IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received July 15, 2020, accepted August 10, 2020, date of publication August 27, 2020, date of current version September 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3019931

Implementing a Deep Learning Model for Intrusion Detection on Apache Spark Platform

MOHAMED HAGGAG[1,3], (Member, IEEE), MOHSEN M. TANTAWY[2], AND MAGDY M. S. EL-SOUDANI[3], (Senior Member, IEEE)

[1]Department of Electronics and Communication Engineering, Misr University for Science and Technology, 6th of October City 12566, Egypt
[2]Network Planning Department, National Telecommunication Institute (NTI), Cairo 11768, Egypt
[3]Electronics and Electrical Communications Department, Faculty of Engineering, Cairo University, Cairo 12613, Egypt

Corresponding author: Mohamed Haggag (mohamed.morsy@must.edu.eg)

ABSTRACT Internet evolution produced a connected world with a massive amount of data. This connectivity advantage came with the price of more complex and advanced attacks. Intrusion Detection System (IDS) is an essential component for security in modern networks. The IDS methodology is either signature-based detection or anomaly behavior detection. Recently, researchers adopted Deep Learning (DL) because it has a better performance than traditional machine learning algorithms. The use of DL to produce a model for the IDS may take a long time because of computation complexity and a large number of hyperparameters. Different DL models for IDS on Apache Spark have been implemented in this article. This article uses the famous Network Security Lab - Knowledge Discovery and Data Mining (NSL-KDD) dataset and presents a computation delay comparison between Apache Spark and regular implementation. Moreover, an enhanced model is used to improve attack detection accuracy.

INDEX TERMS Intrusion detection, bigdata, Hadoop, apache spark, deep learning.

I. INTRODUCTION

Computer networks have proliferated over the years, adding to social and economic growth. The Internet Security Threat Report (ISTR) states that 1 in 13 Web requests is malware. The spam rate in e-mails had increased to 55%, ransomware had risen to 46 %, and other Internet threats [1]. Cybercrime and threat actions have grown and have become a critical threat. This growth promoted an increase in network security importance. By analyzing packets captured from the network, IDS helps to detect threats [2].

There are many threats like Denial of Service (DoS), which denies or prevents legitimate user's resources on a network by introducing undesired traffic. Also, malware is malicious software that uses a vulnerability in the computer network machines to gain some advantage [2]. IDS developed to counter these attacks.

The conventional IDS suffered from false detection, which is categorized into positive or negative. These false detections are a burden on the network administrator. This burden made the researchers try to develop an IDS that has a high accuracy of detection and low false detection rate [3]. Signature-based IDS identifies only the known attacks, which makes signature-based IDS unable to detect unknown attacks. Anomaly-based IDS trains on regular traffic and abnormal traffic dataset to identify an attack. Diverse machine learning models have been presented to operate the IDS functions but produced numerous imperfections viz low throughput and high false detection rates.

Machine learning model training has issues that slow down the process, such as the size of the dataset and the optimization parameters to build the most fitting model. These difficulties made the researchers look for a more appropriate approach. A possible solution to these problems is the use of the Apache Spark tool, which is one of the fastest cluster computing frameworks, and it is an open-source distributed programming tool for clusters. Also, Spark executes the operations in memory [4].

The development in computational capabilities expedited Deep Learning (DL) for various applications in many areas such as image processing, natural language processing, computer vision, and the focus of this article, the IDS [5].

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

The article workflow will follow these steps:

1) NSL-KDD dataset has been investigated rigorously. Two features need treatment before it enters the model training.
2) Also, the NSL-KDD dataset suffers from a class imbalance problem, and a hybrid solution is presented for it.
3) This article presents the use of Apache Spark in the IDS implementation process.
4) For the sake of comparison, this article implements traditional machine learning algorithms, which are Decision Tree classifier, K-Neighbor classifier, and Support Vector Machine (SVM).
5) Three Apache Spark cluster configuration is used in the implementation to study the reduction in the computational delay.
6) IDS have been implemented on Apache Spark using three DL models, which are Multilayer perceptron(MLP), Recurrent Neural Network(RNN), and Long-Short Term Memory(LSTM).
7) This article presents different arrangements for each network type.
8) The performance of the selected model with the highest detection accuracy outperformed many proposed schemes in terms of accuracy and time.

The main contributions of this article can be summarized as follows.

1) Propose a Deep learning-based Intrusion detection system with high accuracy compared with previously developed systems.
2) Use Spark Cluster configuration to reduce the training process while implementing the IDS with different hyperparameters.
3) Solve problems related to the selected dataset, NSL-KDD, such as class imbalance.

The paper is organized as follows; the next section provides a brief of IDS previously developed algorithms. Section 3 gives the details of the system model for this article. Section 4 describes the experimental setup. The proposed model performance analyses are presented in Section 5. Finally, Section 6 concludes the paper.

II. RELATED WORK

Many papers proposed the use of Machine Learning (ML) in IDS, and there are two methods. The first one is the traditional machine learning algorithms, and the second method is Deep Learning algorithms. For the first method, the authors in [6] presented a K-nearest neighbor(KNN) to build an IDS for Wireless Sensor Network(WSN). The training dataset was not provided, but the primary function of that IDS was to prevent flooding attacks. The use of the Random-Forest (RF) classifier for IDS is introduced in [7] to implement an IDS. They used the NSL-KDD dataset to evaluate their model and presented the detection accuracy on the training dataset, but their evaluation did not contain any test data. A hybrid system was introduced in [8]. The model had two classifiers; the first is the Support Vector Machine(SVM), and the second is a Decision Tree. The hybrid system offered better detection accuracy.

The work in [9] summarizes the IDS ML algorithms' accuracy of the traditional methods. They presented the intrusion detection accuracy of six algorithms. The list of intrusion detection accuracies was as follows; (1) 74.6% for J48 which is an open-source Java implementation of the C4.5 algorithm of a decision tree, (2) 74.40% for Naive Bayes, (3) 75.40% for the C5.0 algorithm of a decision tree, (4) 74.00% for Random Forest, and (5) 74.00% for SVM. Another IDS ML method is proposed in [10], which is called an ensemble because many weak learners build it. Weak learners are classifiers that have poor detection accuracy. The weak learners used in the paper were J48, C5.0, Naïve Bayes, and Rule-Based classifiers (PART). The accuracy reached by this algorithm is 78.14%.

The second method, which is DL, has shown that its accuracy effectively exceeds traditional approaches [11]. In [12], the authors use self-taught learning (STL) on the NSL-KDD dataset for anomaly detection, and the accuracy of the outcome was 79%. A Restricted Boltzmann(RBM) model had been proposed in [13], and this model had feature selection using one Hidden Layer and utilized the KDD Cup '99 dataset. An artificial neural network (ANN), consists of two hidden layers, each with one hundred neurons, is proposed in [14]. The IDS accuracy for this system was 78.51%.

The authors in [15] aspired a Deep Neural Network (DNN) with 100 hidden units. They used a GPU to enhance the performance and used the KDD 99' dataset. The authors suggested that both Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM) models are better for improving detection accuracy.

The authors in [16] presented a convolutional neural network (CNN). The model has two convolutional layers, two pooling layers, and two fully connected MLP layers. The paper tested their model against KDD 99' dataset, and the accuracy was 99.84%.

Then CNN is also implemented on the NSL-KDD dataset in [17]. The CNN needs an additional step than any other DL models. Since CNN's primary purpose is image recognition, it has a stage that is converting the 41 features into matrices that can be dealt with as an image. CNN reached a detection accuracy of 79.48%. On the other hand, RNN is introduced in [9] to create an IDS. A regular processor is used, which increased the training time, so they suggested the use of GPU and Tensorflow to reduce the training time. This model had a good accuracy of 81.29%.

A deep learning method, which is a self-taught learning-based IDS that combines a sparse autoencoder and an SVM, is presented in [18]. The primary function of the sparse autoencoder(SAE) is to reduce the features of the NSL-KDD before it is classified. The SVM classifies the output of the SAE. The accuracy achieved with this method was 80.48%.

The authors in [19]presented a model based on autoencoder, followed by Multilayer Perceptron (MLP). The model is structured as one input layer, five hidden layers, and one output layer; the input layer has 122 neurons, while the output is five neurons. The output layer uses the softmax function. The model accuracy is 79.74%.

A hybrid model introduced using RNN and LSTM is given in [20]. The dataset used for evaluation was the KDD 99' dataset. The authors did not mention the duration of the training time, but a long time is expected for this model.

Another approach presented for LSTM is hierarchical LSTM[21]. The authors produced a three layers' network, which contains two layers of LSTM and one Layer of a fully connected layer. This network reached an accuracy of 83.5% for the NSL-KDD test dataset and 69.73% for KDD-test-21.

The main drawback of using DL is that the training duration to get the best model takes a long time[9]. The authors in [22]and [23] used the Spark as a platform for machine learning. They showed that Spark reduces the execution delay of the training process for Machine learning, but they did not use it for DL. The authors in [24] used Spark for MLP implementation and presented substantial work. They considered that the implementation process is confidential, so it was not given. They tested many datasets, including NSL-KDD. The detection accuracy for NSL-KDD reached 78.5%.

From the above, it can be concluded that Apache Spark can reduce the training time process, and the DL algorithms improved intrusion detection accuracy. RNN and LSTM algorithms perform better than other algorithms, and the NSL-KDD dataset is well known to test the new models. Therefore, we build our model on Apache Spark with the aim to solve dataset problems and to achieve better accuracy.

III. SYSTEM MODEL

This article proposes an IDS system based on deep learning algorithms for the attacks included in the NSL-KDD dataset. The training process will be implemented on Apache Spark. For short, we refer to this model as DLS-IDS (Deep Learning Spark Intrusion Detection System). The DLS-IDS solves the NSL-KDD dataset related problems, defines the best model arrangement and model elements to produce a high intrusion detection accuracy, as well as determines the best Apache Spark cluster configurations to reduce the implementation process time.

The DLS-IDS workflow consists of four main blocks; the first block is to choose and explore the dataset, and the second block is dataset preprocessing. The third block is the class imbalance solution, and the last block is model training using Apache Spark, as shown in Figure 1. The model training will be on three different networks MLP, RNN, and LSTM.

The last block in Figure 1 is the Apache Spark cluster. Figure 2 illustrates the architecture and the workflow within the Spark cluster that is used in the DLS-IDS.

Spark architecture contains three main parts; the driver, the cluster manager, and the worker. The driver includes the Spark context, the cluster manager distributes the workload between the worker nodes, and the worker node performs the tasks as shown in Figure 2. The Spark cluster components workflow goes as follows. The user sends the code with the data and the number of workers. The Spark context receives the task from the user. It uses the cluster manager to distribute the workload between the workers, then sends the data, the model arrangement, and initiates parameters for each worker. Also, it sends the number of Resilient Distributed Datasets (RDD). The worker performs feedforward then computes the gradients to update the parameters. After completing the training process, the worker generates the partial model, which the Spark driver will receive. The Spark driver uses all partial models from the workers to average the parameters of the model to obtain the deep model.

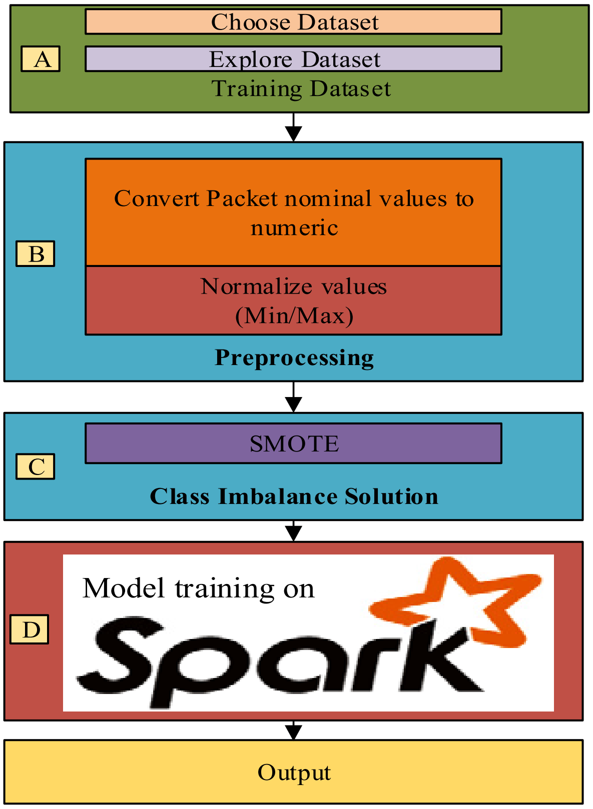


FIGURE 1. The DLS-IDS model block diagram.

The next subsections will illustrate the DLS-IDS model workflow in details.

A. DATASET EXPLORATION

The first block in the DLS-IDS, shown in Figure 1, is the training dataset. This block has two steps, which choose the dataset, and explore it.

As stated before, there are two well-known datasets KDD-Cup and NSL-KDD. The KDD-Cup99 has a tremendous amount of redundant data points. The NSL-KDD was built by reducing these repetitive points of the KDD-Cup99 dataset [25],[26]. Therefore, we use NSL-KDD dataset for training and testing.

The second step in the first block is data exploration of the chosen training dataset. The NSL-KDD dataset includes four files, two of them for training the model, and the other

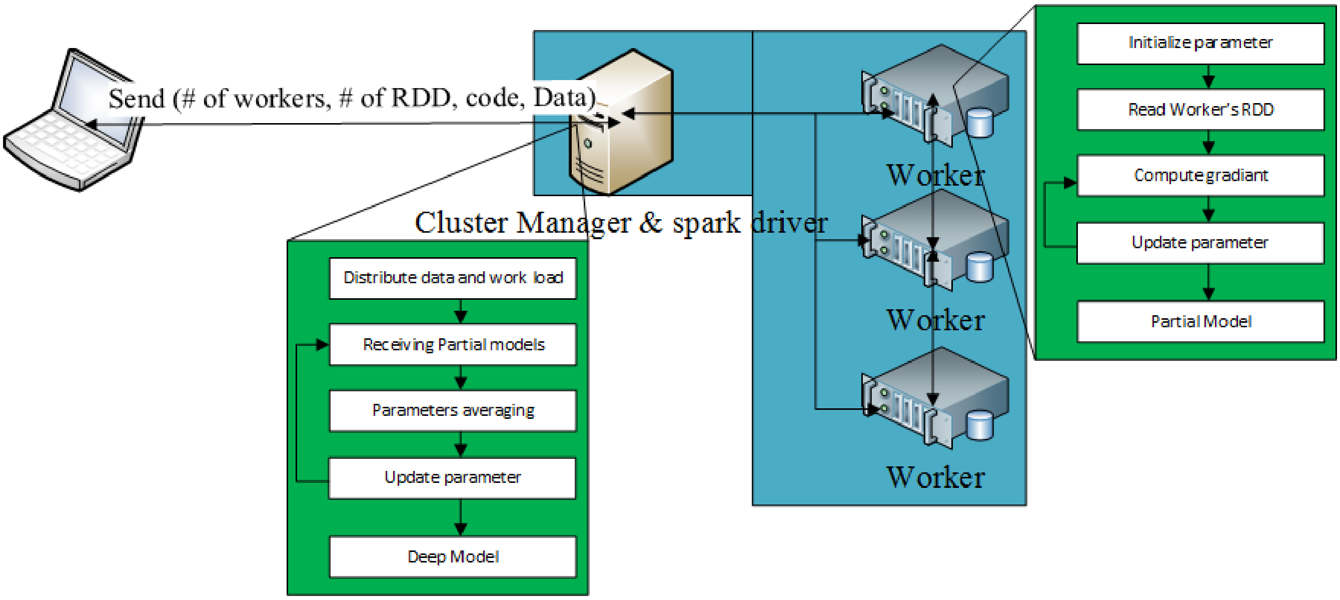


FIGURE 2. Spark model for deep learning.

two for testing the model. These four files are the primary training dataset KDDTrain+, the smaller training dataset KDDTrain+_20%, the primary testing dataset KDDTest+, and KDDTest-21, which is a smaller testing dataset[27].

The attacks that exist in NSL-KDD dataset are one of the following four types:

- Denial of Service Attack (DoS), which is an attack that targets service availability by consuming computing and memory resources.
- User to Root Attack (U2R), which is an attack that begins with access as a legitimate user on the network and then endeavors to exploit a vulnerability to obtain root access.
- Remote to Local Attack (R2L), which is an attack in which a user signs in as a remote user, then try to detect the system vulnerabilities and exploit the privileges as if it is a local user.
- Probe Attack (Probe), which is a trial to collect data about computer networks to use these data in later attacks.

In the NSL-KDD dataset, each data point is composed of 41 features and a label that is maybe normal or an attack [27]. Table 1 shows the NSL-KDD dataset features.

The dataset exploration process resulted in three discoveries. The first is that feature number 15 "su_attempted", which is an attempt to log in as a superuser, has three values (0,1,2). The value 2 is not possible because this feature is binary, so the user is either tried to log in as a superuser or not. The second is that feature number 20 "num_outbound_cmds", which is the number of outbound commands in a File Transfer Protocol (FTP) session, has only one value, which is 0. Figure 3 shows the third problem, which is the class imbalance problem because the U2R attack has only 52 elements, and the R2L attack has 995 while normal is 67343 for Normal.

TABLE 1. NSL-KDD features.

Type	Features
Nominal	Protocol_type(2), Service(3), Flag(4)
Binary	Land(7), logged_in(12), root_shell(14), su_attempted(15), is_host_login(21), is_guest_login(22)
Numeric	Duration(1), src_bytes(5), dst_bytes(6), wrong_fragment(8), urgent(9), hot(10), num_failed_logins(11),num_compromised(13), num_root(16), num_file_creations(17), num_shells(18), num_access_files(19), num_outbound_cmds(20), count(23), srv_count(24), serror_rate(25), srv_serror_rate(26), rerror_rate(27), srv_rerror_rate(28), same_srv_rate(29) diff_srv_rate(30), srv_diff_host_rate(31), dst_host_count(32), dst_host_srv_count(33), dst_host_same_srv_rate(34), dst_host_diff_srv_rate(35), dst_host_same_src_port_rate(36), dst_host_srv_diff_host_rate(37), dst_host_serror_rate(38), dst_host_srv_serror_rate(39), dst_host_rerror_rate(40), dst_host_srv_rerror_rate(41)

B. DATA PREPROCESSING

The second block in Figure 1 is the preprocessing, which contains two steps; the first is feature preparing, and the second is the feature scaling.

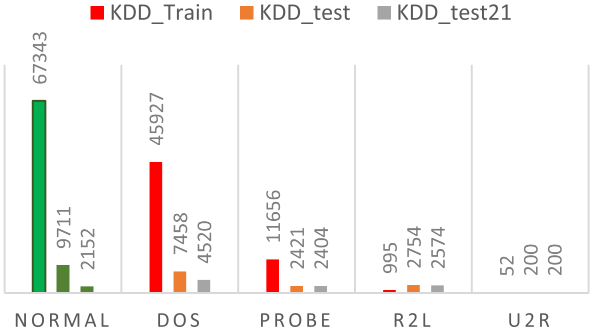


FIGURE 3. Dataset points indicate a class imbalance.

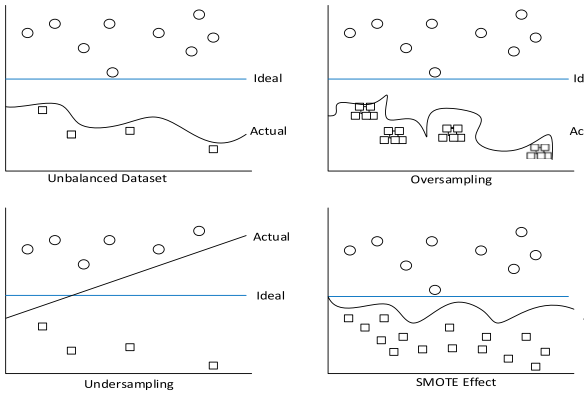


FIGURE 4. Oversampling problem.

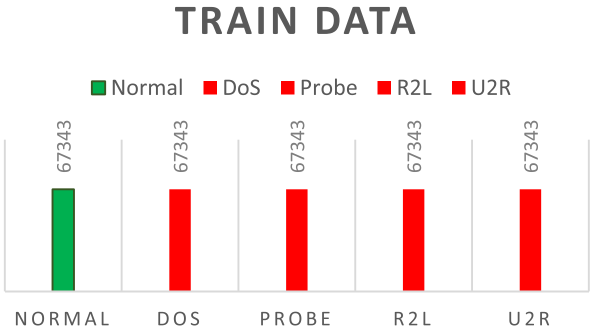


FIGURE 5. Data points after applying SMOTE technique.

1) FEATURE PREPARING

The first step is to change the values of "su_attempted" to be only (0,1) by converting the value 2 to 0. The "num_outbound_cmds" feature will be dropped as it does not affect the model. There are 37 numeric features and three nominal features in the NSL-KDD dataset. The input value of any model should be numeric. Nominal features, such as "protocol_type", service, and flag, must be converted into a numeric form. The feature "protocol_type" has three types of attributes, TCP, UDP, and ICMP. This feature will change to binary vectors [1 0 0], [0 1 0], and [0 0 1]. Furthermore, the service feature has 67 classes of attributes, and the flag feature has 11 types of attributes. So, the 40 features will become 118 features after conversion, which is less than 122 features like the model in [9].

2) FEATURES SCALING

If the features have significant variance and values, the model will be biased to these features. These features must be scaled. Three features have a notable deviation in their values, which are duration, source bytes (src_bytes), and destination bytes (dst_bytes). Min-Max normalization will be used to scale all features. Min-Max normalization is given by equation 1.

$$x_{i,j} = \frac{x_{i,j} - \text{Min}}{\text{Max} - \text{Min}} \tag{1}$$

where $x_{i,j}$ is the feature j value of sample i, Min is the lowest feature value in all samples, and Max is the highest feature value of all samples.

C. CLASS IMBALANCE

The third block, in Figure 1, is the proposed class imbalance solution for the NSL-KDD dataset.

The NSL-KDD dataset suffers from class imbalance distributions. Some researchers use oversampling, which is duplicating the minority class points, but this method has the disadvantage of overfitting on these points.

Others use undersampling, which is removing some points from the majority class. The problem with this methodology is that some removed points may be critical to represent the class. There is a Hybrid solution that duplicates minority class points and removes some majority class points. This method will enhance the model but will inherit the problems of the two procedures.

A new technique was introduced [28], called Synthetic Minority Over-Sampling Technique(SMOTE). This technique is a combination of oversampling and undersampling. Still, the oversampling is done by creating new points of the minority class rather than duplicating, which reduces the effect of overfitting, as shown in Figure 4.

The NSL-KDD data points of all categories (normal and attacks) are equal after applying the SMOTE, as shown in Figure 5.

D. SPARK MODEL

The last block in Figure 1 is the use of a Spark cluster to train the Deep Learning (DL) models.

The model training process has three main steps. The first is to input the training data point to the feedforward network to produce a predicted output, then use the predicted output against the actual output to compute the loss, and use it to optimize the weights of the feedforward network, then repeat the process.

1) FEEDFORWARD NETWORKS

DL has many network models. It has been proven that the CNN model has low detection accuracy [17], so in this work, we will test three different architectures; MLP, RNN, and LSTM. These architectures will be described in the following subsections.

a: MULTILAYER PERCEPTRON

MLP is a set of feedforward artificial neural network. An MLP is constructed by the input stage, the output stage, and the hidden stage[29]. The hidden stage may contain many layers or at least one. All nodes in the hidden stage are a neuron that sums all inputs with weights and then applies a nonlinear activation function as in equation (2).

$$a_k = Relu\left(\sum_{i=1}^{n} W_{xh}.x_i + b_k\right) \tag{2}$$

where a_k is the activation function, W_{xh} is the weight between the input and the hidden layer, b_k is the bias value of the current layer, and Relu is the activation function of the neuron. Relu has been used lately in DL training because it reduces the vanishing and error gradient. Moreover, Relu faster than other activation functions[24]. Relu can be evaluated using equation (3).

$$f(x) = \max(0, x) \tag{3}$$

The output stage will be used for all three feedforward techniques in the DLS-IDS model. The next equation explains the output stage operation.

$$y_k = f\left(\sum_{i=1}^{n} W_{hy}.a_{k-1} + b_k\right) \tag{4}$$

where y_k is the predicted output, W_{hy} is the weight between the hidden layer and the output layer, a_{k-1} is the output of the previous layer, and f is the activation function, which could be sigmoid or Softmax. The activation function for the binary classification is sigmoid, and multiclass classification is Softmax. Both functions are given by equations (5) and (6).

$$\text{sigmoid} = \frac{1}{1+e^{-x}} \tag{5}$$

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{M} e^{x_j}} \tag{6}$$

where i represents the sample, and j represents the class, and M is the number of classes.

b: RECURRENT NEURAL Network(RNN)

It is called recurrent because each node depends on the previous computation. RNN treats the input as a time series[30]. The following equations can evaluate the activation function and output.

$$a_t = \tanh(W_{hh}.h_{t-1} + W_{xh}.x_t) \tag{7}$$

$$y_t = W_{hy}.a_t \tag{8}$$

where W_{hh} are the weights of hidden of previous computation to present hidden layer, h_{t-1} is the output of the last calculation, x_t is the input at time t, and y_t is the output at time t. Figure 6 shows the RNN architecture.

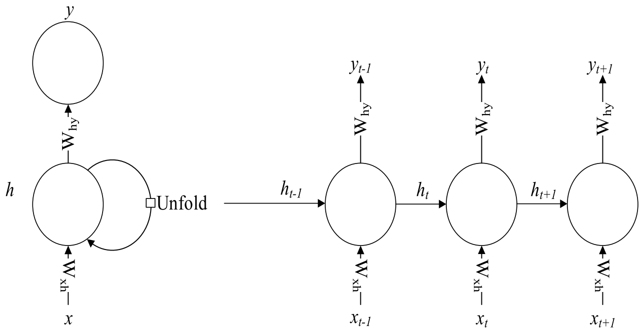


FIGURE 6. Recurrent Neural Network.

c: LONG SHORT TERM MEMORY

RNN model suffers from vanishing gradient descent problem, which leads to creating LSTM. LSTM architecture is composed of a cell (the memory part of the LSTM unit) and three gates. The three entrances are the input gate, the output gate, and the forget gate[31]. The forget gate function is to discard inessential details, while the input gate function is to modify the memory according to the input. Finally, the output gate function is to determine the output based on the input and memory gates.

$$f_t = \text{sigmoid}(W_f.[h_{t-1}, x_t] + b_f) \tag{9}$$

$$i_t = \text{sigmoid}(W_i.[h_{t-1}, x_t] + b_i) \tag{10}$$

$$\tilde{c}_t = \tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{11}$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{12}$$

$$o_t = \text{sigmoid}(W_O.[h_{t-1}, x_t] + b_O) \tag{13}$$

$$h_t = o_t * tanh(c_t) \tag{14}$$

where x_t is the input vector to the LSTM unit, f_t is the forget gate's activation vector, i_t is the input gate's vector, $\tilde{c}_t$ is the cell input activation vector, c_t is the cell state vector o_t is the output gate's activation vector, h_t is the hidden state vector, also known as output vector of the LSTM unit, W is the weight matrices, and b is the bias vector parameters.

The following subsection will discuss the overfitting problem.

d: OVERFITTING SOLUTION

The deep neural network tends to overfit the decision boundary on the training dataset. In this work, we use two methods to reduce the effect of overfitting; the first is the SMOTE technique, which is described before in the class imbalance section, and the second is network regularization.

One of the most effective techniques for neural network regularization is the dropout layer [32]. The dropout layer function is to generate a mask **s** that samples the output of the previous layer. The sampling has a Bernoulli distribution with a probability 'p':

$$s_k \sim \text{Bernoulli}(p), s_k \in \mathbf{s}. \tag{15}$$

s_k is the dropout probability in layer k. This mask will be applied to the activation function:

$$a_t = s_k \circ f(W * a_{t-1} + b_t), \tag{16}$$

where '∘' denotes the Hadamard product.

All the mentioned network elements are the feedforward propagation that will be used in the DLS-IDS model, and the following sections describe the Backpropagation in the DLS-IDS model.

2) LOSS COMPUTATION

After the feedforward, the predicted output is used to calculate the loss. Then an update algorithm will be applied to the weights to decrease the loss and eventually increase the accuracy. Categorical cross-entropy and binary cross-entropy are used in the DLS-IDS model to evaluate the loss in the case of multiclass and binary classification, respectively, as follows [33]:

$$L\left(y, \hat{y}\right) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * \log(\hat{y}_{ij})) \tag{17}$$

where $L\left(y, \hat{y}\right)$ is the loss function, y_{ij}, $\hat{y}_{ij}$ represents the actual and predicted output of sample i for class j, respectively. The binary cross-entropy is given by

$$L\left(y, \hat{y}\right) = -\frac{1}{N}\sum_{i=0}^{N}(y_i * \log\left(\hat{y}_i\right) + (1 - y_i) * \log(1 - \hat{y}_i)) \tag{18}$$

3) ADAM OPTIMIZERS

The next step after calculating the loss is using an optimizer to update the weights. The optimizer used in DLS-IDS is Adaptive moment estimation(Adam), which is an adaptive learning rate method [34]. Adam is a combination between the RMSprop and Momentum algorithms. Adam stores the past gradient descent m_t, and the past squared gradient descent v_t, which are an exponential moving average of the first and the second moment of the gradient, respectively. Adam algorithm has six computations which are as follows:

$$g_t = \nabla J(W_{t,i}) \tag{19}$$

where g_t is the gradient, J is the loss function, and ∇ is the gradient.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t \tag{20}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2 \tag{21}$$

where β_1, β_2 are decay terms for the first and second momentum. The next step is to calculate a bias-corrected first and second momentum estimates.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{22}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{23}$$

where $\hat{v}_t$, $\hat{m}_t$ are the corrected bias estimates. The last step is to update weight, which is given by equation 24.

$$w_t = w_{t-1} - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{24}$$

IV. EXPERIMENTAL DESIGN

The training of the models was implemented on Google Cloud Dataproc. Dataproc has Spark version 2.4.4 over a Hadoop version 2.9. The training dataset will be divided into the training dataset and the validation dataset. The validation dataset is essential to make sure that the model will perform well on the test dataset.

A. SPARK CLUSTER CONFIGURATION

This article presents three different Spark cluster configuration. The use of these configurations will illustrate the impact of Spark in the DLS-IDS model to reduce the training process time. The main advantage of using Spark is that the Spark cluster can be made by commodity hardware. Although this article uses the Google Cloud Dataproc, which gives the ability to choose powerful machines, commodity hardware configurations were chosen. The first configuration contains one master node with two workers, and each node has two processors with 7.5 GB memory. The second configuration consists of one master node and two worker nodes, where each node has four processors with 15 GB memory. The last configuration is one master and four workers where each node has two processors with 7.5 GB memory. All nodes are within the same rack. TABLE 2 summarizes the Spark cluster configurations.

TABLE 2. Spark Cluster Configuration Summary.

	No. of Drivers	No. of Workers	No.of processors/node	Memory size/node
Conf.1	1	2	2	7.5 GB
Conf .2	1	2	4	15 GB
Conf.3	1	4	2	7.5 GB

B. MODEL ARCHITECTURE SETTINGS

Nine different model settings on each Spark cluster configuration will be trained. The hyperparameters that will be adjusted are the number of hidden layers and the network type. As mentioned earlier in the feature preparing section, the 41 features have been converted to 118 features. So, the basic model arrangement has 118 nodes for the input layer, 80 for the hidden layer, and 5 for the output layer. The model settings differ in the neural network type and network architecture. This experiment implements the three neural network types mentioned earlier, i.e., MLP, RNN, and LSTM, with three different arrangements. The network architecture training runs for three values of the hidden layer; the first only one hidden layer; the second two hidden layers; the last is three hidden layers. TABLE 3 shows the different model arrangements that will be used for the three types mentioned earlier.

TABLE 3. Different Model Arrangement.

	One hidden Layer	Two hidden Layers	Three Hidden Layers
Layer 1	Input layer 118 node	Input layer 118 node	Input layer 118 node
	Dropout(0.01)	Dropout(0.01)	Dropout(0.01)
Layer 2	Hidden Layer 80 node	Hidden Layer 80 node	Hidden Layer 80 node
	Dropout(0.01)	Dropout(0.01)	Dropout(0.01)
Layer 3	Output 5 neurons	Hidden Layer 80 node	Hidden Layer 80 node
		Dropout(0.01)	Dropout(0.01)
Layer 4		Output 5 neurons	Hidden Layer 80 node
			Dropout(0.01)
Layer 5			Output 5 neurons

C. STATISTICAL MEASURES

After Spark completes the training for each model, it evaluates the models using statistical measures. The model uses the test dataset to compare the classification output with the actual label. The output yields the following statistical values; True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

These values are used in the model performance evaluation metrics, which are defined and calculated below [24],[35].

1) Accuracy: It is the ratio of the correctly classified packets (normal or attacks) to the total dataset. It can be calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{25}$$

2) Precision: It is the ratio of correctly classified attacks to the total number of identified attacks. It can be calculated as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{26}$$

3) Recall: It is the ratio of accurately classified attacks to the total number of attacks in the test dataset. It can be calculated as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{27}$$

4) F1-Score: It is the average of the precision and the Recall with a weight of 2. It can be calculated as:

$$\text{F1} - \text{Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{28}$$

5) False Positive Rate (FPR) is the number of normal connections that are recognized as an attack on the total number of normal connections.

$$\text{FPR} = \frac{FP}{TN + FP} \tag{29}$$

6) Receiver Operating Characteristics (ROC) curve is the relation between the True Positive Rate(TPR) on the y-axis, and the F(false) PR on the x-axis. Area Under the ROC Curve (AUC) is the area under the ROC curve.

$$\text{AUC} = \int_0^1 \frac{TP}{TP + FN} d \frac{FP}{TN + FP} \tag{30}$$

7) Sensitivity metric: it measures the detection accuracy of the attacks. It is the number of detected attacks on the total number of attacks in the dataset, which is the same as Recall eq. 27.

8) Specificity: it is the detection accuracy of the normal packets. It is the number of correctly classified normal packets on the total number of packets in the dataset.

$$\text{Specificity} = \frac{TN}{FP + TN} \tag{31}$$

9) Geometric Mean (G-Mean) metric is the measure of the balance between normal and attacks classification accuracy. A low G-Mean means poor performance. This measure is essential in the avoidance of overfitting the normal packets and underfitting the attack packets.

$$\text{G} - \text{Mean} = \sqrt{\text{Specificity} \times \text{Sensitivity}} \tag{32}$$

V. RESULTS

This section presents and discusses the results of the experiments. It is divided into two parts the DL algorithms on Apache Spark results and rigorous analysis for the selected model.

A. DEEP LEARNING ALGORITHM ON APACHE SPARK

The delay cost computation for the three configurations mentioned earlier is investigated, along with model settings accuracy. Each configuration trains nine different models for one hundred epochs. At last, some failure implementation scenarios are presented.

1) MODEL SETTINGS ACCURACY

TABLE 4 shows the accuracy of each model settings.

The results show that for MLP, the accuracy is 78.305%, 80.292%, and 79.462%. The authors in [24] used a five-layer MLP model, and the output accuracy was less than 78.6%.

The RNN accuracy is 81.88%, 81.371%, 80.897%. The authors in [9] used a two-layer RNN model and found that the best accuracy was 81.29%. The RNN determines the input using the previous state and the input, and that is why it has a better performance than MLP because there is a relation between the attacks and different fields. For example, the ping of death attack, which lay in the DoS category, has a protocol type of ICMP and lengthy payload. This reason drives the use of LSTM for intrusion detection. The LSTM accuracy is 82.440%,83.57%, and 81.535%. LSTM has a better performance than MLP and RNN.

Figure 8 illustrates the enhancement due to the use of the SMOTE technique. A test has been made for LSTM with two

TABLE 4. **Accuracy results for each model settings.**

Model Setting	Accuracy
MLP	
One Layer	78.305%
Two Layers	80.292%
Three layers	79.462%
RNN	
One Layer	81.889%
Two Layers	81.371%
Three layers	80.897%
LSTM	
One Layer	82.440%
Two Layers	83.574%
Three layers	81.535%

TABLE 5. **Training time for model settings on a different configuration.**

Model Setting	Conf.1	Conf.2	Conf.3
MLP			
One Layer	178.66	93.29	104.08
Two Layers	238.01	119.05	124.97
Three layers	309.38	149.96	158.56
RNN			
One Layer	241.27	126.57	135.49
Two Layers	123.93	58.87	64.92
Three layers	171.55	81.90	82.80
LSTM			
One Layer	146.37	81.08	123.21
Two Layers	826.20	429.59	521.46
Three layers	1138.77	617.90	667.14

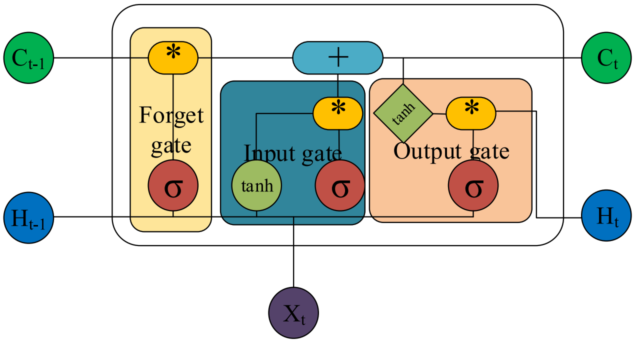
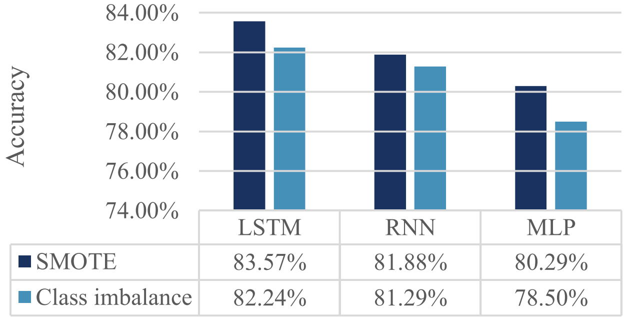


FIGURE 7. **LSTM architecture.**



	LSTM	RNN	MLP
■ SMOTE	83.57%	81.88%	80.29%
■ Class imbalance	82.24%	81.29%	78.50%

FIGURE 8. **DL accuracy for class imbalance and SMOTE.**

hidden layers without applying SMOTE to get the effect, and the accuracy result is 82.24%.

From the above, it can be concluded that The use of LSTM with two hidden layers is the best model.

2) DELAY COST FOR EACH CLUSTER CONFIGURATION

TABLE 5 shows the training time in seconds for different model settings on a three cluster configuration.

Each cluster treats the memory in all workers as memory containers. The third configuration and the second configuration have the same numbers of containers. The containers in the second configuration on two workers while in the third configuration, the containers on four workers, which add communication overhead, that explains the difference in the training time between the second and the third configuration. The nine models are trained in a sequence manner with the following order: RNN, LSTM, then MLP. After the training for RNN and LSTM has been finished, a conversion process is done from the RDD form to the pyspark, which is python for Apache Spark, data frame form to be suitable for the MLP input.

Since the first layer of RNN is the first to train, it takes more time than expected. This delay caused due to workers' initialization and setting the memory containers on each worker.

The authors in [9] trained their model in 11444 seconds, while the delay cost for the DLS-IDS model to train nine models is only 1758.21 seconds. Another platform that may be considered to perform the training process is Hadoop. However, it has been found that Spark is faster than Hadoop by almost 100 times. This advantage is because Hadoop executes the operations in the storage, and Spark executes the operations in the memory, as stated by Apache. This considerable difference demonstrates that the use of Spark is better than conventional training techniques. The mentioned reasons prove that Spark is the most suitable platform for the training process in the DLS-IDS model.

Spark can train the data in three forms RDD, dataset, or data frame. Spark has no built-in libraries for DL. Developers have made a library called Elephas, which enables the use of Spark in DL. The library supports the MLP to train the pyspark data frame, while RNN and LSTM failed to train on the pyspark data frame since they must have an input in a three-dimensional structure. RNN and LSTM use the RDD form as input to satisfy the input structure requirement.

One of the main features of the DLS-IDS is the use of Spark to speed the training process. The implementation code runs all model arrangements in a sequence to present all the results at once. Since Spark does the operations on the memory, the first configuration failed to train all models in one run. This failure happened due to the lack of memory, which was

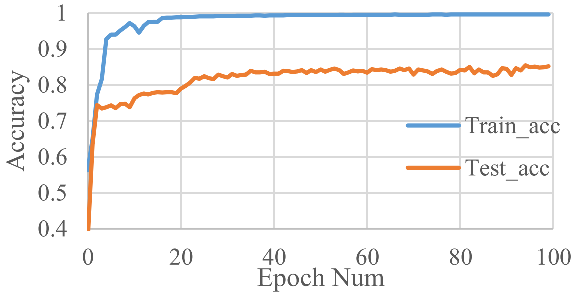


FIGURE 9. Binary classification accuracy for train and test dataset.

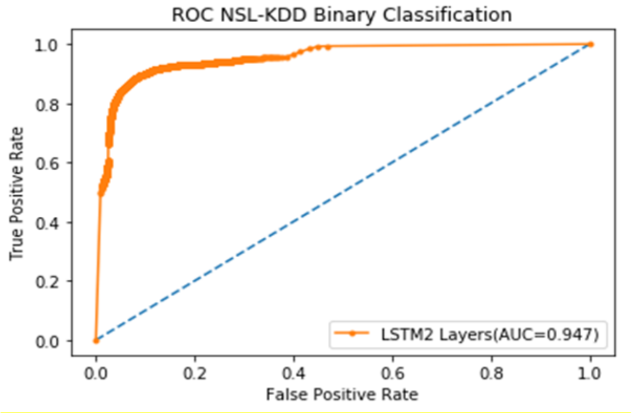


FIGURE 10. Receiver Operation Characteristic (ROC) for Binary classification.

7.5 GB only, while other configurations were able to train all the arrangements.

B. RIGOROUS ANALYSIS FOR THE SELECTED MODEL

We select the LSTM model with two hidden layers because it has the highest accuracy among all models, and the analysis will be for the binary test and the multiclass test. The model determines if the packet is an attack or normal only in the binary test. In the multiclass test, the model obtains the attack belonging to which class. Then, this model is applied to the KDDTest-21 dataset. Finally, a comparison between the resulted model of the DLS-IDS approach and the previously presented IDS attack detection accuracy is presented.

1) BINARY TEST ANALYSIS

Figure 10 illustrates the accuracy of binary classification on the train and test dataset for one hundred epoch. The training accuracy reached 99.61%, and the test accuracy reached 85.44%.

The model output has been evaluated against the KDDTest+ dataset. The output of the confusion matrix is TP = 9846, TN = 9417, FN = 2987, and FP = 294. The equations will determine The statistical evaluation of the model is shown in TABLE 6.

TABLE 6. Model evaluation metrics for binary classification.

Accuracy	Precision	Recall	F-Score	FPR
85.44%	97.1%	76.7%	85.7%	3.03%

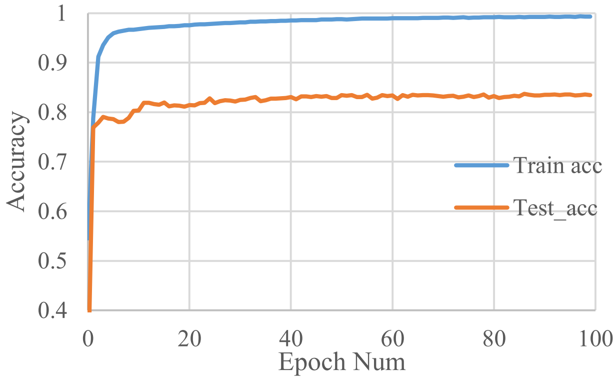


FIGURE 11. Multiclass classification accuracy for train and test dataset.

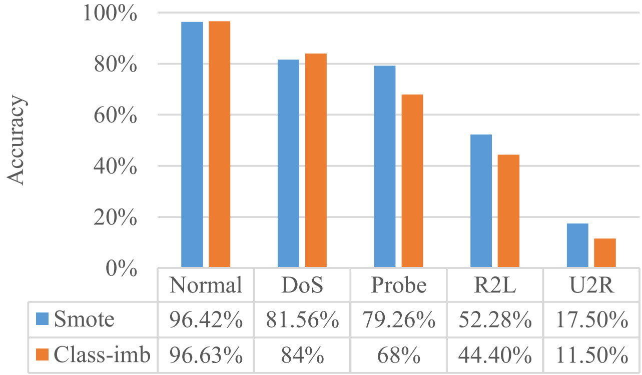


FIGURE 12. Multiclass accuracy comparison for SMOTE and Class imbalance.

TABLE 7. Confusion matrix for LSTM without SMOTE of KDDTest +.

	Normal	DoS	Probe	R2L	U2R
Normal	9384	63	259	4	1
DoS	1339	6265	32	0	0
Probe	598	170	1646	9	0
R2L	1347	0	3	1223	1
U2R	168	0	2	7	23

2) MULTICLASS TEST ANALYSIS

Figure 12 illustrates the accuracy of binary classification on the train and test dataset for one hundred epoch. The training accuracy reached 99.32%, and the test accuracy reached 83.57%.

The confusion matrix is built for both class imbalance and SMOTE applied shown in TABLE 7 and TABLE 8.

The confusion matrix is used to generate TP, TN, FN, and FP. The equations will determine the statistical evaluation of the model shown in TABLE 9.

The overall performance has increased with SMOTE. However, it is evident that the detection of the major classes, which are normal and DoS, accuracy is reduced because SMOTE has added new points to the minority classes, which

TABLE 8. Confusion matrix for LSTM with SMOTE of KDDTest+.

	Normal	DoS	Probe	R2L	U2R
Normal	9364	55	271	12	9
DoS	1258	6083	53	240	2
Probe	264	176	1919	63	1
R2L	975	1	106	1440	52
U2R	157	0	0	8	35

TABLE 9. Statistical evaluation for multiclass classification of the LSTM model of KDDTest+.

Class	Accuracy	Precision	Recall	F-score	FPR
LSTM without SMOTE					
Overall	82.24%	96.55%	72.62%	82.95%	3.37%
LSTM with SMOTE					
Overall	83.57%	96.46%	78.12%	86.32%	3.57%
Class	Specificity	Sensitivity	G-Mean		
LSTM without SMOTE					
Overall	96.63%	72.62%	71.38%		
LSTM with SMOTE					
Overall	96.43%	78.12%	76.71%		

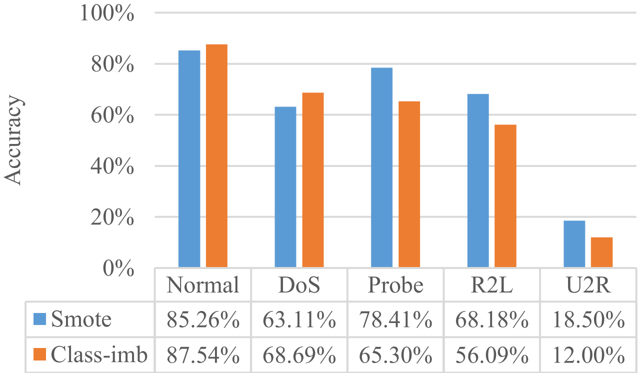


FIGURE 13. Multiclass accuracy comparison for SMOTE and Class imbalance.

affected the model bias toward the majority class. This reduction explains the increase in FPR. The difference of G-Mean after using the SMOTE shows the reduction of the overfitting of the model. A comparison of the accuracy of the multiclass is presented in Figure 13. The comparison illustrates the decrease in the accuracy in the dominant classes and the enhancement in the minor classes.

The model has been tested on the KDDTest-21 dataset, Figure 3 shows the dataset information, and the same analogy will be used. The confusion matrix is built for both class imbalance and SMOTE applied shown in TABLE 10 and TABLE 11.

The confusion matrix is used to generate TP, TN, FN, and FP. The equations will determine the statistical evaluation of the model shown in Table 12.

Figure 14 presents a chart of the accuracy of the multiclass. The graph illustrates the decrease in the accuracy in the dominant classes and the enhancement in the minor classes.

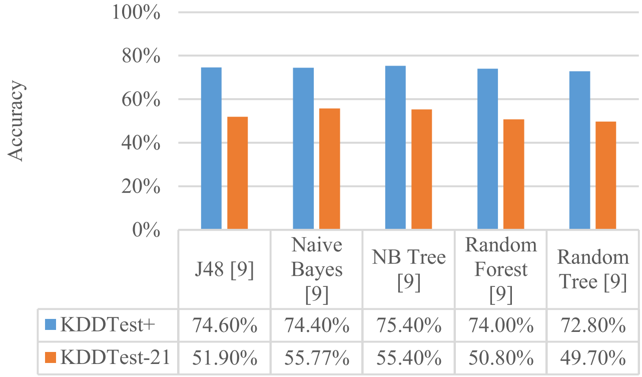


FIGURE 14. Performance of the traditional machine learning models.

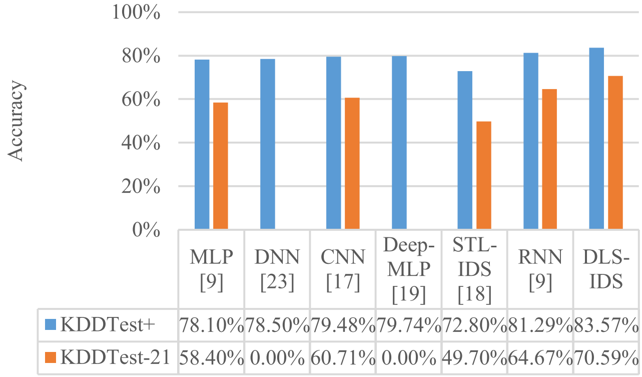


FIGURE 15. Performance of the proposed model and the other state of the art Deep learning models.

TABLE 10. Confusion matrix for LSTM without SMOTE of KDDTest-21.

	Normal	DoS	Probe	R2L	U2R
Normal	1884	59	199	4	6
DoS	1172	3105	54	189	0
Probe	643	174	1570	17	0
R2L	1120	0	6	1444	4
U2R	154	0	14	8	24

TABLE 11. Confusion matrix for LSTM with SMOTE of KDDTest-21.

	Normal	DoS	Probe	R2L	U2R
Normal	1835	55	224	24	14
DoS	1435	2853	37	193	2
Probe	299	182	1885	34	4
R2L	792	1	3	1755	23
U2R	148	0	4	11	37

3) COMPARISON BETWEEN DLS-IDS AND PREVIOUSLY PRESENTED IDS

A comparison is listed below between the state of the art Intrusion detection algorithms and the DLS-IDS model. Figure 15 shows the traditional machine learning algorithms accuracies against the KDDTest+ and KDDTest-21, while Figure 16 shows the deep learning algorithms accuracies

TABLE 12. Statistical evaluation for multiclass classification of the LSTM model of KDDTest-21.

Class	Accuracy	Precision	Recall	F-score	FPR
LSTM without SMOTE					
Overall	67.74%	95.82%	66.54%	78.53%	4.18%
LSTM with SMOTE					
Overall	70.59%	95.37%	70.95%	81.37%	4.63%
Class	Specificity	Sensitivity	G-Mean		
LSTM without SMOTE					
Overall	87.54%	66.54%	76.32%		
LSTM with SMOTE					
Overall	85.28%	70.95%	77.78%		

against the earlier mentioned datasets. In Figure 16, DNN and Deep-MLP did not test their models against the KDDTest-21. Figure 15 and Figure 16 show that the DLS-IDS model of this article enhances the overall attack detection accuracy.

VI. CONCLUSION

This article presented a new intrusion detection system based on deep learning. This system is called Deep Learning Spark Intrusion Detection System or DLS-IDS for short. The DLS-IDS model has four main building blocks, and we use the NSL-KDD dataset for training and testing purposes. The NSL-KDD dataset has a class imbalance problem. Therefore, the four system blocks are to choose and explore, preprocess, class imbalance solution, and the last block is training over Apache Spark. This DLS-IDS proved that the use of Spark is better than a regular implementation for DL. The Spark cluster enables model training with different hyperparameters, such as the model elements type and the number of hidden layers. Since Spark uses memory to execute its operations, then memory size must be taken into consideration of the design process of new models to avoid the system halt. When the Spark cluster contains many workers, there will be a communication overhead delay, but this delay is less than the overall computation delay. When dealing with a dataset that contains class imbalance, it is better to use Synthetic Minority Over-Sampling Technique (SMOTE) as a preprocessing step to enhance the detection accuracy of the model and reduce the overfitting effect of DL. The DLS-IDS found that the use of LSTM with SMOTE improves the detection accuracy to reach 83.57%. In future work, we consider the use of more datasets to cover more types of attacks hence train the model on these new attacks. Also, the use of the Kafka Hadoop tool to test the proposed model in real-time configuration would be considered in the future.

REFERENCES

[1] *2018 Internet Security Threat Report*, Symantec Corp., Tempe, AZ, USA, 2018, vol. 23, pp. 1–89.

[2] A. A. Ghorbani, W. Lu, and M. Tavallaee, *Network Intrusion Detection and Prevention Concepts and Techniques*. vol. 47. Springer, 2010, pp. 27–54.

[3] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson, "Shallow and deep networks intrusion detection system: A taxonomy and survey," *arXiv:1701.02145*, pp. 1–43, 2017. [Online]. Available: https://arxiv.org/abs/1701.02145

[4] M. A. Alsheikh, D. Niyato, S. Lin, H.-P. Tan, and Z. Han, "Mobile big data analytics using deep learning and apache spark," *IEEE Netw.*, vol. 30, no. 3, pp. 22–29, May 2016.

[5] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Comput.*, vol. 22, pp. 949–961, Aug. 2017.

[6] W. Li, P. Yi, Y. Wu, L. Pan, and J. Li, "A new intrusion detection system based on KNN classification algorithm in wireless sensor network," *J. Electr. Comput. Eng.*, vol. 2014, no. 1, pp. 1–8, 2014.

[7] N. Farnaaz and M. A. Jabbar, "Random forest modeling for network intrusion detection system," *Procedia Comput. Sci.*, vol. 89, pp. 213–217, Aug. 2016.

[8] S. A. Mulay, P. R. Devale, and G. V. Garje, "Intrusion detection system using support vector machine and decision tree," *Int. J. Comput. Appl.*, vol. 3, no. 3, pp. 40–43, Jun. 2010.

[9] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[10] N. Paulauskas and J. Auskalnis, "Analysis of data pre-processing influence on intrusion detection using NSL-KDD dataset," in *Proc. Open Conf. Electr., Electron. Inf. Sci. (eStream)*, Apr. 2017, pp. 1–5.

[11] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring: A survey," vol. 14, no. 8, pp. 1–14, 2016.

[12] Q. Niyaz, W. Sun, A. Y. Javaid, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. BICT*, 2016, pp. 21–26.

[13] K. Alrawashdeh and C. Purdy, "Toward an online anomaly intrusion detection system based on deep learning," in *Proc. 15th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2016, pp. 195–200.

[14] S. Sapre, P. Ahmadi, and K. Islam, "A robust comparison of the KDD-Cup99 and NSL-KDD IoT network intrusion detection datasets through various machine learning algorithms," 2019, *arXiv:1912.13204*. [Online]. Available: https://arxiv.org/abs/1912.13204

[15] J. Kim, N. Shin, S. Y. Jo, and S. Hyun Kim, "Method of intrusion detection using deep neural network," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Feb. 2017, pp. 313–316.

[16] N. Ding, Y. Liu, Y. Fan, and D. Jie, *Network Attack Detection Method Based on Convolutional Neural Network*, vol. 593. Singapore: Springer, 2020.

[17] K. Wu, Z. Chen, and W. Li, "A novel intrusion detection model for a massive network using convolutional neural networks," *IEEE Access*, vol. 6, pp. 50850–50859, 2018.

[18] M. Al-Qatf, Y. Lasheng, M. Al-Habib, and K. Al-Sabahi, "Deep learning approach combining sparse autoencoder with SVM for network intrusion detection," *IEEE Access*, vol. 6, pp. 52843–52856, 2018.

[19] C. Zhang, F. Ruan, L. Yin, X. Chen, L. Zhai, and F. Liu, "A deep learning approach for network intrusion detection based on NSL-KDD dataset," in *Proc. IEEE 13th Int. Conf. Anti-Counterfeiting, Secur., Identificat. (ASID)*, Oct. 2019, pp. 41–45.

[20] J. Kim, J. Kim, H. L. Thi Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *Proc. Int. Conf. Platform Technol. Service (PlatCon)*, Feb. 2016, pp. 1–5.

[21] H. Hou, Y. Xu, M. Chen, Z. Liu, W. Guo, M. Gao, Y. Xin, and L. Cui, "Hierarchical long short-term memory network for cyberattack detection," *IEEE Access*, vol. 8, pp. 90907–90913, 2020.

[22] M. Belouch, S. El Hadaj, and M. Idhammad, "Performance evaluation of intrusion detection based on machine learning using apache spark," *Procedia Comput. Sci.*, vol. 127, pp. 1–6, Jan. 2018.

[23] P. Dahiya and D. K. Srivastava, "Network intrusion detection in big dataset using spark," *Procedia Comput. Sci.*, vol. 132, pp. 253–262, Jan. 2018.

[24] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41525–41550, 2019.

[25] W. Lu, "A detailed analysis of the KDD CUP 99 data set NRC publications archive (NPArC)," in *Proc. IEEE Conf. Comput. Intell. Secur. Defense Appl.*, Jul. 2009, pp. 53–58.

[26] S. Revathi and A. Malathi, "A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection," *Int. J. Eng. Res. Technol.*, vol. 2, no. 12, pp. 1848–1853, 2013.

[27] R. Bala and R. Nagpal, "A review on KDD CUP99 and NSL-KDD dataset," *Int. J. Adv. Res. Comput. Sci.*, vol. 10, no. 2, pp. 64–67, Apr. 2019.

[28] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 63–77, Jan. 2006.

[29] A. A. Heidari, H. Faris, S. Mirjalili, I. Aljarah, and M. Mafarja, "Ant lion optimizer: Theory, literature review, and application in multi-layer perceptron neural networks," in *Nature-Inspired Optimizers* (Studies in Computational Intelligence), vol. 811, S. Mirjalili *et al.*, Eds. Cham, Switzerland: Springer, 2020, doi: 10.1007/978-3-030-12127-3_3.

[30] T. A. Tang, D. McLernon, L. Mhamdi, S. A. R. Zaidi, and M. Ghogho, "Intrusion detection in SDN-based networks: Deep recurrent neural network approach," in *Deep Learning Applications for Cyber Security* (Advanced Sciences and Technologies for Security Applications), M. Alazab and M. Tang, Eds. Cham, Switzerland: Springer, 2019, pp. 175–195, doi: 10.1007/978-3-030-13057-2_8.

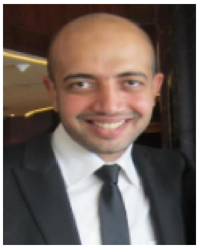
[31] A. Chu, Y. Lai, and J. Liu, "Industrial control intrusion detection approach based on multiclassification GoogLeNet-LSTM model," *Secur. Commun. Netw.*, vol. 2019, no. 2, pp. 1–11, Dec. 2019.

[32] S. H. Khan, M. Hayat, and F. Porikli, "Regularization of deep neural networks with spectral dropout," *Neural Netw.*, vol. 110, pp. 82–90, Feb. 2019.

[33] H. Alaiz-Moreton, J. Aveleira-Mata, J. Ondicol-Garcia, A. L. Muñoz-Castañeda, I. García, and C. Benavides, "Multiclass classification procedure for detecting attacks on MQTT-IoT protocol," *Complexity*, vol. 2019, pp. 1–11, Apr. 2019.

[34] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15.

[35] J. C. Bansal, N. Delhi, K. Deep, and A. K. Nagar, *Evolutionary Machine Learning Techniques*. Singapore: Springer, 2020.



MOHAMED HAGGAG (Member, IEEE) received the B.S. degree in electronics and communications engineering from the Misr University for Science and Technology, Giza, Egypt, in 2006, and the M.Sc. degree in electronics and communications engineering from the Arab Academy for Science and Technology and Maritime Transport Engineering, Cairo, Egypt, in 2012. He is currently pursuing the Ph.D. degree with Cairo University, Giza. His research interest includes bigdata security.

MOHSEN M. TANTAWY received the M.Sc. degree from Cairo University, Egypt, in 1998, and the Ph.D. degree from Ain Shams University, Egypt, in 2003. He is currently a Professor with the Network Planning Department, National Telecommunication Institute (NTI), an affiliate of the Ministry of Communication and Information Technology.

MAGDY M. S. EL-SOUDANI (Senior Member, IEEE) is currently a Professor with the Communications, Electronics, and Electrical Communications Department, Faculty of Engineering, Cairo University. He has published more than 70 articles in specialized technical journals or presented at international conferences. His main research interests include channel and network coding, data, and network security.