Detection and Prevention of Advanced Persistent Threat (APT) activities in heterogeneous networks using Deep Learning

1Dharaneish V C   
*Department of Computer Science and Engineering (CSE)*  
*Amrita Vishwa Vidyapeetham*Coimbatore - 641112, India   
[dharaneishvc@gmail.com](mailto:dharaneishvc@gmail.com)

4Dr. Senthil Kumar T  
*Department of Computer Science and Engineering (CSE)*  
*Amrita Vishwa Vidyapeetham*Coimbatore - 641112, India  
[t\_senthilkumar@cb.amrita.edu](mailto:t_senthilkumar@cb.amrita.edu) 2Nithin Kumar  
*Department of Computer Science and Engineering (CSE)*  
*Amrita* *Vishwa Vidyapeetham*Coimbatore - 641112, India  
[itsnithinkumar@gmail.com](mailto:itsnithinkumar@gmail.com)

5Kartik Srinivasan  
*Sr. Architect, IBM Security*  
*IBM India Pvt Ltd*  
[kartiksr@in.ibm.com](mailto:kartiksr@in.ibm.com)

7Sulakshan Vajipayajula  
*Architect – CTO, IBM Security*  
*IBM India Pvt Ltd*Bengaluru, India   
[svajipay@in.ibm.com](mailto:svajipay@in.ibm.com) 3Hari Varsha  
*Department of Electrical and Electronics Engineering(EEE)*  
*Amrita Vishwa Vidyapeetham*Coimbatore - 641112, India  
[v.harivarsha@gmail.com](mailto:v.harivarsha@gmail.com)

6Anjali Tibrewal  
*Sr. Architect, IBM Security*  
*IBM India Pvt Ltd*  
[anjtibre@in.ibm.com](mailto:anjtibre@in.ibm.com)

*Abstract*—Security vulnerabilities and breaches caused by malicious software attacks are a major security concern in this digital age. Malware detection continues to be a hot topic as many computer users, networks, organizations, businesses and governments are affected by the rapid growth of malware attacks. Many intrusion detection Systems have been developed to protect the data and resources from attacks. Sadly, new attacks and threats are developed every day, making it more difficult to these systems to detect those attacks. Not only the system has to detect an attack, but also it should prevent the attacks in network. To achieve this purpose an Advanced Persistent Threat detection system based on Deep Learning model can be developed as it has the potential to perform better in extracting features of data considering the massive cyber traffic in real life. In order to determine the efficiency of identifying anomalies, this work aims to examine deep learning artificial neural network algorithms like Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) and Recurrent Neural Network (RNN) and compare their efficiency in this research.

Keywords—Deep Learning, Artificial Neural Networks, Malware Detection, Intrusion detection system, Data Mining, Convolutional Neural Networks (CNN), Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN)

# Introduction

An Advanced Persistent Threat (APT) is a type of cyberattack that is carried out by a group, often nation states or state-sponsored organizations, where a hacker gains unauthorized access to a computer network and remains undetected for an extended period of time. APT is one of the major information security threats that industry is currently facing. APT attacks are particularly dangerous because they target an organization's sensitive data and exfiltrate the data to remote hosts. APT attacks are typically carried out by highly skilled and well-funded attackers, making them extremely difficult to detect and prevent using traditional security measures. Notable examples of APT attacks include the loss of 40 GB of Ashley Madison's database in 2015 and the theft of 9 GB of encrypted password data from Adobe Leak in 2013.

Conventional techniques for detecting APT attacks are inadequate when such attacks occur in a dynamic and complex infrastructure like the cloud. These attacks are challenging to identify due to their long-lasting nature on the network, and the possibility that the system would crash owing to the enormous traffic. Moreover, APT attacks often maintain their anonymity and frequently employ Zero-Day attack, a type of cyberattack that exploits a software security flaw that the developer may be unaware of. As a result, existing Intrusion Detection System solutions are unable to identify APTs. For many years, most of these attacks go unreported, like the Red October APT attack that has been operating for more than five years.

To detect and defend against APT-type attacks before exfiltration occurs, network intrusion systems using new Deep learning techniques and relevant analytical tools must be developed. The network intrusion detector is a predictive model that distinguishes between intrusions or attacks and normal connections. Deep learning is preferred due to its capacity to thoroughly analyse network data and automatically produce the feature vector. Deep learning algorithms greatly improve the performance of network intrusion detection systems by producing higher detection rates and lower false alarm rates.[1]

Both straightforward and sophisticated neural network models have been developed to identify cyberattacks on hosts and network systems. Deep learning is highly preferred due to its ability to examine the computer process that replicates the normal activity of the human brain. Therefore, to effectively detect and defend against APT attacks, new network intrusion detection systems utilizing deep learning algorithms and analytical tools must be developed.[2][3]

## Roadmap

This paper is organised into several sections. The first section, Related Works, presents a literature survey of existing intrusion detection systems. This is followed by the Materials and Methods section, which outlines the entire process of data pre-processing, transformation, and deep learning methods used in this study. The Experiment and Analysis section details the study of the dataset and the methods used in the experiment, while the Results and Conclusion section presents the accuracy and prediction metrics of the models developed, as well as the future scope of the work. Overall, this paper provides a comprehensive analysis of the intrusion detection problem, along with a detailed description of the methodology employed in the study.

# Related Works

This section examines the existing Intrusion Detection Systems, APT attack methods, and the challenges faced by them. The objective is to compare different models in a black box that produces the best accuracy which can be then used to quantify how well a cast performs in various domains.

Currently, APT attacks can be detected using tools like User and Entity Behaviour Analytics UEBA, deception technology, and network monitoring. In recent trends many Machine learning algorithms like Decision tree, Bayesian network, Support Vector Machine have been employed and those gave a reasonable Accuracy, sensitivity, specificity and false-negative rate.[4][5][6]

The Intrusion Detection System (IDS) is a tool that monitors a system for harmful activities. IDS can be categorized depending on where the data came from and uses a variety of sources that might be implemented at the network or host level, employing signature or using anomaly-based intrusion detection algorithms which is the primary focus of this paper with additional analysis of data both online and offline for a proper safety measure.

It is difficult to quantify the level of advancement that exists in the field of intrusion detection systems compared to other sub-areas of Machine learning. Getting a real-time and good dataset for comparison is problematic and the whole approach to it has become quite repetitive thus this lack of proper innovative methodology and a lack of crucial elements, such as ground-truth labels and publicly available & real-world environment traffic in datasets are among the chief problems that make it difficult to build production level systems that closely matches academic research. In this paper, more focus is given on closing the gap thus making such systems more precise and efficient.

Techniques for anomaly detection look for variations from the usual to find unidentified assaults. An anomaly detector can characterize novel behaviour based on how it differs from previously observed benign behaviour. Because it can be difficult to discriminate between unknown benign activity, unknown hostile behaviour, and system failures, anomaly detectors frequently suffer from a high false alarm rate while trying to identify unknown assaults like zero-day attacks. Additionally, system evolution makes it more challenging to define typical behaviour, which might reduce detection performance.

Some of the difficulties encountered in identifying APTs are consistent with the survey on advanced persistent threat detection by Adam Khalid et al.[7] APT assaults last for a very long period. While it might be challenging to identify sophisticated assaults, it can be more difficult to correlate attacks over a lengthy period. The condition of the devices exhibiting unusual behaviour must be monitored and recorded to detect an APT assault making it necessary to link this unusual behaviour to more instances. This is a hurdle for a sizable network as more often these APT attacks are sponsored thus making detection more difficult because of the attackers' tenacity, expertise, and funding. Systems for cloud computing are expanding quickly today which also adds stress to assessing how vulnerable these cloud computing platforms are.

Due to the growing number of malware variations, the conventional approach of feature matching is chosen, which is challenging to deal with and has weak resistance in order to make it better. Weina Niu et al.[8] discussed and compared diverse methods of detecting the existence of APT attacks and moved away from the traditional way citing its high false-negative rate for unidentified samples. This paper also aims to fix this by combining learnings of different methods other researchers have implemented in diverse domains.

Since attackers can assess a single network flow or host event in real-time, machine learning algorithms are more flexible. As the malicious payload is transmitted before the data is exfiltrated, the presence of the harmful payload can be used to infer the presence of an APT. To differentiate between legitimate network traffic and network traffic carrying malicious payloads, Micheal Zipperle Lu et al.[9] suggest employing temporal transform characteristics before feeding them to machine learning algorithms. It records regular traffic moving through gates and combines it with APT traffic. The findings indicate that temporal transform characteristics might boost detection efficiency.

Another alternate way for this was a deep learning stack suggested by T. Bodstrom et al.[10] for identifying APT assaults. This method operates first in serial mode and then switches to parallel mode to identify APT layers at a time. The data is then sent to the attack database for storage and analysis when the model recognizes an APT attack. Based on our experimentations, the former method was more suitable in terms of practicality ( inclusive of computation time ) and is far better suited to our methodology.

The work by Hanan et al.[11] aims to identify research gaps and shortcomings in current datasets, as well as their impact on building Network Intrusion Detection Systems due to the increasing number of sophisticated threats. This paper provides us with key pieces of information the majority of researchers overlook as existing datasets show a clear lack of real network threats as well as a large number of deprecated threats, limiting the detection accuracy of current machine learning IDS approaches. It provides us with the much-needed survey and analysis of prominent datasets and their impact on the development of Intrusion Detection Systems over the last decade.

Mhmood Radhi Had et al.[12] used a feature selection strategy where they extracted 12 features from 41 features in the NSL-KDD dataset and deployed classifiers like CNN, DNN, RNN, LSTM, and GRU whose scores were compared. Those techniques generated accuracy results of 98.63%, 98.53%, 98.13%, 98.04%, and 97.78% respectively. This modern approach of employing 5 deep learning classifiers on the pre-processed dataset achieved the best results in binary classification and attack detection.

Praneet Singh et al.[13] discuss the underrated problem in most models: Resource limits in novel network infrastructure tiers that limit the deployment of traditional Network Intrusion Detection Systems. They solve this issue by constructing an extremely light and blazingly accurate model that can function within resource constraints, such as low power, memory, and processor capabilities, to produce correct results at a relevant pace. It is constructed by layering Long Short-Term Memory and creating a viable data science pipeline using a Recurring Neural Network (RNN) to learn from network packet behaviour and determine if it is normal or attack-oriented. The results show that, when the model maintains a high testing accuracy of 99% although using less CPU and memory compared to traditional DLM methodologies. Furthermore, it is roughly three times less in size than the current model and requires significantly less testing time. This approach of combining different classifiers on a more abstract level can prove itself to be bleeding-edge when it comes to protection against zero-day attacks.

The proposed work is to bridge the crack of real-world network intrusion detection systems with a rather unconventional approach derived from other successes in different fields of Machine Learning. Building on the scope of this area, the contribution of this work is to give researchers an appropriate benchmark of each classifier, minimizing their effort of identifying the best ones when it comes to the design of inventive network intrusion architectures that are both performant and accurate.

# Materials and Methods

In this part, process of data pre-processing, transformation and Deep learning techniques are discussed.

## Proposed Work Flow

The proposed workflow involves the development of a deep learning-based classifier model for network intrusion detection. The system aims to accurately differentiate between normal connections and intrusions while minimizing false alarms. To achieve this, the system utilizes artificial neural networks and trains on the patterns of anomalies. The model is designed to adapt to new intrusion patterns and changes in attacker behaviour over time. Specifically, the strategy uses a deep neural network model for binary classification that is trained on the NSL-KDD dataset. The resulting output is binary, with 1 indicating an intruder and 0 indicating a typical user.

## Data Pre-processing

The effective utilization of artificial neural network algorithms to identify network intrusion detection relies heavily on data pre-processing. This phase involves the analysis, filtering, transformation, and encoding of data in a way that allows the deep learning classifier to understand and work with the processed output. The quality of the classifier results can be significantly degraded by the presence of unclean data such as missing attributes, attribute values, noise or outliers, and duplicate or incorrect data. Thus, it is essential to manipulate and transform raw data into a useful and efficient format before using it in Artificial Neural Network model to ensure performance. [15][16][17]

To achieve this objective, various data pre-processing techniques are employed. The presence of missing values in a dataset can negatively impact the performance and accuracy of a deep learning model. In order to mitigate this issue, missing values in the data are removed by deleting rows with more than 25 missing features. For rows with fewer missing values, categorical features are replaced with mode and numerical data with mean. Next, data transformation is performed to produce patterns that are simpler to comprehend by altering its format, structure, or values. The range of raw data values varies widely, and some algorithms do not perform well without feature scaling. Thus, the data value of all numerical features are scaled by subtracting the mean and scaling to unit variance using the StandardScaler class of the sklearn.preprocessing library.[14]

Here, 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.

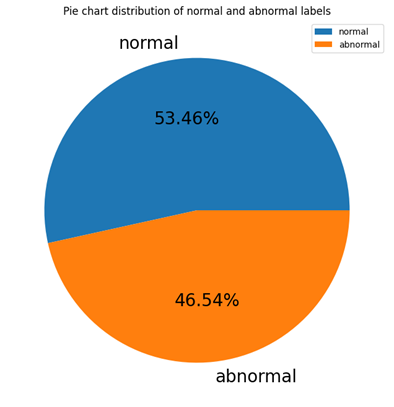
Next, concept hierarchy generation is performed to transform properties from lower to higher in the hierarchy, depending on the prediction feature. In this study, as the model aims to predict only the class of attack, the attribute label, which contains sub-classes of attacks or normal traffic, is converted into either of the four attack classes or normal.

Categorical data is converted into categories, and a dummy encoding is used to encode these categorical variables as numbers, enabling deep learning models to assume correlation between these variables that can produce false results. In the dataset, the data attributes 'protocol\_type', 'service', and 'flag' are one-hot encoded using dummy variables.

Furthermore, feature selection is performed to select only a few relevant features in the dataset for generating the deep learning classifier model. Pearson's correlation coefficient for all columns is calculated, and only attributes that have more than 0.5 correlation with encoded attack label attribute are selected for further deep learning model development.

Finally, the dataset is split into train, validation, and test sets in an 80:10:10 ratio randomly using the train\_test\_split function in sklearn, with the test dataset used to evaluate results using various metrics. The resulting dataset after merging the KDDTrain and KDDTest datasets contains 148,517 rows, with the train, validation, and test sets containing 111,387, 18,565, and 18,565 rows, respectively.

The distribution of the Normal and Abnormal labels in the dataset was found to be equally distributed with 77,054 rows of the normal class and 71,463 rows of the attack class. A pie chart of the distribution of the Normal and attack classes is shown in Fig 1, which indicates that the dataset is well balanced between the two classes.



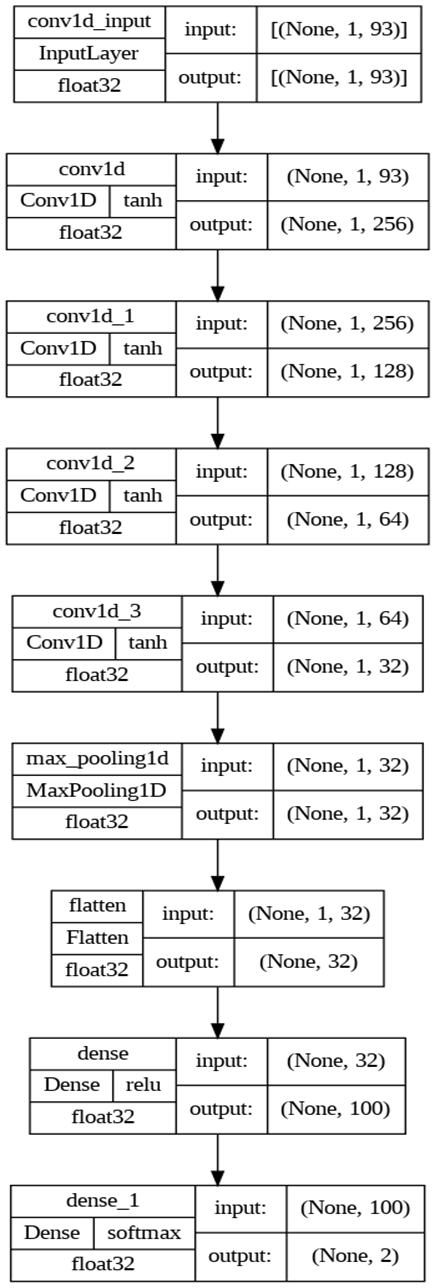
1. Pie Chart distribution of Normal and attacks in dataset

## Artficial Neural Networks

Convolutional Neural Networks (CNNs) are a type of artificial neural network that been extensively used in deep learning for binary classification tasks. The key advantage of CNNs over other machine learning algorithms is their ability to learn hierarchical features directly from raw data, without the need for manual feature engineering.

In this study, a one-dimensional convolutional neural network (1D-CNN) model for binary classification using the Keras library is used. The model architecture is presented in Fig 1 and consists of a total of 9 layers, including an input layer, four 1D convolutional layers, a max pooling layer, a flattening layer, and two fully connected (dense) layers.[20][21][22].

The model begins with a 1D convolutional layer that applies 256 filters of kernel size 1 to the input data. The activation function used is hyperbolic tangent (tanh). A second 1D convolutional layer is added with 128 filters of kernel size 1 and tanh activation. A third 1D convolutional layer is added with 64 filters of kernel size 1 and tanh activation. A fourth 1D convolutional layer is added with 32 filters of kernel size 1 and tanh activation. A max pooling layer is added with pool size 1, which reduces the output size. A flatten layer is added to convert the output of the previous layer into a 1D vector. A fully connected (dense) layer with 100 units and ReLU activation is added. Another fully connected (dense) layer with 2 units and softmax activation is added, which produces the output probabilities for the two classes (binary classification). The model is compiled with binary cross-entropy loss and Adam optimizer, and the metric used for evaluation is accuracy.



1. CNN Architecture

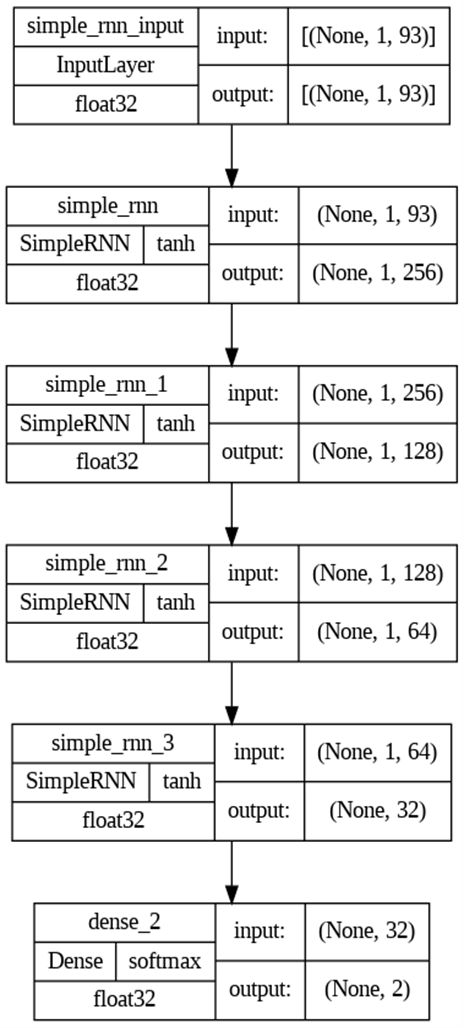
Recurrent Neural Networks (RNNs) have become increasingly popular due to their ability to process sequential data with varying lengths by utilizing internal memory. In this study, a simple RNN architecture for binary classification using the Keras library is used.

The proposed model architecture is presented in Fig 2 and consists of four layers of SimpleRNN cells followed by a Dense layer with a softmax activation function. The first SimpleRNN layer has 256 units, a dropout rate of 0.3, and return\_sequences set to True. The default tanh activation function is used in the RNN layer. A second SimpleRNN layer is added with 128 units, a dropout rate of 0.3, and return\_sequences set to True. A third SimpleRNN layer is added with 64 units, a dropout rate of 0.3, and return\_sequences set to True. Finally, a fourth SimpleRNN layer is added with 32 units, without return\_sequences, meaning that only the final output of the RNN layer will be used as input to the next layer. A fully connected (dense) layer with 2 units and softmax activation is added, which produces the output probabilities for the two classes (binary classification). The model is compiled with binary cross-entropy loss and Adam optimizer, and the metric used for evaluation is accuracy.[18][19]

In the proposed RNN architecture, the input data is processed sequentially, one time step at a time. Each RNN layer has a set of recurrent units that maintain a hidden state, which is updated at each time step based on the input and the previous hidden state. The output of each time step is then used as input for the next time step. The return\_sequences parameter set to True in the first three RNN layers means that the output of each time step will be returned as a sequence, which allows the subsequent RNN layers to process the entire sequence of outputs. The last RNN layer has return\_sequences set to False, which means that it outputs the final hidden state of the RNN layer.

In an RNN, the input data is processed sequentially, one time step at a time. Each RNN layer in this model has a set of recurrent units that maintain a hidden state, which is updated at each time step based on the input and the previous hidden state. The output of each time step is then used as input for the next time step. The return\_sequences parameter set to True in the first three RNN layers means that the output of each time step will be returned as a sequence, which allows the subsequent RNN layers to process the entire sequence of outputs. The last RNN layer has return\_sequences set to False, which means that it outputs the final hidden state of the RNN layer.

To prevent overfitting in the RNN layers, the dropout parameter is used to randomly set a fraction of the inputs to zero during training. The softmax activation function in the output layer produces a probability distribution over the two classes, and binary cross-entropy is used as the loss function to train the model to predict the correct class.

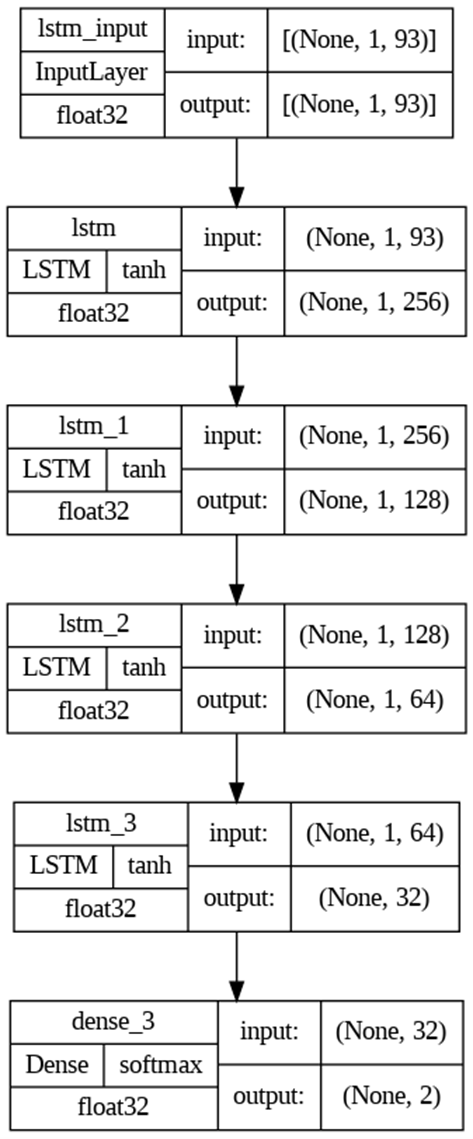


1. RNN Architecture

Long short-term memory (LSTM) is a popular type of artificial neural network that has gained significant attention in recent years due to its ability to handle long-term dependencies in input data. LSTMs are a type of recurrent neural network (RNN) that use feedback connections to process complete data sequences instead of just single data points.[23]

In this study, an LSTM architecture for binary classification using the Keras library is used. The model architecture is presented in the Fig 3 and contains a total of 6 layers: Input layer, 4 LSTM layers, and a Dense layer. The architecture begins with a first layer consisting of 256 units and a dropout rate of 0.3. The return\_sequences parameter is set to True, allowing the output of each time step to be returned as a sequence. The default tanh activation function is used for this LSTM layer. A second LSTM layer is added with 128 units, a dropout rate of 0.3, and return\_sequences set to True. A third LSTM layer is added with 64 units, a dropout rate of 0.3, and return\_sequences set to True. A fourth LSTM layer is added with 32 units, without return\_sequences, meaning that only the final output of the LSTM layer will be used as input to the next layer. The final fully connected (dense) layer has 2 units and uses a softmax activation function to produce the output probabilities for the two classes in binary classification.

The LSTM architecture is compiled with binary cross-entropy loss and Adam optimizer, and the evaluation metric used is accuracy. The dropout parameter and input shape are the same as in the RNN model architecture, allowing the LSTM to process input data sequentially, one time step at a time. The return\_sequences parameter set to True in the first three LSTM layers allows subsequent LSTM layers to process the entire sequence of outputs, while the last LSTM layer with return\_sequences set to False outputs the final hidden state of the LSTM layer.

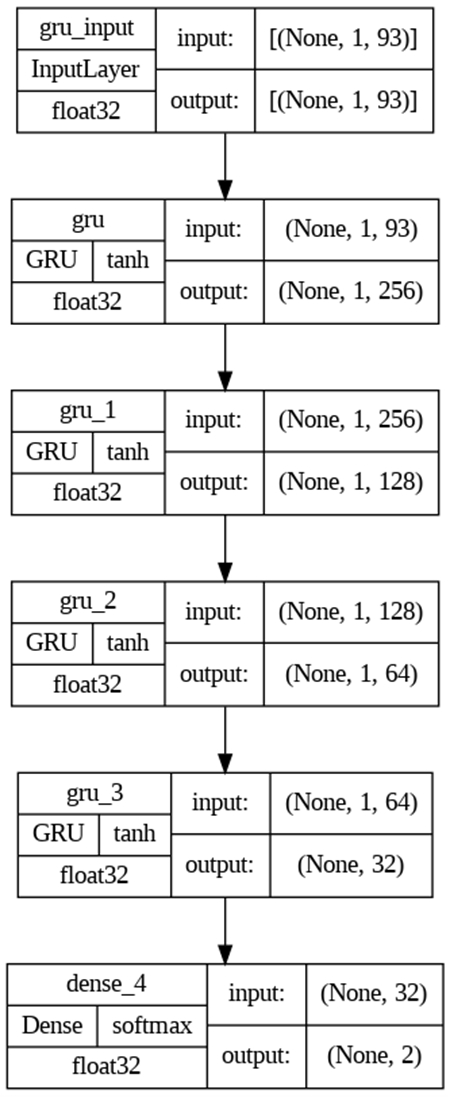


1. LSTM Architecture

Gated Recurrent Unit (GRU), is an improved model over the regular Recurrent Neural Network (RNN) by incorporating a gating mechanism. GRU is similar to an LSTM with a forget gate, but it has fewer parameters as it doesn't have an output gate.

In this study, an GRU architecture for binary classification using the Keras library is used. The model architecture the same is presented in Fig 4 and consists of a total of seven layers, including an input layer, four GRU layers, and a dense layer. The four GRU layers are sequentially connected, with a decreasing number of neurons from 256 to 32, while a dropout rate of 0.3 is incorporated to mitigate overfitting.

The input layer takes the input data in the form of a 2D array with a shape of (1, no of columns), where the first dimension corresponds to the time steps and the second dimension corresponds to the features. The second to fourth layers are the GRU layers, which are a type of recurrent neural network (RNN) that are similar to LSTMs but with fewer parameters. The GRU layer includes a reset gate and an update gate, which enable the network to remember or forget previous input data. The GRU layer also utilizes the hidden state to capture the temporal dependencies of the input data. The final layer is a dense layer with 2 neurons, producing the model's output in the form of a probability distribution over the two classes. The softmax activation function is employed to ensure that the model's outputs sum to one. The model is trained with binary cross-entropy as the loss function and the Adam optimizer.



1. GRU Architecture

# Experiments and Analysis

## Experimental Set-up

**Python Library for Data pre-processing:** The Data pre-processing and Deep learning have been performed on dataset using Python. To do data preprocessing and use Artificial Neural networks in Python, some predefined Python libraries are imported and used. These libraries are used to perform some specific tasks. All libraries used are listed in the table 1.

1. Python Libraries Used

|  |  |
| --- | --- |
| Numpy | Pandas |
| Matplotlib | Seaborn |
| keras | sklearn |
| tensorflow | visualkeras |
| pydot | graphviz |
| pydotplus | sys |

## Study of NSL KDD Dataset

**Dataset Description**: The NSL-KDD dataset was introduced as part of The Third International Knowledge Discovery and Data Mining Tools Competition, which was held alongside KDD-99, The Fifth International Conference on Knowledge Discovery and Data Mining. The primary objective of the competition was to develop a network intrusion detection system capable of accurately distinguishing between "bad" connections, known as intrusions or attacks, and "good" normal connections. The dataset comprises a comprehensive collection of auditable data, including a diverse array of simulated intrusions encountered in a military network environment. It has since become a widely used benchmark dataset in the field of network security and intrusion detection, facilitating the development and evaluation of new and improved models and algorithms.

## Dataset Splits

This data set is comprised of four sub data sets: KDDTest+, KDDTest-21, KDDTrain+, KDDTrain+\_20Percent, although KDDTest-21 and KDDTrain+\_20Percent are subsets of the KDDTrain+ and KDDTest+.

KDDTrain+ is simply referred to as train and KDDTest+ is referred to as test. The KDDTest-21 is a subset of test, without the most difficult traffic records (Score of 21), and the KDDTrain+\_20Percent is a subset of train, whose record count makes up 20% of the entire train dataset. That being said, the traffic records that exist in the KDDTest-21 and KDDTrain+\_20Percent are already in test and train respectively and aren’t new records held out of either dataset.

## Features

The dataset contains 4,94,021 tuples and 43 features per record, with 41 referring to the traffic input itself [independent] and the last two being labels (whether the traffic input is normal or attack) and Score (the severity of the traffic input itself) [dependent].

Within the data set exists 4 different classes of attacks: Denial of Service (DoS), Probe, User to Root(U2R), and Remote to Local (R2L). A brief description of each attack is presented in table 2.

1. Classes of Attacks in NSL KDD Dataset

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Attribute type** | **Purpose** |
| DoS | Explicit | shut down traffic flow from the target system. (IDS is flooded with an abnormal amount of traffic)  Eg: online retailer getting flooded with online orders on a day with a big sale |
| Probe | Implicit | get information from a network  act like a thief and steal important information |
| U2R | Implicit | Exploit the vulnerabilities to gain root privileges  (starts off with a normal user account and tries to gain access to the system or network, as a super-root user) |
| R2L | Implicit | gain local access to a remote machine (kinda hacking) |

Here Important to note is - DoS acts differently from the other three attacks, where DoS attempts to shut down a system to stop traffic flow altogether, whereas the other three attempts to quietly infiltrate the system undetected.

Break- down of sub classes of each attack is summarised in the table 3.

1. Sub-Classes of Attacks in NSL KDD Dataset

|  |  |  |
| --- | --- | --- |
| **Classes** | **Sub-Classes** | **Total Count** |
| DoS | apache2, back, land, Neptune, mailbomb, pod, processtable, smurf, teardrop, udpstorm, worm | 11 |
| Proble | Ipsweep, mscan, nmap, portsweep, saint, satan | 6 |
| U2R | Buffer\_overflow, loadmodule, perl, ps, rootkit, sqlattack, xterm | 7 |
| R2L | ftp\_write, guess\_passwd, httptunnel, imap, multihop, named, phf, sendmail, Snmpgetattack, spy, snmpguess, warezclient, warezmaster, xlock, xsnoop | 15 |

Essentially, more than half of the records that exist in each data set are normal traffic, and the distribution of U2R and R2L are extremely low. Although this is low, this is an accurate representation of the distribution of modern-day internet traffic attacks, where the most common attack is DoS and U2R and R2L are hardly ever seen

## Class level Details

The features can be broken down into four categories: Intrinsic, Content, Host-based, and Time-based.

1. Classification of Features in NSL KDD Dataset

|  |  |  |
| --- | --- | --- |
| **Category** | **Description** | **Features** |
| Intrinsic features | These can be derived from the header of the packet without looking into the payload itself, and hold the basic information about the packet. | Features 1-9 |
| Content features | These hold information about the original packets, as they are sent in multiple pieces rather than one. With this information, the system can access the payload. This category contains features 10–22. | Features 10-22 |
| Time-based features | These features hold the analysis of the traffic input over a two-second window and contain information like how many connections it attempted to make to the same host. These features are mostly counts and rates rather than information about the content of the traffic input | Features 23-31 |
| Host-based features | These features are similar to Time-based features, except instead of analyzing over a 2-second window, it analyzes over a series of connections made (how many requests made to the same host over x-number of connections). These features are designed to access attacks, which span longer than a two-second window time-span. | Features 32-41 |

## Feature Types

These features types can be broken down into Categorical, Binary, Discrete and Continuous

#### 4 Categorical (Features: 2, 3, 4, 42)

#### 6 Binary (Features: 7, 12, 14, 20, 21, 22)

#### 23 Discrete (Features: 8, 9, 15, 23–41, 43)

#### 10 Continuous (Features: 1, 5, 6, 10, 11, 13, 16, 17, 18, 19)

Here is the detailed description about each feature in dataset.

1. Description of Features in NSL KDD Dataset

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | **Feature Name** | **Description** | **Type** | **Value Type** | **Ranges** |
| 1 | Duration | Length of time duration of the connection | Continuous | Integers | 0 - 54451 |
| 2 | Protocol Type | Protocol used in the connection | Categorical | Strings |  |
| 3 | Service | Destination network service used | Categorical | Strings |  |
| 4 | Flag | Status of the connection – Normal or Error | Categorical | Strings |  |
| 5 | Src Bytes | Number of data bytes transferred from source to destination in single connection | Continuous | Integers | 0 - 1379963888 |
| 6 | Dst Bytes | Number of data bytes transferred from destination to source in single connection | Continuous | Integers | 0 - 309937401 |
| 7 | Land | If source and destination IP addresses and port numbers are equal then, this variable takes value 1 else 0 | Binary | Integers | {0, 1} |
| 8 | Wrong Fragment | Total number of wrong fragments in this connection | Discrete | Integers | {0,1,3} |
| 9 | Urgent | Number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated | Discrete | Integers | 0 - 3 |
| 10 | Hot | Number of “hot‟ indicators in the content such as: entering a system directory, creating programs and executing programs | Continuous | Integers | 0 - 101 |
| 11 | Num Failed Logins | Count of failed login attempts | Continuous | Integers | 0 - 4 |
| 12 | Logged In | Login Status: 1 if successfully logged in; 0 otherwise | Binary | Integers | {0, 1} |
| 13 | Num Compromised | Number of "compromised” conditions | Continuous | Integers | 0 - 7479 |
| 14 | Root Shell | 1 if root shell is obtained; 0 otherwise | Binary | Integers | {0 , 1} |
| 15 | Su Attempted | 1 if "su root'' command attempted or used; 0 otherwise | Discrete |  |  |
| 16 | (Dataset contains ‘2’ value) | Integers | 0 - 2 |  |  |
| 17 | Num Root | Number of "root'' accesses or number of operations performed as a root in the connection | Continuous | Integers | 0 - 7468 |
| 18 | Num File Creations | Number of file creation operations in the connection | Continuous | Integers | 0 - 100 |
| 19 | Num Shells | Number of shell prompts | Continuous | Integers | 0 - 2 |
| 20 | Num Access Files | Number of operations on access control files | Continuous | Integers | 0 - 9 |
| 21 | Num Outbound Cmds | Number of outbound commands in an ftp session | Continuous | Integers | {0} |
| 22 | Is Hot Logins | 1 if the login belongs to the "hot'' list i.e., root or admin; else 0 | Binary | Integers | {0, 1} |
| 23 | Is Guest Login | 1 if the login is a "guest'' login; 0 otherwise | Binary | Integers | {0, 1} |
| 24 | Count | Number of connections to the same destination host as the current connection in the past two seconds | Discrete | Integers | 0 - 511 |
| 25 | Srv Count | Number of connections to the same service (port number) as the current connection in the past two seconds | Discrete | Integers | 0 - 511 |
| 26 | Serror Rate | The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 27 | 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) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 28 | Rerror Rate | The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 29 | Srv Rerror Rate | The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv\_count (24) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 30 | Same Srv Rate | The percentage of connections that were to the same service, among the connections aggregated in count (23) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 31 | Diff Srv Rate | The percentage of connections that were to different services, among the connections aggregated in count (23) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 32 | Srv Diff Host Rate | The percentage of connections that were to different destination machines among the connections aggregated in srv\_count (24) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 33 | Dst Host Count | Number of connections having the same destination host IP address | Discrete | Integers | 0 - 255 |
| 34 | Dst Host Srv Count | Number of connections having the same port number | Discrete | Integers | 0 - 255 |
| 35 | Dst Host Same Srv Rate | The percentage of connections that were to different services, among the connections aggregated in dst\_host\_count (32) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 36 | Dst Host Diff Srv Rate | The percentage of connections that were to different services, among the connections aggregated in dst\_host\_count (32) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 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) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 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) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 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) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 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) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 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) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 42 | Class | Classification of the traffic input | Categorical | Strings |  |
| 43 | Difficulty Level | Difficulty level | Discrete | Integers | 0 - 21 |
| 44 | 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) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |

# Results and Discussion

A total of 25,192 data with 41 features was taken from the NSL-KDD dataset for training. The 3 symbolic features (protocol, service, flag) were expanded using 1-N encoding. Then the dataset is standardized with differencing with mean and dividing by standard deviation. Then only 10 features with high correlation with the intrusion column was chosen. The final data contains 93 features, it is subject to vary depending on encoding categories.

## Performance Metrics

The performance of the model is evaluated using the confusion matrix. The following factors are taken into account. True Positive (TP) denotes the correct classification of the attack. A False Positive (FP) is when a normal network is misclassified as an attack. A True Negative (NP) is a correctly classifying normal attack as normal, whereas a False Negative (FN) is the case when the attack is incorrectly identified as a normal network.

1. The accuracy is defined as the proportion of accurately predicted values to total number of test cases.

Accuracy = TP+TN/TP+TN+FP+FN 2

1. The precision is defined as the proportion of accurately predicted positive values to total number of predicted positive values.

Precision = TP/TP+FP 3

1. The recall is defined as the proportion of accurately predicted positive values to the total number of positive values.

Recall = TP/TP+FN 4

1. The Specificity is defined as the proportion of accurately predicted negative test results to the total number of all truly negative values.

Specificity = TN/TN+FP 5

1. F1-score metric combines the precision and recall of a classifier into a single measure. F1-score is calculated by taking harmonic mean of Precision and Recall.

F1-Score = 2\*(Precision\*Recall)/(Precision + Recall) 6

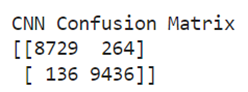
where,

* TP = number of true positives
* TN = number of true negatives
* FP = number of false positives
* FN = number of false negatives

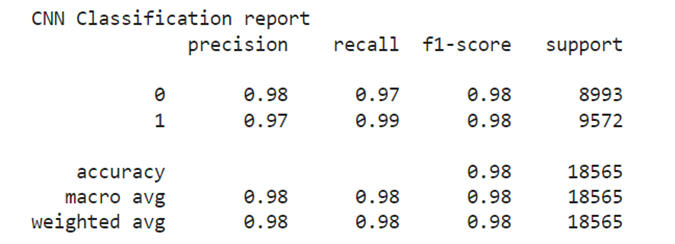
## Performance of Each model

**CNN**: The performance metrics of the CNN model is presented in table VI. The Classification report, Confusion matrix, Accuracy graph, Loss graph and ROC curve for the same are also shown in Fig 6, Fig 7, Fig 8, Fig 9 and Fig 10 respectively.

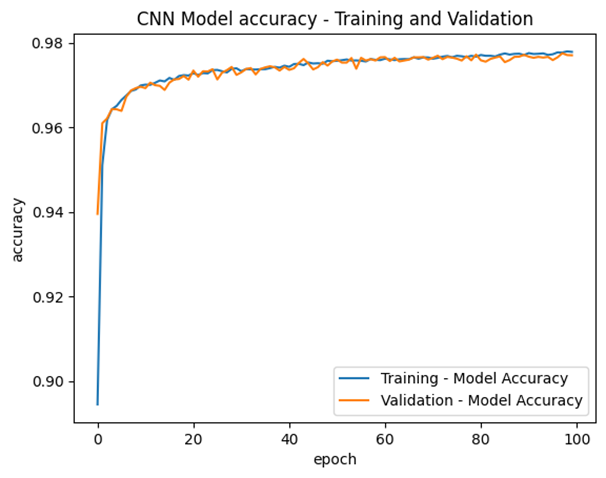
The CNN model has the highest accuracy score of 0.978, which suggests that it classified almost 98% of the instances correctly. The precision score of 0.973 indicates that out of all the instances classified as positive, 97% of them were actually positive. The recall/sensitivity score of 0.986 suggests that out of all the actual positive instances, the model was able to correctly identify almost 99% of them. The specificity score of 0.971 implies that out of all the actual negative instances, the model was able to correctly identify about 97% of them. The F1 score of 0.979 is a measure of the balance between precision and recall, and it is relatively high for this model. Overall, these metrics suggest that the CNN model performed very well in classifying both positive and negative instances, and it may be a suitable choice for this binary classification task.



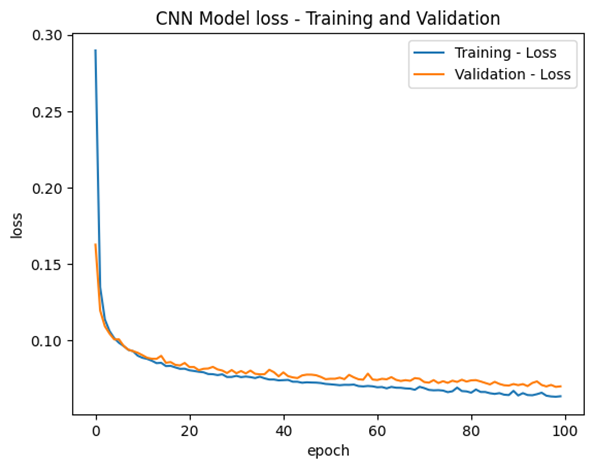
1. CNN Model Confusion Matrix



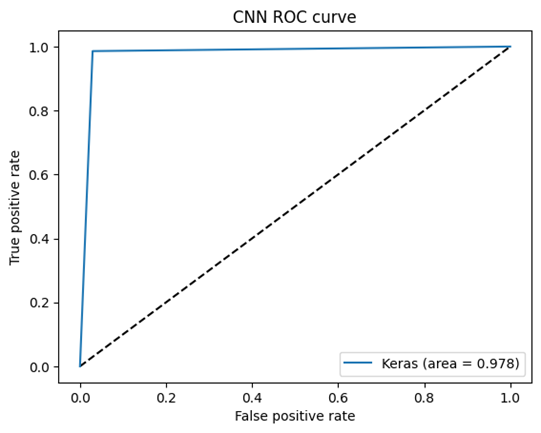
1. CNN Classification Report



1. CNN Model Accuracy graph



1. CNN Model Loss graph

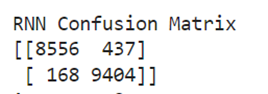


1. CNN ROC Curve
2. Results of CNN Model

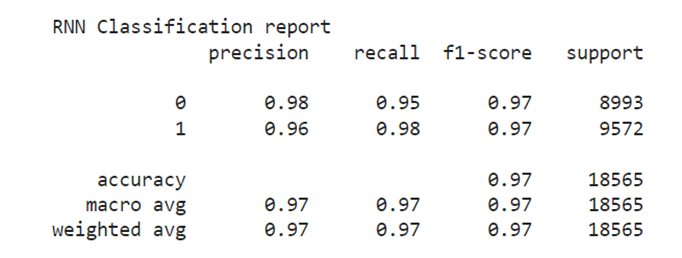
|  |  |
| --- | --- |
| Epoch | 100 |
| Batch size | 2785 |
| Accuracy | 0.9784540802585511 |
| Precision | 0.9727835051546392 |
| Recall | 0.9857918930213122 |
| Specificity | 0.9706438340931836 |
| F1 Score | 0.979244499792445 |

**RNN**:The performance metrics of the RNN model is presented in table VII. The Classification report, Confusion matrix, Accuracy graph, Loss graph and ROC curve for the same are also shown in Fig 11, Fig 12, Fig 13, Fig 14 and Fig 15 respectively.

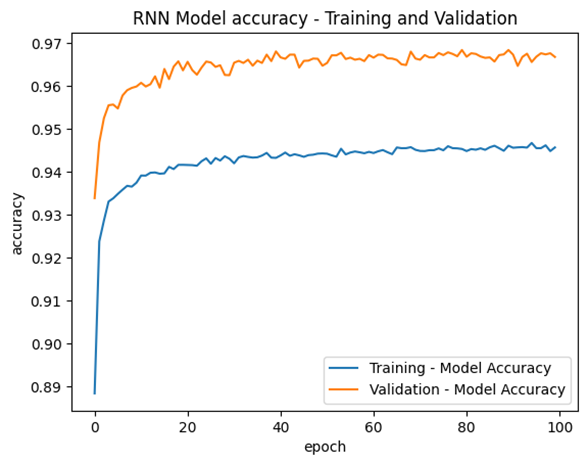
The RNN model has an accuracy score of 0.967, which is slightly lower than that of the CNN model. The precision score of 0.956 indicates that out of all the instances classified as positive, 96% of them were actually positive. The recall/sensitivity score of 0.982 suggests that out of all the actual positive instances, the model was able to correctly identify about 98% of them. The specificity score of 0.951 implies that out of all the actual negative instances, the model was able to correctly identify about 95% of them. The F1 score of 0.969 is relatively high for this model, indicating a good balance between precision and recall. Overall, these metrics suggest that the RNN model is also performing well, but it is slightly less accurate than the CNN model.



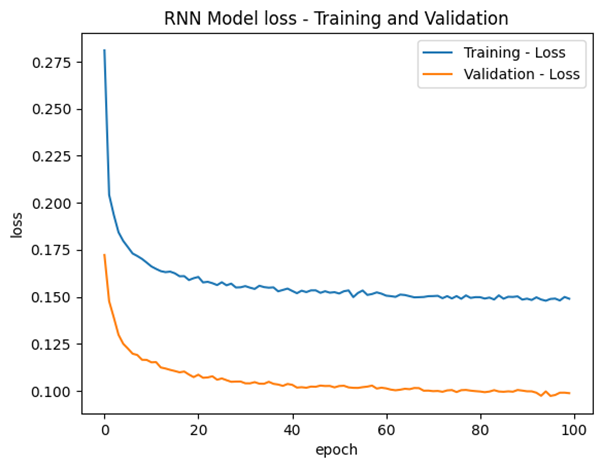
1. RNN Confusion Matrix



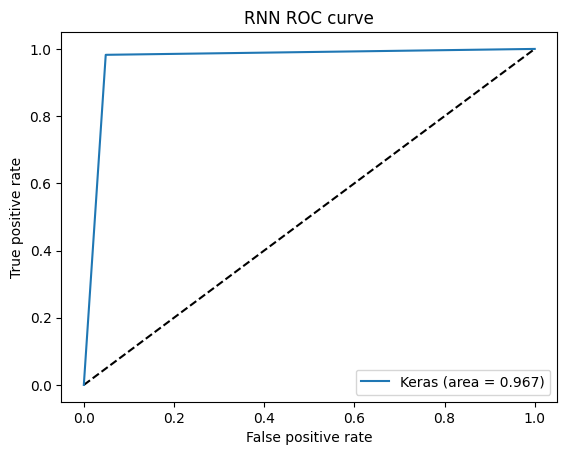
1. RNN Classification Report



1. RNN Model Accuracy



1. RNN Model Loss

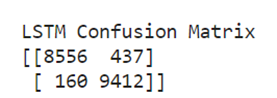


1. RNN ROC Curve
2. Results of RNN Model

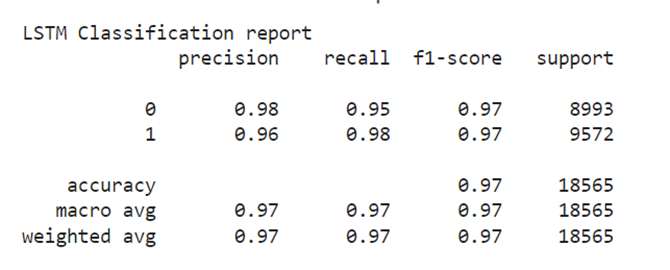
|  |  |
| --- | --- |
| Epoch | 100 |
| Batch size | 2785 |
| Accuracy | 0.9674117963910585 |
| Precision | 0.955593943704908 |
| Recall | 0.9824488090263268 |
| Specificity | 0.9514066496163683 |
| F1 Score | 0.9688353165404625 |

**LSTM**: The performance metrics of the LSTN model is presented in table VIII. The Classification report, Confusion matrix, Accuracy graph, Loss graph and ROC curve for the same are also shown in Fig 16, Fig 17, Fig 18, Fig 19 and Fig 20 respectively.

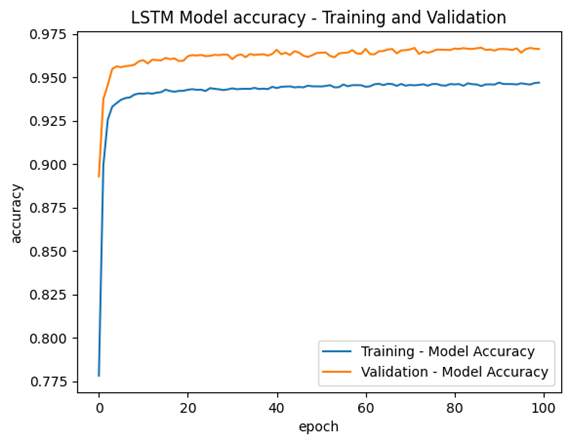
The LSTM model has an accuracy score of 0.968, which is similar to that of the RNN model. The precision score of 0.956 indicates that out of all the instances classified as positive, 96% of them were actually positive. The recall/sensitivity score of 0.983 suggests that out of all the actual positive instances, the model was able to correctly identify almost 98.5% of them. The specificity score of 0.951 implies that out of all the actual negative instances, the model was able to correctly identify about 95% of them. The F1 score of 0.969 is also similar to that of the RNN model, indicating a good balance between precision and recall. Overall, these metrics suggest that the LSTM model is also performing well, but its performance is very similar to that of the RNN model.

****

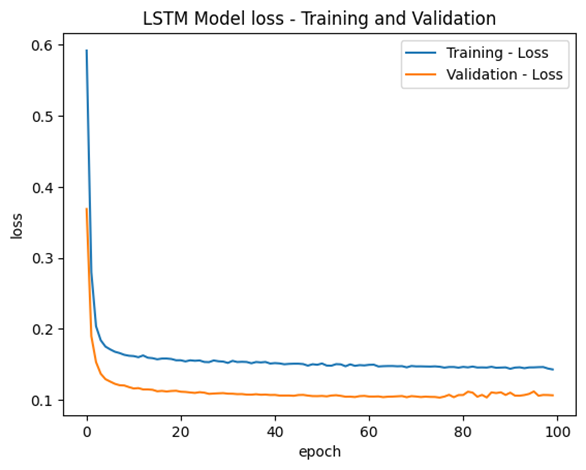
1. LSTM Confusion Matrix



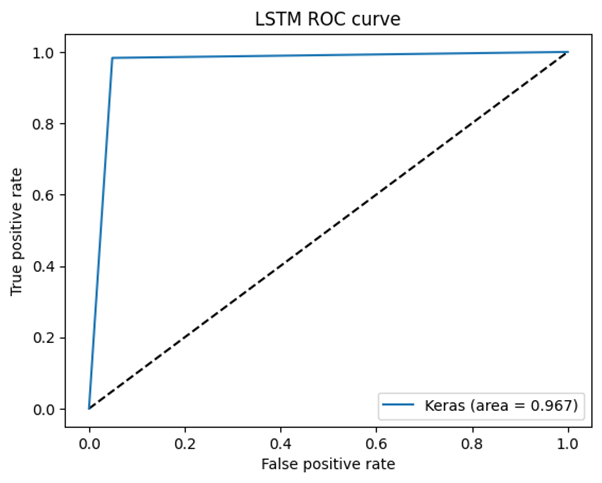
1. LSTM Classification Report



1. LSTM Model Accuracy



1. LSTM Model Loss graph

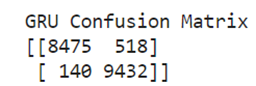


1. LSTM ROC Curve
2. Results of LSTM Model

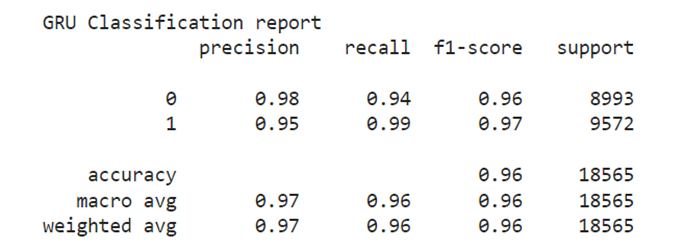
|  |  |
| --- | --- |
| Epoch | 100 |
| Batch size | 2785 |
| Accuracy | 0.9678427147858875 |
| Precision | 0.9556300131993096 |
| Recall | 0.9832845800250731 |
| Specificity | 0.9514066496163683 |
| F1 Score | 0.9692600792956079 |

**GRU**: The performance metrics of the GRU model is presented in table IX. The Classification report, Confusion matrix, Accuracy graph, Loss graph and ROC curve for the same are also shown in Fig 21, Fig 22, Fig 23, Fig 24 and Fig 25 respectively.

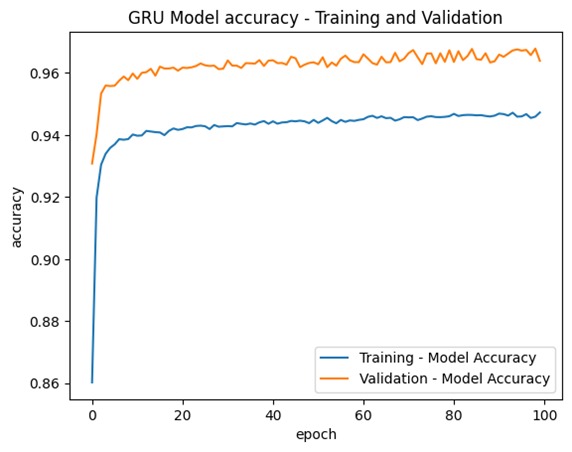
The GRU model has the lowest accuracy score of 0.965, which is slightly lower than that of the RNN and LSTM models. The precision score of 0.948 indicates that out of all the instances classified as positive, 94.8% of them were actually positive. The recall/sensitivity score of 0.985 suggests that out of all the actual positive instances, the model was able to correctly identify almost 99% of them. The specificity score of 0.942 implies that out of all the actual negative instances, the model was able to correctly identify about 94% of them. The F1 score of 0.966 is the lowest among the four models, indicating that there is some imbalance between precision and recall. Overall, these metrics suggest that the GRU model is performing well in identifying actual positive instances, but it has a relatively high false positive rate, which could impact its overall accuracy.



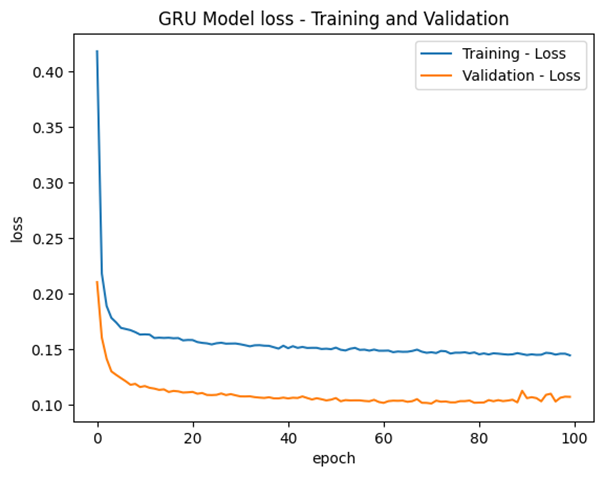
1. GRU Confusion Matrix



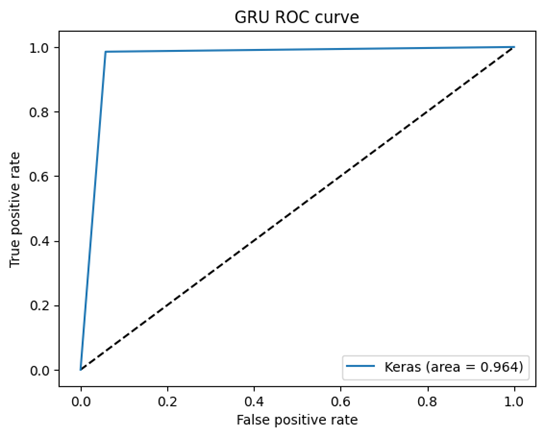
1. GRU Classification Report



1. GRU Accuracy Model



1. GRU Model Loss



1. GRU ROC Curve
2. Results of GRU Model

|  |  |
| --- | --- |
| Epoch | 100 |
| Batch size | 2785 |
| Accuracy | 0.9645569620253165 |
| Precision | 0.9479396984924623 |
| Recall | 0.985374007521939 |
| Specificity | 0.942399644167686 |
| F1 Score | 0.9662944370453848 |

## Results Comparision

Comparison of all 4 model’s Confusion matrix and performance metrics is summarized in Table X and Table XI. Fig 26 presents the comparison of various binary classification performance metrics using Bar Graph.

It can be interfered that, all four models are performing relatively well, with accuracy scores ranging from 0.965 to 0.978. The CNN model has the highest accuracy score, and the GRU model has the lowest accuracy score. However, the precision, recall, specificity, and F1 scores of the four models are all relatively close to each other. The CNN model has the highest F1 score of 0.979, indicating a good balance between precision and recall, while the GRU model has the lowest F1 score of 0.966, indicating some imbalance between precision and recall. Therefore, based on these metrics, the CNN model appears to be the best performing model, followed closely by the RNN and LSTM models, with the GRU model lagging behind. [25][26].

1. Results Comparision of all Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | TP | FP | FN | TN |
| CNN | 8729 | 264 | 136 | 9436 |
| RNN | 8556 | 437 | 168 | 9404 |
| LSTM | 8556 | 437 | 160 | 9412 |
| GRU | 8475 | 518 | 140 | 9432 |

1. Results Comapision Matrics of all Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CNN | RNN | LSTM | GRU |
| Epoch | 100 | 100 | 100 | 100 |
| Batch size | 2785 | 2785 | 2785 | 2785 |
| Accuracy | 0.9784 | 0.9674 | 0.9678 | 0.9645 |
| Precision | 0.9727 | 0.9555 | 0.9556 | 0.9479 |
| Recall | 0.9857 | 0.9824 | 0.9832 | 0.9853 |
| Specificity | 0.97064 | 0.9514 | 0.9514 | 0.9423 |
| F1 Score | 0.97924 | 0.9688 | 0.9692 | 0.9662 |



1. Comparision bargraph of all model metrics

# Conclusion and Future Scope

The research conducted on binary classification of intrusion detection using various deep learning models on the NSL-KDD dataset has yielded promising results. Among the four popular deep learning models used, namely CNN, RNN, LSTM, and GRU, the CNN model emerged as the best-performing model, closely followed by the RNN and LSTM models. The study's evaluation of these models based on performance metrics, such as accuracy, precision, recall, specificity, and F1 score, provides valuable insights into the effectiveness of each model for intrusion detection. This information can help researchers and scientists in selecting the most appropriate model for intrusion detection based on their specific needs and requirements.

The study's comparison of different deep learning models using the widely used NSL-KDD dataset provides a standard baseline for future research in intrusion detection. This information can guide researchers and scientists in selecting the most appropriate model for intrusion detection based on their specific requirements.

Future scope of this research work could involve exploring other deep learning architectures or hybrid models to improve the accuracy and performance of intrusion detection systems. Additionally, expanding the dataset to include more diverse and complex cyber-attacks could help to create more robust models for intrusion detection. Furthermore, exploring the use of transfer learning, data augmentation, and ensembling techniques could help to improve the performance of deep learning models for intrusion detection. Overall, this research work provides a solid foundation for further exploration and development of deep learning-based intrusion detection systems.

##### References

1. A. Alshamrani, S. Myneni, A. Chowdhary and D. Huang, "A Survey on Advanced Persistent Threats: Techniques, Solutions, Challenges, and Research Opportunities," in IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pp. 1851-1877, Secondquarter 2019, doi: 10.1109/COMST.2019.2891891
2. Atif Ahmad, Jeb Webb, Kevin C. Desouza, James Boorman, Strategically-motivated advanced persistent threat: Definition, process, tactics and a disinformation model of counterattack, Computers & Security, Volume 86, 2019, Pages 402-418, ISSN 0167-4048, doi: 10.1016/j.cose.2019.07.001.T.
3. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi and M. Ghogho, "Deep learning approach for Network Intrusion Detection in Software Defined Networking," 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), 2016, pp. 258-263, doi: 10.1109/WINCOM.2016.7777224.
4. J. Hassannataj Joloudari, M. Haderbadi, A. Mashmool, M. Ghasemigol, S. S. Band and A. Mosavi, "Early Detection of the Advanced Persistent Threat Attack Using Performance Analysis of Deep Learning," in IEEE Access, vol. 8, pp. 186125-186137, 2020, doi: 10.1109/ACCESS.2020.3029202.
5. Chu W-L, Lin C-J, Chang K-N. Detection and Classification of Advanced Persistent Threats and Attacks Using the Support Vector Machine. Applied Sciences. 2019; 9(21):4579. Doi: 10.3390/app9214579
6. H. N. Ooi and N. H. Ab Rahman, “A Comparative Study between Deep Learning Algorithm and Bayesian Network on Advanced Persistent Threat(APT) Attack Detection”., aitcs, vol. 2, no. 2, pp. 219–235, Nov. 2021.
7. A. Khalid, A. Zainal, M. A. Maarof and F. A. Ghaleb, "Advanced Persistent Threat Detection: A Survey," 2021 3rd International Cyber Resilience Conference (CRC), Langkawi Island, Malaysia, 2021, pp. 1-6, doi: 10.1109/CRC50527.2021.9392626.
8. Weina Niu, Jie Zhou, Yibin Zhao, Xiaosong Zhang, Yujie Peng, Cheng Huang, Uncovering APT malware traffic using deep learning combined with time sequence and association analysis, Computers & Security, Volume 120, 2022, 102809, ISSN 0167-4048, doi: 10.1016/j.cose.2022.102809.
9. Michael Zipperle, Florian Gottwalt, Elizabeth Chang, and Tharam Dillon. 2022. Provenance-based Intrusion Detection Systems: A Survey. ACM Comput. Surv. 55, 7, Article 135 (July 2023), 36 pages. https://doi.org/10.1145/3539605
10. T. Bodström and T. Hämäläinen, “A Novel Deep Learning Stack for APT Detection,” Applied Sciences, vol. 9, no. 6, p. 1055, Mar. 2019, doi: 10.3390/app9061055.
11. H. Hindy et al., "A Taxonomy of Network Threats and the Effect of Current Datasets on Intrusion Detection Systems," in IEEE Access, vol. 8, pp. 104650-104675, 2020, doi: 10.1109/ACCESS.2020.3000179.
12. HADI, MHMOOD, A Novel Approach to Network Intrusion Detection System Using Deep Learning for Sdn: Futuristic Approach (July 11, 2022). Available at SSRN: doi: 10.2139/ssrn.4160050
13. P. Singh, J. J. P, A. Pankaj and R. Mitra, "Edge-Detect: Edge-Centric Network Intrusion Detection using Deep Neural Network," 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), 2021, pp. 1-6, doi: 10.1109/CCNC49032.2021.9369469.
14. Dhanabal, L. and S. P. Shantharajah. “A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms.” (2015).
15. Sharma, S., Gigras, Y., Chhikara, R. and Dhull, A., 2019. Analysis of NSL KDD dataset using classification algorithms for intrusion detection system. Recent Patents on Engineering, 13(2), pp.142-147. 12. 10.2174/1872212112666180402122150.
16. R. Thomas and D. Pavithran, "A Survey of Intrusion Detection Models based on NSL-KDD Data Set," 2018 Fifth HCT Information Technology Trends (ITT), 2018, pp. 286-291, doi: 10.1109/CTIT.2018.8649498.
17. M. Srikanth Yadav. and R. Kalpana., "Data Preprocessing for Intrusion Detection System Using Encoding and Normalization Approaches," 2019 11th International Conference on Advanced Computing (ICoAC), 2019, pp. 265-269, doi: 10.1109/ICoAC48765.2019.246851.
18. F. Z. Belgrana, N. Benamrane, M. A. Hamaida, A. Mohamed Chaabani and A. Taleb-Ahmed, "Network Intrusion Detection System Using Neural Network and Condensed Nearest Neighbors with Selection of NSL-KDD Influencing Features," 2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS), 2021, pp. 23-29, doi: 10.1109/IoTaIS50849.2021.9359689.
19. Kesavulu Reddy, Dr. (2013). Neural Networks for Intrusion Detection and Its Applications. Lecture Notes in Engineering and Computer Science. 2. 1210-1214.
20. Yalei Ding and Yuqing Zhai. 2018. Intrusion Detection System for NSL-KDD Dataset Using Convolutional Neural Networks. In Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence (CSAI '18). Association for Computing Machinery, New York, NY, USA, 81–85. doi : 10.1145/3297156.3297230
21. Ravi, Vinayakumar & Kp, Soman & Poornachandran, Prabaharan. (2017). Applying convolutional neural network for network intrusion detection. 1222-1228. 10.1109/ICACCI.2017.8126009.
22. X. Zhang, J. Ran and J. Mi, "An Intrusion Detection System Based on Convolutional Neural Network for Imbalanced Network Traffic," 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), 2019, pp. 456-460, doi: 10.1109/ICCSNT47585.2019.8962490.
23. Hsu, CM., Hsieh, HY., Prakosa, S.W., Azhari, M.Z., Leu, JS. (2019). Using Long-Short-Term Memory Based Convolutional Neural Networks for Network Intrusion Detection. In: Chen, JL., Pang, AC., Deng, DJ., Lin, CC. (eds) Wireless Internet. WICON 2018. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 264. Springer, Cham. https://doi.org/10.1007/978-3-030-06158-6\_9
24. Sharma, Srishti & Gigras, Yogita & Chhikara, Rita & Dhull, Anuradha. (2018). Analysis of NSL KDD Dataset Using Classification Algorithms for Intrusion Detection System. Recent Patents on Engineering. 12. 10.2174/1872212112666180402122150.
25. T. Su, H. Sun, J. Zhu, S. Wang and Y. Li, "BAT: Deep Learning Methods on Network Intrusion Detection Using NSL-KDD Dataset," in IEEE Access, vol. 8, pp. 29575-29585, 2020, doi: 10.1109/ACCESS.2020.2972627.
26. Ding, Yalei & Zhai, Yuqing. (2018). Intrusion Detection System for NSL-KDD Dataset Using Convolutional Neural Networks. CSAI '18: Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence. 81-85. 10.1145/3297156.3297230.
27. S. K. Sahu, S. Sarangi and S. K. Jena, "A detail analysis on intrusion detection datasets," 2014 IEEE International Advance Computing Conference (IACC), 2014, pp. 1348-1353, doi: 10.1109/IAdCC.2014.6779523.
28. Ghurab, Mossa & Gaphari, Ghaleb & Alshami, Faisal & Alshamy, Reem & Othman, Suad. (2021). A Detailed Analysis of Benchmark Datasets for Network Intrusion Detection System.
29. I. Abrar, Z. Ayub, F. Masoodi and A. M. Bamhdi, "A Machine Learning Approach for Intrusion Detection System on NSL-KDD Dataset," 2020 International Conference on Smart Electronics and Communication (ICOSEC), 2020, pp. 919-924, doi: 10.1109/ICOSEC49089.2020.9215232.
30. R. Feng, S. Chen, X. Xie, G. Meng, S. -W. Lin and Y. Liu, "A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices," in IEEE Transactions on Information Forensics and Security, vol. 16, pp. 1563-1578, 2021, doi: 10.1109/TIFS.2020.3025436.
31. Wang, Yajing & Ma, Juan & Sharma, Ashutosh & Singh, Pradeep & Gaba, Gurjot & Masud, Mehedi & Baz, Mohammed. (2021). An Exhaustive Research on the Application of Intrusion Detection Technology in Computer Network Security in Sensor Networks. Journal of Sensors. 2021. 1-11. 10.1155/2021/5558860.
32. Sarika Choudhary, Nishtha Kesswani, Analysis of KDD-Cup’99, NSL-KDD and UNSW-NB15 Datasets using Deep Learning in IoT, Procedia Computer Science, Volume 167, 2020, Pages 1561-1573, ISSN 1877-0509, doi: 10.1016/j.procs.2020.03.367.
33. Dorogovs, Pjotrs & Borisov, Arkady & Romanovs, Andrejs. (2011). Building an Intrusion Detection System for IT Security Based on Data Mining Techniques. Scientific Journal of Riga Technical University. Computer Sciences. 45. 10.2478/v10143-011-0040-3