_diff_host_rate | 43 | flag_REJ | 59 | service_csnet_ns | 75 | service_http | 91 | service_nntp | 107 | service_systat | | |
| 13 | num_root | 28 | dst_host_count | 44 | flag_RSTO | 60 | service_ctf | 76 | service_http_443 | 92 | service_ntp_u | 108 | service_telnet | | |
| 14 | num_file_creations | 29 | dst_host_srv_count | 45 | flag_RSTOS0 | 61 | service_daytime | 77 | service_imap4 | 93 | service_other | 109 | service_tftp_u | | |
| 15 | num_shells | 30 | dst_host_same_srv_rate | 46 | flag_RSTR | 62 | service_discard | 78 | service_iso_tsap | 94 | service_pm_dump | 110 | service_tim_i | | |
| 16 | num_access_files | 31 | dst_host_diff_srv_rate | 47 | flag_S0 | 63 | service_domain | 79 | service_klogin | 95 | service_pop_2 | 111 | service_time | | |

Figure 3.8: List of features.

# Chapter 4

# Autoencoders

Autoencoders are artificial neural networks with a symmetric structure. They follow an unsupervised learning approach.

## 4.1 General Structure

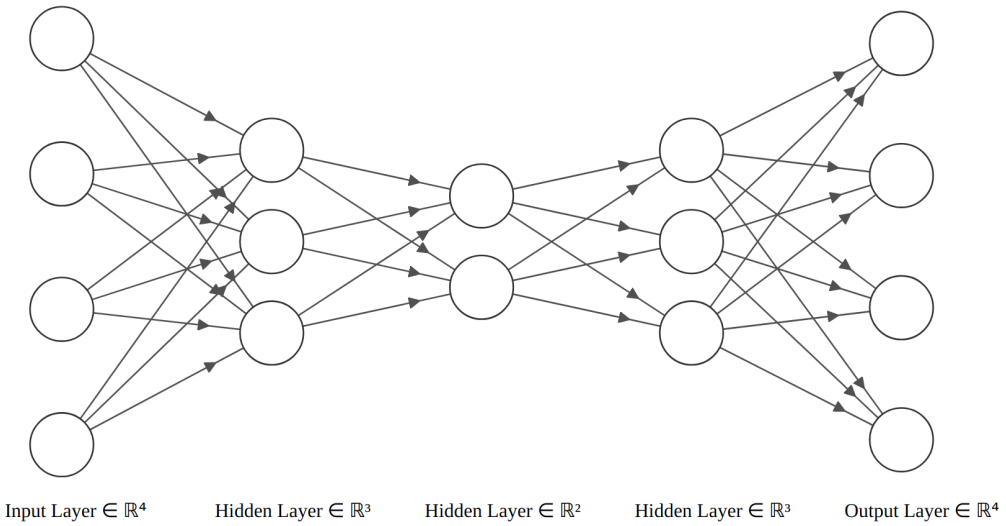


Figure 4.1: General structure of an autoencoder.

An Autoencoder has an input layer, an output layer, and one or more hidden layers. They have an encoder-decoder structure. The middle layer of an autoencoder represents an encoding of the input data. The encoder compresses the input into a latent-space representation. The decoder reconstructs the input from this latent-space representation. These encodings have a much lower dimensionality than the input data. Autoencoders attempt to reconstruct the input. During reconstruction, the output may differ from the original input since they are transformed from a lower dimension to a higher dimension. This difference is quantified in terms of reconstruction error.

## 4.2 Autoencoders for Anomaly Detection

Autoencoders built for anomaly detection train only on normal instances. During prediction, reconstruction error produced during the decoding phase is used for anomaly detection. Instances with high reconstruction error are identified as outliers, that is, anomalies. Stereotypical instances should have low reconstruction error. A threshold is set to identify anomalies. Any data point in the test set that has reconstructed error greater than threshold error will be marked as an outlier.

## 4.3 Deep Autoencoders

The encoders and decoders are made up by stacking multiple fully-connected layers of neurons. Multiple layers help the au-

toencoders learn complex codings.

# Chapter 5

# Methodology

## 5.1 Architecture

In our implementation each autoencoder consists of one input, three hidden, and one output layer. The layers are fully connected.

| **Layer number** | **Type** | **Number of neurons** |
|---|---|---|
| Layer 1 | Input Layer | 122 neurons |
| Layer 2 | Hidden Layer 1 | 32 neurons |
| Layer 3 | Hidden Layer 2 | 16 neurons |
| Layer 4 | Hidden Layer 3 | 32 neurons |
| Layer 5 | Output Layer | 122 neurons |

Table 5.1: Summary of autoencoder architecture

We implemented the model to predict two different types of classes: Normal and anomaly (2-class classification). We study

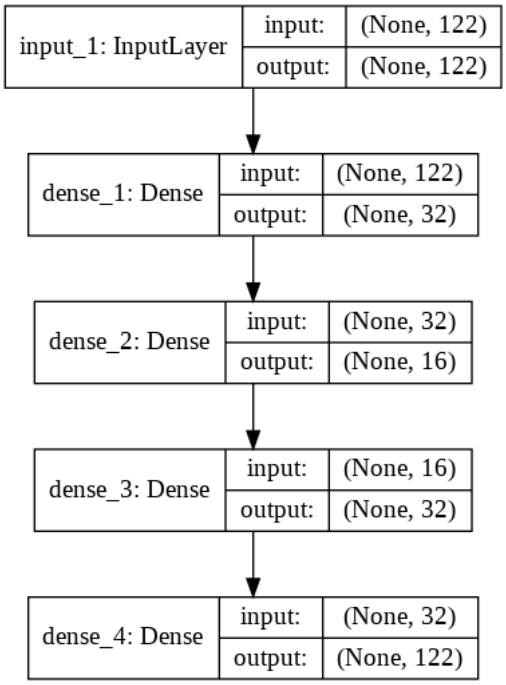


Figure 5.1: Architecture of implemented model.

and compare how different optimizers and activation functions perform in building a predictive model for anomaly detection using deep autoencoders. We implement, train, and evaluate 9 autoencoder models with the following specifications:

| Model number | Number of hidden layers | Neurons | Activation | Optimizer | Loss | Learning rate |
|---|---|---|---|---|---|---|
| 1 | Three hidden layers | [122,32,16,32,122] | linear | adam | mse | 0.001 |
| 2 | Three hidden layers | [122,32,16,32,122] | sigmoid | adam | mse | 0.001 |
| 3 | Three hidden layers | [122,32,16,32,122] | tanh | adam | mse | 0.001 |
| 4 | Three hidden layers | [122,32,16,32,122] | linear | sgd | mse | 0.01 |
| 5 | Three hidden layers | [122,32,16,32,122] | sigmoid | sgd | mse | 0.01 |
| 6 | Three hidden layers | [122,32,16,32,122] | tanh | sgd | mse | 0.01 |
| 7 | Three hidden layers | [122,32,16,32,122] | linear | rmsprop | mse | 0.001 |
| 8 | Three hidden layers | [122,32,16,32,122] | sigmoid | rmsprop | mse | 0.001 |
| 9 | Three hidden layers | [122,32,16,32,122] | tanh | rmsprop | mse | 0.001 |

Table 5.2: Summary of model specifications

## 5.2 Loss Function

For all the implemented models, we use Mean Square Error (mse) as the loss function. This measures the difference or the error between the reconstructed and the original input. Our objective is to minimize the mse, and we reach that by using an optimizer.

$$MSE = \frac{1}{n}\sum(x - z)^2 \tag{5.1}$$

where:
$x$ = original input
$z$ = output obtained by reconstruction
$n$ = number of data points

## 5.3 Optimizers

### 5.3.1 adam

Adam stands for Adaptive Moment Estimation. Adam is a popular and robust optimization algorithm that has been designed specifically for training deep neural networks because it achieves good results fast. It is an adaptive learning rate optimization algorithm. For all the models implemented in this report we use the following configuration parameters:
$learning\_rate = 0.001$
$beta\_1 = 0.9$
$beta\_2 = 0.999$
$epsilon = 1e - 07$

### 5.3.2 sgd

SGD stands for stochastic gradient descent. It is a "classical" optimization technique. SGD work on small batches of training data and computes the gradient of the network loss function to update each individual weight in the network. For all the models implemented in this report we use the following configuration parameters:
$learning\_rate = 0.01$
$momentum = 0.0$

### 5.3.3 rmsprop

RMSprop stands for Root Mean Square Propagation. It is also an adaptive learning rate optimization algorithm. It is an unpublished, yet very widely-known gradient descent optimization algorithm for mini-batch learning of neural networks. This technique was proposed by Geoffrey Hinton at his Neural Networks Coursera course. For all the models implemented in this report we use the following configuration parameters:
$learning\_rate = 0.001$
$rho = 0.9$
$momentum = 0.0$
$epsilon = 1e - 07$

## 5.4 Activation Function

Activation functions take in a real valued number and execute a fixed mathematical operation on it. These functions are attached to each neuron in the neural network, and determine whether the output of the neuron should be transferred to the next layer or not. It enables to create complex mappings between the input and outputs of the neural network by introducing non-linearity into the output of a neuron (when non-linear activation functions like sigmoid, tanh are used).

### 5.4.1 linear

It is a straight line function where activation is proportional to input. It does not introduce non-linearity into the neural network.

**Equation**

$$f(x) = x \tag{5.2}$$

**Derivative**

$$f'(x) = 1 \tag{5.3}$$

**Plot**

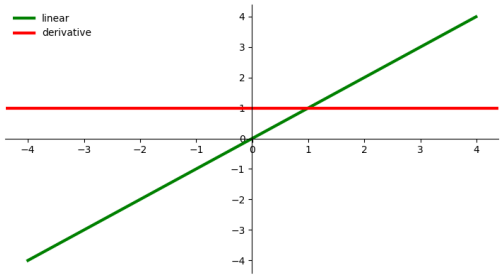


Figure 5.2: Plot for linear activation function.

### 5.4.2 sigmoid

Sigmoid function is a non-linear activation function which maps input values to be between 0 and 1.

**Equation**

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.4}$$

**Derivative**

$$f'(x) = f(x)(1 - f(x)) \tag{5.5}$$

**Plot**

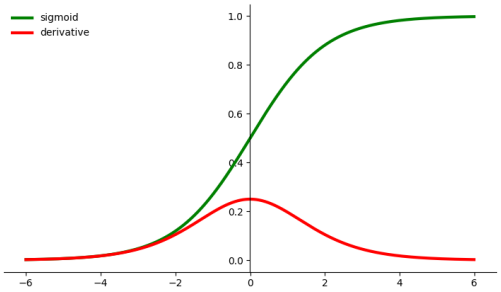


Figure 5.3: Plot for sigmoid activation function

### 5.4.3 tanh

Tanh function is a non-linear activation function which maps input values to be between -1 and 1.

**Equation**

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{5.6}$$

### Derivative

$$f'(x) = 1 - f(x)^2 \tag{5.7}$$

### Plot

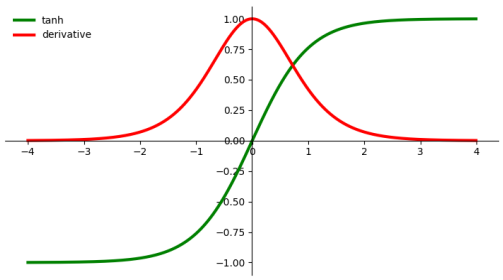


Figure 5.4: Plot for tanh activation function

# Chapter 6

# Implementation

The source code for training and evaluating the neural models was written using Python and Keras deep learning library [1] with the Tensorflow backend on Google Colaboratory.

## 6.1 Training

During the training phase, the autoencoder learns the characteristics of normal instances only hence the original X_train dataset was split to contain normal instances only. There were 67343 data points that were categorized as 'normal' in the X_train set, 20% of the instances from this set were used for validation. Hence, the models were trained on 53874 instances and validated on 13469 instances. The batch size was set to 128 and all models were trained for 100 epochs using Mean Squared Error (mse) as the loss function.

Model 1: Linear activation function was used with adam opti-

mizer.
Model 2: Sigmoid activation function was used with adam optimizer.
Model 3: Tanh activation function was used with adam optimizer.
Model 4: Linear activation function was used with sgd optimizer.
Model 5: Sigmoid activation function was used with sgd optimizer.
Model 6: Tanh activation function was used with sgd optimizer.
Model 7: Linear activation function was used with rmsprop optimizer.
Model 8: Sigmoid activation function was used with rmsprop optimizer.
Model 9: Tanh activation function was used with rmsprop optimizer.

## 6.2 Testing

A reconstruction error is generated upon prediction for each instance in X_test. Higher the reconstruction error, higher the possibility of that instance being an anomaly. This is because the autoencoder was trained on normal instances only and the weights of the autoencoder had been optimized to encode and decode normal instances only. As a result, any anomalous instance will result in a higher magnitude of reconstruction error. A loss threshold is set to distinguish normal and anomalous data instances. If reconstruction error is greater than the set threshold, the data point is classified as an anomaly, else if reconstruction error is less than

the set threshold, it is classified as a normal instance. Testing was carried out on the entire X_test set which contained 22544 data points. A histogram of mean square error for all data points with the threshold value was plotted during the testing phase.

## 6.3 Performance Evaluation

The following performance metrics were used to evaluate the model - Accuracy, Precision, Recall, F1 score.

# Chapter 7

# Results

The following results were obtained after successfully implementing nine autoencoder models.

## 7.1 Summary

| Model | activation | optimizer | loss | threshold | TP | FP | TN | FN | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | linear | adam | mse | 0.001001001001 | 12224 | 1443 | 8268 | 609 | 0.9089779986 | 0.8944172093 | 0.9525442219 | 0.9225660377 |
| 2 | sigmoid | adam | mse | 0.001001001001 | 11814 | 1204 | 8507 | 1019 | 0.9013928318 | 0.9075126748 | 0.9205953401 | 0.9140071951 |
| 3 | tanh | adam | mse | 0.001001001001 | 12058 | 1044 | 8667 | 775 | 0.9193133428 | 0.9203175088 | 0.939608821 | 0.9298631193 |
| 4 | linear | sgd | mse | 0.01601601602 | 12251 | 1323 | 8388 | 582 | 0.9154985806 | 0.9025342567 | 0.9546481727 | 0.9278600371 |
| 5 | sigmoid | sgd | mse | 0.01701701702 | 12473 | 3003 | 6708 | 360 | 0.8508250532 | 0.8059576118 | 0.9719473233 | 0.8812038574 |
| 6 | tanh | sgd | mse | 0.01601601602 | 12353 | 1755 | 7956 | 480 | 0.9008605394 | 0.875602495 | 0.9625964311 | 0.9170409413 |
| 7 | linear | rmsprop | mse | 0.001001001001 | 12252 | 1532 | 8179 | 581 | 0.9062721789 | 0.8888566454 | 0.9547260968 | 0.9206146448 |
| 8 | sigmoid | rmsprop | mse | 0.001001001001 | 12011 | 1311 | 8400 | 822 | 0.9053850248 | 0.9015913526 | 0.9359463882 | 0.9184477155 |
| 9 | tanh | rmsprop | mse | 0.001001001001 | 12063 | 1127 | 8584 | 770 | 0.9158534422 | 0.9145564822 | 0.9399984415 | 0.9271029474 |

Table 7.1: Summary of nine models.

## 7.2 Histogram of mean square error

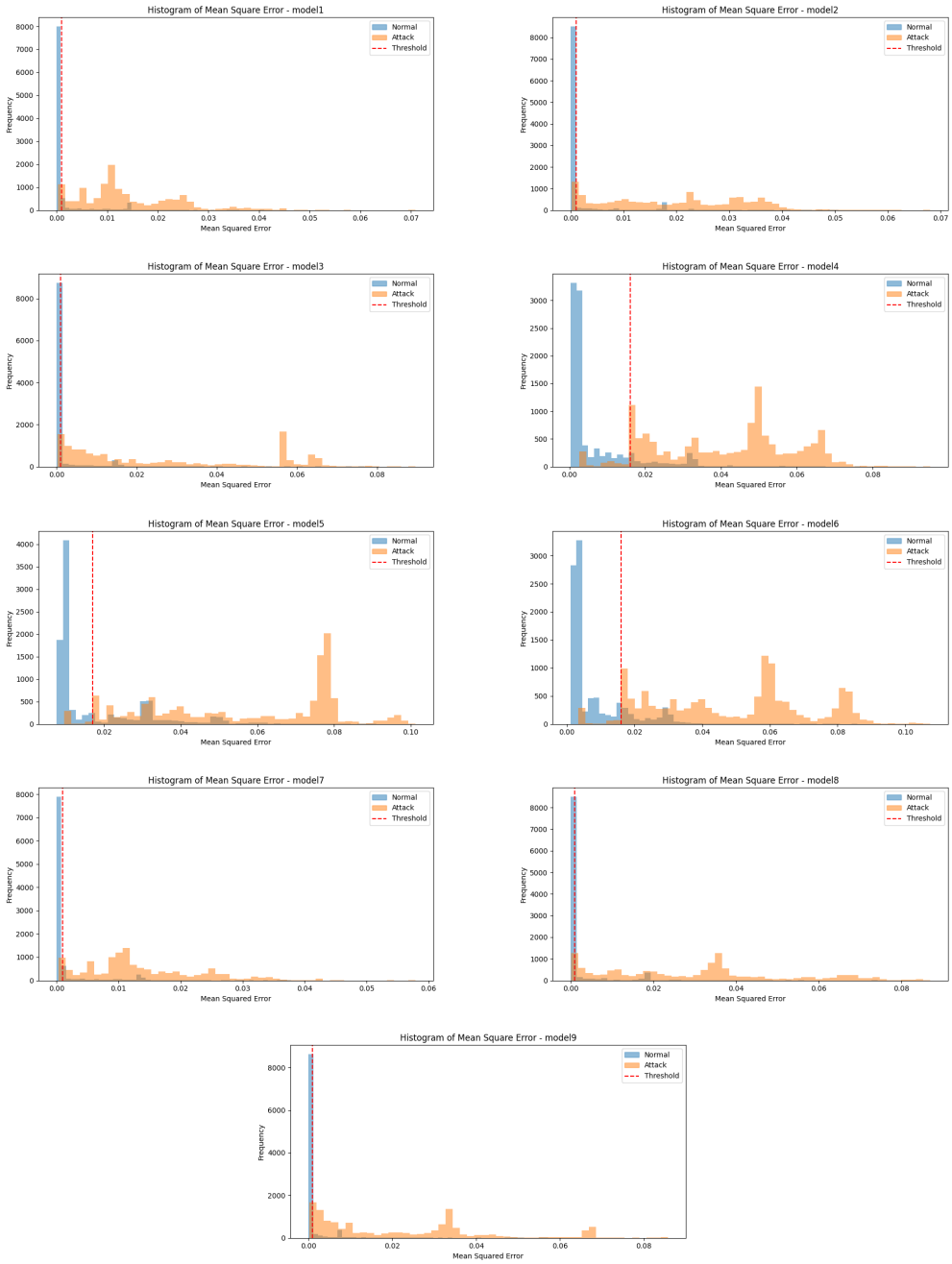


Figure 7.1: Histogram of MSE

## 7.3 Training Loss

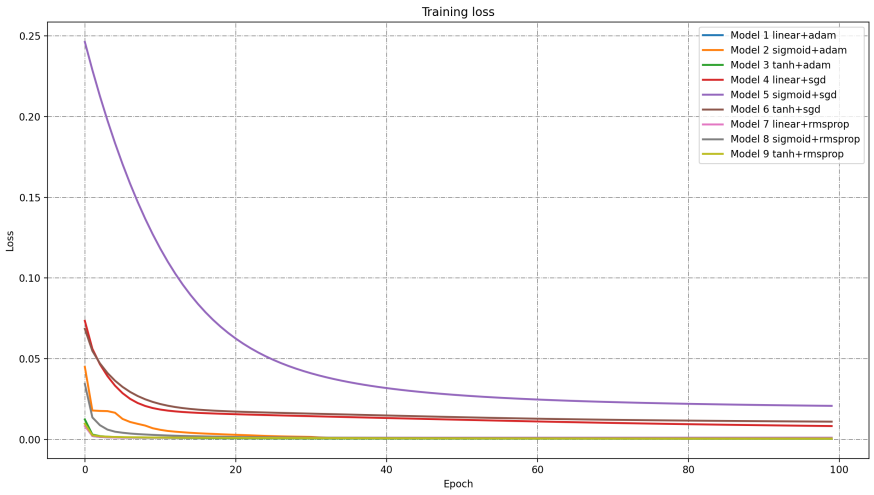


Figure 7.2: Plot for training loss.

## 7.4 Training Accuracy

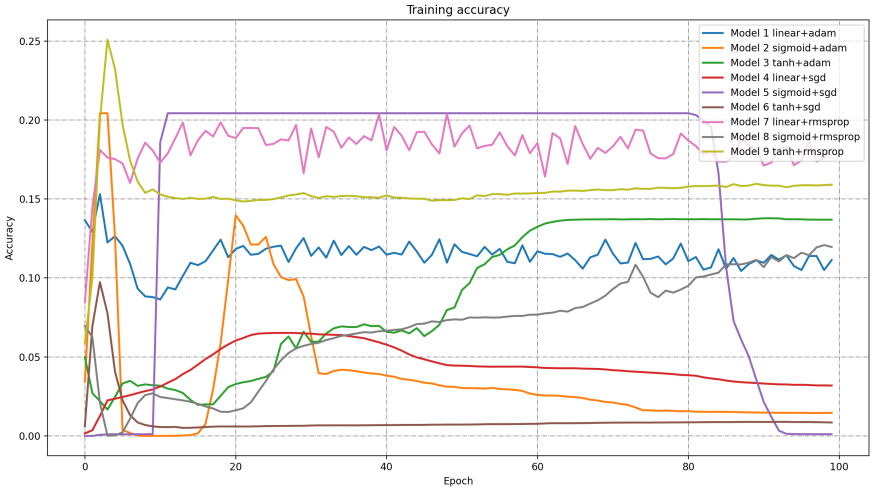


Figure 7.3: Plot for training accuracy.

## 7.5 Validation Loss

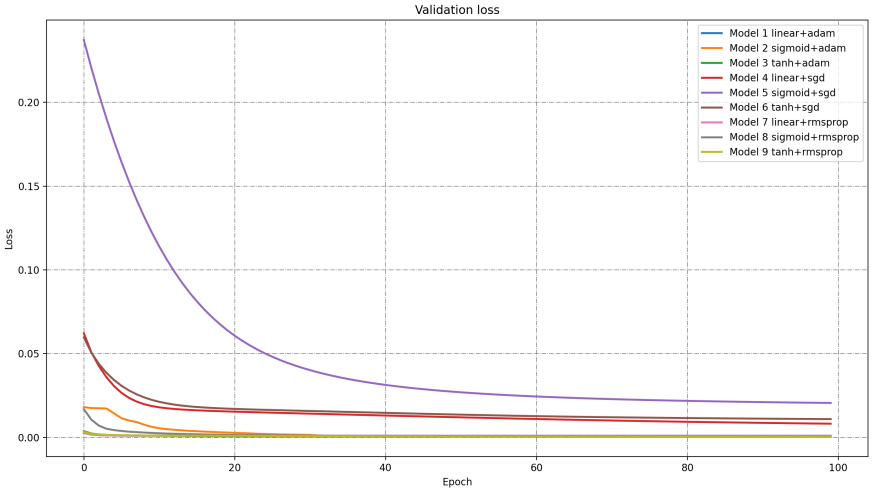


Figure 7.4: Plot for validation loss.

## 7.6 Validation Accuracy

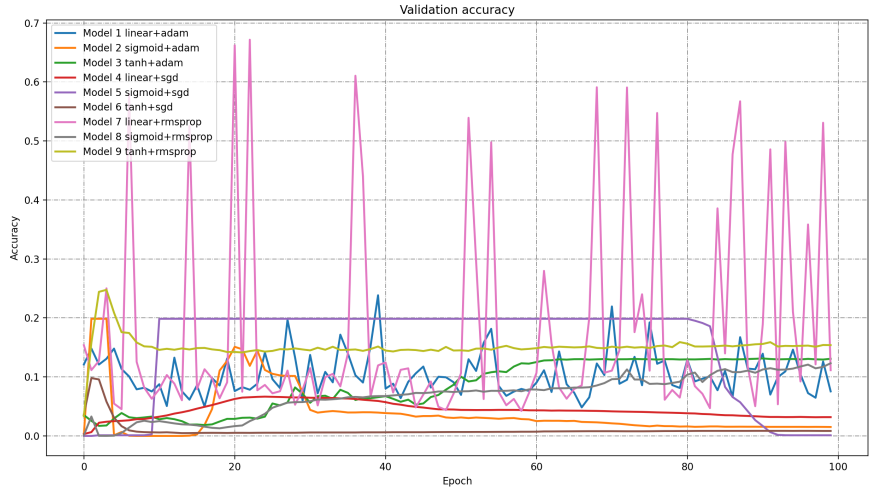


Figure 7.5: Plot for validation accuracy.

## 7.7 Precision Recall Curve

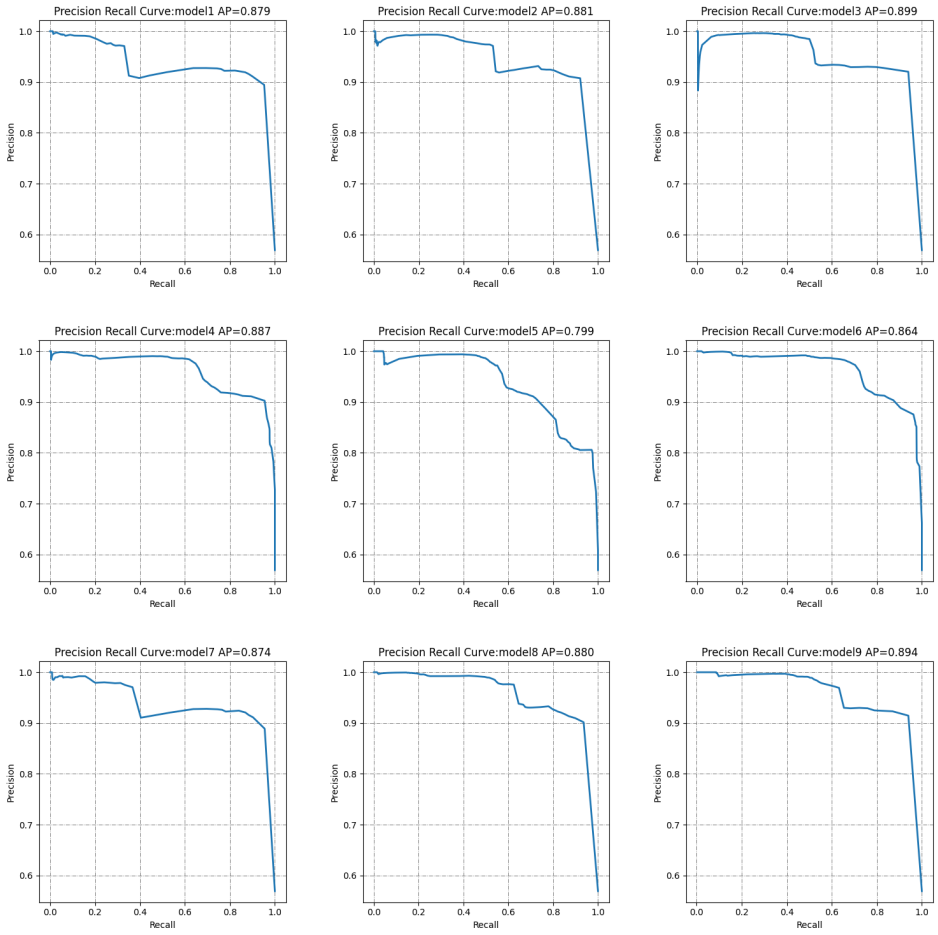


Figure 7.6: Precision - Recall Curve

## 7.8 Performance Metrics

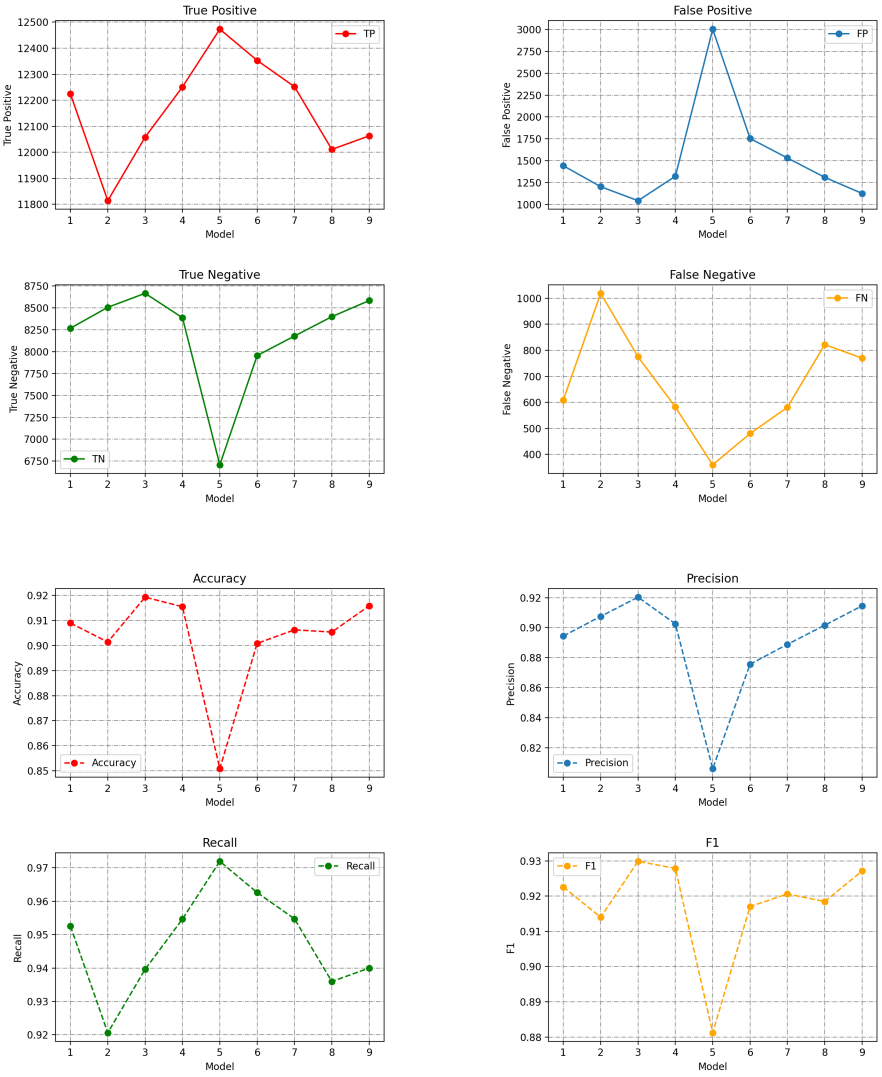


Figure 7.7: Plot for performance metrics.

# Chapter 8

# Conclusion

We implemented nine models of a stacked autoencoder with different activation functions and optimizers for developing an efficient NIDS. We used the benchmark network intrusion dataset - NSL-KDD to evaluate anomaly detection accuracy. We observed that Model-3 with 'tanh' activation function and 'adam' optimizer performed very well compared to the rest of the autoencoder models implemented. A threshold value of 0.001 on mse was set to differentiate between normal and anomalous network connections. An accuracy of 91.93% with an average precision of 0.899 from precision-recall curve was obtained in normal/anomaly detection for Model-3 when evaluated on the test data.

# Bibliography

[1] F. Cholletet al., "Keras," https://github.com/fchollet/keras, 2015.

[2] F. Chollet,Deep Learning with Python, 1st ed. USA: Manning Publications Co., 2017.

[3] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data,"PloS one, vol. 11,no. 4, p. e0152173, 2016.

[4] A. Gron,Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems,1st ed. O'Reilly Media, Inc., 2017.

[5] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," inProceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), 2016, pp. 21–26.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization,"arXiv preprint arXiv:1412.6980, 2014.

[7] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp.1–6.

[8] G. Van Rossum and F. L. Drake,Python 3 Reference Manual. ScottsValley, CA: CreateSpace, 2009.

[9] NSL KDD Dataset - https://www.unb.ca/cic/datasets/nsl.html

[10] NSL KDD Attack Types - http://kdd.ics.uci.edu/databases/kddcup99/training_attack_types