
# School Of Electrical Engineering

# And

# Computer Science

## Artificial Intelligence

**Project Title:** Network Intrusion Detection

**Submitted To:** Mam Seemab Latif

**Lab instructor:** Mam Shakeela Bibi

## Group Members


| Name | CMS |
|---|---|
| Syed Aitezaz Imtiaz | 291093 |
| Waleed Ahmed Shahid | 283290 |
| Hafiz Haseeb Ahmad Butt | 313193 |
| Sana Ullah | 294942 |

- ## Introduction:

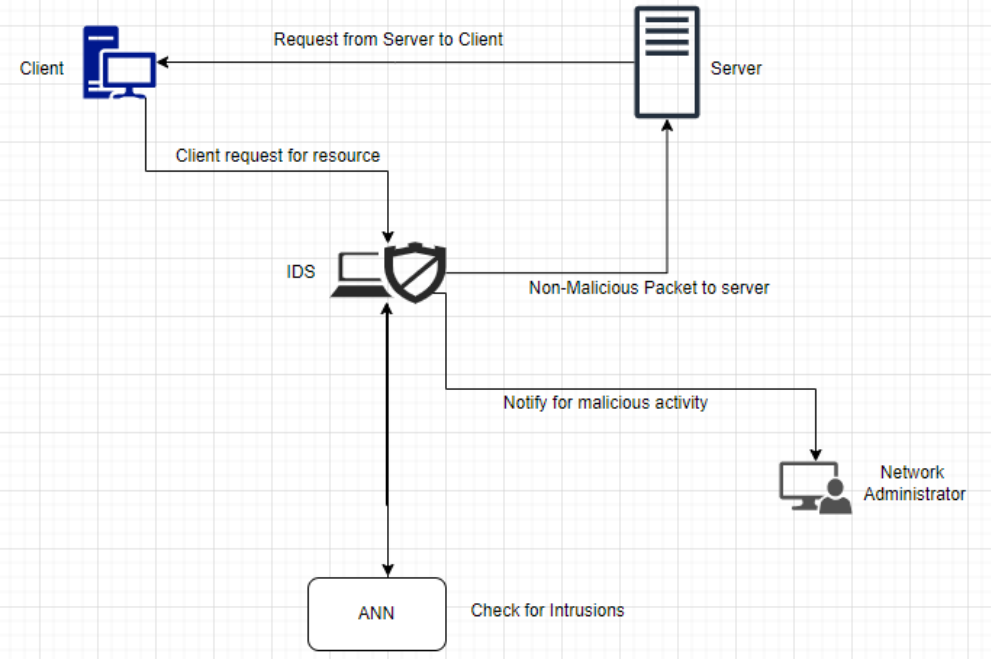
An Intrusion detection system (IDS) is software and/or hardware designed to detect unwanted attempts at accessing, manipulating, and/or disabling of computer systems, mainly through a network, such as the Internet. Firewalls limits access between networks to prevent intrusion and do not signal an attack from inside the network. An IDS evaluates a suspected intrusion once it has taken place and signals an alarm. As the network of computers expands both in number of hosts connected and number of services provided, security has become a key issue for the technology developers. This work presents a prototype of an intrusion detection system for networks. There is often the need to update an installed Intrusion Detection System (IDS) due to new attack methods or upgraded computing environments.

- ## Project Description:

With the enormous growth of computer networks usage and the huge increase in the number of applications running on top of it, network security is becoming increasingly more important. All the computer systems suffer from security vulnerabilities which are both technically difficult and economically costly to be solved by the manufacturers. Therefore, the role of Intrusion Detection Systems (IDSs), as special-purpose devices to detect anomalies and attacks in the network, is becoming more important. We studied the details of the research done in anomaly detection and considered various aspects such as learning and detection approaches, training data sets, testing data sets, and evaluation methods.

- ## Flow Chart:

This the basic flow of our intrusion detection system. Every request that comes to server is first passed through our neural network where it will be checked whether this request is normal, or it has malicious thing in it. If the request is normal and it has nothing harmful than it will be proceed to the server or next user or if the request is not normal than it will notify the server or user that this request has some vulnerability and notify him before any trouble.




- **Dataset:**

The dataset we used in our project is the KDDCUP 99 dataset, which is widely used as one of the few publicly available data sets for network-based anomaly detection systems. KDD CUP 99 data set description: Since 1999, KDD99 has been the most widely used data set for the evaluation of anomaly detection methods. This data set is prepared by Stolfo et al. and is built based on the data captured in DARPA98 IDS evaluation program. DARPA98 is about 4 gigabytes of compressed dump data of 7 weeks of network traffic, which can be processed into about 5 million connection records, each with about 100 bytes. The two weeks of test data have around 2 million connection records. KDD training dataset consists of approximately 4,900,000 single connection vectors each of which contains 41 features. Arbitral Strategy by Neural Network: Artificial Neural network is a powerful tool to solve complex classification problem. We do not need to force much assumption on the problem. We only need to prepare a set of inputs and targets to train it, and let the neural network learn a model.

- **Pre-processing:**

As we have already mentioned above that we are using the KDD Cup 1999 dataset which is very refine for intrusion detection systems. It has 41 features. Most of its features are in int (numeric and float) but some of its features are in string. There are three features that are in string, and we need to convert them into int first so that we can use them in the training first. So, we have first

factorized these three string or categorical columns. After factorizing these three columns all the columns are now in int, but the issue is that they all have different ranges so, we must make their ranges equal so that model don't become biased towards one feature. So, for normalizing our data we have used the standard scalar. StandardScaler removes the mean and scales each feature/variable to unit variance. This operation is performed feature-wise in an independent way. StandardScaler can be influenced by outliers (if they exist in the dataset) since it involves the estimation of the empirical mean and standard deviation of each feature. After normalizing the data, our data has been processed and it was ready to be fed into our models. Some statistics of data before and after the pre-processing are as follows:

**Before pre-processing:**

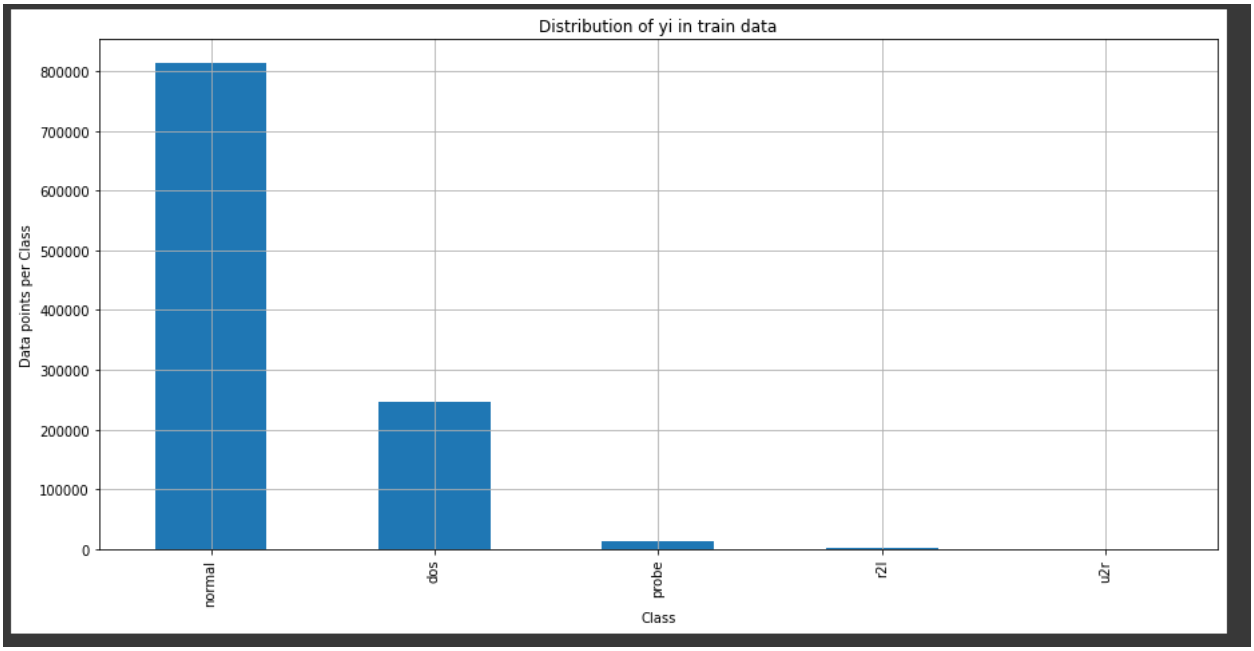
| | duration | protocol_type | service | flag | src_bytes | dst_bytes | land | wrong_fragment | urgent | hot | ... | dst_host_srv_count | dst_host_same_srv_rate | dst_host |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | tcp | http | SF | 215 | 45076 | 0 | 0 | 0 | 0 | ... | 0 | 0.0 | |
| 1 | 0 | tcp | http | SF | 162 | 4528 | 0 | 0 | 0 | 0 | ... | 1 | 1.0 | |
| 2 | 0 | tcp | http | SF | 236 | 1228 | 0 | 0 | 0 | 0 | ... | 2 | 1.0 | |
| 3 | 0 | tcp | http | SF | 233 | 2032 | 0 | 0 | 0 | 0 | ... | 3 | 1.0 | |
| 4 | 0 | tcp | http | SF | 239 | 486 | 0 | 0 | 0 | 0 | ... | 4 | 1.0 | |

**After pre-processing:**

| | duration | src_bytes | dst_bytes | wrong_fragment | urgent | hot | num_failed_logins | num_compromised | root_shell | su_attempted | ... | fla |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.102066 | -0.002798 | 0.029198 | -0.030963 | -0.002356 | -0.054418 | -0.009373 | -0.004338 | -0.01739 | -0.009705 | ... | |
| 1 | -0.102066 | -0.002824 | -0.000251 | -0.030963 | -0.002356 | -0.054418 | -0.009373 | -0.004338 | -0.01739 | -0.009705 | ... | |
| 2 | -0.102066 | -0.002787 | -0.002648 | -0.030963 | -0.002356 | -0.054418 | -0.009373 | -0.004338 | -0.01739 | -0.009705 | ... | |
| 3 | -0.102066 | -0.002789 | -0.002064 | -0.030963 | -0.002356 | -0.054418 | -0.009373 | -0.004338 | -0.01739 | -0.009705 | ... | |
| 4 | -0.102066 | -0.002786 | -0.003187 | -0.030963 | -0.002356 | -0.054418 | -0.009373 | -0.004338 | -0.01739 | -0.009705 | ... | |

## Labels Setting:

We have 26 labels first in the dataset but then we change these 26 labels to five because most of these labels have very less rows. So, we transform these 26 labels into 5 labels based on their properties. The occurrence of these labels and their name in the dataset is show below:




As, you can see that normal has the maximum occurrences in our dataset. So, our model will give maximum accuracy for normal labels because it has trained on its most of its occurrences while on the other labels it has less data, and it can have less accuracy on these labels.

- # Algorithms:

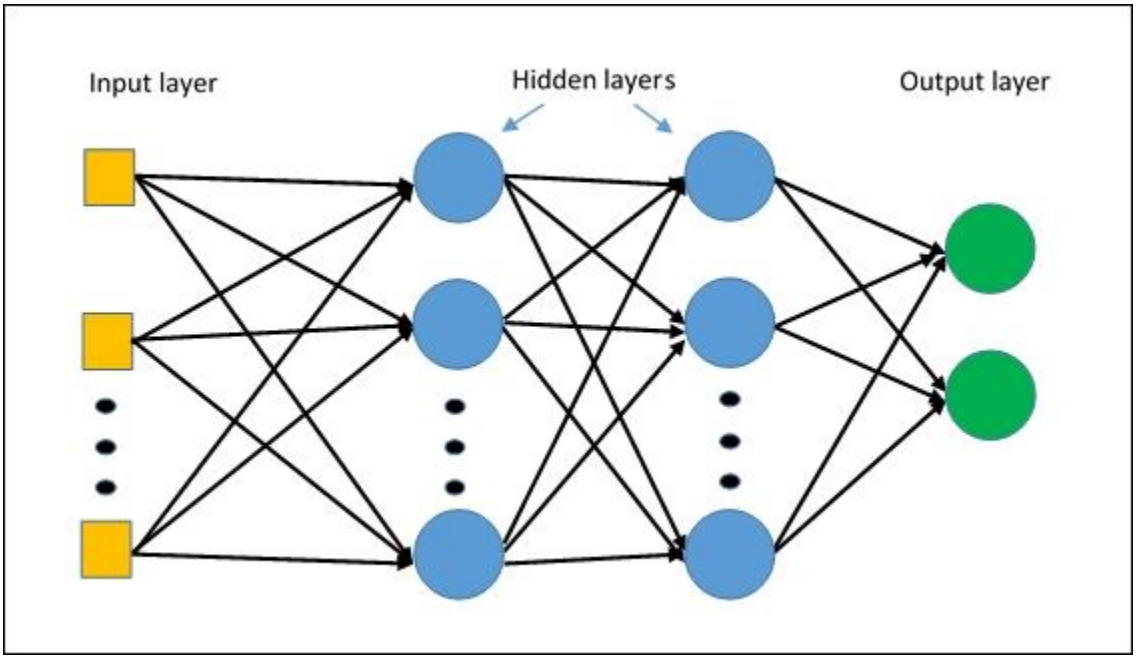
We have used two neural networks algorithms for intrusion detection system. We will discuss each of them in detail. The algorithm with their old parameters and hyper tuned parameters has been mentioned below. The algorithms we have used in our model are as follows.

- Multi-Layer Perceptron
- Recurrent Neural Network

## 1. Multi-Layer Perceptron:

A multilayer perceptron is a fully connected class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to mean any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptron (with threshold activation). Multilayer perceptron are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear

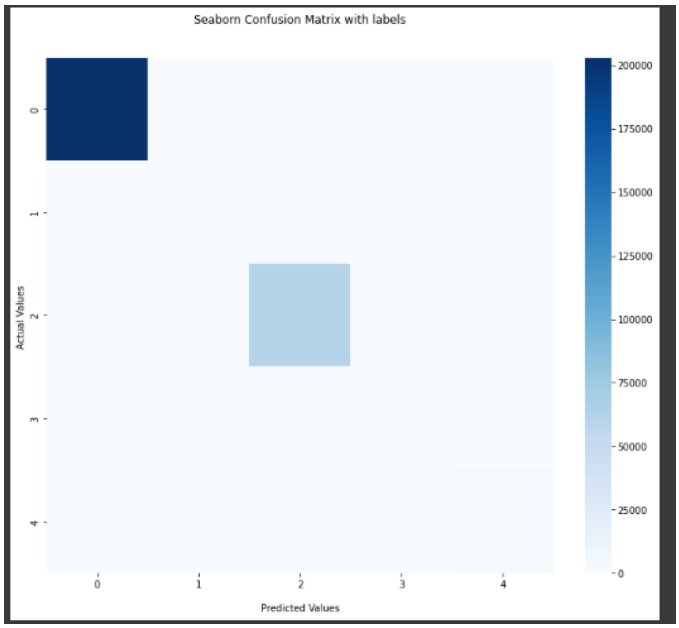
activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable. This diagram can help you in clear understanding of MLP.




**Fine Tuning:**

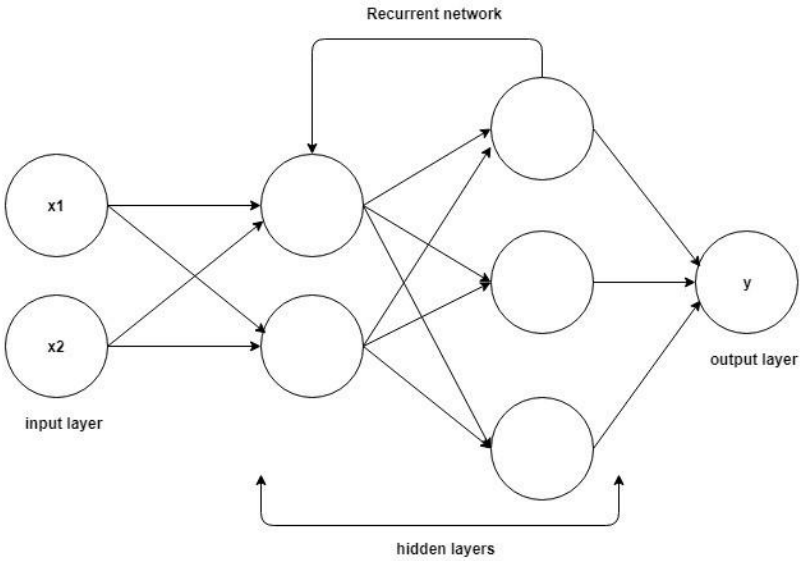
- We have used multi-layer perceptron in our first model, and we are using two number of iterations at first but then we increased the number of iterations to eight and we see a huge increase in accuracy. We have used the default number of hidden layers in our model. The activation function we have used in our model is relu. The hyper tuned parameter model has an **accuracy of more than 97 percent** in our MLP model.

The confusion matrix for test data of multi-layer perceptron model is as follows:




## 2. Recurrent Neural Network:

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed or undirected graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. Recurrent neural networks are theoretically Turing complete and can run arbitrary programs to process arbitrary sequences of inputs. The below image can help us in clear understanding of RNN.

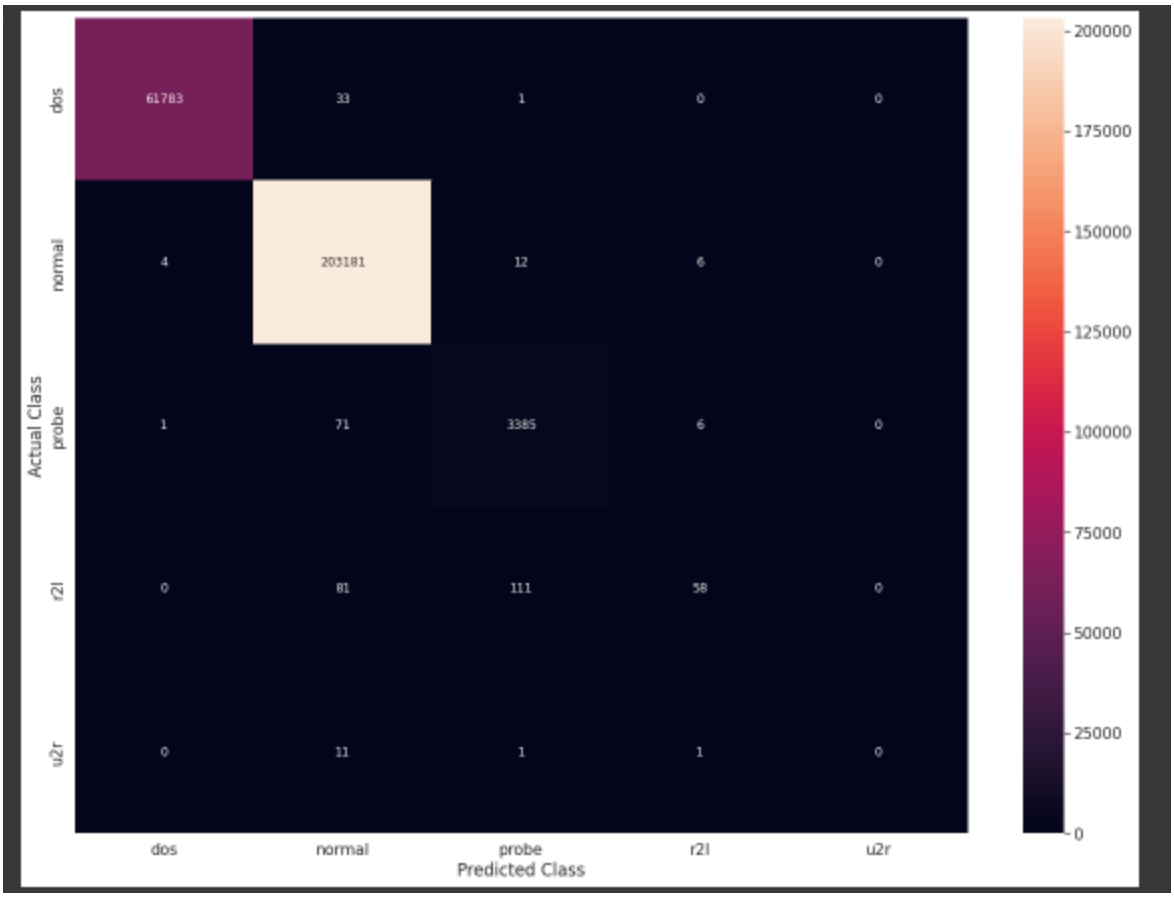


- We have used the sequential model as a recurrent neural network. The total layers in our sequential model are five. We have used the dense layers in our sequential model. We have used the relu and softmax as activation function in these layers. The loss function we have used is categorical_crossentropy and the optimizer we have added during the compile function is adam. The parameters we have used during the training of our model is verbose, epochs and monitors. The value of epoch we have used is 10 and the verbose value is 2. The summary of model is shown below for better understanding of our sequential model.

```
=================================================
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 125)               15750

 dense_6 (Dense)             (None, 70)                8820

 dense_7 (Dense)             (None, 35)                2485

 dense_8 (Dense)             (None, 1)                 36

 dense_9 (Dense)             (None, 23)                46

=================================================================
Total params: 27,137
Trainable params: 27,137
Non-trainable params: 0
_________________________________________________________________
```

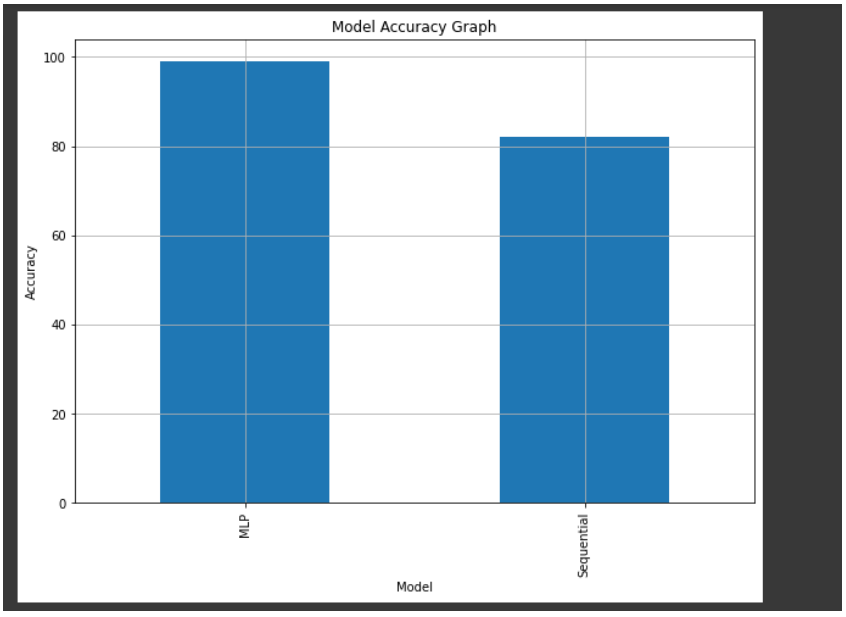
**Fine Tuning:**

- We are first getting the accuracy of 27 percent when we were using the epoch 2 and using higher number of neurons in our model. But later we reduced the neurons used in each layer and increased the number of epochs to 10 and our accuracy got increased to **82 percent**. The confusion matrix for testing data is as follows:




## Accuracy:

The final accuracy we got from our both hyper tuned model is shown below:



You can see from the graph that multi-layer perceptron model has a very higher accuracy than the sequential model and there are more chances of getting right prediction from multi-layer perceptron than sequential model.

## Conclusion:

I have seen from my project that we can increase the accuracy a more by adding some data in our dataset of different labels other than normal. Our model has a very huge data on normal label while on other labels it has very less data. So, for making a good model which can predict almost all network requests correctly we need to enhance our dataset to make it more accurate and precise. On the other hand, network intrusion system scope can be increased by scaling it from detection of requests to block the requests too. We have seen that in many big servers' network intrusion detection is used where it not only detects the request as normal or malicious but also block them and notify the user that the request is malicious and requested. By working on its dataset and trying different models with different parameters, we can more increase its accuracy which we will do after this project and try to implement it on real servers.