World Journal of Advanced Research and Reviews

eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(Research Article)

Analysis of intrusion detection system in cloud computing environment using artificial neural network

Muhammad Abdullahi Onawo [2, *], Binyamin A. Ajayi [2], Muhammad Umar Abdullahi [1], Kene Tochukwu Anyachebelu [2] and Abubakar Suleiman [2]

[1] *Department of Computer Science, Federal University of Technology, Owerri, Nigeria.*
[2] *Department of Computer Science, Nasarawa State University, Keffi, Nasarawa State-Nigeria.*

World Journal of Advanced Research and Reviews, 2023, 19(03), 1007–1019

Publication history: Received on 08 August 2023; revised on 17 September 2023; accepted on 19 September 2023

Article DOI: https://doi.org/10.30574/wjarr.2023.19.3.1334

Abstract

This study developed a novel intrusion detection system (IDS) for cloud computing using artificial neural networks (ANNs) and machine learning techniques. The proposed IDS uses an adaptive architecture capable of detecting malicious activities within a cloud computing environment. To process and optimize the data, Adam optimization techniques were employed, and MiniMaxScaler was used to normalize the data for training. The model was designed using the TensorFlow framework for ANNs, and the LSD methodology was employed in the development. The training was conducted using the University of New Brunswick Intrusion Detection Systems dataset, which had been preprocessed. Results indicate that the proposed architecture was highly effective in detecting various attacks, with low false-positive and false-negative rates. The training and validation accuracies were 99.7% and 99.9%, respectively, using this method. This approach can automatically detect the nature of attacks, saving time and resources.

Keywords: (ABS) Artificial Neural Network; Intrusion Detection System; Deep Learning; Model and Cloud Computing

1. Introduction

Due to the tremendous growth witnessed in the field of information technology IT, cloud computing has become the first choice of every IT organization because of its distributed and scalable nature. Some see the cloud as a novel technical revolution, while others consider it a natural evolution of technology, economy, and culture. Nevertheless, cloud computing is an important paradigm that is flexible, cost-effective, and an efficient delivery platform for providing consumer and business IT services over the Internet [1].

Considering the benefits of cloud computing, its wide-ranging appeal is not surprising. However, this new approach does raise some concerns. Foremost among them is securing data in the cloud. Security controls in cloud computing such as cyber threats, advanced threat detection, real-time protection, enterprise security, and vulnerability management are by far the most part to be considered in IT environments [1]. Cloud computing presents new risks and threats to an organization because of cloud services, operational models, and technologies that enable these services.

Attackers can compromise the integrity, confidentiality, and availability of resources, data, and virtualized infrastructure of cloud computing systems, which may give birth to new types of attacks [2]. The problem can be worse and more critical when a cloud with massive storage capacity and computing power is attacked by intruders that are present in the cloud environment.

[*] Corresponding author: Muhammad Umar Abdullahi

In 2017, Equifax, a US-based consumer credit reporting agency became a victim of a cyber-attack when the problem of identity theft for 145.5 million U.S. consumers arose. The hackers stole personally identifiable information including names, social security numbers, and birth dates of consumers, and left no evidence of unauthorized activity. This was considered to be one of the major breaches that happened in history, costing the company $275 million in loss [4]. Data in the cloud is viewed as big data that has high volume, high velocity, and high variety. Intruder attacks can be detected and restricted using several mechanisms like cryptography and authentication, IDS is one of these (Jangla & Amne, 2015). In today's era of hackers, it becomes very crucial to design the system much stronger to protect the resources, infrastructure, and valuable data from intruders. Here, IDS plays a significant role in securing customers' data, resources, and assets in the cloud against security threats. It is one of the advanced security solutions capable of protecting network data from malicious activities. Therefore, IDS should be designed and deployed such that it can uncover all cloud-related attacks over the entire cloud network with a minimum false alarm rate as possible. Since cloud computing is different from traditional computer systems, specific attacks such as Denial of service attacks, insider attacks, man-in-the-middle, and cloud malware injection attacks must be efficiently detected [5].

2. Literature survey

A cloud-based IDS framework has been proposed by [6] the cloud IDS CIDS involves two types of nodes. Cooperative nodes and Central coordinator. A dedicated network IDS NIDS is deployed on a virtual switch VM at the cloud entry point. This monitors the incoming network packets. The author proposed to use the SNORT tool for this function. The host VMs are monitored by individual HIDS. An open-source tool HIDS was used.

The alerts from the NIDS and HIDS are collected by the central coordinator. The alerts are in the intrusion detection message exchange format IDMEF. The central coordinator is the heart of the framework and is responsible for monitoring, managing, analyzing, and correlating the alerts hence creating comprehensive information on attacks within the public cloud. The central coordinator reports the comprehensive attack information to the user who can then initiate required preventive measures to contain the attack. However, the drawback of this scheme can be a single point of failure at the main central coordinator.

Shenfield, Day, and Ayesh [7] presented a naive approach for the detection of unwanted traffic using ANN that is suitable for use in deep packet inspection-based IDS. The inputs range from normal benign network traffic which includes images, videos, dynamically linked libraries, logs, and music files, including the malicious shell code obtained from online vulnerability exploit dB, experimentally proved that the proposed system can classify the traffic accurately. The proposed malicious network traffic detection system can remarkably improve the utility of IDS applied to both conventional system traffic analysis and traffic analysis of cyber-physical systems like smart grids. The detection rate of 98% and 2% false alarm rate. Various intrusion detection schemes and methods that try to detect intrusion in one way or the other were compared by Prasad et al., (2015). From the comparison, it is evident that many IDS techniques depend on high time, memory, and cost requirements apart from advantages

Liu, Liu, and Zhao [8] established an intrusion detection approach based on a convolutional neural network (CNN) and evaluated the IDS model. In the training phase, it generated datasets by extracting living examples from the KDD Cup1999 dataset and did two depersonalization processes on test data to bring more convenient convolutional neural network learning. In the test phase, it extracted 10 test datasets and tested their performance. Compared with other IDS classifiers, the intrusion detection model based on CNN has the highest detection rate and precision. The feasibility of applying convolutional neural networks in highly-intruded detection has been proved. The detection rate observed was 97.7% with a false alarm rate of 0.099958%.

Ingre [9] analyzed the efficiency of the NSL-KDD dataset by evaluating it with the help of an Artificial Neural Network. In both the binary class and the five-class classification (type of attack) the results were obtained and analyzed according to several performance parameters with better accuracy. The detection rate was 81% for intrusion detection and 78% for attack-type classification using the NSL-KDD dataset. A higher detection rate was observed when the proposed scheme was compared with the existing scheme for both class classifications.

An IDS for cloud computing using neural networks and artificial bee colony optimization algorithm has been developed [10]. The IDS was based on the combination of multilayer perceptron (MLP) networks artificial bee colony (ABC) and fuzzy clustering algorithm. Normal and abnormal traffic pockets were identified by the multilayer perceptron, while the MLP training was done by the ABC algorithm by optimizing the values of linkage weight and biases. Fayaz (2017) presented the usage of intrusion detection and intrusion prevention techniques in the Cloud. The study specifies the locations in the Cloud where IDS/IPS can be positioned for efficient detection and prevention of attacks. The author listed some current solutions to mitigate the risks.

Research suggests that anomaly-based IDS can identify new network threats [11]. This used a real-time big data stream processing framework, Apache Storm, for the implementation of network IDS. The IDS was tested and evaluated using the Knowledge Discovery and Data Mining 1999 (KDD'99) dataset. Hatef et al. [5]. developed a hybrid intrusion detection approach in cloud computing. An open-source cloud computing environment called Eucalyptus was used to implement this method. A training dataset and NSL-KDD set were used. The proposed method shows that intrusion coverage, intrusion detection accuracy, reliability, and availability in cloud computing systems are considerably increased and false warnings are significantly reduced.

An intelligent approach, based on the collaboration of IDS systems, and Multi-Agent Systems has been proposed [12]. The authors have designed seven different agents for both mode host and network which work, operate, run independently, and communicate with each other to check and identify all malicious attacks in the cloud computing system. This approach provides an intelligent self-administered and fault-tolerant IDS with continuous execution time and minimal human intervention with the use of multi-agent security systems.

Rani [13] emphasized existing methods of IDS based on soft computing techniques, data mining, and other approaches. The entire survey is categorized based on detection approaches such as signature and anomaly detection to get a vivid analysis of the type of attack to be detected, the advantages of using the approach, and loopholes that existed in implementing the approach in a tabular manner. As per the survey conducted, an anomaly-based detection approach is adopted by many researchers to detect both known and unknown attacks by monitoring network traffic. They have been capable of detecting both network and host-based attacks and that too old ones and the latest ones whose signatures are not present (Sharma & Kumar, 2019). that are 2.3 percent and 1.5 percent higher than the common base result in face and mask detection.

3. Methodology

Since the late 1980s research in the field of intrusion detection and system security has been held all around the world. Various approaches and techniques have been suggested and some are currently being implemented for intrusion detection. The main aim of implementing IDS with the help of ANN is to incorporate a system that contains an intelligent agent that can bring out latent patterns to classify normal and abnormal records, along with the capability to generalize records belonging to the same class.

3.1. Proposed Method

The development of this system will be broken down into several sections to simplify the development of the project. The proposed method would be an Artificial Neural Network (ANN) that is made of an AutoEncoder and developed using the architecture discussed in section 3.3 using the agile Lean Software Development (LSD) software methodology which is center focused methodology that wastes no time in none essential parts of the research. These three things (ANN, LSD, and System Architecture) come together to form the basis of the program being developed.

3.2. System Architecture

The system presented in this paper is a machine-learning application for detecting several cloud intrusion attacks using Artificial Neural Networks (ANN). The architecture of the system is shown in Figure 1. The system starts by pulling data from the storage repository which is a directory holding the dataset, the dataset has 15 classes to predict, Label_BENIGN being safe and the malicious classes being Label_Bot, Label_DDoS, Label_DoS GoldenEye, Label_DoS Hulk, Label_DoS Slowhttptest, Label_DoS slow loris, Label_FTP-Patator, Label_Heartbleed, Label_Infiltration, Label_PortScan, Label_SSH-Patator, Label_Web Attack Brute Force, Label_Web Attack Sql Injection and Label_Web Attack XSS. The dataset is then processed by removing the null values and normalizing the values using MinMaxScaler this helps in keeping the values within a range and reducing outliers. As machine learning models are prone to overfitting, we then employ k-folds for creating a train and validation set this way the accuracy is calculated per fold and this way the accuracy of each comes to form the total, and if one-fold performs too well we know the model is overfitting to that dataset. Once we have our train and validation set, we then create our ANN, the ANN is then trained using Adam optimizer with a learning rate of 0.001, and the model is then trained and exported with the learned information. This architecture was modified from the research in (Liu, Liu, & Zhao, 2017) by including k-folds and encoder and decoder system of the ANN instead of the CNN used in their work.

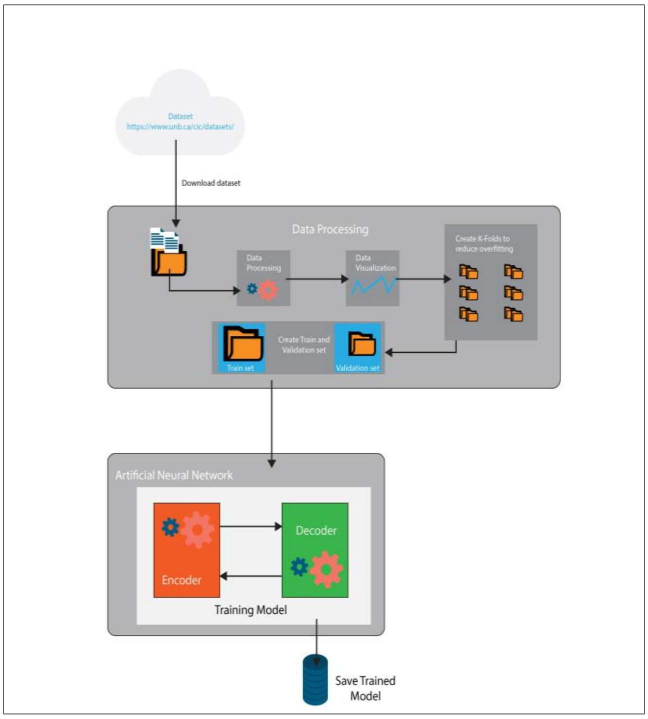


Figure 1 Architecture of the Proposed System adapted from (Liu, Liu, & Zhao, 2017)

3.3. ANN AutoEncoder

Autoencoding is an Artificial Neural Network data compression algorithm where the compression and decompression functions are data-specific, lossy, and learned automatically from train datasets rather than engineered by a human.

Encoder-Decoder models are a family of models that learn to map data points from an input domain to an output domain via a two-stage network: The encoder, represented by an encoding function z = f(x), compresses the input into a latent-space representation; the decoder, y = g(z), aims to predict the output from the latent space representation.

3.4. System Modelling

This section discusses the behavior of the system its environment and its internal components. The Unified Modelling Language (UML) due to its vast library of modeling tools and diagrams from use case modeling, sequence, class, entity relation, activity modeling, etc.; It is the technique chosen for this section and three techniques were chosen to model three different aspects of the computer system which are the use case diagram to show the system interaction with the external environment, the activity diagram that models how each action in the system is handled and a sequence diagram that shows how the internal components of the system interact with each other.

3.5. Use case Diagram

The use case diagram is a high-level diagram that highlights the system requirements provided to various users of the system. Figure 2 shows the use case diagram of the system which has a single user who can view the attack data and the result from the model's prediction.

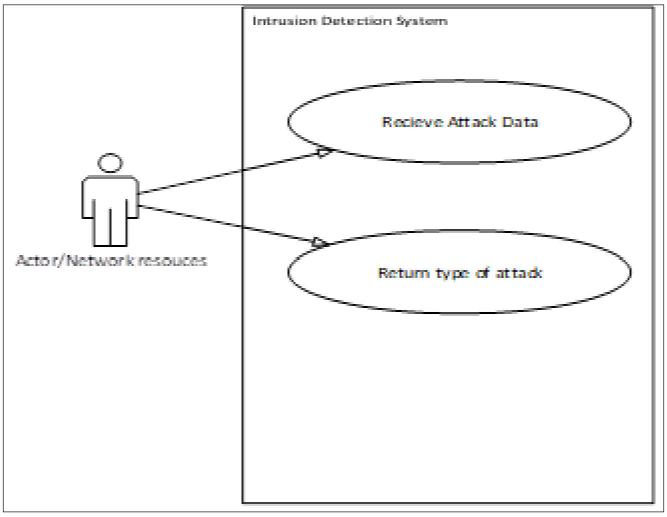


Figure 2 Use case Diagram of the Intrusion System

3.6. Usage Activity Diagram

This activity diagram, Figure 3, shows the deployment flow of the model, the model has to be loaded from this and receive input data, in this case, the attack data. The model then processes the attack to make a prediction and lastly yields the result.

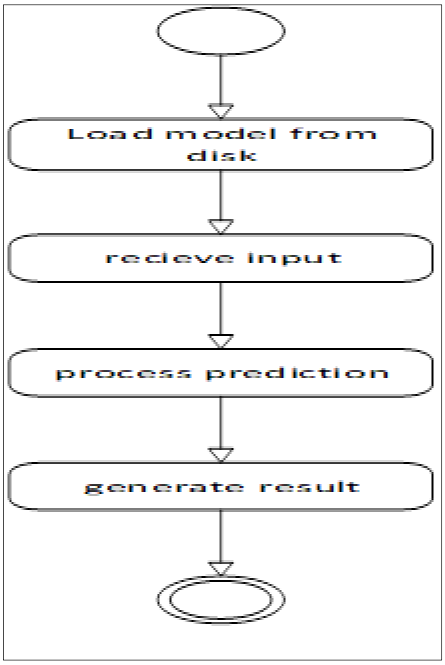


Figure 3 Activity Diagram of the Intrusion System

3.7. Development Activity Diagram

Activity diagrams are used to serve as an advanced flowchart that shows how the system responds to actions or event triggers. They are designed to model the complex transitions in the system from the start of an action to the result or output. Figure 4 shows the activity diagram of the proposed system which starts with loading the model from disk and performing data processing by dropping null values, this balanced data is visualized to view if there were other anomalies. The data set is segmented into k-folds this way we reduce the chances of overfitting, these folds were then stored in memory as a train and validation set. The train and validation sets are then normalized for training. The training is then performed by the neural network's encoder and decoder and then the model is saved for usage.

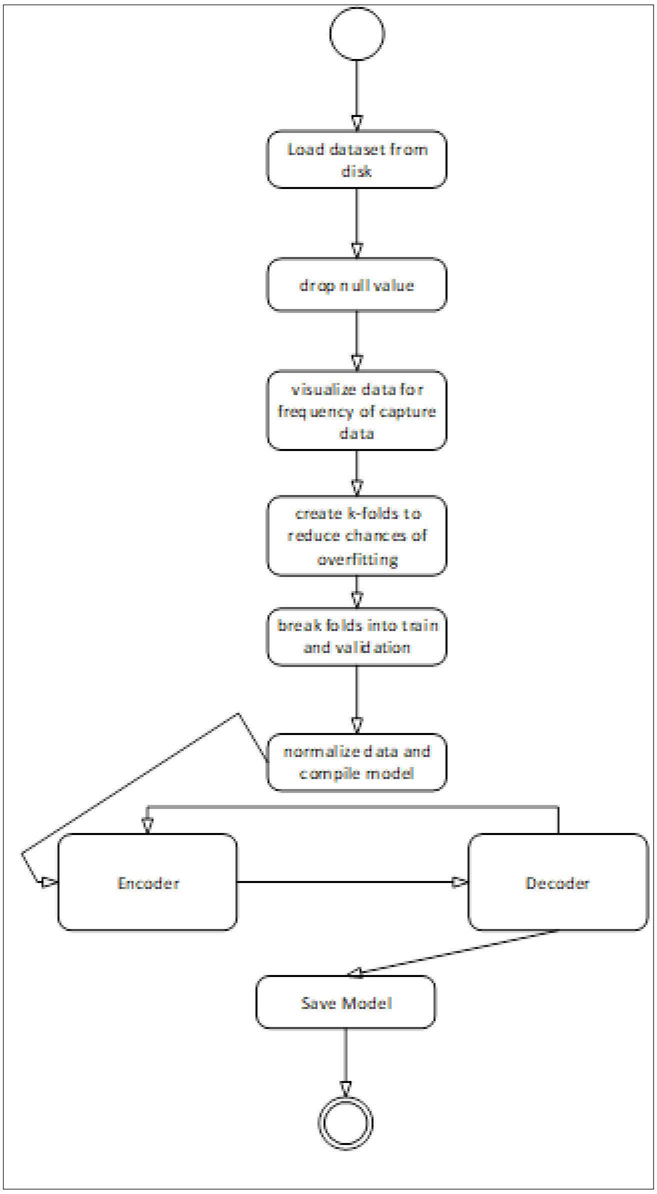

Load dataset from disk
drop null value
visualize data for frequency of capture data
create k-folds to reduce chances of overfitting
break folds into train and validation
normalize data and compile model
Encoder
Decoder
Save Model

Figure 4 Activity Diagram of the Model Development

3.8. Sequence Diagram

The sequence diagram unlike the use case and activity diagram models the internal behaviour of the system which is the interaction between subsystems of the software and process in the application, it gives a very good low-level overview of what is happening within the system. Figure 5 shows the sequence diagram of the system and its internal components, when the system runs the dataset is loaded from the disk for data processing which is streamed into segments to create k-folds from which these folds are normalized and used by the encoder to train a decoder and the smarter the decoder gets the smart the model gets when we achieve a decent result, we can then return the built model for prediction**.**

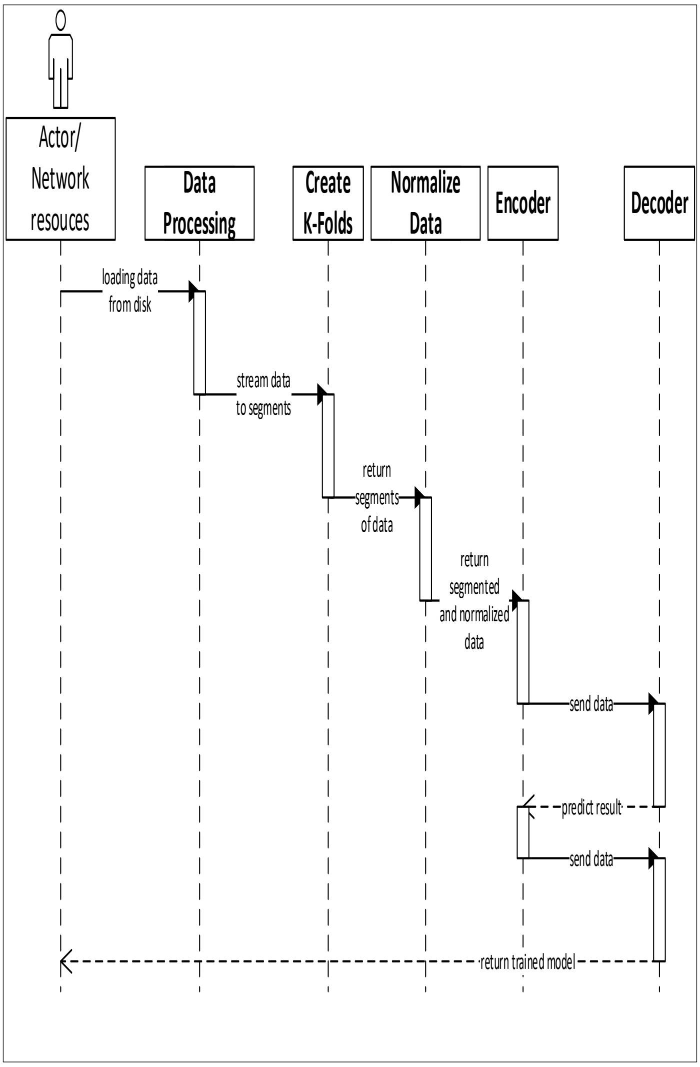


Figure 5 Sequence Diagram of the Model Development

3.9. System Analysis

This system ANN is built to run on a very low computing system and cross-platform on Linux, Windows, and MacOS that meets the minimum system specification, and has Python and TensorFlow installed. The system was tested on a MacOS Yosemite laptop and a Windows 10 computer with a minimum RAM of 1 gigabyte and a minimum storage requirement of 2 gigabytes. The specifications are shown in detail in Table 1.

Table 1 Test Device Specification

Requirement	**Minimum Specification**	**Device Specification**	
		Windows	Mac OS
System OS	Windows /MacOS	10	Yosemite
Memory (RAM)	Minimum: 1 GB	16 GB	8 GB
Storage Capacity	2 GB	2 TB	256 GB

4. Result and discussion

The application was built using Python programming language. The application was deployed on a Windows computer and tested on Mac and Linux-based systems. The application is a desktop application that predicts the type of network attack being carried out by an attacker on a network. TensorFlow is an open-source end-to-end machine learning platform powered by Google which serves as the machine learning backend for this research using Python programming language.

4.1. Dataset

The dataset used during this process is from the University of New Brunswick IDS dataset. The program runs by first loading the model to memory to position it for prediction and then loading it into a model variable, when the model is done loading predictions can be performed. The prediction returns an array of results these results represent the probability of it being either of the classes, and the class index with the highest result represents the predicted class. Once the index is obtained, the type of intrusion is known. During the development of this program, Tensorflow was used and this is an open-source end-to-end machine learning framework developed and backed by Google used to develop top-of-line Machine Learning software. Tensorflow provides a collection of workflows to develop and train models using Python or JavaScript, and to easily deploy in the cloud, in the browser, or on a device no matter what language you use. It was chosen because of its robust nature and flexibility for research papers.

Numpy, Pandas, and MatplotLib are the data processing libraries used for this research paper, python does not have traditional arrays so Numpy provides that with its suite of rich array functions, Pandas was used for data processing of the dataset and MatplotLib serves as a great python data visualization tool.

4.2. Data Visualization Results

During the development process data bins were exported for each of the fields of the dataset to view the frequency of the fields and find the anomalies before data processing this way, we can remove missing values and normalize the dataset as shown in Figure 6.

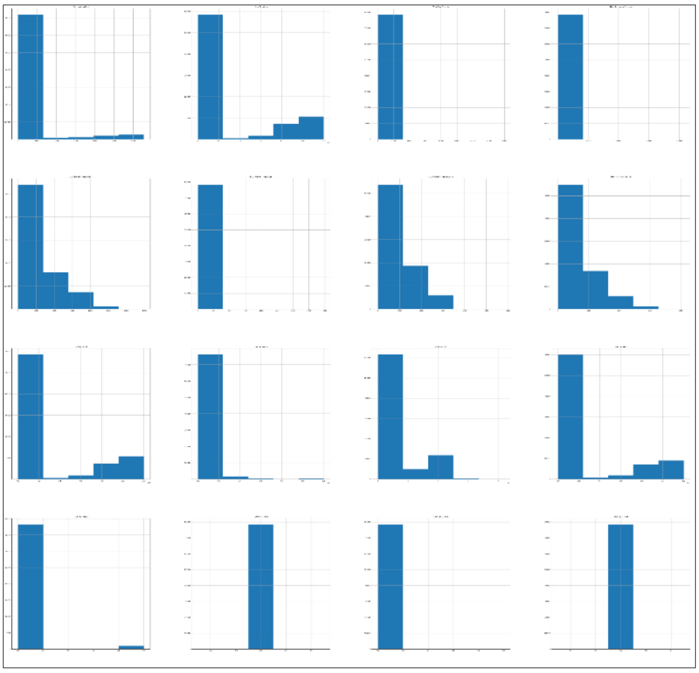


Figure 6 Data Bins of the Dataset

The normalized data is summarized in Table 2 where there is an average of 783966 rows of data in the 94 columns and also the mean, minimum, max, and standard deviation of each field. The frequencies are also displayed this way we know what is within the 25th, 50th, and 75th percentile.

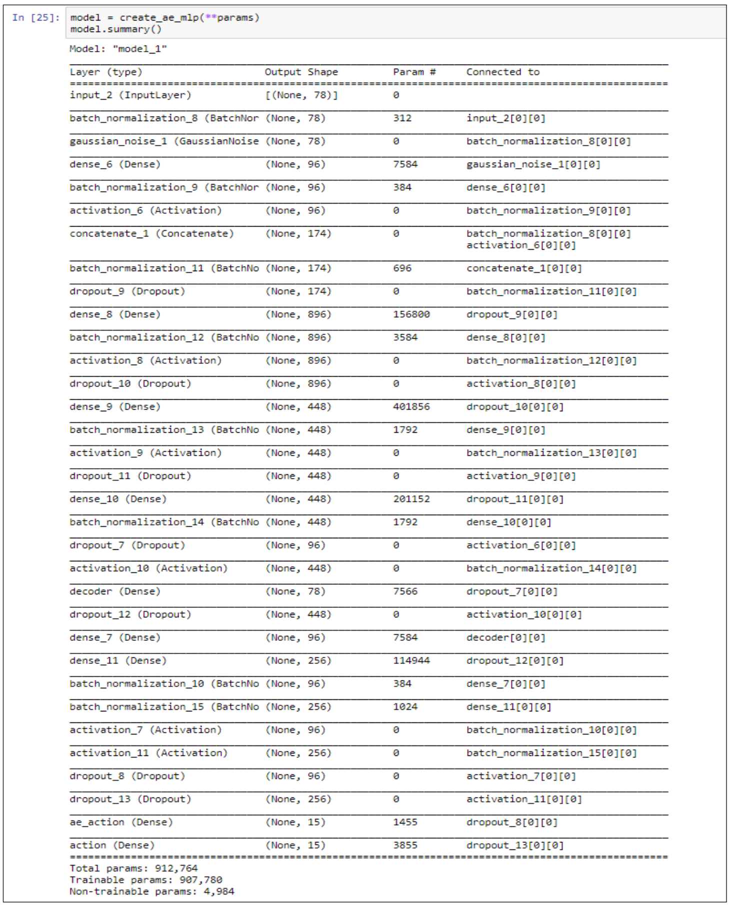
Table 2 Summary of Normalized Data

	Destination Port	**Flow Duration**	**Total Fwd Packets**	**Total Backward Packets**	**Total Length of FWD Packets**	**Total Length of BWD Packets**
Count	783966.000000	7.839660e+05	783966.000000	783966.000000	7.839660e+05	7.839660e+05
Std	13100.767048	4.001842e+07	643.637420	858.662197	6.881230e+03	1.9304080e+06
Min	0.000000	+2.000000e+00	1.000000	0.000000	0.000000e+00	0.000000e+00
25%	80.000000	5.9000000e+01	1.000000	1.000000	2.000000+00	0.000000e+00
50%	80.000000	6.039700e+04	3.000000	2.000000	2.6000000e+01	7.6000000e+01
75%	543.000000	1.200000e+08	214125.00000	283900.000000	2.866110e+06	6.29600e+08

4.3. Structure of the Developed Model

The model was developed using keras tensorflow API, this way neural network layers could be easily stacked. In this case, we use batch normalization layers to make the training faster and balanced by normalizing the values and by recentering and rescaling the values through the training iterations.

Overfitting is also resolved using batch normalization but this model goes a step further by adding dropout layers, these dropout layers turn off certain neurons at certain positions during the training and this reduces the likelihood of a model memorizing the data rather than learning the patterns of the dataset.



```
In [25]: model = create_ae_mlp(**params)
         model.summary()

Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 78)]         0
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 78)           312         input_2[0][0]
__________________________________________________________________________________________________
gaussian_noise_1 (GaussianNoise (None, 78)           0           batch_normalization_8[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 96)           7584        gaussian_noise_1[0][0]
__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 96)           384         dense_6[0][0]
__________________________________________________________________________________________________
activation_6 (Activation)       (None, 96)           0           batch_normalization_9[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 174)          0           batch_normalization_8[0][0]
                                                                 activation_6[0][0]
__________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 174)          696         concatenate_1[0][0]
__________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 174)          0           batch_normalization_11[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 896)          156800      dropout_9[0][0]
__________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 896)          3584        dense_8[0][0]
__________________________________________________________________________________________________
activation_8 (Activation)       (None, 896)          0           batch_normalization_12[0][0]
__________________________________________________________________________________________________
dropout_10 (Dropout)            (None, 896)          0           activation_8[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 448)          401856      dropout_10[0][0]
__________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 448)          1792        dense_9[0][0]
__________________________________________________________________________________________________
activation_9 (Activation)       (None, 448)          0           batch_normalization_13[0][0]
__________________________________________________________________________________________________
dropout_11 (Dropout)            (None, 448)          0           activation_9[0][0]
__________________________________________________________________________________________________
dense_10 (Dense)                (None, 448)          201152      dropout_11[0][0]
__________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 448)          1792        dense_10[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 96)           0           activation_6[0][0]
__________________________________________________________________________________________________
activation_10 (Activation)      (None, 448)          0           batch_normalization_14[0][0]
__________________________________________________________________________________________________
decoder (Dense)                 (None, 78)           7566        dropout_7[0][0]
__________________________________________________________________________________________________
dropout_12 (Dropout)            (None, 448)          0           activation_10[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 96)           7584        decoder[0][0]
__________________________________________________________________________________________________
dense_11 (Dense)                (None, 256)          114944      dropout_12[0][0]
__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 96)           384         dense_7[0][0]
__________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 256)          1024        dense_11[0][0]
__________________________________________________________________________________________________
activation_7 (Activation)       (None, 96)           0           batch_normalization_10[0][0]
__________________________________________________________________________________________________
activation_11 (Activation)      (None, 256)          0           batch_normalization_15[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 96)           0           activation_7[0][0]
__________________________________________________________________________________________________
dropout_13 (Dropout)            (None, 256)          0           activation_11[0][0]
__________________________________________________________________________________________________
ae_action (Dense)               (None, 15)           1455        dropout_8[0][0]
__________________________________________________________________________________________________
action (Dense)                  (None, 15)           3855        dropout_13[0][0]
==================================================================================================
Total params: 912,764
Trainable params: 907,780
Non-trainable params: 4,984
```

Figure 7 Structure of the Model

Lastly, we have a total of 912,764 parameters within the model out of which 907,780 are trainable and 4,984 as shown in Figure 7 This makes the model highly trainable and trains within 1 hour on a 32gig Windows 10 laptop with 6gig of NVidia GTX 1060 graphics memory.

4.4. Train Accuracy Result

After the model was built it was trained and evaluated using the training results accuracy parameter, this way we can monitor the training and the trends of the accuracy and the loss values, the aim is to keep the accuracy high (green graph) and the loss low (red line) using TensorFlow's early stopping we can stop the model when we have a comfortable loss and accuracy. The model was trained for 20 epochs which is each cycle the model gets to see the dataset during training.

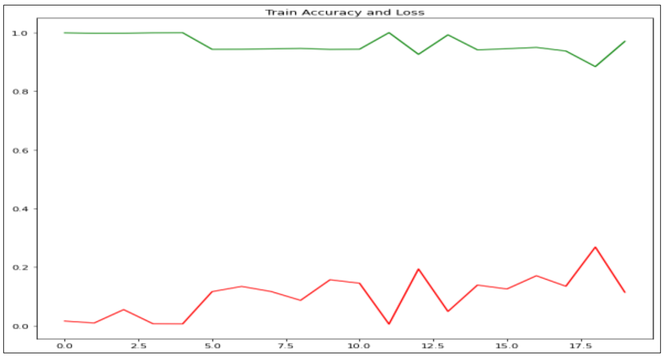


Figure 8 Accuracy Result for Training

4.5. Validation Accuracy Result

Training result alone is not enough to determine our accuracy hence we use a validation set to see if our model still performs well on data, it was not exposed to during the training of the model. Our validation accuracy stands at 99.96% which is very close to our training accuracy at 99.97%. The training and validation process can be seen in Figure 9 and Figure 10.

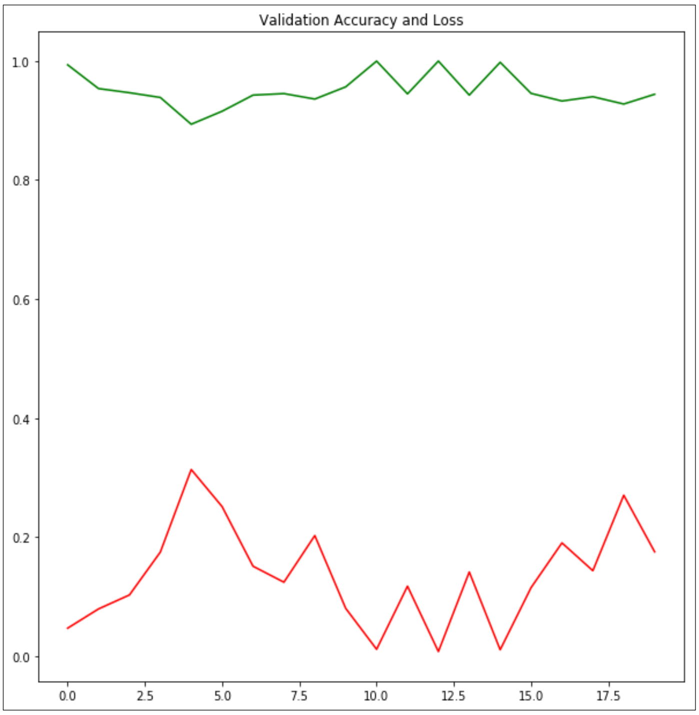
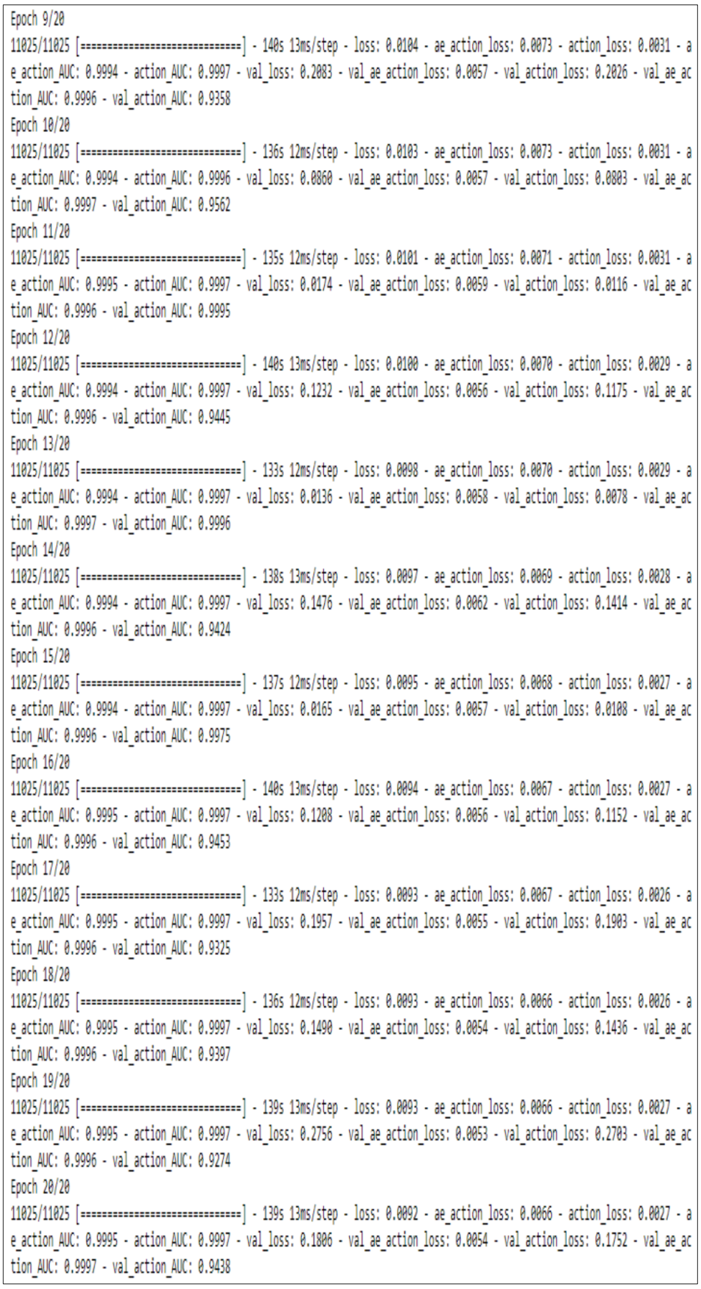


Figure 9 Validation Result from Training

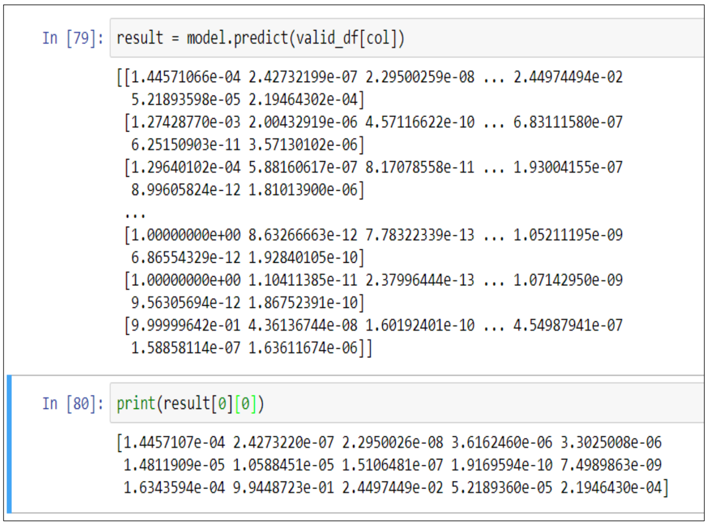


```
Epoch 9/20
11025/11025 [==============================] - 140s 13ms/step - loss: 0.0104 - ae_action_loss: 0.0073 - action_loss: 0.0031 - a
e_action_AUC: 0.9994 - action_AUC: 0.9997 - val_loss: 0.2083 - val_ae_action_loss: 0.0057 - val_action_loss: 0.2026 - val_ae_ac
tion_AUC: 0.9996 - val_action_AUC: 0.9358
Epoch 10/20
11025/11025 [==============================] - 136s 12ms/step - loss: 0.0103 - ae_action_loss: 0.0073 - action_loss: 0.0031 - a
e_action_AUC: 0.9994 - action_AUC: 0.9996 - val_loss: 0.0860 - val_ae_action_loss: 0.0057 - val_action_loss: 0.0803 - val_ae_ac
tion_AUC: 0.9997 - val_action_AUC: 0.9562
Epoch 11/20
11025/11025 [==============================] - 135s 12ms/step - loss: 0.0101 - ae_action_loss: 0.0071 - action_loss: 0.0031 - a
e_action_AUC: 0.9995 - action_AUC: 0.9997 - val_loss: 0.0174 - val_ae_action_loss: 0.0059 - val_action_loss: 0.0116 - val_ae_ac
tion_AUC: 0.9996 - val_action_AUC: 0.9995
Epoch 12/20
11025/11025 [==============================] - 140s 13ms/step - loss: 0.0100 - ae_action_loss: 0.0070 - action_loss: 0.0029 - a
e_action_AUC: 0.9994 - action_AUC: 0.9997 - val_loss: 0.1232 - val_ae_action_loss: 0.0056 - val_action_loss: 0.1175 - val_ae_ac
tion_AUC: 0.9996 - val_action_AUC: 0.9445
Epoch 13/20
11025/11025 [==============================] - 133s 12ms/step - loss: 0.0098 - ae_action_loss: 0.0070 - action_loss: 0.0029 - a
e_action_AUC: 0.9994 - action_AUC: 0.9997 - val_loss: 0.0136 - val_ae_action_loss: 0.0058 - val_action_loss: 0.0078 - val_ae_ac
tion_AUC: 0.9997 - val_action_AUC: 0.9996
Epoch 14/20
11025/11025 [==============================] - 138s 13ms/step - loss: 0.0097 - ae_action_loss: 0.0069 - action_loss: 0.0028 - a
e_action_AUC: 0.9994 - action_AUC: 0.9997 - val_loss: 0.1476 - val_ae_action_loss: 0.0062 - val_action_loss: 0.1414 - val_ae_ac
tion_AUC: 0.9996 - val_action_AUC: 0.9424
Epoch 15/20
11025/11025 [==============================] - 137s 12ms/step - loss: 0.0095 - ae_action_loss: 0.0068 - action_loss: 0.0027 - a
e_action_AUC: 0.9994 - action_AUC: 0.9997 - val_loss: 0.0165 - val_ae_action_loss: 0.0057 - val_action_loss: 0.0108 - val_ae_ac
tion_AUC: 0.9996 - val_action_AUC: 0.9975
Epoch 16/20
11025/11025 [==============================] - 140s 13ms/step - loss: 0.0094 - ae_action_loss: 0.0067 - action_loss: 0.0027 - a
e_action_AUC: 0.9995 - action_AUC: 0.9997 - val_loss: 0.1208 - val_ae_action_loss: 0.0056 - val_action_loss: 0.1152 - val_ae_ac
tion_AUC: 0.9996 - val_action_AUC: 0.9453
Epoch 17/20
11025/11025 [==============================] - 133s 12ms/step - loss: 0.0093 - ae_action_loss: 0.0067 - action_loss: 0.0026 - a
e_action_AUC: 0.9995 - action_AUC: 0.9997 - val_loss: 0.1957 - val_ae_action_loss: 0.0055 - val_action_loss: 0.1903 - val_ae_ac
tion_AUC: 0.9996 - val_action_AUC: 0.9325
Epoch 18/20
11025/11025 [==============================] - 136s 12ms/step - loss: 0.0093 - ae_action_loss: 0.0066 - action_loss: 0.0026 - a
e_action_AUC: 0.9995 - action_AUC: 0.9997 - val_loss: 0.1490 - val_ae_action_loss: 0.0054 - val_action_loss: 0.1436 - val_ae_ac
tion_AUC: 0.9996 - val_action_AUC: 0.9397
Epoch 19/20
11025/11025 [==============================] - 139s 13ms/step - loss: 0.0093 - ae_action_loss: 0.0066 - action_loss: 0.0027 - a
e_action_AUC: 0.9995 - action_AUC: 0.9997 - val_loss: 0.2756 - val_ae_action_loss: 0.0053 - val_action_loss: 0.2703 - val_ae_ac
tion_AUC: 0.9996 - val_action_AUC: 0.9274
Epoch 20/20
11025/11025 [==============================] - 139s 13ms/step - loss: 0.0092 - ae_action_loss: 0.0066 - action_loss: 0.0027 - a
e_action_AUC: 0.9995 - action_AUC: 0.9997 - val_loss: 0.1806 - val_ae_action_loss: 0.0054 - val_action_loss: 0.1752 - val_ae_ac
tion_AUC: 0.9997 - val_action_AUC: 0.9438
```

Figure 10 Results of Data Trends During Training

4.6. Attack Prediction Result

After the model is trained it now fits to make predictions, the predictions are returned as arrays shown in cell 79 from this, we can now return the list of results from which we can then get the row-specific results in cell 80. This final array is a list of probabilities of an attack being in a particular class in this case the highest value is in the 11th index 0.9948732 as shown in Figure 11.



```
In [79]: result = model.predict(valid_df[col])

         [[1.44571066e-04 2.42732199e-07 2.29500259e-08 ... 2.44974494e-02
           5.21893598e-05 2.19464302e-04]
          [1.27428770e-03 2.00432919e-06 4.57116622e-10 ... 6.83111580e-07
           6.25150903e-11 3.57130102e-06]
          [1.29640102e-04 5.88160617e-07 8.17078558e-11 ... 1.93004155e-07
           8.99605824e-12 1.81013900e-06]
          ...
          [1.00000000e+00 8.63266663e-12 7.78322339e-13 ... 1.05211195e-09
           6.86554329e-12 1.92840105e-10]
          [1.00000000e+00 1.10411385e-11 2.37996444e-13 ... 1.07142950e-09
           9.56305694e-12 1.86752391e-10]
          [9.99999642e-01 4.36136744e-08 1.60192401e-10 ... 4.54987941e-07
           1.58858114e-07 1.63611674e-06]]

In [80]: print(result[0][0])

         [1.4457107e-04 2.4273220e-07 2.2950026e-08 3.6162460e-06 3.3025008e-06
          1.4811909e-05 1.0588451e-05 1.5106481e-07 1.9169594e-10 7.4989863e-09
          1.6343594e-04 9.9448723e-01 2.4497449e-02 5.2189360e-05 2.1946430e-04]
```

Figure 11 Attack Prediction Result

5. Conclusion

From the training and validation results during this research it is obvious that machine learning in particular ANN can be used in intrusion detection. Not only in the detection but also the exact type of intrusion being carried out and this would go a long way in helping software engineers to know the exact remedy or deterrent to use.

The first objective was to design an architecture for cloud intrusion detection which was developed based on ANN using AutoEncoder shown in Figure 3.1. During the research process, the second objective was met due to extensive and rigorous research as documented in the literature review where it was noticed that an ANN with an AutoEncoder approach had not been documented.

The developed model can detect attacks automatically from network data supplied to the model and make the appropriate prediction with 99.96% accuracy which covers the third objective and the model tells the exact nature of the attack which helps in reducing response time addressing the fourth objective.

Recommendation

The following are recommendations made due to the discoveries during this study.

- Intrusion data from more attacks on cloud service providers should be made public to make machine learning models more robust.
- Grants can be given by the government to support researchers in this domain so that resources necessary to gather data and build models will be less of a burden on the developer.

Future work

This work can be improved in future work in the field. I hope that all the solutions provided in this work will enable researchers to have more efficient and influential work regarding new proposals on IDS in a cloud computing environment. To expand human knowledge there is a need to highlight some suggestions based on this research. The suggestions for further studies are as follows:

- Further research can be carried out using Caffe2 from Facebook or Azure ML from Microsoft.
- Data sources from less developed countries can also be added to improve the performance of the model in such regions.

Compliance with ethical standards

Disclosure of conflict of interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Reference

[1] Takabi, H., Joshi, J. B. D., & Ahn, G. J. (2010). Security and Privacy Challenges in Cloud Computing Environments. IEEE Security and Privacy, 8(6), 24–31.

[2] Prasad, S. N. S. E., Srinath, M. V., & Basha, M. S. (2015). Intrusion Detection Systems, Tools and Techniques – An Overview. International Journal of Science and Technology. Volume: 8, Issue: 35, pp. 1-7.

[3] Rani, M. G. (2019). A Review of Intrusion Detection Systems in Cloud Computing. Security and Privacy in Smart Sensor Networks, 253–283.

[4] Jangla, G. K., & Amne, D. A. (2015). Development of an Intrusion Detection System based on Big Data for Detecting Unknown Attacks. International Journal of Advanced Research in Computer and Communication Engineering, 4(12), 229–232.

[5] Hatef, M. A., Shaker, V., Jabbarpour, M. R., Jung, J., & Zarrabi, H. (2018). HIDCC: A Hybrid Intrusion Detection Approach in Cloud Computing. Concurrency Computation

[6] Shenfield, A., Day, D., & Ayesh, A. (2018). Intelligent Intrusion Detection Systems Using Artificial Neural Networks. ICT Express, 4(2), 95–99.

[7] Liu, Y., Liu, S., & Zhao, X. (2017). Intrusion Detection Algorithm Based on Convolutional Neural Network. Destech Transactions on Engineering and Technology Research. pp. 9–13.

[8] Ingre, B. (2015). Performance Analysis of NSL-KDD dataset using ANN. International Conference on Processing and Communication Engineering Systems, pp. 92–96.s

[9] Hajimirzaei, B., & Navimipour, N. J. (2019). Intrusion Detection for Cloud Computing Using Neural Networks and Artificial Bee Colony Optimization Algorithm. ICT Express, 5(1), 56–59.

[10] Manzoor, M. A., & Morgan, Y. (2017). Network Intrusion Detection System Using Apache Storm. Advances in Science, Technology and Engineering Systems, 2(3), 812–818.

[11] Fayaz, H. (2017). Cloud Security Enhancement Through Intrusion Detection System. International Journal of Advanced Research in Computer Science, 8(2), 2015–2017.

[12] Rani, M. G. (2019). A Review of Intrusion Detection Systems in Cloud Computing. Security and Privacy in Smart Sensor Networks, 253–283.

[13] Sharma, M., & Kumar, M. (2019). Intrusion Detection in Cloud Computing Environment Dynamic Remote Surveillance Scheme. Proceedings of International Conference on Sustainable Computing in Science, Technology, and Management (SUSCOM), Amity University Rajasthan, Jaipur- India.