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

1Dharaneish V C, 2Nithin Kumar, 3Hari Varsha  
1[dharaneishvc@gmail.com](mailto:dharaneishvc@gmail.com), 2[itsnithinkumar34@gmail.com](mailto:itsnithinkumar34@gmail.com), 3[v.harivarsha@gmail.com](mailto:v.harivarsha@gmail.com)

1,2 Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, Coimbatore – 641112, India

3 Department of Electrical and Electronics Engineering, Amrita Vishwa Vidyapeetham, Coimbatore - 641112, India

## **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.

**Key Words**: 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)

# **1. INTRODUCTION**

An Advanced Persistent Threat (APT) is a broad term used to describe an attack carried by a group often nation states involved in war or state-sponsored organizations, in which a hacker acquires unauthorized access to a computer network and evades detection for a period of time. APT is one of the major information security threats that industry is currently facing. The attack generally targets the private organization’s sensitive data and exfiltrate data to remote hosts. These are carried out by the most skilled and well-funded attackers. The attacks which led to 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 are APT attacks.

It is challenging to detect APT attacks using conventional techniques when they occur in a dynamic and complex infrastructure like cloud. It is difficult to identify this kind of attack because of the APT attack's long-lasting nature on the network and the possibility that the system would crash owing to the enormous traffic. Existing Intrusion Detection System solutions are unable to identify APTs because they work to maintain their anonymity and frequently employ Zero-Day attack, a type of cyberattack that takes use of a potentially dangerous software security flaw that the developer may be ignorant about. For many years, the majority of these attacks go unreported like the Red October APT attack which has been operating for more than five years.

When the APT is effectively organized and prepared to attack, it is regarded as being too late to create the defense in different phases, particularly in the last step. In order to detect APT-type attacks and defend against them before exfiltration is carried out, network intrusion systems using new Deep learning techniques and relevant analytical tools must be developed in the context of such persistent attacks. The network intrusion detector is a predictive model that distinguish between intrusions or attacks and normal connections. Due to its capacity to thoroughly analyze network data and produce the feature vector itself automatically, deep learning is preferred. Deep Learning algorithms greatly improve the performance by producing higher detection rate and lower false alarm rate.[1]

To identify cyberattacks on hosts and network systems, both straightforward and sophisticated neural network models have been built. Due to its capacity to examine in depth the computer process that replicates the normal activity of the human brain, deep learning is highly preferred.[2][3]

Further, this paper is organized as Related works in which a literature survey of existing intrusion detection systems is done, Materials and Methods section which explains the entire process of Data preprocessing, transformation and Deep learning methods, Experiment and Analysis section which includes study of dataset and the methods used and Results Conclusion section which is about accuracy and prediction metrics and future scope of the work.

# **2. RELATED WORKS**

Currently, APT attacks can be detected using tools like User and Entity Behavior 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]

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.

The work by Hanan et al.[7] 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.[8] 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.[9] 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 behavior 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.

# **3. MATERIALS AND METHODS**

# **3.1 PROPOSED WORK FLOW**

A deep-learning based classifier model to identify network intrusion detection is the goal of the presented work. In order to distinguish between normal connections and intrusions, the system uses artificial neural network to train itself on the patterns of anomalies. The strategy also aims to reduce the false alarm rate. The strategy is adaptable to new patterns of intrusion and changes in the attacker’s strategy and behavior that may occur over the course of time. The suggested method uses a deep Neural network model that was trained on the NSL-KDD dataset. It outputs a result of 0 or 1, with 1 signifying an intruder and 0 signifying a typical user.

# **3.1 DATA SET SELECTION**

Only when a useful data collection is available, a good intelligent intrusion detection system be developed. An intrusion detection system can only be trained and tested with a data set that contains a large volume of high-quality data that resembles real-time events. NSL-KDD dataset is used for this work over its successor the standard KDD dataset as it is refined of the former and does not include redundant records in the train set and no duplicate records in the proposed test sets and many such advantages. The NSL-KDD dataset is the best potential data set to simulate and test the performance of Intrusion Detection System, according to numerous research and analyses. Hence, The NSS-KDD dataset available in the University of new Brunkswick(https://www.unb.ca/cic/datasets/nsl.html) is utilized since it is useful to the system, however pre-processing is necessary.[10][11]

# **3.2 DATA PREPROCESSING**

The objective of data pre-processing is to analyze, filter, transform, and encode data so that the Artificial Neural Network algorithm can understand and work with the processed output. The presence of any unclean data like missing attributes, attribute values, containing noise or outliers, and duplicate or wrong data will degrade the quality of the Deep learning classifier results. So, it is important to manipulate or transform the raw data in a useful and efficient format before it is used in Machine learning to ensure or enhance performance. [12][13]

## 3.2.1 DATA CLEANING

Missing values in a data are a problem since they can often skew the results, depending on their type. This means that because the data came from an unrepresentative sample, the findings could not be generalizable to situations outside of this study. So, in order to remove missing values, rows with more than 25 missing features are deleted. Since only a very few of rows contains more than 25 features missing, this is efficient and causes no additional bias. And for rows with few missing values, categorical features are replaced with mode and numerical data is replaced with mean.

## 3.2.2 DATA TRANSFORMATION

Before feeding the dataset into Deep learning model, data transformation is a crucial data preprocessing technique that must be applied to the data in order to produce patterns that are simpler to comprehend. Data transformation transforms the data into clean, useable data by altering its format, structure, or values. As the range of raw data values ​​varies widely, some of the algorithms which works based on Euclidean distance, the objective functions will not perform well without the feature scaling. Hence, the data value of all numerical are scaled by subtracting the mean and scaling to unit variance.

Here, the standard score of a sample x is calculated as:

z = (x - u) / s 🡪 (1)

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

For this feature scaling, StandardScaler class of sklearn.preprocessing library is used.[10]

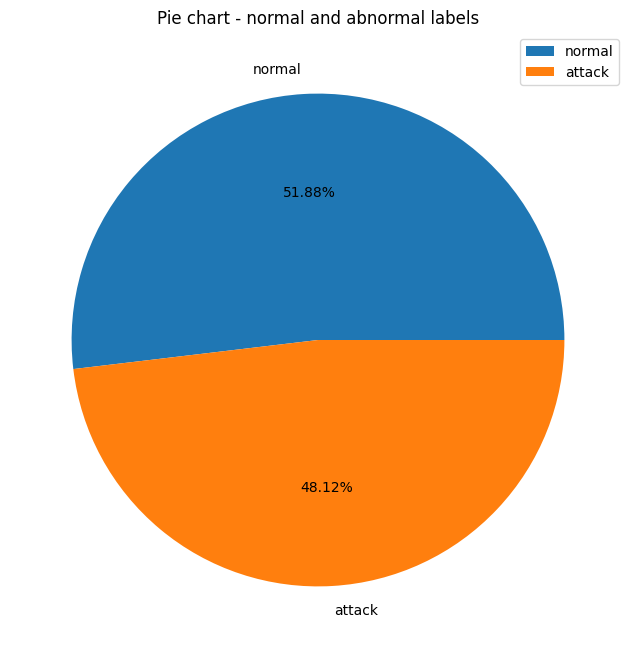
## 3.2.2.1 CONCEPT HIERARCHY GENERATION:

For sake of easy modelling of deep learning and depending on the prediction feature, properties can be transformed from lower to higher in the hierarchy. Since the model aims to predict only the class of attack, in the dataset, from attribute label, which contain sub-class of attacks or normal traffic, it can be converted into either of 4 attack class or normal.

## 3.2.2.2 ENCODING CATEGORICAL VALUES

Categorical data is statistical data consisting of categorical data variables converted into categories. Since most Deep learning models works well on math and numbers, but if the dataset has a categorical variable, it can’t be used to build the model. Therefore, it is necessary to encode these categorical variables as numbers. With these numbers, the deep learning model can assume that there is some correlation between these variables that will produce false results. So, to get rid of this problem a dummy encoding is used. In the dataset, Data Attributes ‘protocol\_type', 'service', 'flag’ are one-hot encoded using Dummy variables.

## 3.2.3 FEATURE SELECTION

Only a few features in the dataset are useful for generating the Deep learning classifier model, and the rest are either redundant or unnecessary. If we use all of these redundant and irrelevant variables in the model, it may have a negative impact on the model's overall performance and accuracy. So, the Pearson’s correlation coefficient for all columns is calculated and only attributes which have more than 0.5 correlation with encoded attack label attribute is selected for further Deep Learning development model.

## 3.2.4 DATASET SPLIT

The dataset KDDTrain and KDDTest were merged before pre-processing steps. The total no of rows after merging is 148517. The dataset is split into 80:10:10 train, validation and test ratio randomly using train\_test\_split function in sklearn. The size of Train, validation and test dataset is 111387, 18565, 18565 respectively. Test dataset is used to evaluate results using various metrics.

The total no of Normal and Abnormal labels in the dataset is equally distributed with normal 77054 rows of normal class and 71463 rows of attack class. Fig 1 shows the pie chart of distribution of Normal and attack class.

# **3.3 ARTIFICIAL NEURAL NETWORKS**

A neural network is a collection of algorithms that attempts to identify underlying links in a set of data using techniques that are modelled after the biological neural networks seen in animal brains. Their structure and nomenclature are modelled after the human brain, mirroring the communication between organic neurons. In this context, neural networks are systems of neurons that can be either organic or synthetic in origin. It will make computer programmes to recognise patterns and solve common problems in the fields of Artificial Intelligence, Machine learning, and Deep learning.

Deep learning techniques are based on neural networks, commonly referred to as artificial neural networks (ANNs), which are a subset of machine learning. A node layer of an artificial neural network (ANN) consists of an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, is connected to others and has a weight and threshold that go along with it. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's uppermost layer. Otherwise, no data is transmitted to the network's next tier. [14][15]

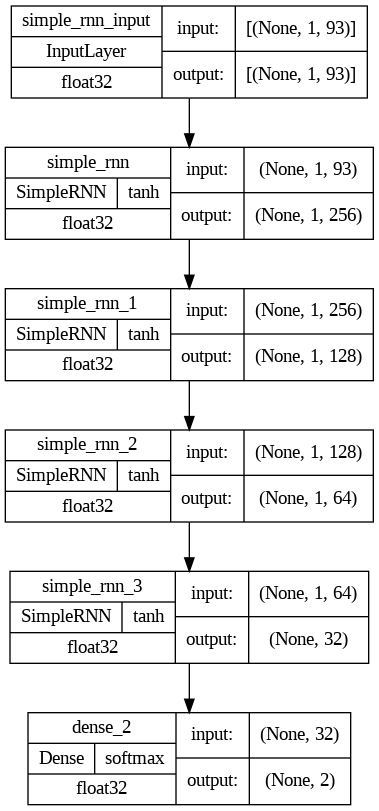
## 3.3.1 CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional neural networks (CNNs, or ConvNets) are a type of artificial neural network (ANN) used most frequently in deep learning to interpret visual data, recognition of images and videos, recommender systems, classification and segmentation of images, analysis of images, natural language processing, brain-computer interfaces, and time series analysis of financial data. Based on the shared-weight architecture of the convolution kernels or filters that slide along input features and produce translation-equivariant responses known as feature maps.

Fig 1 shows the architecture of the CNN model used. It has total of 9 layers - Input layer, 4 1D convolution layers, Max pooling, Flattening layer, Dense layer. [16][17][18]

Figure 1 Architecture of CNN

## 3.3.2 RECURRENT NEURAL NETWORK (RNN)

A recurrent neural network (RNN) is a type of artificial neural network in which connections between nodes can form a cycle, allowing the output of some nodes to influence the input received by other nodes in the same network. It can display temporal dynamic behaviour as a result of this. RNNs, which are derived from feedforward neural networks, may process input sequences of different lengths by using their internal state (memory).

RNN based deep learning is used in well-known products like Siri, voice search, and Google Translate. They are frequently employed for ordinal or temporal issues, such as language translation, natural language processing (NLP), speech recognition, and image captioning. Recurrent neural networks (RNNs) use training data to learn, just like feedforward and convolutional neural networks (CNNs) do. They stand out due to their "memory," which allows them to affect the current input and output by using data from previous inputs. Recurrent neural networks' outputs are dependent on the previous parts in the sequence, unlike typical deep neural networks, which presume that inputs and outputs are independent of one another. Unidirectional recurrent neural networks are unable to take into account future events in their forecasts, despite the fact that they would be useful in deciding the output of a particular sequence.

Fig 2 shows the architecture of the RNN model used. It has total of 6 layers - Input layer, 4 Simple RNN layers and Dense layer.

Figure 2 Architecture of Simple RNN

## 3.3.3 LONG SHORT-TERM MEMORY (LSTM)

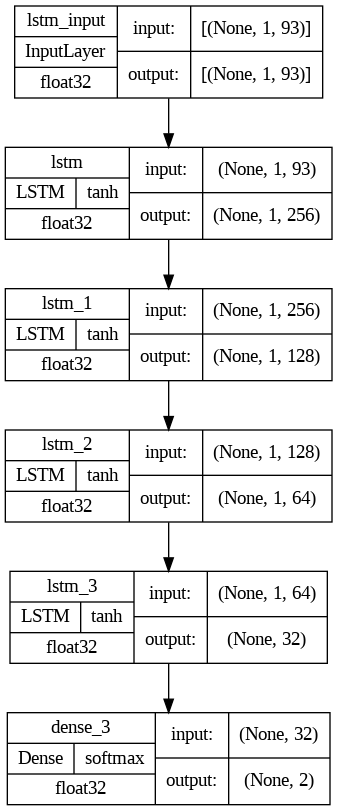
Long short-term memory (LSTM) is a type of artificial neural network used in artificial intelligence and deep learning. LSTM features feedback connections as opposed to typical feedforward neural networks. Such a recurrent neural network may process complete data sequences in addition to single data points.

Figure 3 Architecture of LSTM

LSTM is similar to Recurrent neural networks (RNN). The output from the previous phase is sent into the current step of an RNN as input. LSTM was later developed by Hochreiter & Schmidhuber which addressed the issue of long-term RNN dependency, in which the RNN can predict words from current data but cannot predict words held in long-term memory. RNN's performance becomes less effective as the gap length increases. By default, LSTM can save the data for a very long time. It is utilised for time-series data processing, forecasting, and classification. We have fed our data pre-processed into LSTM model. [19]

Fig 3 shows the architecture of the LSTM model used. It has total of 6 layers - Input layer, 4 LSTM layers and Dense layer.

## 3.3.4 GATED RECURRENT UNIT (GRU)

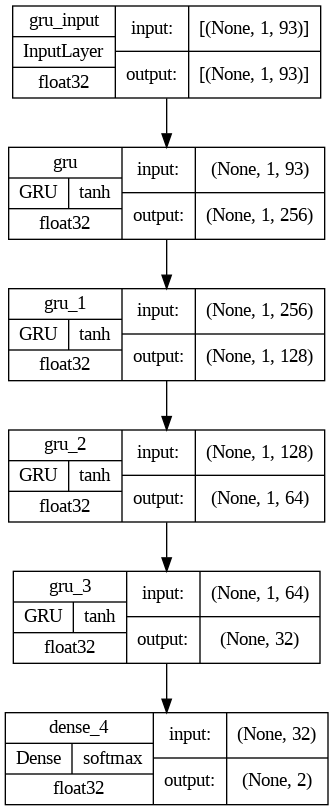
Gated Recurrent Unit (GRU), is an improved model over the regular Recurrent Neural Network (RNN) by incorporating a gating mechanism. GRU is similar to an LSTM with a forget gate, but it has fewer parameters as it doesn't have an output gate. GRU's performance on few tasks like polyphonic music modelling, speech signal modelling and natural language processing was found to be similar to that of LSTM. On some smaller, less frequent datasets, GRU has been proven to perform better.

Figure 4 Architecture of GRU

Fig 4 shows the architecture of the GRU model used. It has total of 7 layers - Input layer, 4 GRU layers and Dense layer.

# 4. EXPERIMENTS AND ANALYSIS

## 4.2 EXPERIMENTAL SETUP

|  |  |
| --- | --- |
| Numpy | Pandas |
| Matplotlib | Seaborn |
| keras | sklearn |
| tensorflow | visualkeras |
| pydot | graphviz |
| pydotplus | sys |
| Table1: Python Libraries used | |

**Python Libraries for Data Preprocessing:**

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.

# **4.2 STUDY OF KDD NSL DATASET**

# 4.2.1 DATASET DESCRIPTION:

# NSL-KDD dataset is used for The Third International Knowledge Discovery and Data Mining Tools Competition, which took place in conjunction with KDD-99, The Fifth International Conference on Knowledge Discovery and Data Mining. The task for the competition was to create a network intrusion detector, a predictive model capable of distinguishing between "bad" connections, known as intrusions or attacks, and "good" normal connections. This database contains a standard set of auditable data, including a wide range of intrusions simulated in a military network environment.

## 4.2.2 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.

# 4.2.3 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 can be seen below:

|  |  |  |
| --- | --- | --- |
| **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) |
| Table 2: Classes of attacks in NSL KDD dataset | | |

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:

|  |  |  |
| --- | --- | --- |
| **Classes** | **Sub-Classes** | **Total Count** |
| DoS | apache2, back, land, Neptune, mailbomb, pod, processtable, smurf, teardrop, udpstorm, worm | 11 |
| Probe | 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 |
| Table 3: Sub-Classes of attacks in NSL KDD dataset | | |

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.

## 4.2.4 CLASS LEVEL DETAILS:

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

|  |  |  |
| --- | --- | --- |
| **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 |
| Table 4: Classification of features in NSL KDD dataset | | |

## 4.2.5 FEATURES 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | **Feature Name** | **Description** | **Type** | **Value Type** | **Ranges (Between both train and test)** |
| 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 (Dataset contains ‘2’ value) | Integers | 0 - 2 |
| 16 | Num Root | Number of "root'' accesses or number of operations performed as a root in the connection | Continuous | Integers | 0 - 7468 |
| 17 | Num File Creations | Number of file creation operations in the connection | Continuous | Integers | 0 - 100 |
| 18 | Num Shells | Number of shell prompts | Continuous | Integers | 0 - 2 |
| 19 | Num Access Files | Number of operations on access control files | Continuous | Integers | 0 - 9 |
| 20 | Num Outbound Cmds | Number of outbound commands in an ftp session | Continuous | Integers | { 0 } |
| 21 | Is Hot Logins | 1 if the login belongs to the "hot'' list i.e., root or admin; else 0 | Binary | Integers | { 0 , 1 } |
| 22 | Is Guest Login | 1 if the login is a "guest'' login; 0 otherwise | Binary | Integers | { 0 , 1 } |
| 23 | Count | Number of connections to the same destination host as the current connection in the past two seconds | Discrete | Integers | 0 - 511 |
| 24 | Srv Count | Number of connections to the same service (port number) as the current connection in the past two seconds | Discrete | Integers | 0 - 511 |
| 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) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 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) | Discrete | Floats (hundredths of a decimal) | 0 - 1 |
| 27 | 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 |
| 28 | 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 |
| 29 | 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 |
| 30 | 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 |
| 31 | 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 |
| 32 | Dst Host Count | Number of connections having the same destination host IP address | Discrete | Integers | 0 - 255 |
| 33 | Dst Host Srv Count | Number of connections having the same port number | Discrete | Integers | 0 - 255 |
| 34 | 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 |
| 35 | 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 |
| 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) | 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 |
| Table 4: Description of features in NSL KDD dataset | | | | | |

# **4.3 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 89 features, it is subject to vary depending on encoding categories.

## **4.3.1 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.

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

Accuracy = TP + TN / TP + TN + FP + FN 🡪 (1)

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

Precision = TP / TP + FP 🡪 (2)

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

Recall = TP / TP + FN 🡪(3)

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 🡪 (4)

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)] 🡪 (5)

where,  
TP = number of true positives   
TN = number of true negatives  
FP = number of false positives  
FN = number of false negatives

## **4.3.2. PERFORMANCE OF EACH MODEL**

## 4.3.2.1 CNN

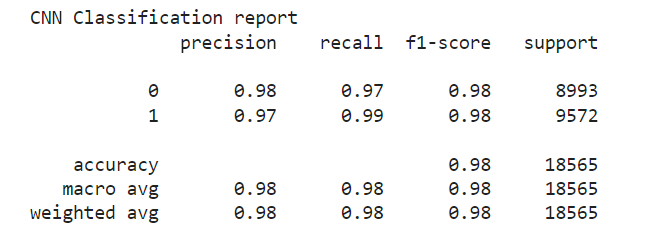
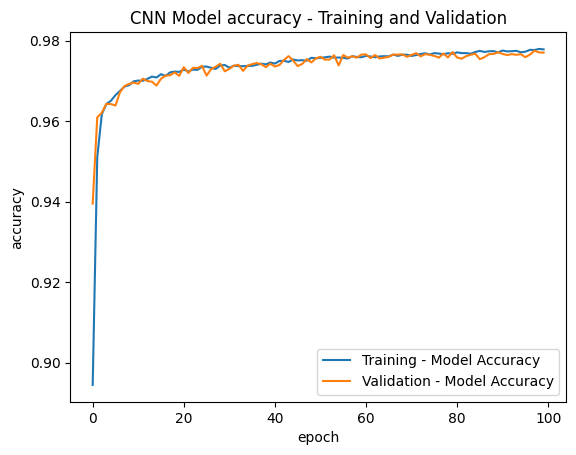
Results of Convolutional Neural Networks (CNN) is best among all the 4 models. 18565 test values have been passed and it classified 18165 of it into correct class and misclassified the rest 370 rows. The Classification report is shown in Fig 5 and Confusion matrix in Fig 6.

Figure 5 CNN Classification Report

Figure 5 CNN Classification report

Figure 5 CNN Classification report

Various metrics of the classification report is displayed in the Table. It has accuracy of 0.978, F1 score of 0.979 and are under ROC curve as 0.978 which means the model is good.

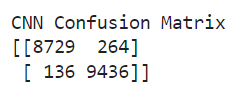


Figure 6 CNN Confusion Matrix

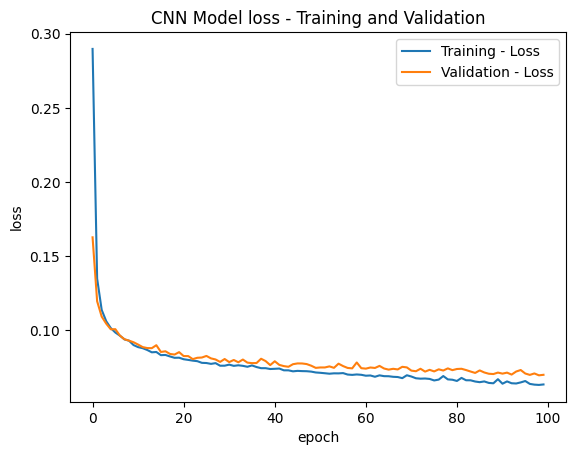


Figure 8 CNN Model Loss Graph

Figure 7 CNN Model Accuracy graph

|  |  |
| --- | --- |
| Epoch | 100 |
| Batch size | 2785 |
| Accuracy | 0.9784540802585511 |
| Precision | 0.9727835051546392 |
| Recall | 0.9857918930213122 |
| Specificity | 0.9706438340931836 |
| F1 Score | 0.979244499792445 |
| Table 5: Results of CNN Model | |

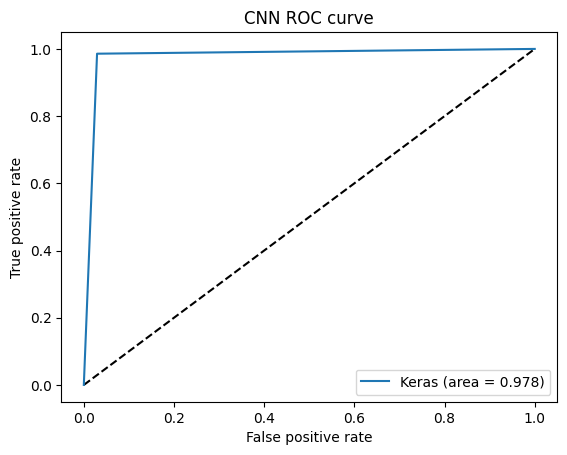


Figure 9 CNN ROC Curve

## 4.3.2.2 RNN

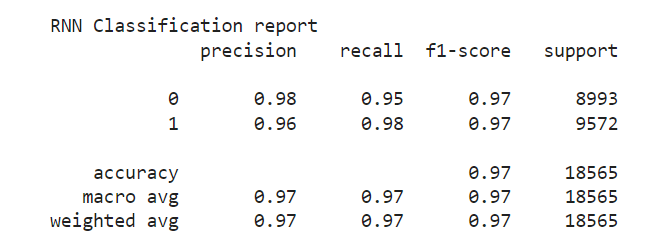
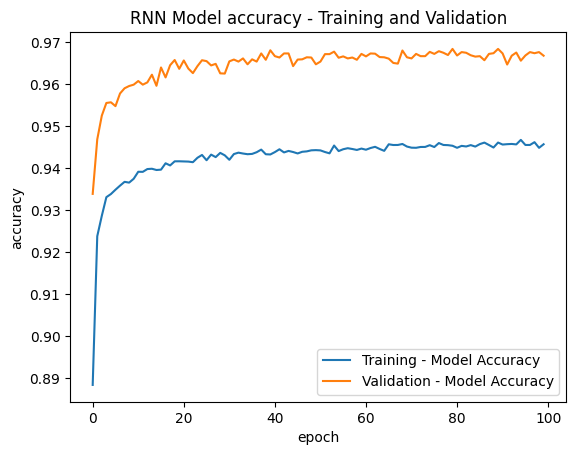
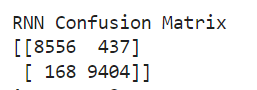
Results of Recurrent Neural Network (RNN) is also better among comparing to other 2 models. 18565 test values have been passed and it classified 17960 of it into correct class and misclassified the rest 605 rows. The Classification report is shown in Figure 3. Various metrics of the classification report is displayed in the Table. It has accuracy of 0.967, F1 score of 0.968 and are under ROC curve as 0.967 which means the model is good

Figure 10 RNN Classification Report

Figure 11 RNN Confusion Matrix



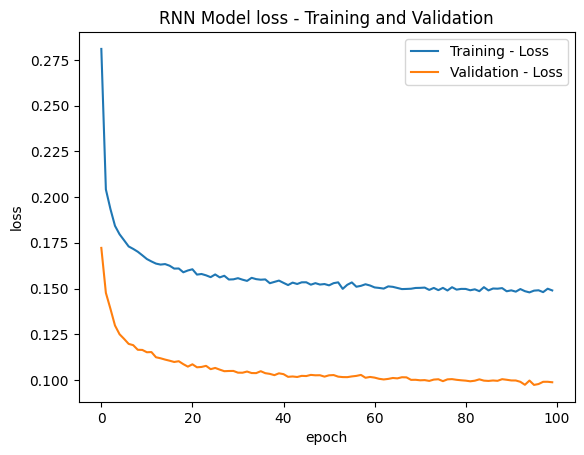
.

Figure 12 RNN Model Accuracy Graph

Figure 13 RNN Model Loss Graph

|  |  |
| --- | --- |
| Epoch | 100 |
| Batch size | 2785 |
| Accuracy | 0.9674117963910585 |
| Precision | 0.955593943704908 |
| Recall | 0.9824488090263268 |
| Specificity | 0.9514066496163683 |
| F1 Score | 0.9688353165404625 |
| Table 6: Results of RNN | |

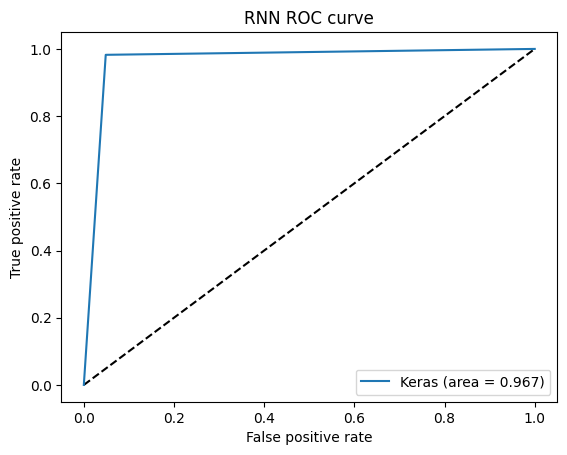


Figure 14 RNN ROC Curve

## 4.3.2.3 LSTM

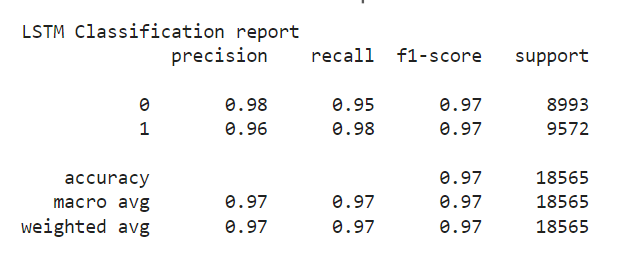
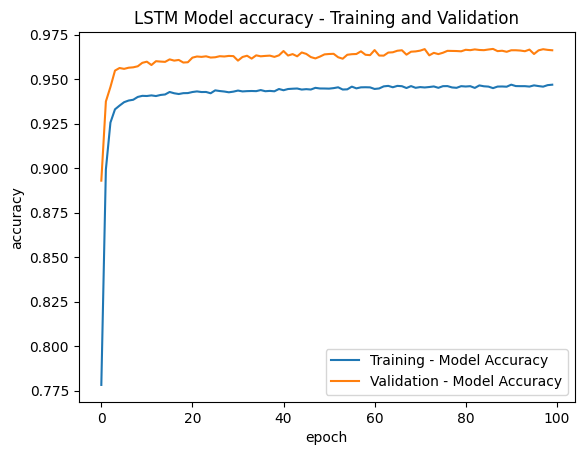
Results of Long Short-term Memory Networks (LSTM) is also as good as Simple RNN but worser than and CNN. 18565 test values have been passed and it classified 17968 of it into correct class and misclassified the rest 597 rows. The Classification report is shown in Figure 5.

Figure 15 LSTM Classification report

Various metrics of the classification report is displayed in the Table. It has accuracy of 0.967, F1 score of 0.969 and are under ROC curve as 0.967 which means the model is good.

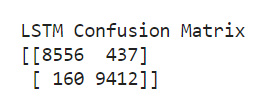


Figure 16 LSTM Confusion Matrix

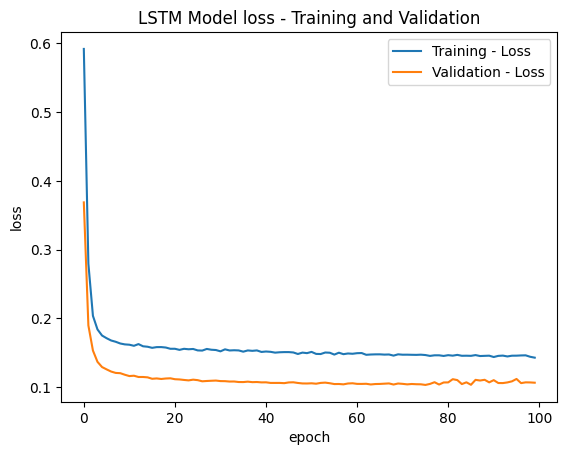


Figure 17 LSTM Model Accuracy Graph

Figure 18 LSTM Model Loss Graph

|  |  |
| --- | --- |
| Epoch | 100 |
| Batch size | 2785 |
| Accuracy | 0.9678427147858875 |
| Precision | 0.9556300131993096 |
| Recall | 0.9832845800250731 |
| Specificity | 0.9514066496163683 |
| F1 Score | 0.9692600792956079 |
| Table 7: Results of LSTM model | |

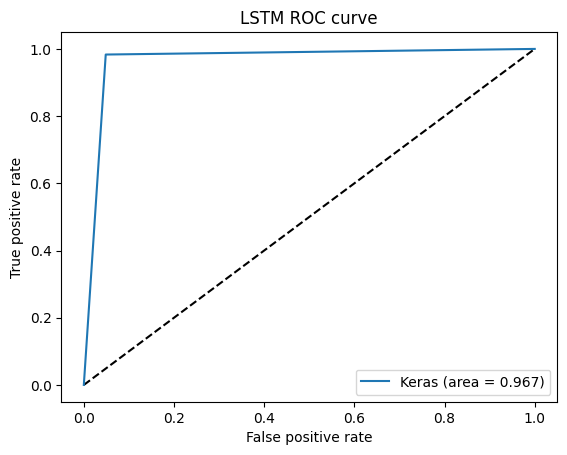


Figure 19 LSTM ROC Curve

## 4.3.2.4 GRU

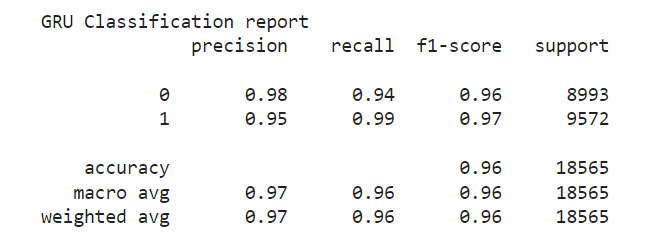
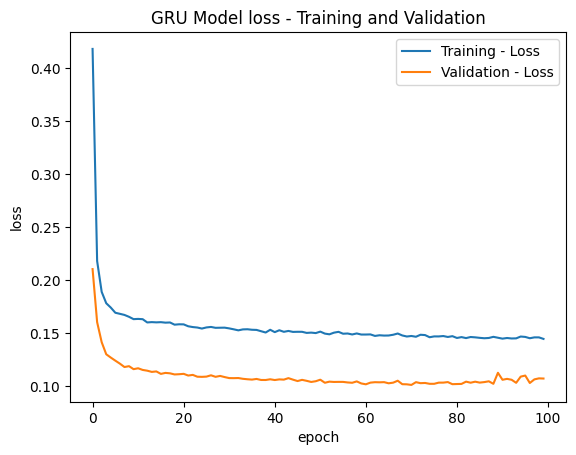
Results of Gated Recurrent Unit (GRU) is also better among comparing to LSTM model. 18565 test values have been passed and it classified 17907 of it into correct class and misclassified the rest 658 rows. The Classification report is shown in Figure 7. Various metrics of the classification report is displayed in the Table. It has accuracy of 0.964, F1 score of 0.966 and are under ROC curve as 0.964 which means the model is good.

Figure 20 GRU Classification Report

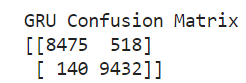


Figure 21 GRU Confusion Matrix

Figure 22 GRU Model Accuracy Graph

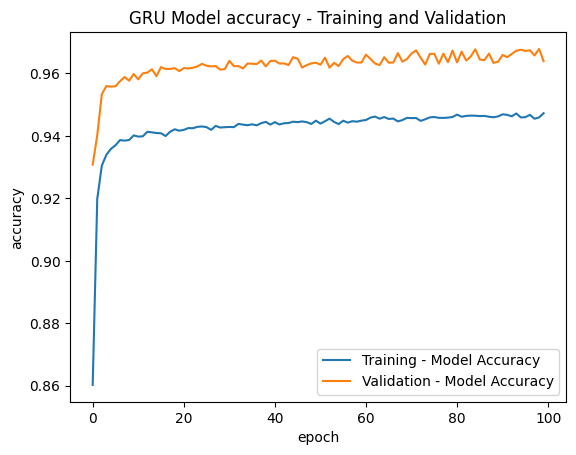


Figure 23 GRU Model Loss Graph

|  |  |
| --- | --- |
| Epoch | 100 |
| Batch size | 2785 |
| Accuracy | 0.9645569620253165 |
| Precision | 0.9479396984924623 |
| Recall | 0.985374007521939 |
| Specificity | 0.942399644167686 |
| F1 Score | 0.9662944370453848 |
| Table 8: Results of GRU Model | |

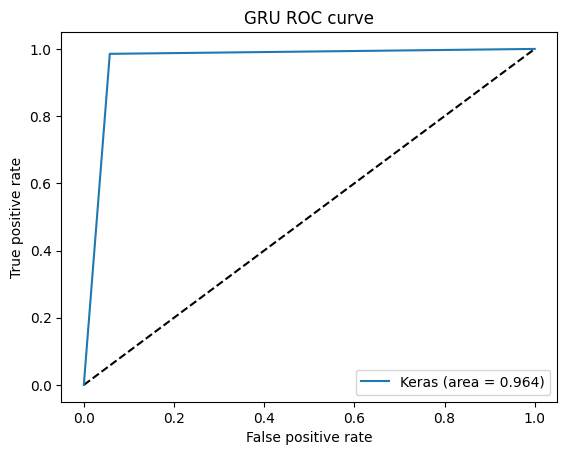


Figure 24 GRU ROC curve

## **4.3.3 RESULTS COMPARISON**

The performance of classic machine learning algorithms and deep learning techniques varies quite a lot when applied over different contexts. Convolutional Neural Networks (CNN) outperforms Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) and Recurrent neural network (RNN) when it comes to the numerical datasets given the architectures in terms of binary classification. Our findings show that all these neural networks achieve satisfactory to high predictive power provided sufficiently large datasets and elucidate the gap between these four models. LSTMs, though robust tend to consume more memory and have slightly higher build time compared to other models. [21][22]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **TP** | **FP** | **FN** | **TN** |
| CNN | 8729 | 264 | 136 | 9436 |
| RNN | 8556 | 437 | 168 | 9404 |
| LSTM | 8556 | 437 | 160 | 9412 |
| GRU | 8475 | 518 | 140 | 9432 |
| Table:9 Comparison of Confusion Matrix values of Deep Learning Algorithms | | | | |

Given below are the metrics and comparisons between the 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 |
| Table:10 Comparison of various Accuracy Metrics of Deep Learning Algorithms | | | | |

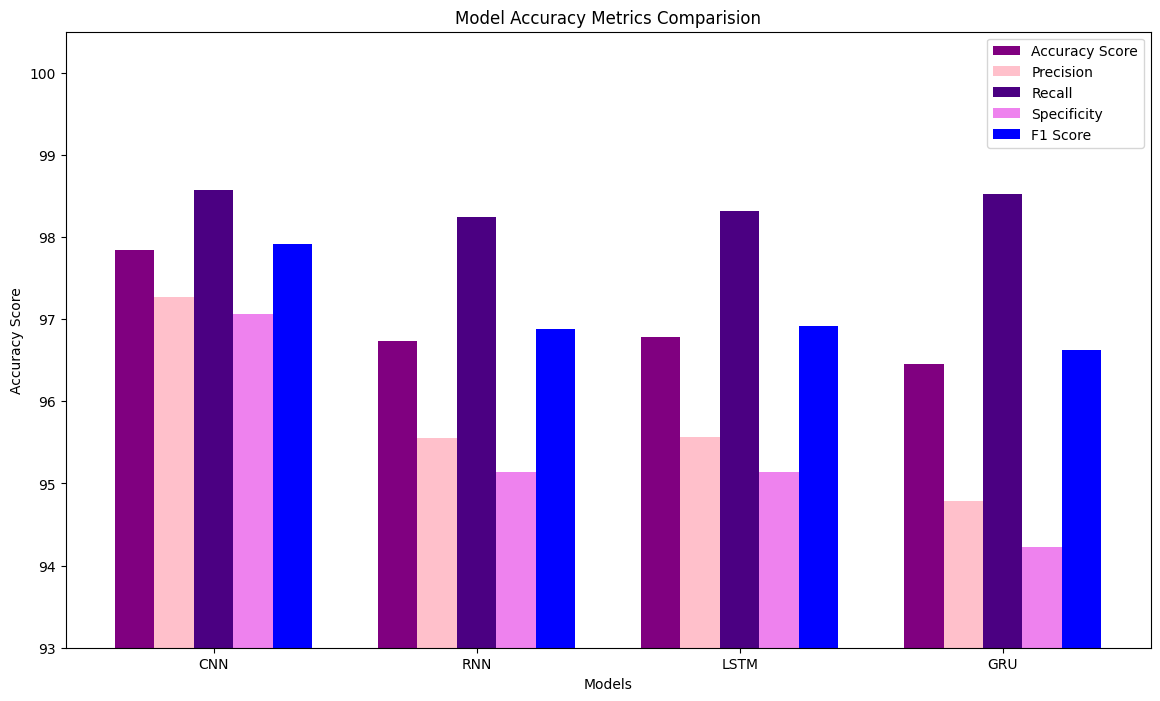
****

Figure 25 Comparison of Evaluation Metrics of Various Models

# **5. CONCLUSION AND FUTURE SCOPE**

It is observed that from the above experiments that after applying feature engineering and feature selection on the dataset. the results were inaccurate and all columns can’t be fed into the model when we used the dataset without much of pre-processing. Hence, handling missing data, Data Transformation using Concept level hierarchy, Data Normalization, One-Hod encoding of categorical data fields, Feature extraction column, Binary Classification are very essential operations before carrying out data mining using any algorithms. A comprehensive study of NSL-KDD dataset and its features is also done as a part of this work. In the undertaken research activity, Deep learning models CNN, RNN, LSTM and GRU neural networks algorithms are used to find which is best for prediction. Convolutional neural network (CNN) predicted the category and subcategory of attack with high accuracy (0.98) than other three models. It is found that the Convolutional neural network (CNN) was the best performing model for the considered test train split, since the findings are so wide, it can be deduced that CNN will do better on the entire dataset or a larger subset. Hence it can be concluded that Convolutional neural network (CNN) can be used for predicting the attacks on datasets. This helps us to build a robust deep learning model around the NSL-KDD dataset. In the future, these results can be used as a benchmark while developing or optimizing Deep learning algorithms for threat detection in network.

**6. 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. 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.
8. 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
9. 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.
10. Dhanabal, L. and S. P. Shantharajah. “A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms.” (2015).
11. 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.
12. 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.
13. 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.
14. 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.
15. Kesavulu Reddy, Dr. (2013). Neural Networks for Intrusion Detection and Its Applications. Lecture Notes in Engineering and Computer Science. 2. 1210-1214.
16. 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
17. Ravi, Vinayakumar & Kp, Soman & Poornachandran, Prabaharan. (2017). Applying convolutional neural network for network intrusion detection. 1222-1228. 10.1109/ICACCI.2017.8126009.
18. 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.
19. 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
20. 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.
21. 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.
22. 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.
23. 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.
24. Ghurab, Mossa & Gaphari, Ghaleb & Alshami, Faisal & Alshamy, Reem & Othman, Suad. (2021). A Detailed Analysis of Benchmark Datasets for Network Intrusion Detection System.
25. 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.
26. 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.
27. 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.
28. 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.
29. 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.