TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS AND DEFINITIONS

SNO	ACRONYM	DEFINITION
01	DNN	Deep Neural Network
02	HIDS	Host based IDS
03	ITC	Information technology & Communication
04	IDS	Intrusion Detection System
05	KDD	Knowledge Discovery Database
06	NIDS	Network based IDS
07	SHIA	Scale-Hybrid-IDS-Alert-Net
08	UML	Unified Modelling Language

1. INTRODUCTION

1.1 NECESSITY OF INTRUSION DETECTION SYSTEM

Information and communications technology (ICT) systems and networks carry different data from various users to prevent attacks. These attacks can be manual and machine-generated, diverse and are gradually advancing in obfuscations resulting in undetected data breaches. With the advancement of hardware, software, and network topologies, including recent advances in the Internet of Things, cyberattacks are constantly evolving with every algorithm. Malicious cyberattacks pose significant security risks, necessitating the development of a new, versatile, and effective system. An intrusion detection system (IDS) is a proactive intrusion detection method that detects and classifies intrusions, attacks, and violations of security policies at the network and host level infrastructure in real-time. Fig 1.1 is an example of Intrusion Detection system.

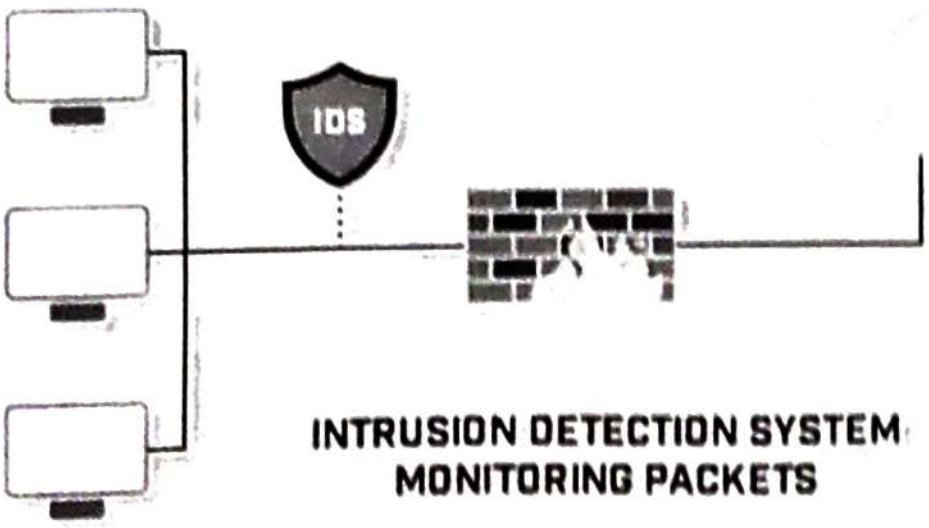

Fig.1.1: Intrusion Detection System Model

1.2 TYPES OF INTRUSION DETECTION SYSTEMS

Intrusion detection is divided into two categories:

- **Network-based intrusion detection systems (NIDS):** The NIDS can monitor incoming, outgoing, and local traffic. Inspecting outgoing or local traffic can yield valuable insight into malicious activities, as can inspecting incoming traffic. Some attacks can originate and stay with the local network or be staged inside the network with an outside-the-network target. The NIDS also works with other systems, like a firewall, to help better protect against known attack sources (e.g., a suspected attacker IP address).

- **Host-based intrusion detection systems (HIDS):** A host-based intrusion detection system (HIDS) is a system that monitors a computer system on which it is installed to detect an intrusion and/or misuse, and responds by logging the activity and notifying the designated authority. A HIDS can be thought of as an agent that monitors and analyzes whether anything or anyone, whether internal or external, has circumvented the system's security policy.

1.3 DIFFERENT APPLICATIONS OF INTRUSION DETECTION SYSTEMS

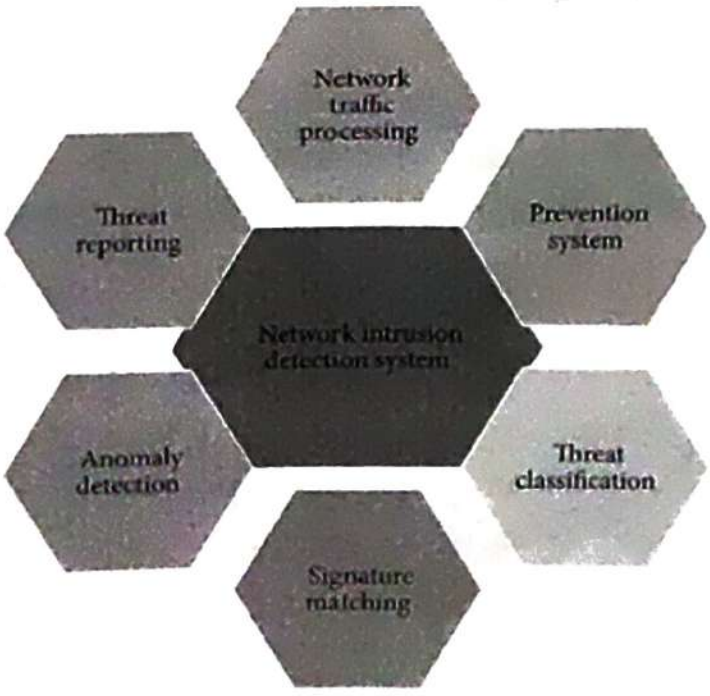

Fig.1.2: Different Application of Intrusion Detection System

In above Fig 1.2 shows the different applications of Intrusion Detection System.

- **Network Traffic Processing**

The Internet has opened new avenues for information accessing and sharing in a variety of media formats. Such popularity has resulted in an increase in the number of resources consumed in backbone links, whose capacities have witnessed numerous upgrades to cope with the ever-increasing demand for bandwidth. Consequently, network traffic processing at today's data transmission rates is a very demanding task, which has been traditionally accomplished utilizing specialized hardware tailored to specific tasks. However, such approaches lack either flexibility or extensibility—of both. As an alternative, the research community has pointed to the utilization of commodity hardware, which may provide flexible and extensible cost-aware solutions, ergo entailing large reductions of the operational and capital expenditure investments. In this chapter, we provide a survey-like introduction to high-performance network traffic processing using commodity hardware. We present the required background to understand the different solutions proposed in the literature to achieve high-speed lossless packet capture, which is reviewed and compared.

- **Prevention System**

Prevention System is also known as Intrusion Detection, is a network security application that monitors network or system activities for malicious activity. The major functions of intrusion prevention systems are to identify malicious activity, collect information about this activity, report it and attempt to block or stop it. Intrusion prevention systems are contemplated as augmentation of Intrusion Detection Systems (IDS) because both IPS and IDS operate network traffic and system activities for malicious activity. IPS typically record information related to observed events, notify security administrators of important observed events and produce reports. Many IPS can also respond to a detected threat by attempting to prevent it from succeeding. They use various response techniques, which involve the IPS stopping the attack itself, changing the security environment or changing the attack's content.

- **Threat Classification**

Threats classifications are important because they mainly allow identifying and understanding threats, characteristics and sources to protect systems assets. Moreover, it articulates the security risks that threaten these systems and assists in understanding the capabilities and selection of security solutions. A literature review has identified several attempts of classifications. In this section, we present an overview of the most commonly used information security threat classifications. A threat is the adversary's goal, or what an adversary might try to do to a system. It is also described as the capability of an adversary to attack a system. Thus, a threat may be defined in two ways: techniques that attackers use to exploit the vulnerabilities in your system components or the impact of threats to your assets. Therefore, we can categorize threat classification approaches into two main classes: x Classification methods that are based on attacks techniques x Classification methods that are based on threats impacts.

- **Signature Matching**

Software reuse is only effective if it is easier to locate (and appropriately modify) a reusable component than to write it from scratch. Signature matching is a method for achieving this goal by using signature information easily derived from the component. We consider two kinds of software components, functions and modules, and hence two kinds of matching, function matching and module matching. The signature of a function is simply its type; the signature of a module is a multiset of user-defined types and a multiset of function signatures. For both functions and modules, we consider not just exact match, but also various flavours of relaxed match.

- **Anomaly Detection**

Successful anomaly detection hinges on an ability to accurately analyze time-series data in real-time. Time series data is composed of a sequence of values over time. That means each point is typically a pair of two items — a timestamp for when the metric was measured, and the value associated with that metric at that time. Time series data isn't a projection in and of itself. Rather, it's a record that contains the information necessary for making educated guesses about what can be reasonably expected in the future. Anomaly detection systems use those expectations to identify actionable signals within your data, uncovering outliers to alert you to key events in your organization

With all the analytics programs and various management software available, it's easier than ever for you to effectively measure every single aspect of business activity. That includes the operational performance of applications and infrastructure components as well as key performance indicators (KPIs) that evaluate the success of your business.

- **Threat Reporting**

Threat intelligence reporting, or cyber threat intelligence, is information an organization uses to understand the threats that have, will, or are currently targeting the organization. This info is used to prepare, prevent, and identify cyber threats looking to take advantage of valuable resources. The great unknown; can be exciting in many situations, but in a world where any number of cyber threats could bring an organization to its knees, it can be downright terrifying. Threat intelligence can help organizations gain valuable knowledge about these threats, build effective defence mechanisms and mitigate the risks that could damage their bottom line and reputation. After all, targeted threats require targeted defence, and cyber threat intelligence delivers the capability to defend more proactively. While the promise of cyber threat intel is alluring in itself, it is important to understand how it works so you can choose the right cyber threat tools and solutions to protect your business.

2. LITERATURE SURVEY

2.1 SURVEY ON BACKGROUND

The performance of IDS depends highly on data features. Selecting the most informative features eliminating the redundant and irrelevant features from network traffic data for IDS is still an open research issue. The key impetus of this paper is to identify and benchmark the potential set of features that can characterize network traffic for intrusion detection. In this correspondence, an ensemble approach is proposed. As a first step, the approach applies four different feature evaluation measures, such as correlation, consistency, information, and distance, to select the more crucial features for intrusion detection. Second, it applies the subset combination strategy to merge the output of the four measures and achieve the potential feature set. Along with this, a new framework that adopts the data analytic lifecycle practices is explored to employ the proposed ensemble for building an effective IDS. The effectiveness of the proposed approach is demonstrated by conducting several experiments on four intrusion detection evaluation datasets, namely KDDCup'99, NSL-KDD, UNSW-NB15, and CICIDS2017 [1].

The increasing number of cyberattacks in recent years has expedited development of innovative tools to quickly detect new threats. A recent approach to this problem is to analyze the content of online social networks to discover the rising of ransomware attacks. Twitter is a popular micro-blogging platform which allows millions of users to share their opinions on what happens all over the world. The subscribers can tweet messages of maximum 280 characters to share general information with URLs and hash tags. In this paper, we analysed 25 families of ransomware over a period of 7 years, from 2010 to 2017. They proposed a deep learning architecture to categorize ransomware tweets to their corresponding family. The proposed method can continuously monitor the online posts in social media data and thus is able to provide early warnings about ransomware spreads [2].

Considering the characteristics of network traffic on the data link layer, such as massive high-speed data flow, information camouflaged easily, and the phenomenon that abnormal traffic is much smaller than the normal one, an intrusion detection system (IDS) based on the quantitative model of interaction mode between ports is proposed.

The model gives the quantitative expression of Port Interaction Mode in Data Link Layer (PIMDL), focusing on improving the accuracy and efficiency of the intrusion detection by taking the arrival time distribution of traffic. The feasibility of the model proposed is proved by the phase space reconstruction and visualization method. According to the characteristics of long and short sessions, a neural network based on CNN and LSTM is designed to mine the differences between normal and abnormal models. On this basis, an improved Intrusion Detection algorithm based on a multi-model scoring mechanism is designed to classify sessions in model space [3].

One of the major challenges in cybersecurity is the provision of an automated and effective cyber-threats detection technique. In this paper, we present an AI technique for cyber-threats detection, based on artificial neural networks. The proposed technique converts multitude of collected security events to individual event profiles and use a deep learning-based detection method for enhanced cyber-threat detection. For this work, we developed an AI-SIEM system based on a combination of event profiling for data preprocessing and different artificial neural network methods, including FCNN, CNN, and LSTM. The system focuses on discriminating between true positive and false positive alerts, thus helping security analysts to rapidly respond to cyber threats. All experiments in this study are performed by authors using two benchmark datasets (NSLKDD and CICIDS2017) and two datasets collected in the real world. To evaluate the performance comparison with existing methods, we conducted experiments using the five conventional machine-learning methods (SVM, k-NN, RF, NB, and DT [4].

The tremendously growing problem of phishing e-mail, also known as spam including spear phishing or spam borne malware, has demanded a need for reliable intelligent anti-spam e-mail filters. This survey paper describes a focused literature survey of Artificial Intelligence (AI) and Machine Learning (ML) methods for intelligent spam email detection, which we believe can help in developing appropriate countermeasures. In this paper, we considered 4 parts in the email's structure that can be used for intelligent analysis: (A) Headers Provide Routing Information, contain mail transfer agents (MTA) that provide information like email and IP address of each sender and recipient of where the email originated and what stopovers, and final destination. (B) The SMTP Envelope, containing mail exchangers' identification, originating source and destination domains\users. (C) First part of SMTP Data, containing information

like from, to, date, subject - appearing in most email clients (D) Second part of SMTP Data, containing email body including text content, and attachment. Based on the number the relevance of an emerging intelligent method, papers representing each method were identified, read, and summarized [5].

Network intrusion detection systems (NIDS) are essential tools in ensuring network information security, and neural networks have become an increasingly popular solution for NIDS. However, with the gradual complexity of the network environment, the existing solutions using the conventional neural network cannot make full use of the rich information in the network traffic data due to its single structure. More importantly, this will lead to the existing NIDS have incomplete knowledge of the intrusion detection domain, and making it unable to achieve a high detection rate and good stability in the new environment. In this paper, we take a step forward and extract the different level features from the network connection, rather than a long feature vector used in the traditional approach, which can process feature information separately more efficiently. And further, we propose multimodal-sequential intrusion detection approach with special structure of hierarchical progressive network, which is supported by multimodal deep auto encoder (MDAE) and LSTM technologies. By design the special structure of hierarchical progressive network, our approach can efficiently integrate the different level features information within a network connection and automatically learn temporal information between adjacent network connections at the same time [6].

Powerful weapon in today's world is ones emotion is social media. They have the power to make an individual trending overnight or even may pull down anyone reputation. In this paper, the sentiment analysis task has been performed by collecting the dataset from the publically available sources and by merging them together to form a new dataset for the sentiment analysis task that is positive or negative sentiment based on the context of the subject. A new reliable dataset is subjected to various pre-processing techniques and then the feature extraction techniques aftermath they are passed to the deep learning techniques out of which by using the text representation method, global vectors (glovec) with the long short-term memory (lstm) has the highest accuracy of 75%, which is the benchmark accuracy for this dataset. For the research purpose the dataset used in this paper is made available publically for research purpose [7].

A Web attack protection system is extremely essential in today's information age. Classifier ensembles have been considered for anomaly-based intrusion detection in Web traffic. However, they suffer from an unsatisfactory performance due to a poor ensemble design. This paper proposes a stacked ensemble for anomaly-based intrusion detection systems in a Web application. Unlike a conventional stacking, where some single weak learners are prevalently used, the proposed stacked ensemble is an ensemble architecture, yet its base learners are other ensembles learners, i.e. random forest, gradient boosting machine, and XGBoost. To prove the generalizability of the proposed model, two datasets that are specifically used for attack detection in a Web application, i.e. CSIC-2010v2 and CICIDS-2017 are used in the experiment. Furthermore, the proposed model significantly surpasses existing Web attack detection techniques concerning the accuracy and false positive rate metrics. Validation result on the CICIDS-2017, NSL-KDD, and UNSW-NB15 dataset also ameliorate the ones obtained by some recent techniques [8].

In cyberspace, anomalies including intentional attacks grow up in their size and diversity. Although using the Intrusion Detection System (IDS) as a solution is helpful to some degree, there is an unsolved problem; the low performance of IDS due to lack of enough attack data. Recent approaches to solving this problem use an unsupervised deep learning-based technique called Generative Adversarial Networks (GANs). Because GAN variants show great performance in image augmentation, some research tries to apply GANs to cyberspace by domain conversion from binary to image. However, the attribute of cyberspace benchmarks is different from that of images. In this paper, we propose using sequence-based generative models such as Sequence Generative Adversarial Nets (SeqGAN) and Sequence to Sequence (Seq2Seq) to augment the ADFA-LD dataset, a sequence call based benchmark. Experimental results show that the performance is better when training ADFA-LD with augmented data from SeqGAN and Seq2Seq than training only the original dataset [9].

To explore the advantages of adversarial learning and deep learning, we propose a novel network intrusion detection model called SAVAER-DNN, which can not only detect known and unknown attacks but also improve the detection rate of low-frequent attacks. SAVAER is a supervised variational auto-encoder with regularization, which uses WGAN-GP instead of the vanilla GAN to learn the latent distribution of the original data. SAVAER's decoder is used to synthesize samples of low-frequent and

unknown attacks, thereby increasing the diversity of training samples and balancing the training data set. SAVAER's encoder is used to initialize the weights of the hidden layers of the DNN and explore high-level feature representations of the original samples. The benchmark NSL-KDD (KDDTest+), NSL-KDD (KDDTest-21) and UNSW-NB15 datasets are used to evaluate the performance of the proposed model. The experimental results show that the proposed SAVAER-DNN is more suitable for data augmentation than the other three well-known data oversampling methods. Moreover, the proposed SAVAER-DNN outperforms eight well-known classification models in detection performance and is more effective in detecting low-frequent and unknown attacks [10].

In this paper, they present an Intrusion Detection System (IDS) using the hybridization of the deep learning technique and the multi-objective optimization method for the detection of Distributed Denial of Service (DDoS) attacks in the Internet of Things (IoT) networks is proposed in this paper. IoT networks consist of different devices with unique hardware and software configurations communicating over different communication protocols, which produce huge multidimensional data that make IoT networks susceptible to cyber-attacks. In a network the IDS is a vital tool for securing it from cyber-attacks. Detection of new emerging cyber threats are becoming difficult for existing IDS, and therefore advanced IDS is required. A DDoS attack is a cyber-attack that has posed substantial devastating losses in IoT networks recently. In this paper, we propose an IDS founded on the fusion of a Jumping Gene adapted NSGA-II multi-objective optimization method for data dimension reduction and the Convolutional Neural Network (CNN) integrating Long Short-Term Memory (LSTM) deep learning techniques for classifying the attack [11].

Existing technology trends have led to an abundance of household items containing microprocessors all connected within a private network. Thus, network intrusion detection is essential for keeping these networks secure. However, network intrusion detection can be extremely taxing on battery operated devices. Thus, this work presents a cyberattack detection system based on a multilayer perceptron neural network algorithm. To show this system is capable of operating at low power, the algorithm was executed on two commercially available minicomputer systems including the Raspberry PI 3 and the Asus Tinkerboard. An accuracy, power, energy, and timing analysis was performed to study the tradeoffs necessary when executing

these algorithms at low power. Our results show that these low power implementations are feasible, and a scan rate of more than 226,000 packets per second can be achieved from a system that requires approximately 5W to operate with greater than 99% accuracy [12].

With the tremendous growth of the internet, cyberspace is facing several threats from the attackers. Threats like spam emails account for 55% of total emails according to the Symantec monthly threat report. Over time, the attackers moved on to image spam to evade the text-based spam filters. To deal with this, the researchers have several machine learning and deep learning approaches that use various features like metadata, color, shape, texture features. But the Deep Convolutional Neural Network (DCNN) and transfer learning-based pre-trained CNN models are not explored much for Image spam classification. Therefore, in this work, 2 DCNN models along with few pre-trained ImageNet architectures like VGG19, Xception are trained on 3 different datasets. The effect of employing a Cost-sensitive learning approach to handle data imbalance is also studied. Some of the proposed models in this work achieves an accuracy up to 99% with zero false positive rate in best case [13].

Networks had an increasing impact on modern life since network cybersecurity has become an important research field. Several machine learning techniques have been developed to build network intrusion detection systems for correctly detecting unforeseen cyber-attacks at the network-level. For example, deep artificial neural network architectures have recently achieved state-of-the-art results. In this paper a novel deep neural network architecture is defined, in order to learn flexible and effective intrusion detection models, by combining an unsupervised stage for multi-channel feature learning with a supervised one exploiting feature dependencies on cross channels. The aim is to investigate whether class-specific features of the network flows could be learned and added to the original ones in order to increase the model accuracy. In particular, in the unsupervised stage, two autoencoders are separately learned on normal and attack flows, respectively. As the top layer in the decoder of these autoencoders reconstructs samples in the same space as the input one, they could be used to define two new feature vectors allowing the representation of each network flow as a multi-channel sample. In the supervised stage, a multi-channel parametric convolution is adopted, in order to learn the effect of each channel on the others [14].

Intrusion detection is one of the most prominent and challenging problem faced by cybersecurity organizations. Intrusion Detection System (IDS) plays a vital role in identifying network security threats. It protects the network for vulnerable source code, viruses, worms and unauthorized intruders for many intranet/internet applications. Despite many open source APIs and tools for intrusion detection, there are still many network security problems exist. These problems are handled through the proper pre-processing, normalization, feature selection and ranking on benchmark dataset attributes prior to the enforcement of self-learning-based classification algorithms. In this paper, we have performed a comprehensive comparative analysis of the benchmark datasets NSL-KDD and CIDDS-001. For getting optimal results, we have used the hybrid feature selection and ranking methods before applying self-learning (Machine / Deep Learning) classification algorithmic approaches such as SVM, Naïve Bayes, k-NN, Neural Networks, DNN and DAE [15].

2.2 Conclusion on survey

In this project, we have presented, in detail, a survey of intrusion detection system methodologies, types, and technologies with their advantages and limitations. Several machine learning techniques that have been proposed to detect zero-day attacks are reviewed. However, such approaches may have the problem of generating and updating the information about new attacks and yield high false alarms or poor accuracy. In addition, the most popular public datasets used for IDS research have been explored and their data collection techniques, evaluation results and limitations have been discussed. As normal activities are frequently changing and may not remain effective over time, there exists a need for newer and more comprehensive datasets that contain wide-spectrum of malware activities

3. SYSTEM DESIGN

3.1 UML DIAGRAMS

UML (Unified Modeling Language) is a standard language for specifying, visualizing, constructing, and documenting the artefacts of software systems.

3.1.1 USE CASE DIAGRAM

Use case diagrams in Unified Modelling Language (UML)are usually referred to as behaviour diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Here the user will interact with the system and perform a set actions to get the prediction results for the output.

In below Fig.3.1 the user interacts with system to perform various actions

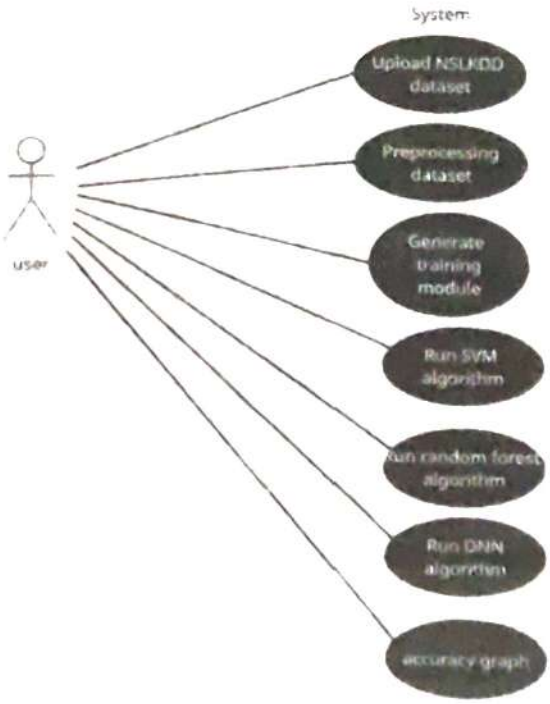

Fig.3.1: Use Case Diagram for Intrusion Detection System

3.1.2 CLASS DIAGRAM

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts.

The upper part holds the name of the class. The middle part contains the attributes of the class. The bottom part gives the methods or operations the class can take or undertake

In the below fig 3.2 class diagram is interaction between the two classes called IDS and TEST. In IDS where the data is given as input where as in Test the data processing and modules are being executed and output is generated.

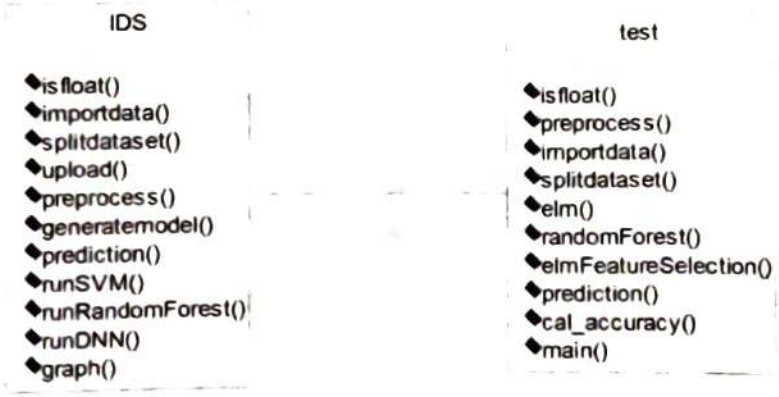

Fig.3.2: Class Diagram for Intrusion Detection System

3.2 REQUIRMENTS

The project involved analyzing the design of few applications so as to make the application more user friendly. To do so, it was really important to keep the navigations from one screen to the other well-ordered and at the same time reducing the amount of typing the user needs to do.

3.2.1 Hardware Requirements

- Processor - Intel(R) Xeon(R)
- RAM - 16 GB
- Hard Disk - 1 TB

3.2.2 Software Requirements

For developing the application, the following are the software requirements

Operating System: Windows 10

Programming Language: Python 3.6.7.

3.3 PROJECT PLANNING

Project planning is the main part of the project which decides whether the project will be successful or not. We decided to complete the whole project in limited days. As we were new to the systematic environment, we initially used to set our daily goals to complete learning. We completed our learning in a few days. Later we split the project into three parts we worked on it and completed it in the latter days. Next, we tried to find bugs, solved them and added some more features for the convenience of users. Thus, our project was planned and done by setting the cut-off date, splitting the work and achieving daily goals. We had to understand what we needed to implement so first we listed out the features and then tried to implement each one step by step. In the Fig 3.3 we have shown the system architecture has been divided into three parts and those parts are explained in the below.

3.3.1 System Architecture

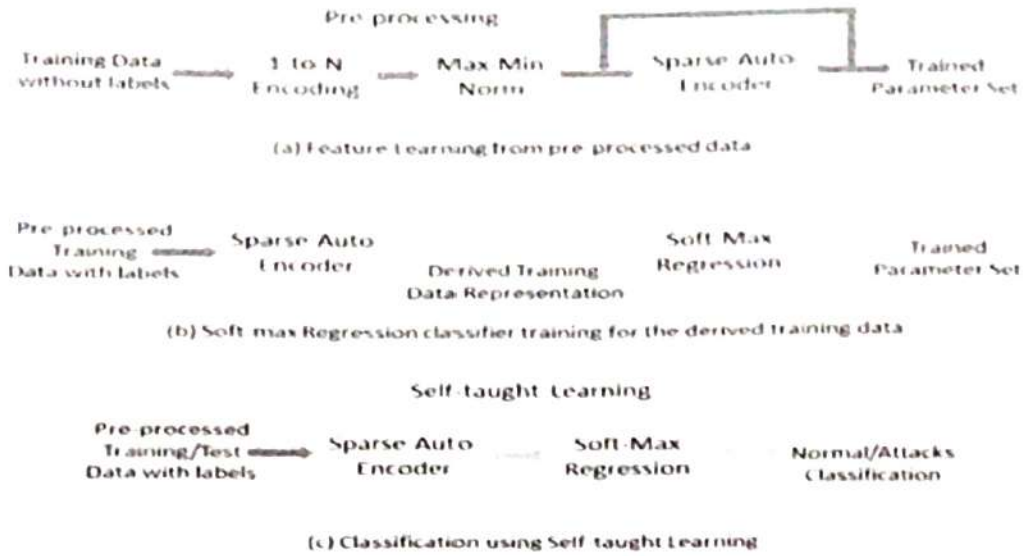

Fig.3.3: Architecture Diagram for Intrusion Detection System

The project process is completed in 3 parts:

1. Feature learning from pre-processing data: in this phase, the input data is stored and if any data is missing or any data is corrupted then it will be replaced with zero or null values, all non-numeric values are converted into numerical will be given in labels which will be used to optimize the system for easy understanding.
2. Classifier trainer for derived training data: in this phase, we will use some pre-processed data will be used to generate a training model to test the remaining data.
3. Classification using self-taught learning: in this phase the if attack data is similar to attack data in the training database, then it will be detected and it will alert the system.

3.4 FEASIBILITY STUDY

Feasibility studies aim to objectively and rationally uncover the strengths and weaknesses of the existing system or proposed venture. In its simplest term, the two criteria to judge feasibility are cost required and value to be attained. As such, a well-designed feasibility study should provide a historical background of the project. Generally, feasibility studies precede technical development and project implementation. The assessment of the feasibility study is based on the following factors:

1. Technical Feasibility
2. Operational Feasibility

3.4.1. Technical Feasibility

Generally, feasibility studies precede technical development and project implementation. The assessment is based on a system requirement in terms of Input, Processes, Output, Fields, Programs, and Procedure. This can be quantified in terms of volumes of data, trends, frequency of updating, etc., to estimate whether the new system will perform adequately or not. Technological feasibility is carried out to determine the capability, in terms of software, hardware, personnel and expertise, to handle the completion of the project.

3.4.2. Operational Feasibility

Operational feasibility is a measure of how well a proposed system solves the problems and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. The operational feasibility of the system can be checked as it solves the problems and reduces the complications occurring in the paper-pencil test.

4.IMPLEMENTATION

4.1 MODULES OVERVIEW

The System design Document describes the system requirements, operating environment, system and subsystem architecture, files, and database design, input formats, output layouts, human machine interfaces, detailed design, processing logic and external interfaces.

This application has six modules which are listed in the following.

1: Upload NSL Kdd dataset

2: Pre process dataset

3: Generate training model

4: Support Vector Machine Algorithm

5: Random Forest Algorithm

6: Deep Neural Network Algorithm

4.2 DEFINE THE MODULES

In this project there are six modules to achieve our expected result. These are the major functionalities of the project. Generating a training model which is processed after uploading datasets is one of the critical yet important task in this project.

4.2.1 Upload NSL KDD Dataset

The KDD data set is a well-known benchmark in the research of Intrusion Detection techniques. A lot of work is going on for the improvement of intrusion detection strategies while the research on the data used for training and testing the detection model is equally of prime concern because better data quality can improve offline intrusion detection. This paper presents the analysis of KDD data set with respect to four classes which are Basic, Content, Traffic and Host in which all data attributes can be categorized. NSL-KDD is a data set suggested to solve some of the inherent problems of the KDD'99 data set. Furthermore, the number of records in the

NSL-KDD train and test sets are reasonable. This advantage makes it affordable to run the experiments on the complete set without the need to randomly select a small portion.

4.2.2 Pre-process Dataset

When we receive NSL-KDD dataset, if we have any data missing or any data is corrupted then it will be replaced by zero or null values. It also covert all non-numeric values into numerical. It is used for optimizing the system and less computer resources.

4.2.3 Generate Training Model

When the data is pre-processed, the data will be divided into computer ratio in a way that some data will be used for training to generate the training model and remaining data will be used for testing purposes.

4.2.4 Support Vector Machine Algorithm

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

4.2.5 Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the

random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

4.2.6 Deep Neural Network Algorithm

At its simplest, a neural network with some level of complexity, usually at least two layers, qualifies as a deep neural network (DNN), or deep net for short. Deep nets process data in complex ways by employing sophisticated math modelling. Deep nets allow a model's performance to increase in accuracy. They allow a model to take a set of inputs and give an output. The use of a deep net is as simple as copying and pasting a line of code for each layer. Deep Neural Networks (DNN) is otherwise known as Feed Forward Neural Networks (FFNNS). In this network, data will be flowing in the forward direction and not in the backward direction, and hence node can never be accessed again. These Networks need a huge amount of data to train, and they have the ability to classify millions of data.

4.3 SOURCE CODE

IDS.py

```
from tkinter import messagebox
from tkinter import *
from tkinter import simpledialog
import tkinter
from tkinter import filedialog
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
from tkinter.filedialog import askopenfilename
import numpy as np
import pandas as pd
from sklearn import *
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn_extensions.extreme_learning_machines.elm import GenELMClassifier
from    sklearn_extensions.extreme_learning_machines.random_layer
import RBFRandomLayer, MLPRandomLayer
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
main = tkinter.Tk()
main.title("Deep Learning")
main.geometry("1300x1200")
global filename
global labels
```

```
global columns
global balance_data
global data
global X, Y, X_train, X_test, y_train, y_test
global svm_acc, random_acc, dnn_acc
def isfloat(value):
  try:
    float(value)
    return True
  except ValueError:
    return False
def importdata():
    global balance_data
    balance_data = pd.read_csv("clean.txt")
    return balance_data
def splitdataset(balance_data):
    X = balance_data.values[:, 0:37]
    Y = balance_data.values[:, 38]
    X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size = 0.2, random_state = 0)
    return X, Y, X_train, X_test, y_train, y_test
def upload():
    global filename
    text.delete('1.0', END)
    filename = askopenfilename(initialdir = "dataset")
    pathlabel.config(text=filename)
    text.insert(END,"Dataset loaded\n\n")
def preprocess():
    global labels
    global columns
    global filename
    text.delete('1.0', END)
```

```
columns =
["duration","protocol_type","service","flag","src_bytes","dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins","logged_in","num_compromised","root_shell","su_attempted","num_root","num_file_creations","num_shells","num_access_files","num_outbound_cmds","is_host_login","is_guest_login","count","srv_count","serror_rate","srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate","diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count","dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate","dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate","dst_host_rerror_rate","dst_host_srv_rerror_rate","label"]

labels =
{"normal":0,"neptune":1,"warezclient":2,"ipsweep":3,"portsweep":4,"teardrop":5,"nmap":6,"satan":7,"smurf":8,"pod":9,"back":10,"guess_passwd":11,"ftp_write":12,"multihop":13,"rootkit":14,"buffer_overflow":15,"imap":16,"warezmaster":17,"phf":18,"land":19,"loadmodule":20,"spy":21,"perl":22,"saint":23,"mscan":24,"apache2":25,"snmpgetattack":26,"processtable":27,"httptunnel":28,"ps":29,"snmpguess":30,"mailbomb":31,"named":32,"sendmail":33,"xterm":34,"worm":35,"xlock":36,"xsnoop":37,"sqlattack":38,"udpstorm":39}

balance_data = pd.read_csv(filename)

dataset = ''

index = 0

cols = ''

for index, row in balance_data.iterrows():

 for i in range(0,42):

  if(isfloat(row[i])):

   dataset+=str(row[i])+','

   if index == 0:

    cols+=columns[i]+','

 dataset+=str(labels.get(row[41]))

 if index == 0:

  cols+='Label'

 dataset+='\n'

 index = 1;

f = open("clean.txt", "w")

f.write(cols+"\n"+dataset)

f.close()

text.insert(END,"Removed non numeric characters from dataset and saved inside clean.txt file\n\n")
```

```
    text.insert(END,"Dataset Information\n\n")
    text.insert(END,dataset+"\n\n")
def generateModel():
    global data
    global X, Y, X_train, X_test, y_train, y_test
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    text.delete('1.0', END)
    text.insert(END,"Training model generated\n\n")
def prediction(X_test, cls):
    y_pred = cls.predict(X_test)
    for i in range(len(X_test)):
      print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))
    return y_pred
# Function to calculate accuracy
def cal_accuracy(y_test, y_pred, details):
    cm = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test,y_pred)*100
    text.insert(END,details+"\n\n")
    text.insert(END,"Accuracy : "+str(accuracy)+"\n\n")
    text.insert(END,"Report : "+str(classification_report(y_test, y_pred))+"\n")
    text.insert(END,"Confusion Matrix : "+str(cm)+"\n\n\n\n\n")
    return accuracy
def runSVM():
    global svm_acc
    global X, Y, X_train, X_test, y_train, y_test
    text.delete('1.0', END)
    cls = svm.SVC(C=2.0,gamma='scale',kernel = 'rbf', random_state = 2)
    cls.fit(X_train, y_train)
    text.insert(END,"Prediction Results\n\n")
    prediction_data = prediction(X_test, cls)
```

```
    svm_acc = cal_accuracy(y_test, prediction_data,'SVM Accuracy, Classification Report & Confusion Matrix')

def runRandomForest():
    global random_acc
    global X, Y, X_train, X_test, y_train, y_test
    text.delete('1.0', END)
    cls = RandomForestClassifier(n_estimators=1,max_depth=0.9,random_state=None)
    cls.fit(X_train, y_train)
    text.insert(END,"Prediction Results\n\n")
    prediction_data = prediction(X_test, cls)
    random_acc = cal_accuracy(y_test, prediction_data,'Random Forest Algorithm Accuracy, Classification Report & Confusion Matrix')

def runDNN():
    global dnn_acc
    global X, Y, X_train, X_test, y_train, y_test
    text.delete('1.0', END)
    srhl_tanh = MLPRandomLayer(n_hidden=8, activation_func='tanh')
    cls = GenELMClassifier(hidden_layer=srhl_tanh)
    cls.fit(X_train, y_train)
    text.insert(END,"Prediction Results\n\n")
    prediction_data = prediction(X_test, cls)
    dnn_acc = cal_accuracy(y_test, prediction_data,'DNN Algorithm Accuracy, Classification Report & Confusion Matrix')

def graph():
    height = [svm_acc,random_acc,dnn_acc]
    bars = ('SVM Accuracy', 'Random Forest Accuracy','DNN Accuracy')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()

font = ('times', 16, 'bold')
title = Label(main, text='Deep Learning Approach for Intelligent Intrusion Detection System')
```

```
title.config(bg='brown', fg='white')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0,y=5)
font1 = ('times', 14, 'bold')
upload = Button(main, text="Upload NSL KDD Dataset", command=upload)
upload.place(x=50,y=100)
upload.config(font=font1)
pathlabel = Label(main)
pathlabel.config(bg='brown', fg='white')
pathlabel.config(font=font1)
pathlabel.place(x=300,y=100)
preprocess = Button(main, text="Preprocess Dataset", command=preprocess)
preprocess.place(x=50,y=150)
preprocess.config(font=font1)
model = Button(main, text="Generate Training Model", command=generateModel)
model.place(x=330,y=150)
model.config(font=font1)
runsvm = Button(main, text="Run SVM Algorithm", command=runSVM)
runsvm.place(x=610,y=150)
runsvm.config(font=font1)
runrandomforest = Button(main, text="Run Random Forest Algorithm",
command=runRandomForest)
runrandomforest.place(x=870,y=150)
runrandomforest.config(font=font1)
runeml = Button(main, text="Run DNN Algorithm", command=runDNN)
runeml.place(x=50,y=200)
runeml.config(font=font1)
graph = Button(main, text="Accuracy Graph", command=graph)
graph.place(x=330,y=200)
graph.config(font=font1)
font1 = ('times', 12, 'bold')
```

```
text=Text(main,height=30,width=150)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=10,y=250)
text.config(font=font1)
main.config(bg='brown')
main.mainloop()
#test.py module
test.py:
import numpy as np
import pandas as pd
from sklearn import *
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn_extensions.extreme_learning_machines.elm import GenELMClassifier
from sklearn_extensions.extreme_learning_machines.random_layer import
RBFRandomLayer, MLPRandomLayer
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
global labels
global columns
global balance_data
def isfloat(value):
 try:
  float(value)
  return True
```

```
  except ValueError:
    return False
def preprocess():
    global labels
    global columns
    columns =
["duration","protocol_type","service","flag","src_bytes","dst_bytes","land","wrong_fr
agment","urgent","hot","num_failed_logins","logged_in","num_compromised","root_
shell","su_attempted","num_root","num_file_creations","num_shells","num_access_f
iles","num_outbound_cmds","is_host_login","is_guest_login","count","srv_count","s
error_rate","srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate","diff_srv
_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count","dst_host_same_sr
v_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate","dst_host_srv_diff_h
ost_rate","dst_host_serror_rate","dst_host_srv_serror_rate","dst_host_rerror_rate","ds
t_host_srv_rerror_rate","label"]
    labels =
{"normal":0,"neptune":1,"warezclient":2,"ipsweep":3,"portsweep":4,"teardrop":5,"nm
ap":6,"satan":7,"smurf":8,"pod":9,"back":10,"guess_passwd":11,"ftp_write":12,"multi
hop":13,"rootkit":14,"buffer_overflow":15,"imap":16,"warezmaster":17,"phf":18,"lan
d":19,"loadmodule":20,"spy":21,"perl":22,"saint":23,"mscan":24,"apache2":25,"snmp
getattack":26,"processtable":27,"httptunnel":28,"ps":29,"snmpguess":30,"mailbomb":
31,"named":32,"sendmail":33,"xterm":34,"worm":35,"xlock":36,"xsnoop":37,"sqlatta
ck":38,"udpstorm":39}
    balance_data = pd.read_csv("dataset.txt")
    dataset = ''
    index = 0
    cols = ''
    for index, row in balance_data.iterrows():
      for i in range(0,42):
        if(isfloat(row[i])):
          dataset+=str(row[i])+','
          if index == 0:
            cols+=columns[i]+','
      dataset+=str(labels.get(row[41]))
      if index == 0:
        cols+='Label'
      dataset+='\n'
      index = 1;
```

```
    f = open("clean.txt", "w")
    f.write(cols+"\n"+dataset)
    f.close()
def importdata():
    global balance_data
    balance_data = pd.read_csv("clean.txt")
    # Printing the dataswet shape
    print ("Dataset Lenght: ", len(balance_data))
    print ("Dataset Shape: ", balance_data.shape)
    # Printing the dataset obseravtions
    print ("Dataset: ",balance_data.head())
    return balance_data
def splitdataset(balance_data):
    X = balance_data.values[:, 0:37]
    Y = balance_data.values[:, 38]
    # Spliting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size = 0.2, random_state = 0)
    return X, Y, X_train, X_test, y_train, y_test
# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
        # Creating the classifier object
        clf_gini = svm.SVC(C=2.0,gamma='scale',kernel = 'rbf', random_state = 2)
# Performing training
        clf_gini.fit(X_train, y_train)
        return clf_gini
        def elm(X_train, X_test, y_train):
    srhl_tanh = MLPRandomLayer(n_hidden=8, activation_func='tanh')
    cls = GenELMClassifier(hidden_layer=srhl_tanh)
    cls.fit(X_train, y_train)
    return cls
def randomForest(X_train, X_test, y_train):
```

```
    cls = RandomForestClassifier(n_estimators=1,max_depth=0.9,random_state=None)
    cls.fit(X_train, y_train)
    return cls
def elmFeatureSelection(X_train, X_test, y_train):
    srhl_tanh = MLPRandomLayer(n_hidden=15, activation_func='tanh')
    cls = GenELMClassifier(hidden_layer=srhl_tanh)
    print('Original features:', X_train.shape[1])
    total = X_train.shape[1];
    X_train = SelectKBest(chi2, k=1).fit_transform(X_train, y_train)
    print('features set reduce after applying features concept:', (total - X_train.shape[1]))
    cls.fit(X_train, y_train)
    return cls
# Function to make predictions
def prediction(X_test, clf_object):
    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    #for i in range(len(X_test)):
     #print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))
    return y_pred
# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
    print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))
    print ("Accuracy : ", accuracy_score(y_test,y_pred)*100)
    print("Report : ", classification_report(y_test, y_pred))
def main():
    #preprocess()
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    #print("Results Using Gini Index:")
```

```
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)
    clf_gini = elm(X_train, X_test, y_train)
    #print("Results Using Gini Index:")
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)
    clf_gini = randomForest(X_train, X_test, y_train)
    #print("Results Using Gini Index:")
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)
    clf_gini = elmFeatureSelection(X_train, X_test, y_train)
    #print("Results Using Gini Index:")
    X_test = SelectKBest(chi2, k=1).fit_transform(X_test,y_test)
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)
# Calling main function
if __name__=="__main__":
        main()
```

5. PROJECT TESTING

5.1. INTRODUCTION

In general, testing is finding out how well something works. In terms of human beings, testing tells what level of knowledge or skill has been acquired. In computer hardware and software development, testing is used at key checkpoints in the overall process to determine whether objectives are being met.

5.2. TYPES OF TESTING

5.2.1. Code Testing

This examines the logic of the program. For example, the logic for updating various sample data and with the sample files and directories were tested and verified.

5.2.2 System Testing

Executing this specification starts with what the program should do and how it should perform under various conditions. Test cases for the various situations and combinations of conditions in all the modules are tested.

5.2.3 Unit Testing

In the unit testing, we test each module individually and integrate it with the overall system. Testing focuses verification efforts on the smallest unit of software design in the module. This is also known as module testing. The module of the system is tested separately. This testing is carried out during the programming stage itself. In the testing step, each module is found to work satisfactorily about the expected output from the module. There are some validation checks for fields also. It is very easy to find errors debut the system. Each Module can be tested using the following two Strategies:

1. White Box Testing
2. Black Box Testing

5.2.4. White Box Testing

White box testing is a testing case design method that uses the control structure of the procedure design to derive test cases. All independent paths in a module are exercised at least once, all logical decisions are exercised at once, execute all loops at boundaries and within their operational bounds exercise internal data structure to ensure their validity.

5.2.5. Black Box Testing

Black Box Testing attempts to find errors in the following areas or categories, incorrect or missing functions, interface error, errors in data structures, performance error and initialization and termination error. Here all the input data must match the data type to become a valid entry.

6. SCREENSHOTS

Fig.6.1: Installation of tenserflow module version(1.14.0)

Tenserflow is installed using the command

Python -m pip install tenserflow==1.14.0

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and gives developers the ability to easily build and deploy ML-powered applications. The installation of the tenserflow is shown in the above Fig.6.1.

TensorFlow provides a collection of workflows with intuitive, high-level APIs for both beginners and experts to create machine learning models in numerous languages. Developers have the option to deploy models on a number of platforms such as on servers, in the cloud, on mobile and edge devices, in browsers, and on many other JavaScript platforms. This enables developers to go from model building and training to deployment much more easily.

Fig.6.2: Installation of keras module version(2.3.1)

Keras is installed using the command

Python -m pip install keras==2.3.1

Keras runs on top of open source machine libraries like TensorFlow, Theano or Cognitive Toolkit (CNTK). Theano is a python library used for fast numerical computation tasks. TensorFlow is the most famous symbolic math library used for creating neural networks and deep learning models. TensorFlow is very flexible and the primary benefit is distributed computing. CNTK is deep learning framework developed by Microsoft. It uses libraries such as Python, C#, C++ or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries but difficult to understand for creating neural networks. The installation of the keras module is shown in the above Fig.6.2.

Keras is based on minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano.

7. RESULTS

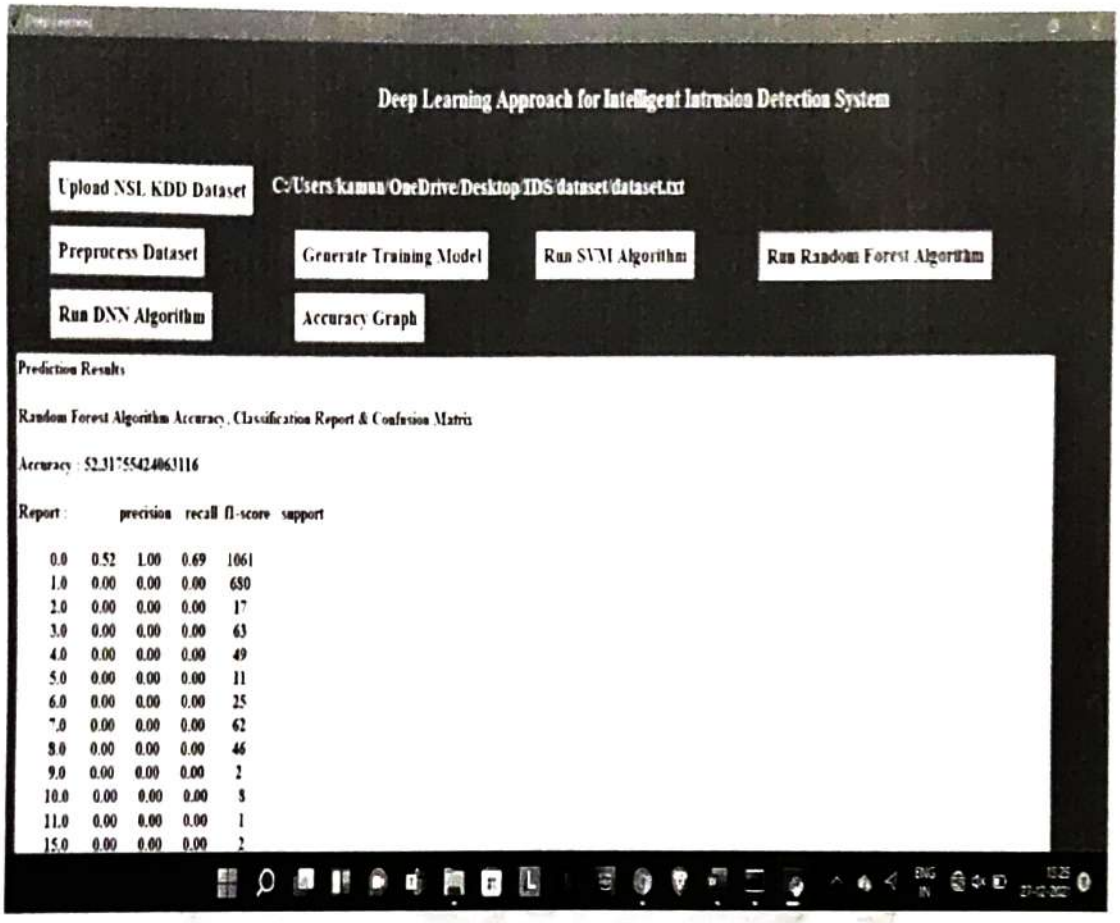

Fig.7.1: Prediction results of random forest algorithm

when the NSLKDD data set is uploaded then the output results of the randomforest algorithm are shown in Fig.7.1

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

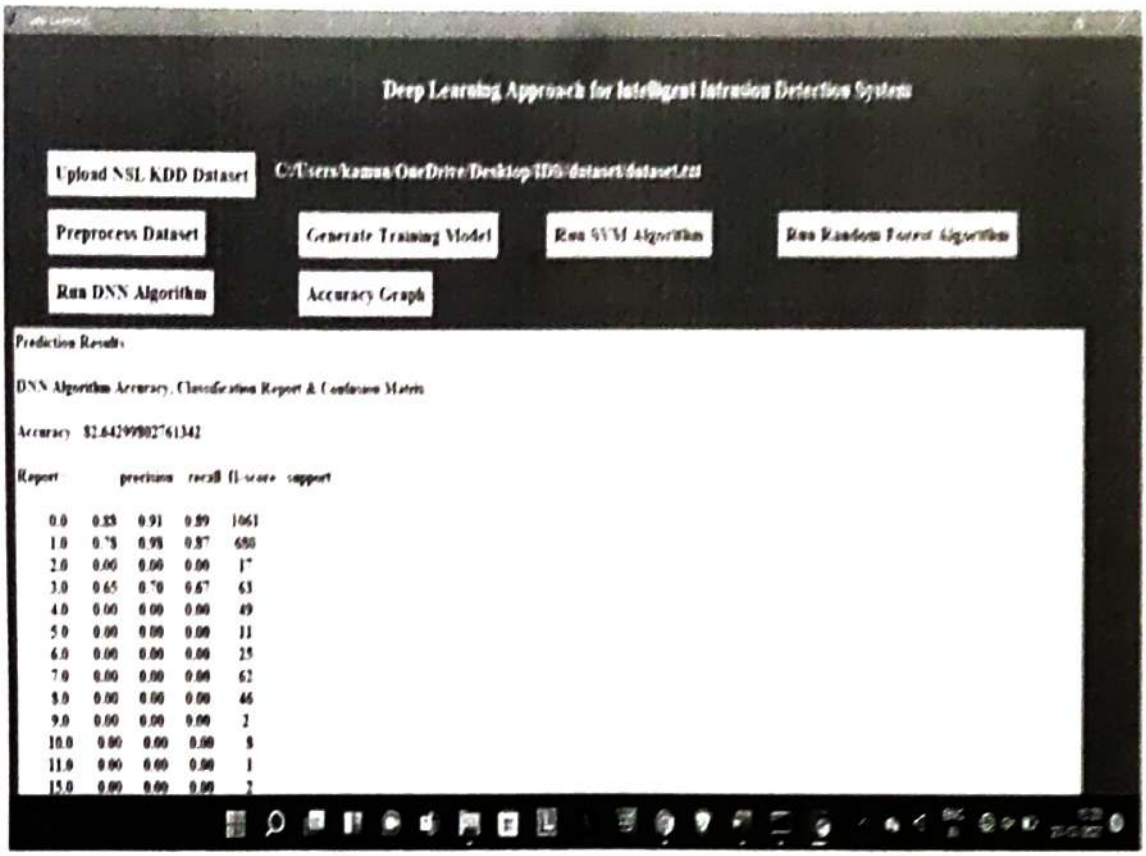

Fig.7.2: Prediction results of SVM algorithm

when the NSLKDD data set is uploaded then the output results of the SVM algorithm are shown in Fig.7.2.

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.

The support vector machines(SVMs) are a set of supervised learning methods used for classification, regression and outliners detection. The advantages of SVM are effective in high dimensional spaces.Uses a subset of training points in decision function, so it is memory efficient and it is versatile.

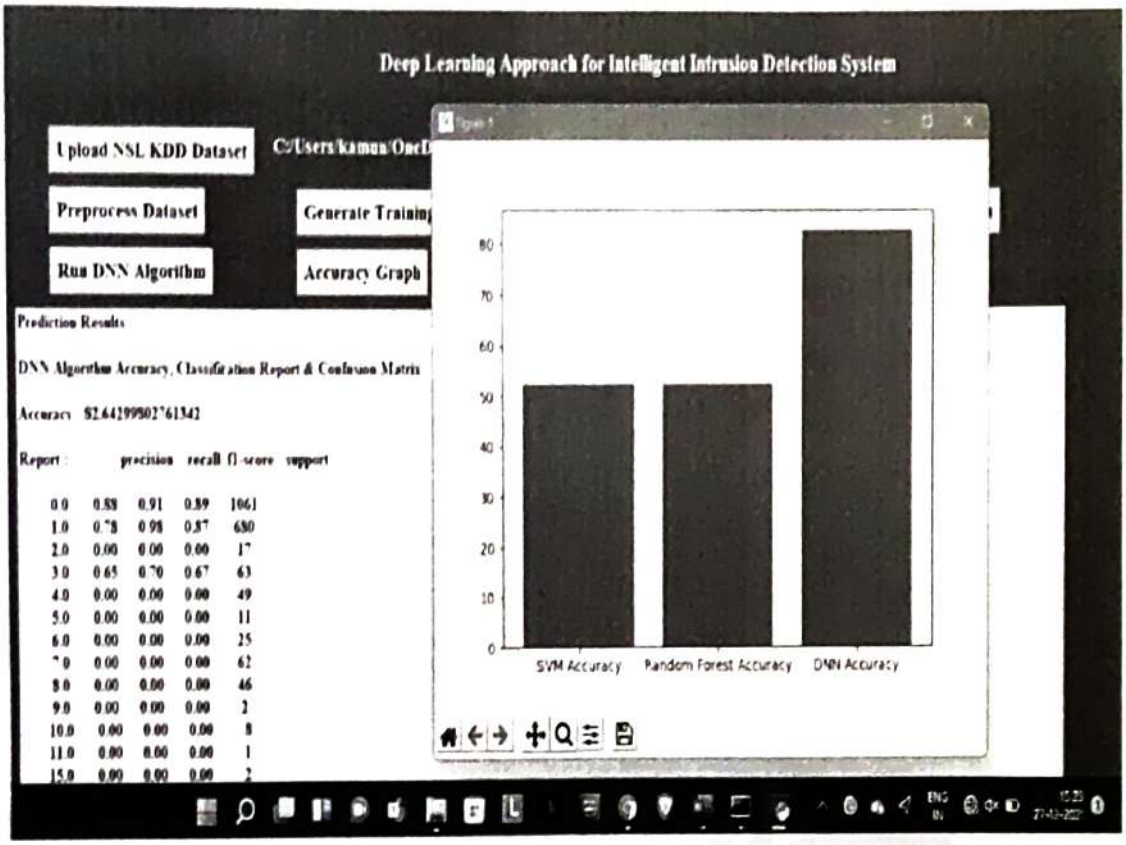

Fig.7.3: Prediction results of DNN algorithm and Accuracy graph

when the NSLKDD data set is uploaded then the output results of the DNN algorithm are shown in Fig.7.3 and graphical representation of all the three algorithms are shown in above Fig.7.3.

DNN is a type of machine learning that mimics the way the brain learns. It's been used for a variety of tasks; some that you might be familiar with, like language translation and image search tool

The general idea of a DNN is that it learns through repetitive action from a collection of samples, like 100 pictures of different dogs, as opposed to a set of man-made rules, like "a dog has a black nose and floppy ears." In this way, a DNN learns in the same way the human brain does – through practice and making mistakes.

The graph shows the comparison of performance SVM, randomforest, DNN algorithms.

8. CONCLUSIONS AND FUTURE SCOPE

8.1 CONCLUSION

We proposed a hybrid intrusion detection alert system using a highly scalable framework on commodity hardware server which has the capability to analyze the network and host-level activities. The framework employed distributed deep learning model with DNNs for handling and analyzing very large scale data in real time.

The DNN model was chosen by comprehensively evaluating their performance in comparison to classical machine learning classifiers on various benchmark IDS datasets. In addition, we collected host-based and network-based features in real-time and employed the proposed DNN model for detecting attacks and intrusions.

8.2 FUTURE SCOPE

In all the cases, we observed that DNNs exceeded in performance when compared to the classical machine learning classifiers. Our proposed architecture can perform better than previously implemented classical machine learning classifiers in both HIDS and NIDS. To the best of our knowledge, this is the only framework that can collect network-level and host-level activities in a distributed manner using DNNs to detect attacks more accurately.

9.REFERENCES

[1]. Adel Binbusayyis, Thavavel Vaiyapuri, "Identifying and Benchmarking Key Features for Cyber Intrusion Detection: An Ensemble Approach"2019.

[2]. Vinayakumar R., Mamoun Alazab, Alireza Jolfaei, Soman K.P., Prabaharan Poornachandran, "Ransomware Triage Using Deep Learning: Twitter as a Case Study"2019

[3]. Ao Liu, Bin Sun, "An Intrusion Detection System Based on a Quantitative Model of Interaction Mode Between Ports"2019

[4]. Junghoon Lee, Junghyun Kim, Ikkyun Kim, Kijun Han, "Cyber Threat Detection Based on Artificial Neural Networks Using Event Profiles"2019

[5]. Asif Karim, Sami Azam, Bharanidharan Shanmugam, Krishnan Kannoorpatti, Mamoun Alazab, "A Comprehensive Survey for Intelligent Spam Email Detection" 2019

[6]. Haitao He, Xiaobing Sun, Hongdou He, Guyu Zhao, Ligang He, Jiadong Ren, "A Novel Multimodal-Sequential Approach Based on Multi-View Features for Network Intrusion Detection"2019

[7]. K S Naveenkumar, R Vinayakumar, K P Soman, "Amrita-CEN-SentiDB 1: Improved Twitter Dataset for Sentimental Analysis and Application of Deep learning"2019

[8]. Bayu Adhi Tama, Lewis Nkenyereye, S.M. Riazul Islam, Kyung-Sup Kwak, "An Enhanced Anomaly Detection in Web Traffic Using a Stack of Classifier Ensemble"2020

[9]. SungUk Shin, Inseop Lee, Changhee Choi, "Anomaly Dataset Augmentation Using the Sequence Generative Models"2019

[10]. Yanqing Yang, Kangfeng Zheng, Bin Wu, Yixian Yang, Xiujuan Wang, "Network Intrusion Detection Based on Supervised Adversarial Variational Auto-Encoder With Regularization"2020

[11]. Monika Roopak, Gui Yun Tian, Jonathon Chambers, "An Intrusion Detection System Against DDoS Attacks in IoT Networks"2020

[12]. Kayla Chisholm, Chris Yakopcic, Md. Shahanur Alam, Tarek M. Taha, "Multilayer Perceptron Algorithms for Network Intrusion Detection on Portable Low Power Hardware"2020

[13]. Sriram Srinivasan, Vinayakumar Ravi, Sowmya V., Moez Krichen, Dhouha Ben Noureddine, Shashank Anivilla, Soman K.P., "Deep Convolutional Neural Network Based Image Spam Classification"2020

[14]. Giuseppina Andresini, Annalisa Appice, Nicola Di Mauro, Corrado Loglisci, Donato Malerba, "Multi-Channel Deep Feature Learning for Intrusion Detection"2020

[15]. Azam Rashid, Muhammad Jawaid Siddique, Shahid Munir Ahmed, "Machine and Deep Learning Based Comparative Analysis Using Hybrid Approaches for Intrusion Detection System"20

ABSTRACT

With the astonishing development of the Internet and its applications in the last decade, cyberattacks are changing quickly, and the necessity of protection for communication networks has improved tremendously. As the primary defence, the intrusion detection system plays a crucial role in making sure network security. The key to an intrusion detection system is actually to determine a variety of attacks effectively as well as to adjust to a constantly changing threat scenario.

In this project, deep neural network (DNN), a type of deep learning model is explored to develop flexible and effective IDS to detect and classify unforeseen and unpredictable cyber-attacks. The continuous change in network behaviour and rapid evolution of attacks makes it necessary to evaluate various datasets which are generated over the years through static and dynamic approaches.

DNN or Deep Neural Network on NSL-KDD dataset for effective detection of an attack. Firstly, the dataset was preprocessed and normalized and then fed to the DNN algorithm to create a model. For testing purposes, an entire dataset of NSL-KDD was used. Finally, to analyze the accuracy and precision of the DNN model, we use accuracy and precision matrices. The proposed DNN-based strategy enhances network anomaly detection and opens a new analysis gateway for intrusion detection systems.

Finally, we propose a highly scalable and hybrid DNNs framework called Scale-Hybrid-IDS-AlertNet (SHIA) which can be used in real-time to effectively monitor the network traffic and host-level events to proactively alert possible cyber-attacks.