Malware Analytics: Intrusion Detection using CNN

by

Anila A Joseph 18BLC1107
Andrew John 18BEC1278
Vighnesh M 18BEC1223

A project report submitted to

Dr. Berlin Hency V

SCHOOL OF ELECTRONICS ENGINEERING (SENSE)

in partial fulfilment of the requirements for the course of

ECE3502 – IoT Domain Analyst

in

B.Tech. Electronics and Communications Engineering

VIT CHENNAI, Vandalur - Kelambakkam Road

Chennai – 600127

June - 2021

BONAFIDE CERTIFICATE

Certified that this project report entitled **“Malware Analytics: Intrusion Detection using CNN”** is a Bonafide work of **Anila A Joseph - 18BLC1107, Andrew John - 18BEC1278, and Vighnesh M - 18BEC1223** who carried out the Project work under my supervision and guidance **for ECE3502 - IoT Domain Analyst.**

Dr. Berlin Hency V

Associate Professor

School of Electronics Engineering (SENSE),

VIT University, Chennai

Chennai – 600 127.

ABSTRACT

One of the most vital parts of system and network administration and security is keeping a network safe from intrusion. If a network is penetrated by a malicious attacker, it can lead to massive losses for a company, including potential downtime, data breaches, and loss of customer trust. A Network Intrusion Detection System (NIDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. Intrusion detection systems work by either looking for signatures of known attacks or deviations from normal activity. These deviations or anomalies are pushed up the stack and examined at the protocol and application layer. There are 2 types of intrusion detection system, Network Intrusion Detection System (NIDS) and Host Intrusion Detection System (HIDS), NIDS works in real-time, which means it tracks live data and flags issues as they happen. On the other hand, HIDS examines historical data to catch savvy hackers that use non-conventional methods that might be difficult to detect in real-time.

We propose a network intrusion detection model using Convolutional Neural Networks (CNNs). As CNN's perform well with image data we convert the raw traffic vector format into image format. In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. The dataset used is the standard KDD Cup 1999 Dataset. The types of attacks faced are classified into five categories in this paper and they are as follows: Normal, DoS, Probe, U2R and R2L. When the traffic is classified into an abnormal attack, the maintainers would take measures to protect the network accordingly.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Berlin Hency V**., Associate Professor, School of Electronics Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Sivasubramanian. A.**, Dean of School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Head of the Department **Dr. Vetrivelan. P** & **Dr. D Thirupusundari** for their support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

[illegible] [illegible] [illegible]

ANILA **ANDREW** **VIGHNESH**

TABLE OF CONTENTS

CHAPTER - I
INTRODUCTION

1.1 Objective

This project's objective is to create an Intrusion Detection System (IDS) for massive networks using Convolutional Neural Networks (CNN) where the model classifies the types of attack on a system based on 41 Features while solving the Dataset Imbalance Problem and provides an Accurate (>98%) classification model by increasing Detection Rate (DR) and Accuracy and decrease False Alarm Rate.

1.2 Motivation

The motivation behind this project aimed at having secure servers by detecting any attack. An intrusion detection system (IDS) is crucial for network security because it enables you to detect and respond to malicious traffic. The primary benefit of an intrusion detection system is to ensure IT personnel are notified when an attack or network intrusion might be taking place.

1.3 Features

The features or novelty of the project are,

I. CNN is performed on two-dimensional / image data,
II. CNN is used on non-image data by converting its one-dimensional attributes into 2D image data,
III. CNN is computationally efficient. It uses special convolution and pooling operations and performs parameter sharing. This enables CNN models to run on any device, making them universal,
IV. The model will be trained on a huge dataset (over 100k data points),
V. CNN parameters can be fine-tuned (number of convolutional/pooling layers, filter size, initial weights etc.) to get optimal results and
VI. This model will also address the dataset imbalance problem that previous models did not.

1.4 Applications

Some of the fields Intrusion detection applications find themselves are in Multivector threat identification where detailed inspection of Layer 2-7 traffic protects your network from policy violations, vulnerability exploitations and anomalous activities. Some of the Accurate and Novel Prevention technologies such as that of Cisco's System are innovative Risk Rating features and Meta Event Generator that provide the confidence to take preventive actions on a broader range of threats without the risk of dropping legitimate traffic.

CHAPTER - II
LITERATURE REVIEW

W. Stallings in **[1]** has given that there are several measures for protecting network security, such as rewalls, encryption, authentication, and intrusion detection but intrusion detection can identify the illegal behaviors of these attacks through an assumption. Concerning Preprocessing and Classification, S. Ganapathy *et al*. **[2]** has conducted a survey on intelligent techniques for feature selection and classification for intrusion detection in networks based on intelligent software agents, neural networks, genetic algorithms, neuro-genetic algorithms, fuzzy techniques, rough sets, and particle swarm intelligence and has presented a new feature selection algorithm called Intelligent Rule based Attribute Selection algorithm and a novel classification algorithm named Intelligent Rule-based Enhanced Multiclass Support Vector Machine. Similarly, for feature selection of higher dimensional data, V. Bolón-Canedo *et al.* **[3]** has proposed a fast filter method which can identify relevant features as well as redundancy among relevant features without pairwise correlation analysis called FCBF (Fast Correlation Based Filter) that is implemented and evaluated through extensive experiments, comparing with related feature selection algorithms. Meanwhile for efficient and robust feature extraction, H. Shi *et al*. **[4]** presents a novel feature extraction and selection approach where, multifractal features are extracted from traffic flows using a Wavelet Leaders Multifractal Formalism (WLMF) to depict the traffic flows; and, a Principal Component Analysis (PCA)-based FS method is applied on these multifractal features to remove the irrelevant and redundant features. The results demonstrate significant improvement in accuracy of Support Vector Machines (SVMs) compared to the TLS features studied in existing ML-based approaches. Furthermore, the proposed approach is suitable for real time traffic classification because of the ability of classifying traffic at the early stage of traffic transmission. Since Lecun *et al*. **[5]** proposed deep learning (DL), it has been widely applied in visual recognition, speech recognition and natural language processing (NLP). As DL can abstract high level features deeply from raw data, many fields have actively utilized DL algorithms. Additionally, some DL algorithms have been used for intrusion detection.

For Intrusion Detection using Fuzzy Logic, A. Chaudhary *et al.* **[6]** have emphasized on the proposed fuzzy based intrusion detection systems in mobile ad hoc networks and presented their effectiveness to identify the intrusions and have analyzed the working style of proposed fuzzy based IDSs and reached on decision that they still do not have any promising solution for this dynamic environment because most of Proposed fuzzy based IDSs emphasized on very limited features for data collection towards detection of very specific range of attacks. Hence, MANETs (Mobile Adhoc NETworks) are required for more concentration of researchers. For Intrusion Detection using SVM Classifier, H. Wang *et al.* **[7]**, proposes an effective intrusion detection framework based on a Support Vector Machine (SVM) with augmented features. More specifically, this paper implements the logarithm marginal density ratios transformation to form the original features with the goal of obtaining new and better quality transformed features that can greatly improve the detection capability of an SVM-based detection model.

In this detection framework, the feature augmented technique is used to provide concise, high-quality training data for the SVM classifier, which not only improves the detection capability of the SVM, but also reduces the required training time. The results show that the proposed detection can achieve a robust performance with high accuracy, and a high detection rate, a low false alarm rate and a rapid training speed. For Intrusion Detection using Decision Tree, S. S. S. Sindhu *et al.* **[8]**, has the design of IDS investigated from these three perspectives. The goals of this paper were (i) removing redundant instances that causes the learning algorithm to be unbiased (ii) identifying suitable subset of features by employing a wrapper based feature selection algorithm (iii) realizing proposed IDS with neurotree to achieve better detection accuracy. The proposed system is better even when the dataset is presented with a different number of classes. This justifies that the proposed features and learning paradigm neurotree is a promising strategy to be applied on intrusion detection. For Intrusion Detection using Cuttlefish Optimization, A. S. Eesa *et al.* **[9]**, have proposed a model that uses the cuttlefish algorithm (CFA) as a search strategy to ascertain the optimal subset of features and the decision tree (DT) classifier as a judgement on the selected features that are produced by the CFA. The KDD Cup 99 dataset is used to evaluate the proposed model. Empirical results reveal that the produced features are performed the DR and AR especially when the number of produced features was equal to or less than 20 features. In general, whenever the number of features is decreased, the AR and DR are increased. For Intrusion Detection using Artificial Neural Networks, G. Wang *et al.* **[10]**, where the general procedure of FC-ANN is as follows: firstly, fuzzy clustering technique is used to generate different training subsets. Subsequently, based on different training subsets, different ANN models are trained to formulate different base models. Finally, a meta-learner, fuzzy aggregation module, is employed to aggregate these results. Through FC-ANN, the heterogeneous training set is divided into several homogeneous subsets. Thus complexity of each sub training set is reduced and consequently the detection performance is increased. The experimental results using the KDD CUP 1999 dataset demonstrates the effectiveness of the new approach especially for low-frequent attacks, i.e., R2L and U2R attacks in terms of detection precision and detection stability. For Intrusion Detection using Naïve Bayes Classifier, S. Mukherjee *et al.* **[11]** proposes a method called: Feature Vitality Based Reduction Method, which identifies important reduced input features and it applies one of the efficient classifier naive bayes on reduced datasets for intrusion detection. FVBRM model for feature selection is used and makes its comparison with three feature selectors: Correlation Based Feature Selection, Information Gain and Gain Ratio. Experimental results illustrate that the feature subset identified by CFS has improved Naïve Bayes classification accuracy when compared to IG and GR.

The paper, proposed by M. A. Salama *et al.,* firstly Introduced usage of Deep Learning for IDS where a hybrid scheme that combines the advantages of deep belief network and support vector machine meanwhile an application of intrusion detection imaging has been chosen and hybridization scheme have been applied to see their ability and accuracy to classify the intrusion into two outcomes: normal or attack, and the attacks fall into four classes; R2L, DoS, U2R, and Probing. Deep Belief network has proved a good addition to the field of network intrusion classification. DBN provides a good result as a separate classifier and as a feature reduction method. The DBN-SVM scheme shows a higher percentage of classification than

SVM and enhances the testing time due to data dimensions' reduction **[12]**. J. Kim *et al.* has achieved good results using Recurrent Neural Network where they apply Long Short Term Memory(LSTM) architecture to a Recurrent Neural Network(RNN) and train the IDS model using KDD Cup 1999 dataset. The IDS classifier based on LSTM-RNN has evaluated the IDS model. In order to find the proper learning rate and hidden layer size, they took an experiment with changing the values. By comparing it to other IDS classifiers, they found that the attacks are well detected by the LSTM-RNN classifier. Because they have the highest DR and Accuracy. Through the performance test, this paper confirms that the deep learning approach is effective for IDS **[13]**.

T. A. Tang *et al.* confirms DL has the potential for strong detection by applying a deep learning approach for flow-based anomaly detection in an SDN environment and building a Deep Neural Network (DNN) model for an intrusion detection system by training the model with the NSLK-DD Dataset. In this work, they use 6 basic features (that can be easily obtained in an SDN environment) taken from the 41 features of NSL-KDD Dataset. The results obtained were not yet good enough to be adopted in any commercial product or an alternative solution for signature-based IDS but by comparing the results with those of other classifiers, they have shown the potential of using deep learning for the flow-based anomaly detection system. In the context of the SDN environment, the deep learning approach also has potential **[14]**.

For Intrusion Detection with Imbalanced Dataset using Imbalanced Data Gravitation-based Classification (DGC), L. Peng *et al.* **[15]** constructed six imbalanced traffic data sets from three original traffic data sets and then extracted their early stage features according to the packet sizes. In identification experiments, it compared the performance of six standard algorithms, including DGC and four imbalanced algorithms with IDGC. Showing that it needs high computational cost. Results demonstrate that the standard classification models could achieve high accuracy with imbalanced traffic data, but their imbalanced performance was not as good, and their generalizability was also a problem. By contrast, IDGC performed very well and in a stable manner in terms of imbalanced performance measures, thereby demonstrating its effectiveness for imbalanced traffic identification.

Hence, there are certain limitations in **[6]-[15]** that are, (i) the traditional ML classifiers for intrusion detection are difficult to adapt to massive networks, (ii) with the rapid growth of network traffic data, the ML classifiers reduce the accuracy of intrusion detection & increase the calculation cost and time, (iii) furthermore, they are easy to over fit, (iv) Deep Learning (DL) models such as DBN (reduced feature sets), LTSM, and DNN which confirmed that DL had strong potential for intrusion detection, and (v) the RNN model also had a high number of parameters which in turn would mean a high computation cost.

CHAPTER - III

DESIGN & IMPLEMENTATIONS

3.1 About Software and Dataset

3.1.1 The Software used:

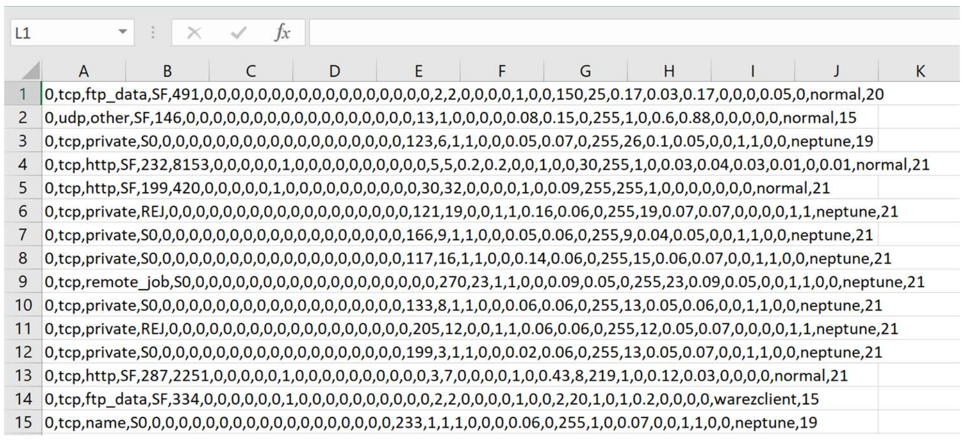
Google Colab and *Jupyter Notebook* are the python coding platforms used.

Some of the important python libraries used are TensorFlow's Keras, Pandas, Numpy, SciKit Learn, Time, OS and MatPlotLib's PyPlot.

- **TensorFlow:** TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Tensorflow is a symbolic math library based on dataflow and differentiable programming.
- **pandas:** pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
- **Numpy:** NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **SciKit Learn:** Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
- **OS:** The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality.
- **MatPlotLib:** Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

3.1.2 About the Dataset Used:

KDD Cup 1999 Dataset



```
L1
   A  B  C  D  E  F  G  H  I  J  K
1  0,tcp,ftp_data,SF,491,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,0,0,0,1,0,0,150,25,0.17,0.03,0.17,0,0,0,0.05,0,normal,20
2  0,udp,other,SF,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,1,0,0,0,0,0.08,0.15,0,255,1,0,0.6,0.88,0,0,0,0,0,normal,15
3  0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,123,6,1,1,0,0,0.05,0.07,0,255,26,0.1,0.05,0,0,1,1,0,0,neptune,19
4  0,tcp,http,SF,232,8153,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,5,5,0.2,0.2,0,0,1,0,0,30,255,1,0,0.03,0.04,0.03,0.01,0,0.01,normal,21
5  0,tcp,http,SF,199,420,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,30,32,0,0,0,0,1,0,0.09,255,255,1,0,0,0,0,0,0,0,normal,21
6  0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,121,19,0,0,1,1,0.16,0.06,0,255,19,0.07,0.07,0,0,0,0,1,1,neptune,21
7  0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,166,9,1,1,0,0,0.05,0.06,0,255,9,0.04,0.05,0,0,1,1,0,0,neptune,21
8  0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,117,16,1,1,0,0,0.14,0.06,0,255,15,0.06,0.07,0,0,1,1,0,0,neptune,21
9  0,tcp,remote_job,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,270,23,1,1,0,0,0.09,0.05,0,255,23,0.09,0.05,0,0,1,1,0,0,neptune,21
10 0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,133,8,1,1,0,0,0.06,0.06,0,255,13,0.05,0.06,0,0,1,1,0,0,neptune,21
11 0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,205,12,0,0,1,1,0.06,0.06,0,255,12,0.05,0.07,0,0,0,0,1,1,neptune,21
12 0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,199,3,1,1,0,0,0.02,0.06,0,255,13,0.05,0.07,0,0,1,1,0,0,neptune,21
13 0,tcp,http,SF,287,2251,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,3,7,0,0,0,0,1,0,0.43,8,219,1,0,0.12,0.03,0,0,0,0,normal,21
14 0,tcp,ftp_data,SF,334,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,2,2,0,0,0,0,1,0,0,2,20,1,0,1,0.2,0,0,0,0,warezclient,15
15 0,tcp,name,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,233,1,1,1,0,0,0,0.06,0,255,1,0,0.07,0,0,1,1,0,0,neptune,19
```

Fig-1: *The dataset used for Intrusion Detection*

This is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between ``bad'' connections, called intrusions or attacks, and ``good'' normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.

3.1.3 Types of Attacks faced:

Generally, there are four attacks: **DoS** (denial of service), **Probe**, **U2R** (user to root) and **R2L** (remote to local).

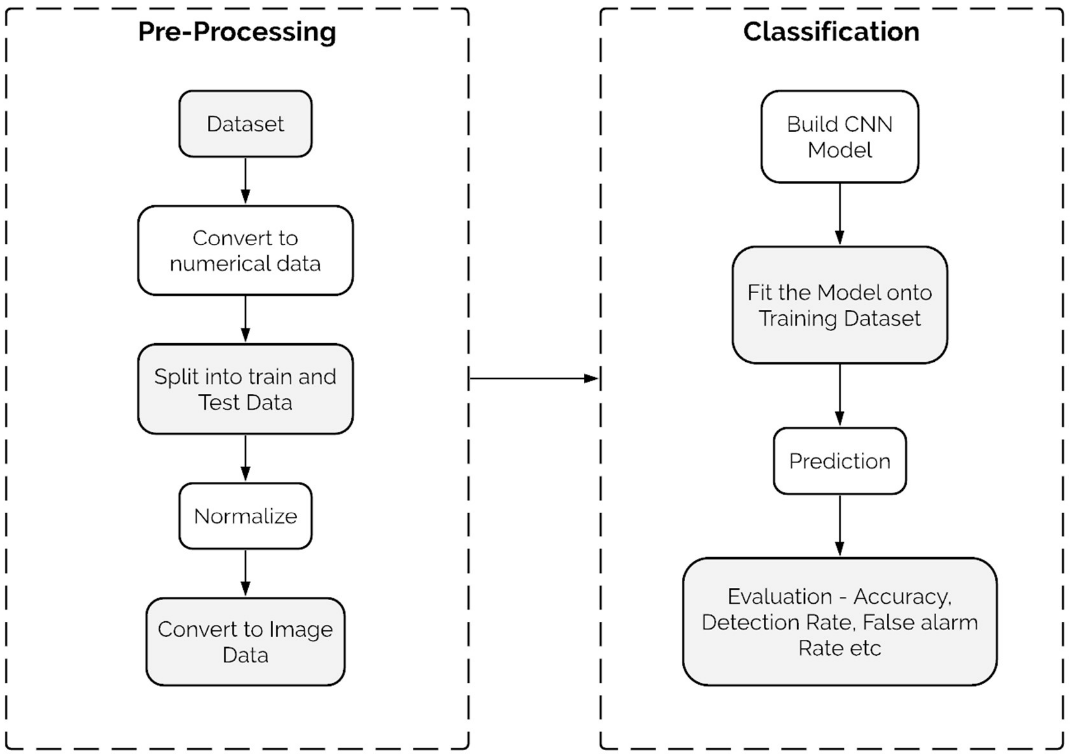
- **Denial of Service:** In computing, a denial-of-service attack is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet.
- **Probe:** A probe is an attempt to gain access to a computer and its files through a known or probable weak point in the computer system. Network Probes are not an immediate threat. However, they do indicate that someone is casing your system for possible entry points for attack.
- **User to Root:** User-to-Root attack is usually launched for illegally obtaining the root's privileges when legally accessing a local machine by exploiting the vulnerabilities of the system.
- **Remote to Local:** Remote-to-Local has been widely known to be launched by an attacker to gain unauthorized access to a victim machine in the entire network.

The goal of intrusion detection is to classify the network traffic into five types: **Normal**, **DoS**, **Probe**, **U2R** and **R2L**. When the traffic is classified into an abnormal attack, the maintainers would take measures to protect the network accordingly.

3.1.4 CNN & Advantages of CNN:

Convolutional neural networks (CNNs) is a classical DL algorithm, and it has been applied in many fields. It can not only select features but also classify the traffic data. CNN can learn better features automatically than traditional feature selection algorithms. The more traffic data, the more useful features the CNN can learn, the better classification the CNN performs. However, machine learning algorithms are easy to over-fit in a massive data environment. Hence, CNN is suitable for the massive network environment. Besides, compared with other DL algorithms, the greatest advantage of CNN is that it shares the same convolutional kernels, which would reduce the number of parameters and calculation amount of training once greatly, it can more quickly identify attack type of traffic data.

3.2 Block Diagram:


Pre-Processing
Dataset
Convert to numerical data
Split into train and Test Data
Normalize
Convert to Image Data
Classification
Build CNN Model
Fit the Model onto Training Dataset
Prediction
Evaluation - Accuracy, Detection Rate, False alarm Rate etc

3.3 Algorithm:

- *Step 1:* **Data Preprocessing**
  - The NSL-KDD Dataset contains 22 usual attack types plus the normal category.
  - The labels are converted into 5 main categories (normal, DoS, Probe, U2R, R2L) using the information provided in the analysis of the dataset.
  - After that, each attack is encoded into one of 5 numbers where normal is 0, DoS is 1, Probe is 2, R2L is 3 and U2R is 4.

- *Step 2:* **Converting to Numerical Data - One Hot Encoding**
  - The NSL-KDD dataset has four symbolic data types: the protocol_type feature, flag feature, service feature and attack label.
  - In this step, the one-hot encoder is used to map the symbolic data into numeric data.
  - One protocol_type feature turns into three features, one flag feature turns into 11 features, and one service feature changes into 70 features.
  - Therefore, the 41 initial features change into 122 features.

- *Step 3:* **Data Normalization and Shuffling**
  - Eliminates the differences between different dimensional data. Normalized to the range [0,1] using min-max normalization.
  - The transformation function is as follows.
  - The data is shuffled to prevent any bias during the training and to prevent the model from learning the order of the training

- *Step 5:* **Solving Dataset Imbalance — COST function based method**
  - The cost function-based method adjusts the weights of the cost function according to the proportion of different samples.
  - This method not only improves the performance of the model but also maintains the efficiency of the initial model.
  - Let the number of normal, DoS, probe, U2R, and R2L samples be n1, n2, n3, n4, and n5, respectively.
  - In addition, the cost function weights of the sample of category i can be written as:

$$w_i = \textit{total no of samples}/n_i$$

- *Step 6:* **Converting Normalized Data to image format**
    - CNNs perform well in the field of image processing. Hence, we first convert the initial data format into image format.
    - Convert the 1x122 vector into nxn format. n should be maximized. As 121=11x11, the optimal result is n=11. Therefore, the 122-dimension feature set should remove a feature.
    - The feature with a zero mean or the feature with the least coefficient of variance is removed.

- *Step 7:* **CNN Model**
    - Parameters like learning rate, batch size, number of epochs are initialized
    - The model has 2 convolution layers and 2 pooling layers. The input is an 11x11 image. The image data is inputted into the first convolution layer.
    - The convolution function can be written as:

$$h_j = f\,(h_j\text{-}1 * w_j + b_j)$$

    - f (x) is the activation function - (ReLU)
    - The pooling layer works after the convolution layer. It can reduce the size of feature image hj and avoid over-fitting. It can be written as:

$$h_j = pool(h_j\text{-}1)$$

Output yi achieved through the Dense layer with SoftMax activation function. The training aims to minimize the error. In addition, the back propagation of the error generally uses the gradient descent method. Loss and accuracy are calculated.

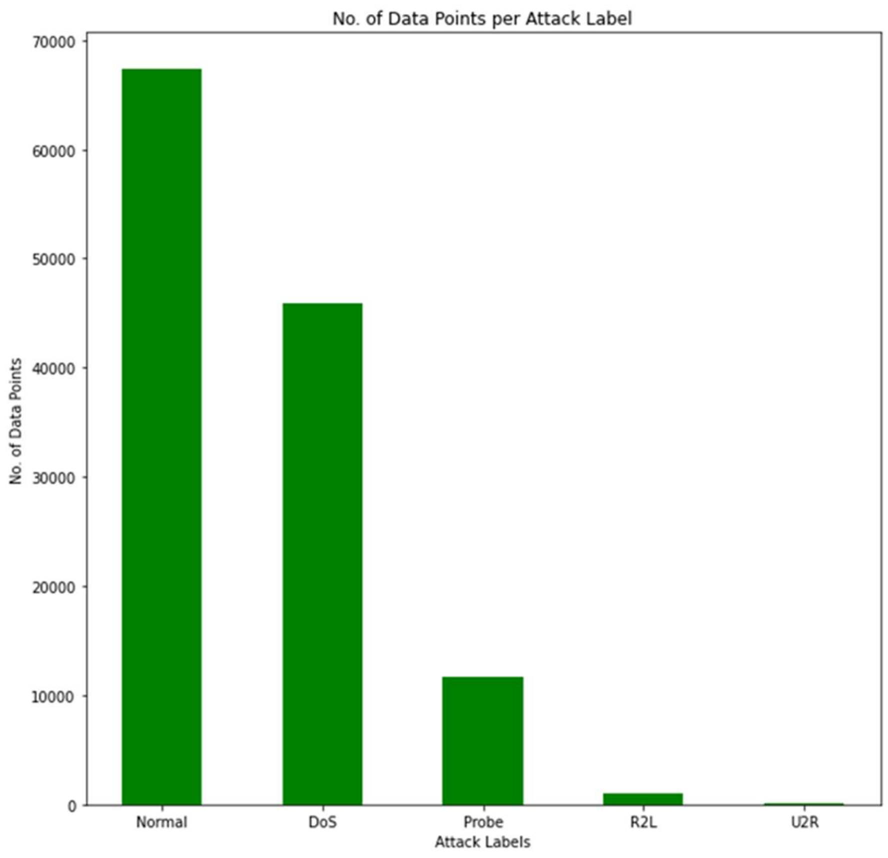
3.4 Coding and Implementation:

```
"""# Main Libraries"""

from google.colab import drive
drive.mount('/content/drive')

import numpy as np
import pandas as pd
import time
import os
import pickle
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras import initializers

"""# Clear"""

clear = lambda:os.system('clear')

"""# Getting the dataset"""

def getDataSet():# Getting the path of the dataset
  #path="KDDTrain+.csv"
  path = "/content/drive/My Drive/IoT 2/KDDTrain+.csv"
  return path

"""# Reading the dataset"""

def readingData(path): #Reading the Dataset

      print("\nReading Dataset...\n")

      dataSet = pd.read_csv(path,low_memory=False)

      return dataSet

"""# Check if missing data"""

def checkMissing(X):#This checks if the dataset given has missing values.
      isMissing  =  str(X.isnull().values.any())  #Using  String  instead  of
Boolean because ("cannot unpack non-iterable numpy.bool object")

      if isMissing == "True":
      #if data set has infinity values replace them with none
      X = X.replace('Infinity', np.nan) #Replacing Infinity values with nan
values

      missingValIndex = []
      total = X.isnull().sum().sum()
      percent = (total / (X.count().sum() + X.isnull().sum().sum())) * 100

      for rows in X:

            if X[rows].isnull().sum() != 0:
                  missingValIndex.append(rows)
      print("\n\n**************************************************")
      print("Data has missing values")
      print("**************************************************")
      print("Features with missing values:",missingValIndex)
      print("Total missing Values -> " , total)
      print(percent,"%")

      return X

      else:

      return X
```

```
"""# Getting the features"""

#Getting The data we want to test for the clustering algorithms
def gettingVariables(dataSet):# If the dataset is NSL-KDD it would get the
features and the labels for it and if it's IDS 2017 it would take the features
and the labels for it and take careof missing values.

      X = dataSet.iloc[:,:-2].values # Data, Get all the rows and all the
columns except all the columns - 2
      Y = dataSet.iloc[:,41].values# Labels
      return X,Y
```

```
"""# Encoding Labels"""

def encodingLabels(Y):# Encoding the labels onto 5 classes

   #4 Main Categories
                  #normal = 0
                  #DoS = 1
                  #Probe = 2
                  #R2L = 3
                  #U2R = 4
   attackType  = {'normal': 'normal', 'saint':'Probe', 'mscan':'Probe',
'neptune':'DoS', 'warezclient': 'R2L', 'ipsweep': 'Probe', 'back': 'DoS',
'smurf': 'DoS', 'rootkit': 'U2R','satan': 'Probe', 'guess_passwd': 'R2L',
'portsweep': 'Probe', 'teardrop': 'DoS','nmap': 'Probe','pod': 'DoS',
'ftp_write': 'R2L', 'multihop': 'R2L','buffer_overflow': 'U2R', 'imap':
'R2L', 'warezmaster': 'R2L', 'phf': 'R2L', 'land': 'DoS', 'loadmodule':
'U2R', 'spy': 'R2L', 'perl': 'U2R', 'apache2':'DoS', 'mailbomb':'DoS',
'processtable':'DoS', 'sendmail' : 'R2L', 'named' : 'R2L','snmpgetattack' :
'R2L' , 'snmpguess' : 'R2L', 'xlock' : 'R2L', 'xsnoop' : 'R2L' , 'worm' :
'R2L','httptunnel':'U2R', 'ps':'U2R', 'sqlattack':'U2R',
'xterm':'U2R','udpstorm':'DoS' }
   attackEncodingCluster  = {'normal':0,'DoS':1,'Probe':2,'R2L':3, 'U2R':4}
#Main Categories
   att=[0,0,0,0,0]

   Y[:] = [attackType[item] for item in Y[:]] #Encoding the main 4 categories
   Y[:] = [attackEncodingCluster[item] for item in Y[:]]# Changing the names
of attacks into 4 main categories

   for i in range (0,len(Y)):
      for x in range(0,5):
      if Y[i]==x:
      att[x]=att[x]+1
   return Y,att
```

```
"""# One Hot Encoding"""

#Encoding the categorical features using one hot encoding onto 5 attack
categories
def oneHotEncodingData(X):

      from sklearn.preprocessing import OneHotEncoder
      from sklearn.compose import ColumnTransformer
      transform = ColumnTransformer([("Servers", OneHotEncoder(categories =
"auto"), [1,2,3])], remainder="passthrough")
      X = transform.fit_transform(X)
      return X

"""# Scaling """

def scaling(X):#Scaling the data with min max normalization

      from sklearn.preprocessing import MinMaxScaler
      #Transforms features by scaling each feature to a given range.
      X =  MinMaxScaler(feature_range=(0, 1)).fit_transform(X)
      return X

"""# Shuffle"""

def shuffleData(X): #the data points are shuffled

      from sklearn.utils import shuffle
      X = pd.DataFrame(X)
      X = shuffle(X)
      X.reset_index(inplace=True,drop=True)
      X = np.array(X)

      #print("Data has been successfully shuffled.")
      return X

"""# Main- Calling Functions"""

clear()
#Calling the functions
path = getDataSet()

dataSet = readingData(path)

dataSet = checkMissing(dataSet)

data,labels = gettingVariables(dataSet) #Getting the Data we want to use for
the algorithms
labels,att = encodingLabels(labels) #encoding labels into 0,1,2,3,4
data = oneHotEncodingData(data) #one hot encoding 41 features into 121
data=scaling(data) #min max normalization
data=shuffleData(data) #shuffling

fig = plt.figure(figsize = (10, 10))
attacks=['Normal','DoS','Probe','R2L','U2R']
# creating the bar plot
plt.bar(attacks, att, color ='green',width = 0.5)
plt.xlabel("Attack Labels")
plt.ylabel("No. of Data Points")
plt.title("No. of Data Points per Attack Label")
plt.show()
```

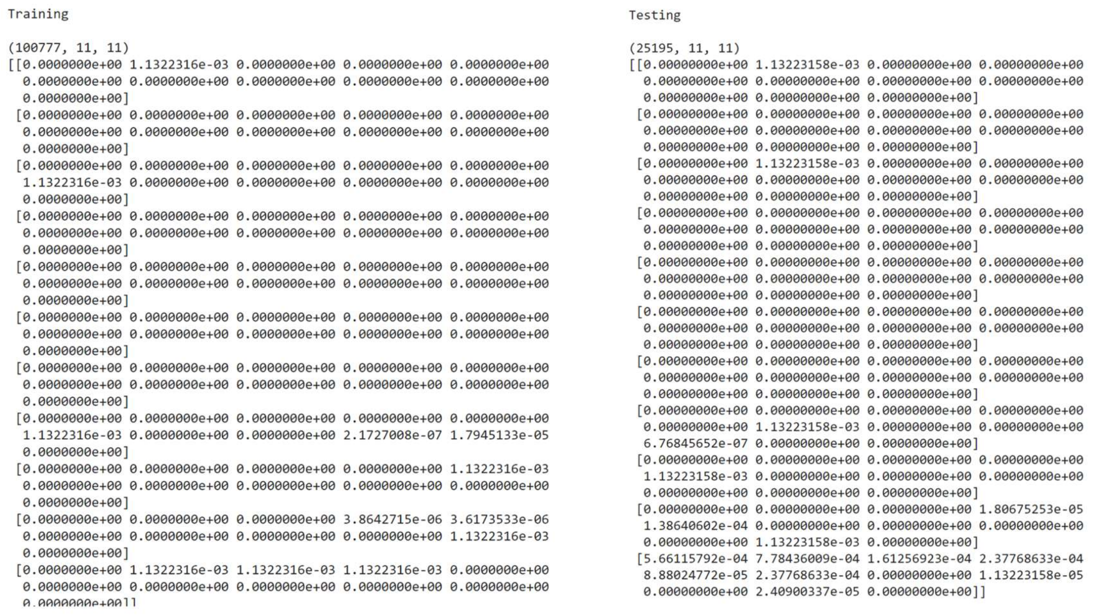


```
"""# Finding column to remove"""

x=pd.DataFrame(data)
sd=x.std(axis = 0, skipna = True)
mean=x.mean(axis = 0, skipna = True)
rem=sd[0]/mean[0];
min=0;

for i in range(len(sd)):
  if mean[i]==0:
      min=i
      rem=mean[i]

print(rem)
print("Feature with zero mean:",end=" ")
print(min)
del(x[min])

print(x)
```

```
    0.0
    Feature with zero mean: 100
            0    1    2    3    4    5    ...   116   117   118  119   120  121
    0       0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  1.00  1.0  0.00  0.0
    1       1.0  0.0  0.0  0.0  0.0  0.0  ...  1.00  0.00  0.00  0.0  0.00  0.0
    2       0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  0.11  0.0  0.89  1.0
    3       0.0  1.0  0.0  0.0  0.0  0.0  ...  0.02  0.03  0.00  0.0  0.02  0.0
    4       0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  0.00  0.0  0.00  0.0
    ...     ...  ...  ...  ...  ...  ...  ...   ...   ...   ...  ...   ...  ...
    125967  0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  0.00  0.0  0.00  0.0
    125968  0.0  1.0  0.0  0.0  0.0  0.0  ...  0.20  0.06  0.00  0.0  0.00  0.0
    125969  0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  1.00  1.0  0.00  0.0
    125970  0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  0.00  0.0  1.00  1.0
    125971  0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  1.00  1.0  0.00  0.0

    [125972 rows x 121 columns]
```

```
"""# Weights"""

c0=0
c1=0
c2=0
c3=0
c4=0
for i in range(0,len(x)):
  if labels[i]==0:
      c0=c0+1
  elif labels[i]==1:
      c1+=1
  elif labels[i]==2:
      c2+=1
  elif labels[i]==3:
      c3+=1
  else:
      c4+=1

cc=c0+c1+c2+c3+c4

w=[0,0,0,0,0]

w[0]=cc/c0
w[1]=cc/c1
w[2]=cc/c2
w[3]=cc/c3
w[4]=cc/c4

"""# Splitting into Test and Train """

from sklearn.model_selection import train_test_split

#the dataset is split into test(20%) and train(80%) datasets

x_train,x_test=train_test_split(x,test_size=0.2) #test,train data
x_train_labels,x_test_labels=train_test_split(labels,test_size=0.2)
#test,train labels


print(x_train)
print(x_test)
```

```
#print(x_train_labels)
#print(x_test_labels)
```



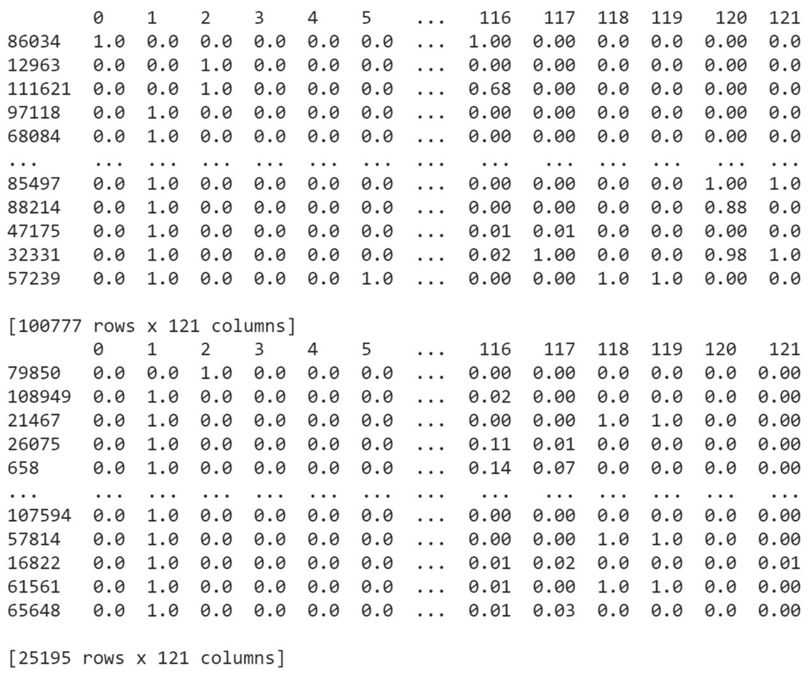
```
Training

(100777, 11, 11)
[[0.0000000e+00 1.1322316e-03 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  1.1322316e-03 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  1.1322316e-03 0.0000000e+00 0.0000000e+00 2.1727008e-07 1.7945133e-05
  0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 1.1322316e-03
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 3.8642715e-06 3.6173533e-06
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 1.1322316e-03
  0.0000000e+00]
 [0.0000000e+00 1.1322316e-03 1.1322316e-03 1.1322316e-03 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00]]
```

```
Testing

(25195, 11, 11)
[[0.00000000e+00 1.13223158e-03 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 1.13223158e-03 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 1.13223158e-03 0.00000000e+00 0.00000000e+00
  6.76845652e-07 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  1.13223158e-03 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 1.80675253e-05
  1.38640602e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 1.13223158e-03 0.00000000e+00]
 [5.66115792e-04 7.78436009e-04 1.61256923e-04 2.37768633e-04
  8.88024772e-05 2.37768633e-04 0.00000000e+00 1.13223158e-05
  0.00000000e+00 2.40900337e-05 0.00000000e+00]]
```

```
"""# Converting Normalized Data"""

#converting each row of data points to 11*11 matrices

#train

df=x_train
arr = df.to_numpy().astype(np.float32)
for i in range(0,len(arr)):
  s=int(x_train_labels[i])
  s=w[s]
  arr[i]=arr[i]*s
arr=scaling(arr)

arr
train_data=[]
train_labels=np.array(x_train_labels).astype(np.float32)
for i in range(0,len(arr)):
  train_data.append(arr[i].reshape(11,11))

train=np.array(train_data)

#test

df1=x_test
arr1 = df1.to_numpy()
for i in range(0,len(arr1)):
  s=int(x_test_labels[i])
  s=w[s]
  arr1[i]=arr1[i]*s
arr1=scaling(arr1)

arr1
test_data=[]
test_labels=np.asarray(x_test_labels).astype(np.float32)
for i in range(0,len(arr1)):
  test_data.append(arr1[i].reshape(11,11))

test=np.array(test_data)
```

"""**Preprocessed Dataset**"""

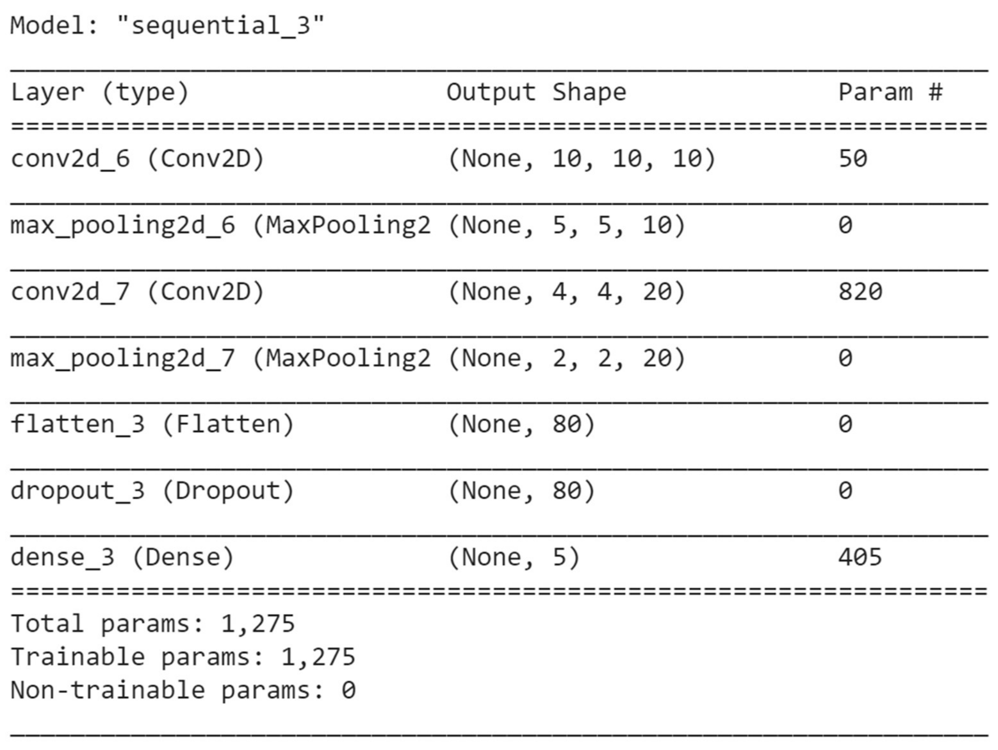
```
print("Training\n")
print(train.shape)
print(train[0])
print("\nTesting\n")
print(test.shape)
print(test[0])
```



```
        0    1    2    3    4    5    ...   116   117  118  119   120  121
86034   1.0  0.0  0.0  0.0  0.0  0.0  ...  1.00  0.00  0.0  0.0  0.00  0.0
12963   0.0  0.0  1.0  0.0  0.0  0.0  ...  0.00  0.00  0.0  0.0  0.00  0.0
111621  0.0  0.0  1.0  0.0  0.0  0.0  ...  0.68  0.00  0.0  0.0  0.00  0.0
97118   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  0.0  0.0  0.00  0.0
68084   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  0.0  0.0  0.00  0.0
...     ...  ...  ...  ...  ...  ...  ...   ...   ...  ...  ...   ...  ...
85497   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  0.0  0.0  1.00  1.0
88214   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  0.0  0.0  0.88  0.0
47175   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.01  0.01  0.0  0.0  0.00  0.0
32331   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.02  1.00  0.0  0.0  0.98  1.0
57239   0.0  1.0  0.0  0.0  0.0  1.0  ...  0.00  0.00  1.0  1.0  0.00  0.0

[100777 rows x 121 columns]
        0    1    2    3    4    5    ...   116   117  118  119  120   121
79850   0.0  0.0  1.0  0.0  0.0  0.0  ...  0.00  0.00  0.0  0.0  0.0  0.00
108949  0.0  1.0  0.0  0.0  0.0  0.0  ...  0.02  0.00  0.0  0.0  0.0  0.00
21467   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  1.0  1.0  0.0  0.00
26075   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.11  0.01  0.0  0.0  0.0  0.00
658     0.0  1.0  0.0  0.0  0.0  0.0  ...  0.14  0.07  0.0  0.0  0.0  0.00
...     ...  ...  ...  ...  ...  ...  ...   ...   ...  ...  ...  ...   ...
107594  0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  0.0  0.0  0.0  0.00
57814   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.00  0.00  1.0  1.0  0.0  0.00
16822   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.01  0.02  0.0  0.0  0.0  0.01
61561   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.01  0.00  1.0  1.0  0.0  0.00
65648   0.0  1.0  0.0  0.0  0.0  0.0  ...  0.01  0.03  0.0  0.0  0.0  0.00

[25195 rows x 121 columns]
```

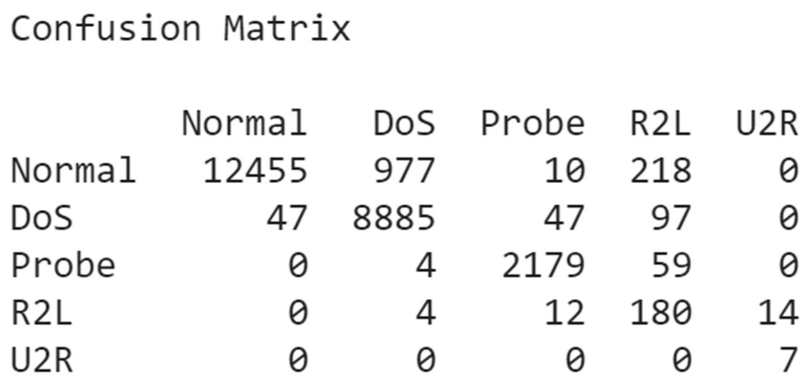
"""**#CNN Model**"""

```
model = models.Sequential()
initializer=tf.keras.initializers.GlorotNormal(seed=None)

model.add(layers.Conv2D(10, (2, 2), activation='relu', input_shape=(11, 11,
1),kernel_initializer=initializer,bias_initializer=initializers.Zeros()))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(20, (2, 2),
activation='relu',kernel_initializer=initializer,bias_initializer=initializ
ers.Zeros()))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dropout(0.1))
model.add(layers.Dense(5, activation='softmax'))

model.summary()
```



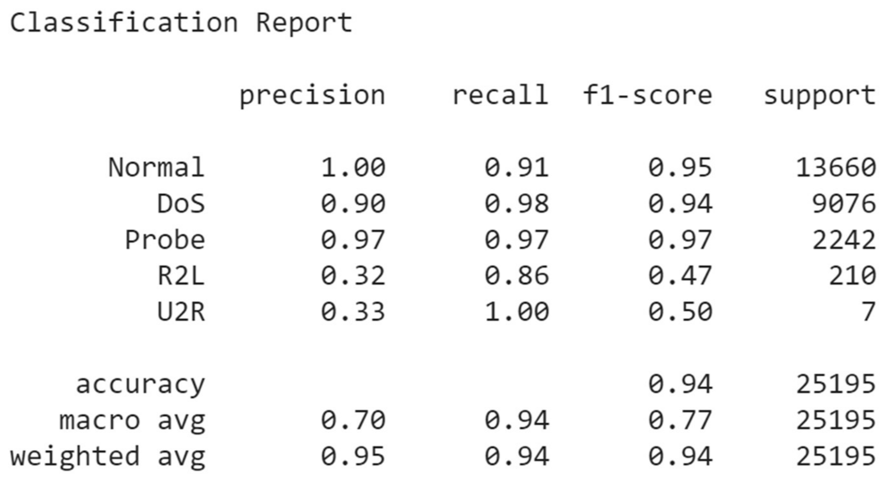
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 10, 10, 10)        50
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 5, 5, 10)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 4, 4, 20)          820
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 2, 2, 20)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 80)                0
_________________________________________________________________
dropout_3 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 405
=================================================================
Total params: 1,275
Trainable params: 1,275
Non-trainable params: 0
_________________________________________________________________
```

```
"""#Fitting"""

from tensorflow import keras

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import RMSprop

INIT_LR = 7e-3
EPOCHS = 100
BS = 64

opt = Adam(lr=INIT_LR,decay=INIT_LR/EPOCHS)
model.compile(optimizer=opt,loss="sparse_categorical_crossentropy",metrics=
["accuracy"])

train=tf.reshape(train,[-1,11,11,1])
test=tf.reshape(test,[-1,11,11,1])

history = model.fit(train, train_labels,
epochs=100,steps_per_epoch=len(train)//BS, validation_data=(test,
test_labels),shuffle=True)

"""Plotting"""

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
plt.figure(figsize=(10,6))
plt.plot(epochs,acc)
plt.plot(epochs,val_acc)
plt.figure(figsize=(10,6))
plt.plot(epochs,loss)
plt.plot(epochs,val_loss)

"""Predicted values"""

pred=model.predict(test)
y_pred=[]
for i in range (0,len(pred)):
  y_pred.append(np.argmax(pred[i]))


"""Confusion Matrix and Classification Report"""

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix

print('\nConfusion Matrix\n')
print(pd.DataFrame(confusion_matrix(test_labels,y_pred),index=['Normal',
'DoS', 'Probe', 'R2L', 'U2R'],columns=['Normal', 'DoS', 'Probe', 'R2L',
'U2R']))
```



```
Confusion Matrix

        Normal   DoS  Probe  R2L  U2R
Normal   12455   977     10  218    0
DoS         47  8885     47   97    0
Probe        0     4   2179   59    0
R2L          0     4     12  180   14
U2R          0     0      0    0    7
```

```
print('\nClassification Report\n')
print(classification_report(test_labels, y_pred, target_names=['Normal',
'DoS', 'Probe', 'R2L', 'U2R']))
```



```
Classification Report

              precision    recall  f1-score   support

      Normal       1.00      0.91      0.95     13660
         DoS       0.90      0.98      0.94      9076
       Probe       0.97      0.97      0.97      2242
         R2L       0.32      0.86      0.47       210
         U2R       0.33      1.00      0.50         7

    accuracy                           0.94     25195
   macro avg       0.70      0.94      0.77     25195
weighted avg       0.95      0.94      0.94     25195
```

```
cnf_matrix=confusion_matrix(test_labels,y_pred)
FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)
```

```
FAR= FP/(FP+TN)
DR= TP/(TP+FN)
attacks=['Normal','DoS','Probe','R2L','U2R']
#print(FAR)
fig = plt.figure(figsize = (10, 7))

# creating the bar plot
plt.bar(attacks, DR, color ='purple', width = 0.4)
plt.ylabel("Detection Rate")
plt.xlabel("Attack Labels")
plt.title("Detection Rate")
plt.show()

fig = plt.figure(figsize = (10, 7))

# creating the bar plot
plt.bar(attacks, FAR, color ='pink', width = 0.4)
plt.ylabel("False Alarm Rate")
plt.xlabel("Attack Labels")
plt.title("False Alarm Rate")
plt.show()
```

CHAPTER - IV

RESULTS & DISCUSSION

4.1 Accuracy: It is the ratio of the number of correct predictions to the total number of input samples. A higher accuracy implies a better performance in a model. It is calculated by:

$$\frac{TP+TN}{TP+FP+TN+FN}$$

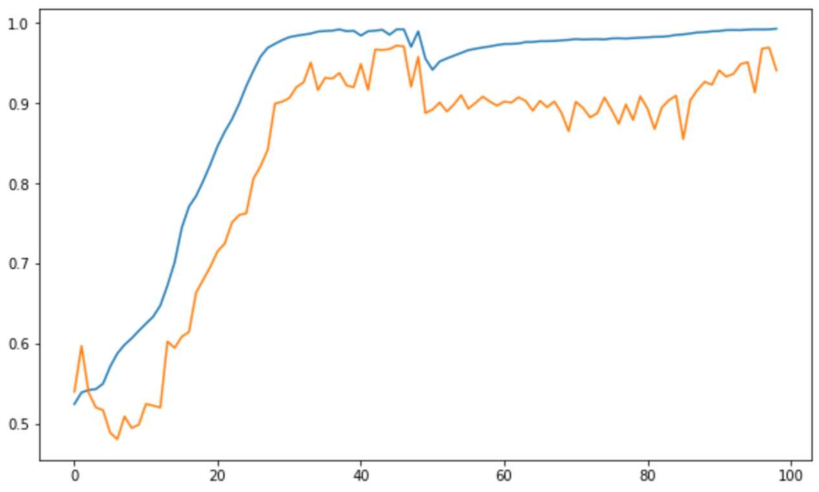


Fig-2: *Accuracy of the Model vs number of Epochs*

4.2 Loss: Sparse Categorical Cross Entropy calculates the compares each of the predicted probabilities to actual class output. A lower loss implies a better model.

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

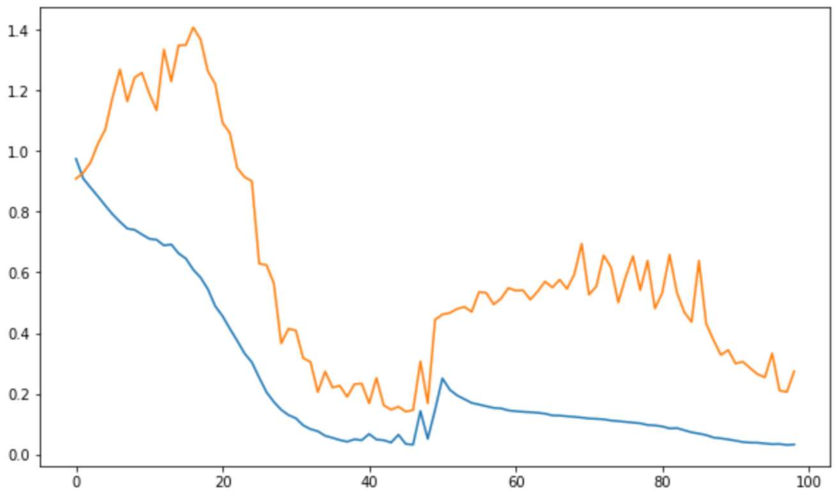


Fig-3: *Loss of the Model vs number of Epochs*

4.3 Confusion Matrix:

Table-1: *Confusion Matrix*

	Normal	**DoS**	**Probe**	**R2L**	**U2R**
Normal	**12455**	977	10	218	0
DoS	47	**8885**	47	97	0
Probe	0	4	**2179**	59	9
R2L	0	4	12	**180**	14
U2R	0	0	0	0	**7**

4.4 Classification Report:

Table-2: *Classification Report*

	Precision	**Recall**	**F1 - score**	**Support**
Normal	1.00	0.91	0.95	13660
DoS	0.90	0.98	0.94	9076
Probe	0.97	0.97	0.97	2240
R2L	0.32	0.86	0.47	210
U2R	0.33	1.00	0.50	7

Training Accuracy: 99.3% **Validation Accuracy**: 94.09%

Detection Rate and False Alarm Rates:

- **Detection Rate (DR)** is defined as the proportion of the whole sample where the events were detected correctly. The higher the DR, the better the model performance. It can be calculated as:

$$\frac{TP}{TP+FN}$$

- **False Alarm Rate (FAR)** is defined as the expectancy of the false positive ratio. For any Machine/Deep Learning model, a lower false alarm rate is preferred. It is calculated as:

$$\frac{FP}{FP+TN}$$

The proposed model provides an excellent detection rate (over 90%) and a very low false alarm rate (less than 6%) for each class. They are described in the figures below:

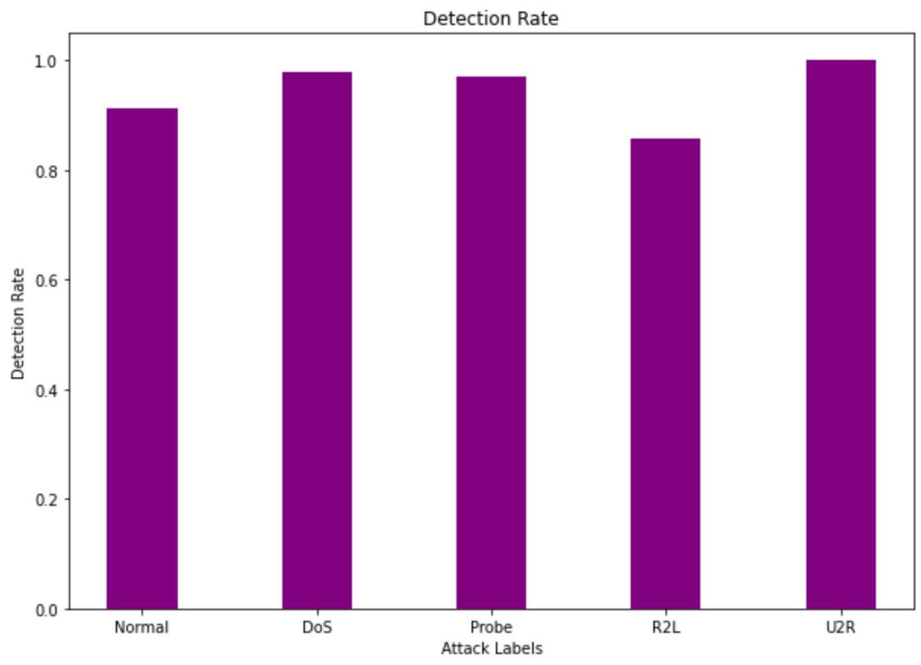


Fig-4: *Detection Rates*

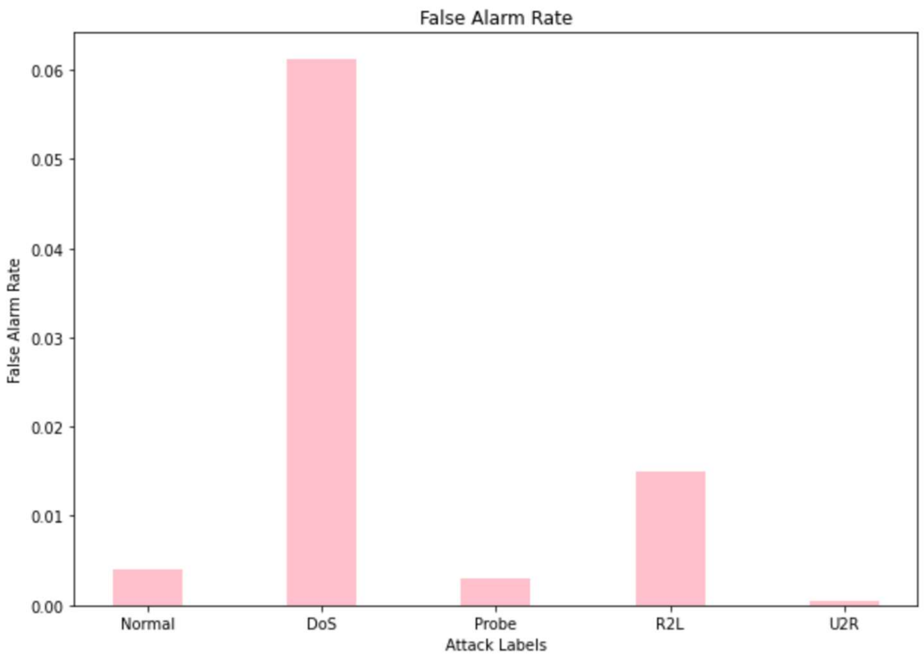


Fig-5: *False Alarm Rates*

CHAPTER - V

CONCLUSION

5.1 Conclusion

This project proposed a novel approach to intrusion detection, using Deep Learning - Convolutional Neural Networks on the NSL-KDD dataset. In this approach, non-numerical features were converted into numerical ones by one-hot-encoding. Then, the features were scaled using min-max normalization. Then, the one-dimensional data (containing attributes like network protocol, flag features, etc.) into two-dimensional image data.

This project also addressed the dataset imbalance problem that previous models did not, by adjusting the cost function of each class. The model provided higher accuracy (99.3%) and detection rates and lower false alarm rates than its predecessors.

This approach was carried out on a dataset with over 120,000 samples, proving that it can be implemented in real-time for massive networks.

5.1 Future Scope

- Increasing the detection rate and precision of R2L and U2R attacks.
- Decreasing the false alarm rate of DoS attacks.
- Improving the F1-score of R2L and U2R attacks.

REFERENCES

[1] W. Stallings, Cryptography and Network Security: Principles and Practice,vol. 11, no. 7. Englewood Cliffs, NJ, USA: Prentice-Hall, 1999, pp. 655-660.

[2] S. Ganapathy, K. Kulothungan, S. Muthurajkumar, M. Vijayalakshmi, P. Yogesh, and A. Kannan, ``Intelligent feature selection and classification techniques for intrusion detection in networks: A survey," EURASIP J. Wireless Commun. Netw., vol. 2013, no. 1, pp. 116, 2013.

[3] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos, ``Feature selection for high-dimensional data," Prog. Artif. Intell., vol. 5, no. 2, pp. 65-75, 2016.

[4] H. Shi, H. Li, D. Zhang, C. Cheng, and W. Wu, ``Efficient and robust feature extraction and selection for traffic classification," Comput. Netw. Int. J. Comput. Telecommun. Netw., vol. 119, pp. 116, Jun. 2017.

[5] Y. Lecun, Y. Bengio, and G. Hinton, ``Deep learning," Nature, vol. 521, no. 7553, pp. 436444, 2015.

[6] A. Chaudhary, V. N. Tiwari, and A. Kumar, ``Analysis of fuzzy logic based intrusion detection systems in mobile ad hoc networks," in Proc. Elect. Perform. Electron. Package., 2014, pp. 690696.

[7] H.Wang, J. Gu, and S.Wang, ``An effective intrusion detection framework based on SVM with feature augmentation," Knowl.-Based Syst., vol. 136, pp. 130139, Nov. 2017.

[8] S. S. S. Sindhu, S. Geetha, and A. Kannan, ``Decision tree based lightweight intrusion detection using a wrapper approach," Expert Syst. Appl., vol. 39, no. 1, pp. 129141, 2012.

[9] A. S. Eesa, Z. Orman, and A. M. A. Brifcani, ``A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems," Expert Syst. Appl., vol. 42, no. 5, pp. 26702679, 2015

[10] G. Wang, J. Hao, J. Ma, and L. Huang, ``A new approach to intrusion detection using artificial neural networks and fuzzy clustering," Expert Syst. Appl., vol. 37, no. 9, pp. 62256232, 2010.

[11] S. Mukherjee and N. Sharma, ``Intrusion detection using Naïve Bayes classifier with feature reduction," Procedia Technology., vol. 4, no. 11, pp. 119128, 2012.

[12] M. A. Salama, H. F. Eid, R. A. Ramadan, A. Darwish, and A. E. Hassanien, ``Hybrid intelligent intrusion detection scheme," in Soft Computing in Industrial Applications. Berlin, Germany: Springer, 2011, pp. 293303

[13] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, ``Long short term memory recurrent neural network classifier for intrusion detection," in Proc. Int. Conf. Platform Service, 2016, pp. 15.

[14] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, ``Deep learning approach for network intrusion detection in software defined networking," in Proc. Int. Conf. Wireless Netw. Mobile Commun., 2016, pp. 258263.

[15] L. Peng, H. Zhang, Y. Chen, B. Yang, ``Imbalanced traffic identification using an imbalanced data gravitation-based classification model," Comput. Commun., vol. 102, pp. 177189, 2016.

BIODATA

Name	: Anila A Joseph
Mobile Number	: 86987 29763
E-mail	: anilaa.joseph2018@vitstudent.ac.in
Permanent Address	: 1B, Bhaggyam Enclave, 23, North Mada Street, Srinagar Colony, Saidapet, Chennai-15.

Name	: Andrew John
Mobile Number	: 90037 35112
E-mail	: andrewjohn.j2018@vitstudent.ac.in
Permanent Address	: E405, Radiance Mandarin 200 Feet Thoraipakkam - Pallavaram Radial Road Thoraipakkam, Chennai-97.

Name	: Vighnesh M
Mobile Number	: 81481 66514
E-mail	: vighnesh.m2018@vitstudent.ac.in
Permanent Address	: 4A/25, Andavar Street, Choolaimedu, Chennai-94.

DEMO OF THE PROJECT:

https://drive.google.com/file/d/1yYy-LKvvlgkH3FuuxIyM_CSEX_9G5pUK/view?usp=sharing