INTERNATIONAL JOURNAL OF NATURAL AND APPLIED SCIENCES (IJNAS), VOL. 12, Special Edition (2019); P. 15 – 28, 6 TABLES, 8 FIGS.

Conceptualisation of Cyberattack Prediction using a Deep Learning Model

A. E. Ibor[1]*, F. A. Oladeji[2], O. B. Okunoye[2] and B I. Ele[1]

ABSTRACT

The state of the cyberspace portends uncertainty for the future Internet and its accelerated number of users. New paradigms add more concerns with big data collected through device sensors divulging large amounts of information, which can be used for targeted attacks. Though a plethora of extant approaches, models and algorithms have provided the basis of cyberattack predictions, there is the need to consider new models and algorithms, which dwell more on data representations other than task-specific techniques. Deep learning, which is underpinned by representation learning, has found widespread relevance in computer vision, speech recognition, natural language processing, audio recognition, and drug design. However, its non-linear information processing architecture can be adapted towards learning the different data representations of network connection vectors to classify benign and malicious network packets. In this paper, we model cyberattack prediction as a classification problem. Furthermore, the deep learning architecture was co-opted into a new model using rectified linear units (ReLU) as the activation function in the hidden layers of a deep feed forward neural network to realise a greedy layer-by-layer learning process that best represents the features useful for predicting cyberattacks in a dataset of connection vectors. The underlying algorithm of the model also performs feature selection, dimensionality reduction, and clustering at the initial stage, to generate a set of input vectors called hyper-features. The model is evaluated using CICIDS2017 dataset on a Python environment test bed. The results obtained show a 99.99% accuracy for the modeled attack types, and a very low false positive rate of 0. 00001. The proposed model is therefore fit for purpose in the prediction of cyberattacks.

INTRODUCTION

The expansion in attack surfaces has affected a huge number of resources in the cyberspace. According to Sharafaldin et al., (2018a) and Sharafaldin et al., (2018b), attacks involving Botnets, Bruteforce, SQL Injection, Denial of Service (DoS), Infiltration, Heartbleed and Distributed Denial of Service (DDoS) are having tremendous impacts on the security of network topologies. Similarly, Tobiyama et al., (2016) and Pai et al., (2017) agree that the attacker is developing new techniques that are able to evade network defences while compromising the internal structure of networks. The accessibility to big data adds more concerns to this already bloated attack surfaces. Though recent researches have delved into the modeling of predictive problems, it has become increasingly difficult to identify a single approach that solves the problem of cyberattacks in recent times.

Most approaches in the literature rely on task specific algorithms, thus requiring the need for an approach that relies more on representation learning. That is, an approach that can learn different attack classes from raw data instead of depending on preprogrammed tasks. Dong and Wang (2016), Erfani et al., (2016), Gulli and Pal (2017), Shone et al., (2018) and Marcus (2018) argue that representation learning can be helpful for extracting the intrinsic features in a modeled dataset in order to generalise on the test cases. In this sense, our model extracts the intrinsic features of connection vectors from network traffic to generate a cascade of concepts for learning the representation of different attack scenarios.

Furthermore, we optimise the accuracy of the model by combining unsupervised and supervised learning to model seven (7) attack types as shown in Table 3. We evaluate our model for accuracy, false positive rate, precision rate, recall rate, F-measure and entropy using CICIDS2017 dataset on a python environment test bed. Results of experimentation show an accuracy of 99.99% for all attack types. Thus, we clearly show that our model outperforms extant models in the prediction of cyberattacks.

Review of Related Literature

This section introduces the most recent researches in cyberattack detection and prediction with deep learning models in order to establish the relevance of the proposed approach. The extant literature discussed here will also highlight researches that benchmark public datasets such as KDD99, NSL-KDD and most recently, CICIDS2017 datasets. The modeling of cyberattack detection and prediction systems is fast tilting towards deep learning models. This is based on the fact that these models tend to learn the representations of attacks from the data instead of depending on traditional Machine Learning (ML) algorithms,

*Corresponding author. Email: ayei.ibor@gmail.com, foladeji@unilag.edu.ng,
[1]Department of Computer Science, University of Calabar, Calabar, Nigeria
[2]Department of Computer Sciences, University of Lagos, Lagos, Nigeria

which assume that data is static (Folino and Sabatino, 2016; Goodfellow et al., 2016).

For the purpose of clarity, a neural network (NN) is a mathematical model of the information processing and network structure of the human brain. It is a connectionist system consisting of many neurons in layers for communicating signals. A Deep Neural Network (DNN) is a neural network with several hidden layers. DNN typically learns data representations rather than perform task specific functions. In learning data representations, DNN relies on several layers of non-linear information processing. These layers can be adapted for supervised or unsupervised automatic feature learning and abstraction on several architectures such as deep neural networks, deep belief networks and recurrent neural networks (Deng and Yu, 2014; LeCun et al., 2015; Schmidhuber, 2015).

Shen et al., (2018) proposed an attack prediction approach called Tiresias xspace. This approach was based on a Recurrent Neural networks (RNN) to predict the possibility of imminent attacks on a host machine using preceding observations. In Nguyen et al., (2018), an approach that used deep learning to detect and isolate cyberattacks in mobile clouds was studied. The approach achieved an accuracy of 97.11% by applying the greedy layer-wise learning algorithm using Restricted Boltzmann Machine (RBM) for pre-training to perform non-linear transformation on its input vectors. The model is then fine-tuned using labeled data to achieve trained weights suitable for detecting attacks.

Similarly, Rhode et al., (2018) predicted the state of an executable code as either malicious or benign with recurrent neural networks (RNNs). The model depended on a short snapshot of behavioural data to obtain a 94% accuracy within the first five seconds of execution and an accuracy of 96.01% during the first twenty seconds of execution on unseen test set. In Aksu and Aydin (2018), Deep Learning with Support Vector Machine (SVM) algorithm is used to introduce an Intrusion Detection System (IDS) that could detect port scan attempts on a host machine. The approach was evaluated using the CICIDS2017 dataset and reported an accuracy rate of 97.80% for the deep learning model and 69.79% for SVM.

In the same sense, Al-Qatf et al., (2018) proposed a deep learning approach for feature learning and dimensionality reduction. The model could reduce training and testing time and also enhances the attack prediction accuracy of SVM. Sparse autoencoder was used to build the model for unsupervised pre-training and the transformed feature space was fed into the SVM algorithm to detect attacks. The model reported good detection accuracy for the KDD99 and NSL-KDD datasets. Rezvy et al., (2019) applied a deep autoencoded dense neural network algorithm to detect attacks on Fifth Generation (5G) and IoT networks. The paper presented a 2-step detection approach with deep autoencoders used for unsupervised pre-training to reduce high dimensional data to low-dimensional representation. The next stage performs supervised classification with a deep neural network to achieve good performance with an accuracy of 99.9%.

An approach called scale-hybrid-IDS-AlertNet was proposed by Vinayakumar et al., (2019). The approach can be used to monitor network traffic in real time in order to indicate the presence of anomalies representing attacks in network traffic. Scale-hybrid-IDS-AlertNet leveraged distributed and parallel machine learning algorithms with a diversity of optimisation techniques for handling a huge number of network and host-level events. Kasongo and Sun (2019) presented an IDS for detecting attacks on wireless networks. The popularity of wireless networks and ease of use has come with many security issues similar to those that affect conventional wired networks. To this effect, the paper discussed the application of a feed forward deep neural network for achieving an effective IDS using NSL-KDD dataset for evaluation.

Zhang et al., (2019) presented a technique that combined the effect of improved Genetic Algorithm and Deep Belief Network (GA-DBN) to develop an adaptive model for detecting attacks on Internet of Things (IoT). The model was simulated and evaluated using the NSL-KDD dataset to recognise attacks and reported the highest accuracy of 99.45% for DoS attacks. In the GA-DBN model, GA was used to select an optimal network structure through multiple iterations on the attack dataset. The DBN then deploys the optimal network structure for the classifying of attacks thus enhancing the classification accuracy.

Though several works have been proposed in the literature, most of them are computationally expensive, thus consuming very high amount of computing resources and at the same difficult to deploy in practice. The accuracy of most of these approaches still needs significant improvement hence the relevance of our approach in this context. Our approach uses a highly improved deep neural network by initially pre-training the samples through intelligent

clustering to achieve very high prediction accuracy required for the evolving attack surfaces.

MATERIALS AND METHOD

Dataset

The model is evaluated using the CICIDS2017 dataset, which is discussed in Gharib et al., (2016), Sharafaldin et al., (2018a), Sharafaldin et al., (2018b), Panigrahi and Borah (2018), and Sharafaldin et al., (2018c), as suitable for the modeling of cyberattack problems. The dataset was released by the Canadian Institute of Cybersecurity (CIC) in 2017, and contains latest threats and features, which were not addressed in older datasets such as the NSL-KDD and KDD99 datasets respectively. This most recent CICIDS2017 dataset has 83 features with 7 broad attack classes and 1 benign class. However, this paper will only consider the attack classes for the validation and testing of the model. Tables 1 summarises the attack classes in the modeled dataset. From extant literature, this dataset is well known for benchmarking attack detection and prediction models, and as such useful for evaluating the proposed model (Gharib et al., 2016; Yin et al., 2017; Kasongo and Sun, 2019; Sharafaldin et al., 2018c).

Table 1. Broad Attack Classes in the CICIDS2017 Dataset

Attack Class	Description	Number of Instances
Brute force	This attack is used for password cracking as well as the discovery of hidden pages and content in a web application	6708
Heartbleed	The heartbleed attack emanates from a bug in the OpenSSL cryptography library, which is an implementation of the Transport Layer Security (TLS) protocol	11
Botnet	This attack uses a number of devices connected over the Internet to circumvent and exploit vulnerable machines	1966
DoS	The Denial of Service (DoS) attack temporarily or indefinitely disrupts services on a host machine connected to the Internet. These services then become unavailable to the intended users for the period of the attack	11936
DDoS	Usually results from a botnet of compromised machines flooding the bandwidth or resources of a victim machine	127538
Web	Includes SQL injection, Cross-Site Scripting (XSS) and Brute force over HTTP: **SQL injection** is a code injection technique that is used to attack data-driven applications. An attacker can create a string of SQL commands in order to force the database to divulge its contents. **XSS attack** allows attackers to inject client-side scripts into web pages, which are viewed by other users. **Brute force over HTTP** enables an attacker to try a list of passwords to find the administrator's password.	2180
Infiltration	This is an attack that exploits the vulnerability of a software in order to execute a backdoor on the victim's machine. This can lead to attacks such as IP Sweep, port scan and service enumerations.	981

All experiments use a test split of 30% of the original datasets for the validation of the model.

The attack data undergoes two learning processes. First, unsupervised learning is used to perform feature engineering and clustering. Unsupervised pre-training is significant for solving the problem of spontaneous classification in order to improve the process of extracting valuable information, which will serve as input to the DNN.

For the second stage, supervised deep learning is used to train the model for making predictions on test data. The model performs cascaded learning based on a feed forward deep neural network (DNN) with h-hidden dense layers and a Softmax layer for classifying network attacks into one of the classes listed in Table 1. The entre prediction process is modeled as a multi-label classification problem.

The Proposed Model

The architecture of the proposed approach is depicted in Fig. 1.

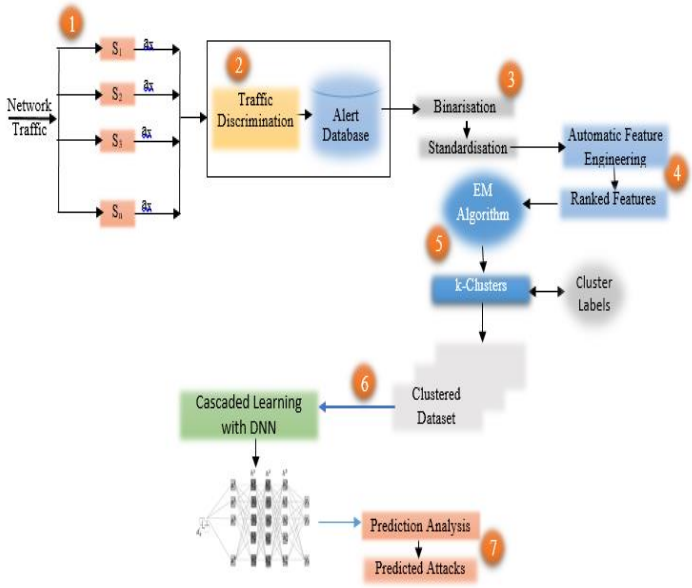

Fig. 1. Architecture of the Proposed Deep Learning Model.

The components of the model in Figure 1 include:

i. Network Traffic Capture

The first component represents the capture of network traffic from different sources across the network perimeter. Each source, say, $S_i; 1 \leq i \leq n$, can generate a set of alerts a_x, where the index x depicts an arbitrary number of alerts depending on the state of the network; x can be sufficiently large or small such as a_{10} representing a set of 10 alerts generated by source S_i at a certain timestamp, t. Each connection vector $x_i, 1 \leq i \leq n$ (alert or non-alert) is represented as an n-tuple, where a tuple consists of m set of features, say, f_m. Where $x_i = a_x$, a connection vector of type a_x is reported. A vector of features for each connection is defined as:

$$x_i = |f_1, f_2, \ldots, f_m| \quad (1)$$

ii. Network Traffic Discrimination

A network traffic filter is used at stage 2 (see Figure 6) to discriminate potential attack vectors from normal connection vectors. At this stage, an attack or normal connection is identified using a feature vector. This feature vector consists of 41 features, some of which include protocol type, service, flag, source and destination bytes. The feature vector depicting an alert is stored in an alert database, and other feature vectors are combined with the flagged alerts for normalisation.

iii. Normalisation

To achieve an error-free prediction, the captured alerts data is normalised. The normalisation process comes in 2 forms. First, the vectors x_i are used to create a matrix of m-features (f_m) and n-instances (x_n). The n x m matrix is processed with all categorical values converted to nominal values using label encoders. This is followed by the multi-label Binarisation of all class labels y_k. Secondly, the dataset is scanned for missing values in order to standardise the range of continuous initial variables or features. In this way, each variable or feature will contribute equally to the analysis of the modeled dataset. Standardisation or Z-score normalisation also involves transforming the dataset to comparable scales in order to achieve unbiased results.

Mathematically, a balanced dataset of connection vectors is achieved by subtracting the mean and dividing by the standard deviation for each value of each variable or feature in the dataset. That is,

$$x' = \frac{x - \acute{x}}{\sigma} \quad (2)$$

This is relevant for enhancing the convergence speed of the optimisation algorithm. The original feature vector is x, $\acute{x}$ is the mean of the feature vector and the standard deviation is σ.

Subsequently, we have a balanced dataset, `say, D, represented as follows:

$$D = \begin{bmatrix} x_1 f_1 & x_1 f_2 \dots & x_1 f_m \\ x_2 f_1 & x_2 f_2 \dots & x_2 f_m \\ x_n f_1 & x_n f_2 \dots & x_n f_m \end{bmatrix}, (x = x') \quad (3)$$

iv. Feature Engineering

Feature engineering is performed on the dataset with Principal Component Analysis (PCA) to generate a set of p uncorrelated principal components from the correlated connection vectors. Some of the features or variables in the dataset are highly correlated in such a way that they contain redundant information. Thus, it is a good practice, when modeling predictive problems, to remove linear correlations among features in a dataset. In this sense, PCA is used to reduce the feature space while still maintaining the variability in the dataset.

Given the dataset, D, with n-instances and m features or variables, PCA generates $min(n-1|,m)$ distinct principal components, which can be used to reconstruct the target output. In this way, the large dataset of connection vectors is easily represented by projecting it on more than a one dimensional vector (De la Hoz et al., 2015). The transformed low-dimensional representation is based on the preservation of the variance of the dataset and the ranking of the principal components. That is, the first principal component contains the largest possible variance, and this threshold decreases with each succeeding principal component. That is,

$$d = D(min(n-1), m) \quad (4)$$

The ranked principal components as the output of the feature engineering process are then used for the clustering process, at which point the attack dataset is completely standardised, thus able to enhance the convergence speed of the optimisation algorithm.

v. Clustering

The compressed dataset denoted by ***d***, comprising of principal components is used at this point to generate k number of clusters based on the EM algorithm. With clustering, the training of the model is improved by automatically categorising attack data. This can be useful in the early steps of an attack. EM performs clustering by initialising the mean and variance as the parameters for k probability distributions. The algorithm then alternates between the 2-step iterative processes as follows:

a. Expectation Step (E-Step): the probabilities required in the M Step are computed using the current estimates of the distribution parameters
b. Maximisation Step (M-Step): the distribution parameters with respect to maximum likelihood estimators are then recomputed using the probabilities from the E Step.

The shape of the cluster changes as these parameters are recomputed iteratively until the k-clusters are generated. Therefore, if we represent the EM algorithm as θ, we have:

$$d_k = \theta(d)_k = \theta(x, y) \quad (5)$$

Where, d_k is the clustered dataset by applying the EM algorithm θ on ***d***, k represents the generated number of clusters on ***d***. Since the dataset is 2-dimensional with the instances as a matrix, x[n, m], and the class labels as a vector, y, fitting x and y into the EM algorithm will generate a function $\theta(x, y)$, to match the instances (data points) to the class labels prior to input to the DNN, which is particularly significant for supervised learning. The labelled clusters from the EM algorithm are used as input to the DNN to improve its training and optimisation. The k^{th} cluster in d_k is represented as μ^k in the DNN as depicted in Figure 2.

vi. Cascaded Learning with Supervised DNN

Cascaded learning is performed at each layer of the DNN. Each layer passes its information to the next layer through the DNN without feedback connections. The model is trained using the constructed k-clusters and cluster labels, generated with the EM algorithm. A Feed Forward (FF) DNN with 5 layers (1 input layer, 3 hidden dense layers $(h^i, 1 \leq i \leq 3)$, and 1 output layer) is used. The DNN is shown in Figure 2.

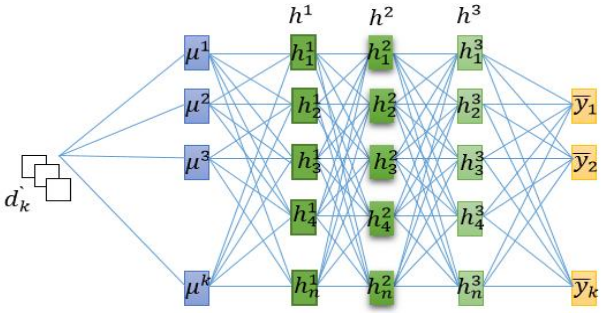

Fig. 2. The Feed Forward Deep Neural Network of the Model

vii. Prediction Module

The deep neural network learns a compressed representation of each cluster μ^k in the hidden layers. At the output layer, the Softmax function is used to classify this compressed representation. In reality, the Softmax function partitions the output such that the total sum is 1, which is equivalent to a categorical probability distribution (Agarap, 2018). Thus, the final layer comprises a single neuron for each of the attack classes. Each attack class yields a value between 0 and 1, which is inferred as a probability. The sum of the probability of the output is 1.

To compute the probability of an attack, we apply the Softmax function to each cluster class value. That is:

$$\acute{y} = Softmax(y_k) = \frac{e^{y_k}}{\sum_{k=1}^{n} e^{y_k}} \quad (6)$$

with $\acute{y}$ as the predicted class. We make predictions using equation (10), and the range of $\acute{y}$ (0, 1) indicates the accuracy of predictions. Next, we analyse the predictions made by the model with the help of a confusion matrix, and then compute the following performance metrics based on the work of Milenkoski et al, (2015):

a) **Accuracy of Prediction (ACC):** the rate of instances of attacks or normal connections predicted correctly. This is calculated as:

$$ACC = \frac{TP+TN}{TP+TN+FN+FP} \quad (7)$$

Where,

TP is True Positive: correct positive prediction; TN is True Negative: correct negative prediction; FN is False Negative: incorrect negative prediction and FP is False Positive: incorrect positive prediction.

b) **False Positive Rate (FPR)**: the rate of instances of attacks predicted as normal connections or vice versa denoted by:

$$FPR = \frac{FP}{TN+FP} \quad (8)$$

c) **Precision Rate (PR)**: the fraction of relevant instances in the dataset given as:

$$PR = \frac{TP}{TP+FP} \quad (9)$$

d) **Recall Rate (RR)**: the retrieved relevant instances over the total amount of relevant instances. RR calculated as shown in equation (13):

$$RR = \frac{TP}{TP+FN} \quad (10)$$

e) **F-Measure (F-Score or F1)**: a measure of the accuracy of the model computed as the weighted harmonic mean of the precision and recall of the model. F-measure is denoted by:

$$F1 = 2x\frac{PR.RR}{PR+RR} \quad (11)$$

f) **Cross Entropy (E)**: a measure of the performance of a classification model whose output is a probability value between 0 and 1. That is,

$$E = -\sum_{i=1}^{n} y_i log\ (\acute{y}_i) \quad (12)$$

In equation (12), n is the number of classes, y is the true class value and $\acute{y}$ is the predicted class value. A good model will have E that is 0 or close to 0. The consideration of the value of E is used to assess the efficiency of the model, i.e. E <0.15 is used as the benchmark for determining good performance by the model. The accuracy of prediction is interpreted by comparing each output from the Softmax layer with its corresponding true value. That is, the true values are one-hot-encoded such that a value of one (1) appears in the column corresponding to the correct attack class, otherwise a value of zero (0) is shown.

Underlying Algorithm of the Model

The model's underlying algorithm is given below.

Initialise all parameters and variables
Initialise number of connection vectors as: $x_i = \{f_1, f_2, \ldots, f_m\}$;
$1 \leq i \leq n$
Split D to obtain x instances and y labels: D = D(x, y)
Generate principal components from D: $d = D(min(n-1), m)$
Generate k clusters from d: $d_k = \theta(d)_k = \theta(x, y)$
For k = 1 to numberofClusters
Assign clusters labels: $\mu^k = \theta(d)_k$
end For
j = 0
For i = 1 to 5
if (i==1) return inputLayer
units = 66
if (i==5) return outputLayer
units = 5
else
return h^i *as hidden layers*
j = j+ 1
endif
endif
end For
i = 1
Repeat
If(i==1) units = 750 for h^1
If(i==2) units = 500 for h^2
If(i==3) units = 250 for h^3
i = i + 1
Until (i>j)
For n = 1 to epochs
$f(\mu) = max(\mu, 0)$
$\dot{y} = Softmax(y_k) = \frac{e^{y_k}}{\sum_{k=1}^{n} e^{y_k}}$
End For

Feature Ranking

The model was trained using a generated number of clusters (k-clusters) from 66 of the 78 features of the CICIDS2017 dataset. The k-clusters were formed from a feature space with a reduced dataset, d, using PCA. PCA generated p-principal components, representing a compressed feature space as mentioned in Vasan and Surendiran (2016). With PCA, the variance in the data was optimised to generate the representative subset of features for training the model.

The model was trained using 70% of the original samples in the dataset and could generalise to an unknown dataset while deriving an accurate estimate of model prediction performance.

Testbed of the Experiments

We implemented the DNN using a TensorFlow backend in Python 3.6 on an Ubuntu 18.04 64-bit operating system with Keras and ScikitLearn libraries (Abadi et al., 2016; Gulli and Pal, 2017; Hackeling, 2017). The system properties of the machine used for experimentation are shown in Table 2.

Table 2. System Properties of the Implementation Machine

Host Operating System	Ubuntu 18.04
Processor	Intel ® Core ™ i3 6100U CPU @2.30 GHz 2.30GHz
RAM	4.00GB
System Type	64-bit Operating System, x-64 based processor

RESULTS

The experimental results for this implementation are discussed in this section. The tuning of the hyperparameters for the deep neural network and the predictions made at the completion of the execution of the Python code used for the implementation are presented. Furthermore, the performance of the model is benchmarked against state-of-the-art approaches and findings show that the proposed model outperforms other models in terms of accuracy and false positive rate.

Configuration and Tuning of Hyperparameters

The deep neural network performs computations on the transformed feature space, which is basically comprised of numeric values. In each layer, the feature space is compressed and abstract features for the optimal representation of the original dataset are learned, transformed and passed on to subsequent layers in a cascaded learning technique. To achieve the required accuracy, the DNN must have an adequate number of layers. Similarly, each layer must have an adequate number of neurons in order to be able to represent the different output classes during predictions.

Hyperparameter tuning helps the model to generalise on the training data while finding the distinctions between the output classes (Rhode et al., 2018). It is noteworthy to mention that there are several ways the hyperparameters of the DNN can be configured and tuned. During the experiments, different configurations were chosen and tested. In tuning the model, it was found that a depth of 3 hidden layers produced the best results.

Additionally, the number of neurons per layer and the number of epochs for training the model were randomly chosen. Through this random search process, the hyperparameter values for the optimal performance of the model were chosen. Subsequently, the hyperparameters that generated the best performance of the model are summarised in Table 3.

Table 3. Hyperparameters for an Optimal Model Performance

Hyperparameter	**Experimented Values**	**Selected Configuration**
Number of hidden dense layers	1 to 10	3
Neurons per layer	100 to 1000	750, 500, 250
Number of epochs	10 to 500	50, 100, 200
Batch Size	32, 54, 128	64
Learning Rate (lr)	0.0001, 0.01, 0.1, 0.2, 0.3	0.01, 0.1

Training and Testing of the Model

The model was trained over 50, 100 and 200 epochs respectively with 74,146 training samples and 31,778 samples for validation and testing. The visualisation of the pre-trained samples is illustrated in Figure 3.

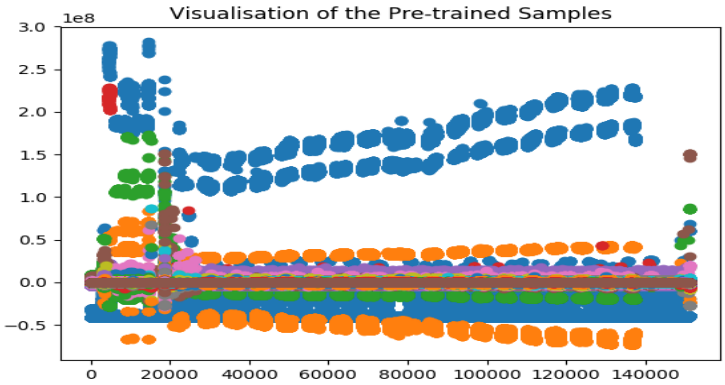

Fig. 3. Visualisation of the Pre-trained Samples

The training of the model is based on three sequence of processes as mentioned in Kasongo and Sun (2019). These processes include:

i) Forward propagation, in which case, each layer passes information to the next layer

ii) Back propagation of the error computed during the cascaded learning process

iii) The update of the weights and biases across the DNN.

The update of all the weights and biases is performed using a backpropagation algorithm, which is optimised with stochastic gradient descent (SGD) and an Adam updater. A standard categorical cross-entropy loss function is used at the output layer. This loss function measures the model's classification performance whose output is a probability value within the range of 0 and 1. When the predicted probability diverges from the actual value, the value of cross-entropy loss increases and tends towards 0 as the predicted probability converges to the actual value. A cross-entropy loss of 0 implies a perfect model.

The model is trained with an initial learning rate of 0.1. A lower learning rate of 0.01 was subsequently introduced to test the model's predictions over the training and test data. However, the optimisation took a longer time due to the tiny steps towards the minimum of the loss function. It is important to note that choosing the appropriate learning rate is significant for achieving optimal predictions as a high learning rate may result in the training not converging, or even diverging. Consequently, changes in weights can be large enough to allow the optimizer overshoot the minimum and make the loss worse.

DISCUSSION

For each training period, the performance of the model was recorded using such metrics as Accuracy, Recall Rate, Precision Rate, F-measure and Cross Entropy for the modeled dataset and selected learning rate. The results obtained are shown in Table 4.

Table 4. Performance Metrics of the Model using lr = 0.1

Metrics	Class	DDoS	DoS	Web attack	Bruteforce	Heartbleed	Botnet	Infiltration
	Epochs							
ACC	50	99.92281	99.96227	99.99339	99.91590	99.99348	99.96719	99.92402
	100	99.88429	99.96699	99.99780	99.90909	99.99348	99.97375	99.92626
	200	99.92281	99.95755	99.99780	99.90908	99.99348	99.96501	99.92849
PR	50	99.98897	99.98015	99.99338	99.93828	100.00000	99.98236	99.99118
	100	99.98236	99.98456	99.99779	99.93388	100.00000	99.98677	99.99118
	200	99.98897	99.97133	99.99779	99.92947	100.00000	99.99118	99.99338
RR	50	99.99779	99.98456	100	99.98015	99.99338	99.98456	99.93388
	100	99.99779	99.98456	100	99.97795	99.99338	99.98677	99.93608
	200	99.99779	99.98897	100	99.98236	99.99338	99.97354	99.93608
F1	50	99.99338	99.98236	99.99669	99.95921	99.99669	99.98346	99.96252
	100	99.99008	99.98456	99.99890	99.95591	99.99669	99.98677	99.96362
	200	99.99338	99.98015	99.99890	99.95591	99.99669	99.98236	99.96472
E	50	0.00011	0.00020	0.00007	0.00062	0.00000	0.00018	0.00009
	100	0.00018	0.00015	0.00002	0.00066	0.00000	0.00013	0.00009
	200	0.00011	0.00029	0.00002	0.00071	0.00000	0.00009	0.00007

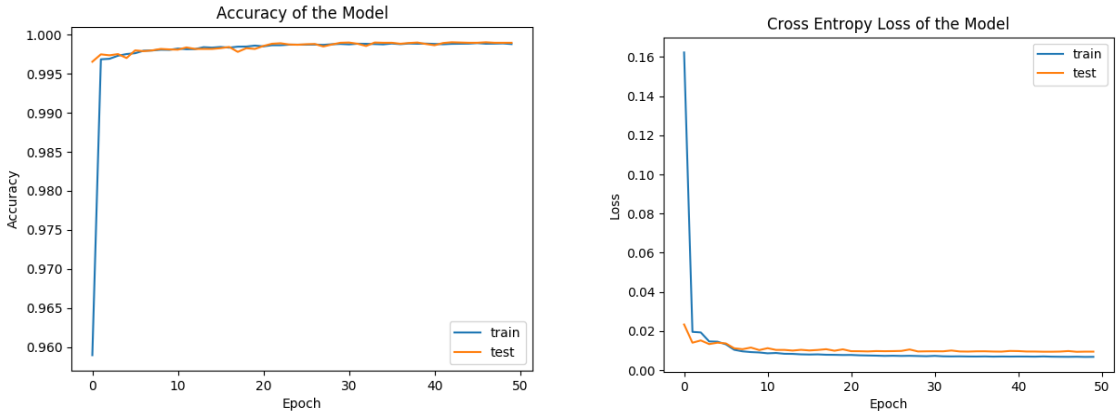

Fig. 4. Accuracy and Cross Entropy Loss of the Model for 50 Epochs

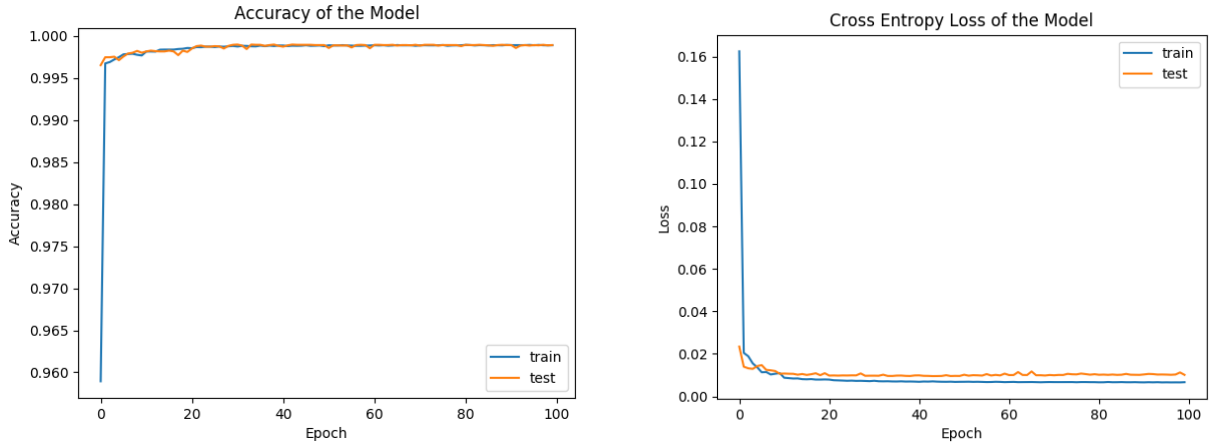

Fig. 5. Accuracy and Cross Entropy Loss of the Model for 100 Epochs

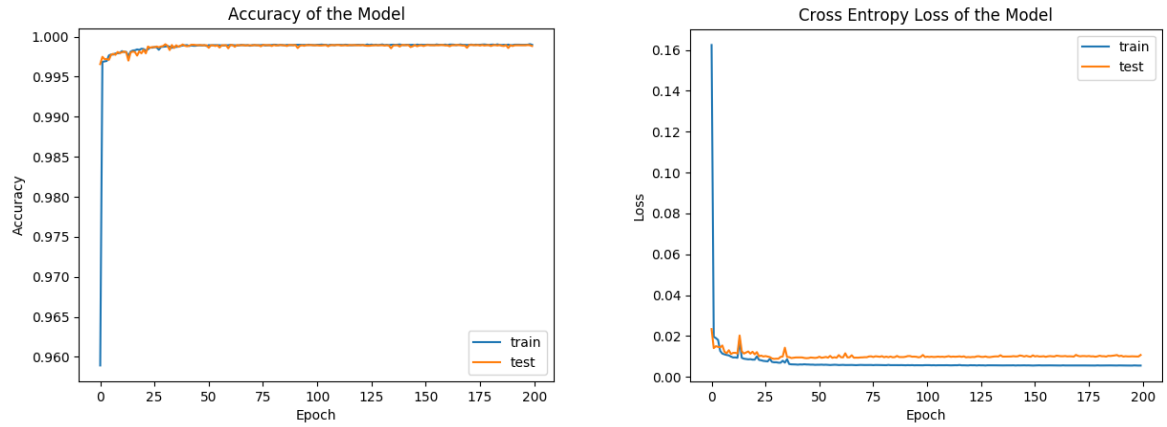

Fig. 6. Accuracy and Cross Entropy Loss of the Model for 200 Epochs

As shown in Figures 4, 5, and 6, there is no significant deviation between the train and test curves showing that the model is able to learn a representation of the attack clusters from the raw attack data. These visualisations were produced by plotting the accuracy and cross entropy loss of the model against the number of epochs during the training and testing phases of the experimentation. For all cases, the model showed an improvement in the training and testing phases over the number of iterations (epochs) used.

At 100 epochs, the accuracy improved while the entropy loss decreased as the model learns the different representations of the attack classes more extensively as illustrated in Figure 5. Further training and testing of the model at 200 epochs, as depicted in Figure 6, fine tunes its performance to achieve stability in the prediction of cyberattacks. This shows that our model has a strong modeling ability and yields very high accuracy for predicting multi-class attacks.

In Table 5, the overall performance of the model is shown. The model showed improved performance as the number of epochs increased. This implies that the model is able to learn more abstracted features of the dataset during the training phase at each layer to make better generalisations for predicting the modeled attack types while minimising the cross entropy loss and the false positive rate.

Table 5. Overall Performance of the Model for lr = 0.1

Epochs / Metrics	50	100	200
ACC	99.99283	99.99522	99.99761
FPR	0.00008	0.00005	0.00003
PR	99.91669	99.94444	99.97223
RR	100	100	100
F!	99.95833	99.97221	99.98611
E	0.00083	0.00056	0.00028

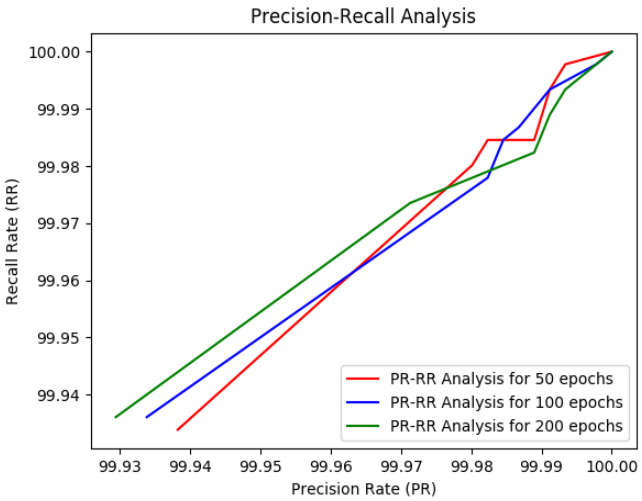

Fig. 7. Precision – Recall Analysis of the Proposed Model

Furthermore, a Precision-Recall analysis is performed by plotting the Precision Rate (PR) on the x-axis against the Recall Rate (RR) on the y-axis using the data in Table 4, to ascertain the stability of the model in making predictions. The PR-RR Analysis is represented in Figure 7. The 3 plots show very high stability and affirm the suitability of the proposed model for multi-class predictions.

Comparison of Results

In this approach, a Softmax function is used at the output layer with classification probability $\acute{y}$ given by equation (6). The range of $\acute{y}$ is (0, 1), equivalently (0, 100), thus the model demonstrates more than 99% prediction accuracy as depicted by the test plots of Figures 4, 5 and 6. Similarly, the cross entropy loss of the model indicates a good classifier. FPR values, which are used as prediction errors of the model were minimal, implying that only a few instances of the attack data were misclassified or predicted incorrectly. We benchmarked the results of experimentation of our model against extant state-of-the-art techniques in deep learning as shown in Table 6. The comparison in Table 6 shows that our model outperforms extant approaches as confirmed by Figure 8.

Table 6. Performance Comparison of the Proposed Model with Extant State-of-the-art Approaches

Approach	Accuracy	False Positive Rate
Proposed Model	99.99%	0.00001
Rezvy et al., (2019)	99.9%	0.1
Vinayakumar et al., (2019)	93.5%	6.45
Kasongo and Sun (2019)	99.54%	0.43
Zhang et al., (2019)	99.45%	0.54

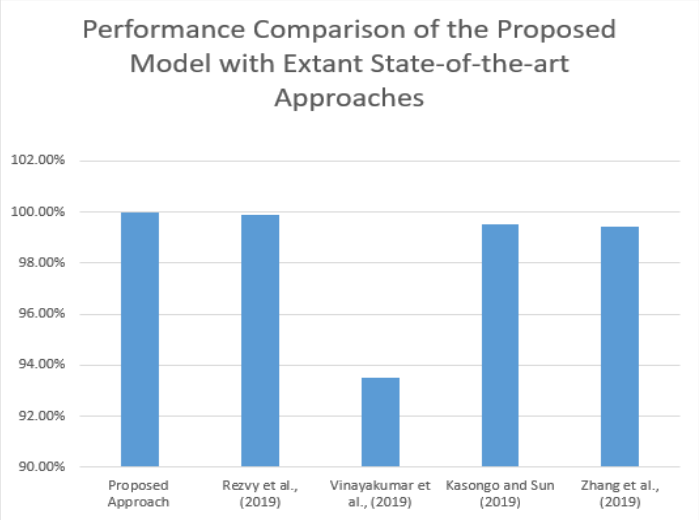

Fig. 8. Visualising the Accuracy of the Proposed Model against other Approaches

CONCLUSION

We introduced a deep learning model for predicting cyberattacks in this paper. We demonstrated the relevance of this model with the use of representation learning to tune the parameters of the model. The model extracts and abstracts features to realise non-linear transformation of the input data to output a compressed feature space, in which it learns the features to optimally choose at each layer to improve the classification of the multi-label problem. Unsupervised cluster-based training of the model improved the learning process of the deep neural network through the use of PCA and EM algorithm prior to supervised training. Furthermore, we evaluated the model using CICIDS2017 as the benchmark dataset. This dataset has a large set of connection vectors of evolving attacks, which enabled us to tune the model to learn different attack types to make accurate predictions. Finally, we obtained a prediction accuracy of 99.99% for the modeled attack types, thus outperforming extant approaches for the same problem domain.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., and Kudlur, M. (2016). Tensorflow: A system for large-scale machine learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (pp. 265-283).

Agarap, A.F., 2018. Deep Learning using Rectified Linear Units (ReLU). arXiv preprint arXiv:1803.08375.

Aksu, D., and Aydin, M. A. (2018), December). Detecting port scan attempts with comparative analysis of deep learning and support vector machine algorithms. In 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT) (pp. 77-80). IEEE.

Al-Qatf, M., Lasheng, Y., Al-Habib, M., and Al-Sabahi, K. (2018). Deep Learning Approach Combining Sparse Autoencoder With SVM for Network Intrusion Detection. IEEE Access, 6, 52843-52856.

Cho, K. (2014). Foundations and advances in deep learning.taxonomy, and future directions. Computer Communications, 107, pp.30-48.

De la Hoz, E., De La Hoz, E., Ortiz, A., Ortega, J. and Prieto, B., 2015. PCA filtering and probabilistic SOM for network intrusion detection. Neurocomputing, 164, pp.71-81.

Deng, L., and Yu, D., 2014. Deep learning: methods and applications. Foundations and Trends® in Signal Processing, 7(3–4), 197-387.

Dong, B., and Wang, X. (2016, June). Comparison deep learning method to traditional methods using for network intrusion detection. In 2016 8th IEEE International Conference on Communication Software and Networks (ICCSN) (pp. 581-585). IEEE.

Erfani, S. M., Rajasegarar, S., Karunasekera, S., and Leckie, C. (2016). High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. Pattern Recognition, 58, 121-134.

Folino, G., and Sabatino, P. (2016). Ensemble based collaborative and distributed intrusion detection systems: A survey. Journal of Network and Computer Applications, 66, 1-16.

Gharib, A., Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. (2016, December). An evaluation framework for intrusion detection dataset. In 2016 International Conference on Information Science and Security (ICISS) (pp. 1-6). IEEE.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. MIT press.

Gulli, A., and Pal, S. (2017). Deep Learning with Keras. Packt Publishing Ltd.

Hackeling, G. (2017). Mastering Machine Learning with scikit-learn. Packt Publishing Ltd.

Kasongo, S. M., and Sun, Y. (2019). A Deep Learning Method With Filter Based Feature Engineering for Wireless Intrusion Detection System. IEEE Access, 7, 38597-38607.

LeCun, Y., Bengio, Y., and Hinton, G., 2015. Deep learning. nature, 521(7553), 436.

Marcus, G. (2018). Deep learning: A critical appraisal. arXiv preprint arXiv:1801.00631.

Milenkoski, A., Vieira, M., Kounev, S., Avritzer, A. and Payne, B.D., 2015. Evaluating computer intrusion detection systems: A survey of common practices. ACM Computing Surveys (CSUR), 48(1), p.12.

Nguyen, K. K., Hoang, D. T., Niyato, D., Wang, P., Nguyen, D., and Dutkiewicz, E. (2018, April). Cyberattack detection in mobile cloud computing: A deep learning approach. In 2018 IEEE Wireless Communications and Networking Conference (WCNC) (pp. 1-6). IEEE.

Pai, S., Di Troia, F., Visaggio, C. A., Austin, T. H., and Stamp, M. (2017). Clustering for malware classification. Journal of Computer Virology and Hacking Techniques, 13(2), 95-107.

Panigrahi, R., and Borah, S. (2018). A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. International Journal of Engineering & Technology, 7(3.24), 479-482.

Rezvy, S., Luo, Y., Petridis, M., Lasebae, A., and Zebin, T. (2019, March). An efficient deep learning model for intrusion classification and prediction in 5G and IoT networks. In 2019 53rd Annual Conference on Information Sciences and Systems (CISS) (pp. 1-6). IEEE.

Rhode, M., Burnap, P., and Jones, K. (2018). Early-stage malware prediction using recurrent neural networks. computers and security, 77, 578-594.

Schmidhuber, J., 2015. Deep learning in neural networks: An overview. Neural networks, 61, 85-117.

Sharafaldin, I., Gharib, A., Lashkari, A. H., and Ghorbani, A. A. (2018a). Towards a reliable intrusion detection benchmark dataset. Software Networking, 2018(1), 177-200.

Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. (2018b, January). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In ICISSP (pp. 108-116).

Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. (2018c, January). A Detailed Analysis of the CICIDS2017 Data Set. In International Conference on Information Systems Security and Privacy (pp. 172-188). Springer, Cham.

Shen, Y., Mariconti, E., Vervier, P. A., and Stringhini, G. (2018). Tiresias. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18. doi:10.1145/3243734.3243811

Shone, N., Ngoc, T. N., Phai, V. D., and Shi, Q. (2018). A deep learning approach to network intrusion detection. IEEE Transactions on Emerging Topics in Computational Intelligence, 2(1), 41-50.

Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., and Yagi, T. (2016, June). Malware detection with deep neural network using process behavior. In 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC) (Vol. 2, pp. 577-582). IEEE.

Vasan, K.K. and Surendiran, B., 2016. Dimensionality reduction using Principal Component Analysis for network intrusion detection. Perspectives in Science, 8, pp.510-512.

Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., and Venkatraman, S. (2019). Deep Learning Approach for Intelligent Intrusion Detection System. IEEE Access, 7, 41525-41550.

Yin, C., Zhu, Y., Fei, J., and He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. Ieee Access, 5, 21954-21961.

Zhang, Y., Li, P., and Wang, X. (2019). Intrusion Detection for IoT Based on Improved Genetic Algorithm and Deep Belief Network. IEEE Access, 7, 31711-31722.