Received: 1 November 2018 | Revised: 26 March 2019 | Accepted: 26 April 2019

DOI: 10.1111/coin.12220

ORIGINAL ARTICLE

WILEY Computational Intelligence

Empirical study on multiclass classification-based network intrusion detection

Wisam Elmasry[1] | Akhan Akbulut[2] | Abdul Halim Zaim[1]

[1]Department of Computer Engineering, Istanbul Commerce University, Istanbul, Turkey

[2]Department of Computer Engineering, Istanbul Kültür University, Istanbul, Turkey

Correspondence
Wisam Elmasry, Department of Computer Engineering, Istanbul Commerce University, 34840 Istanbul, Turkey.
Email: wisam.elmasry@istanbulticaret.edu.tr

Abstract

Early and effective network intrusion detection is deemed to be a critical basis for cybersecurity domain. In the past decade, although a significant amount of work has focused on network intrusion detection, it is still a challenge to establish an intrusion detection system with a high detection rate and a relatively low false alarm rate. In this paper, we have performed a comprehensive empirical study on network intrusion detection as a multiclass classification task, not just to detect a suspicious connection but also to assign the correct type as well. To surpass the previous studies, we have utilized four deep learning models, namely, deep neural networks, long short-term memory recurrent neural networks, gated recurrent unit recurrent neural networks, and deep belief networks. Our approach relies on the pretraining of the models by exploiting a particle swarm optimization–based algorithm for their hyperparameters selection. In order to investigate the performance differences, we also included two well-known shallow learning methods, namely, decision forest and decision jungle. Furthermore, we used in our experiments four datasets, which are dedicated to intrusion detection systems to explore various environments. These datasets are KDD CUP 99, NSL-KDD, CIDDS, and CICIDS2017. Moreover, 22 evaluation metrics are used to assess the model's performance in each of the datasets. Finally, intensive quantitative, Friedman test, and ranking methods analyses of our results are provided at the end of this paper. The results show

a significant improvement in the detection of network attacks with our recommended approach.

KEYWORDS

cyber security, deep learning, network intrusion detection, particle swarm optimization

1 | INTRODUCTION

Nowadays, we are living in the era of "Big Data," the increasing of the growth of data's three Vs-Volume, Velocity, and Variety-. Volume is the size of the generated data and can be measured in gigabytes, terabytes, or even petabytes. On the other hand, velocity refers to the rate at which data is generated per a specific time unit, and variety is the types of that data. With the aim of pursuing these numerous changes on data storing, processing, acquisition, and transition, a revolution on information systems is demanded and put forward by the IT industry. Equally important is that the challenges and threats are mounted up in perspective of security.[1]

In cybersecurity, an intruder is that some anonymous party seeks to impersonate a genuine client by exploiting system vulnerabilities. Primarily, intrusion in computer networking can be derived from launching attacks. A computer network attack takes place when a cyber attacker initiates suspicious activities either to gain an unauthorized access to a legitimate user's information or to make system down. There are various well-known computer network attacks such as denial of service (DoS), probing (Probe), Remote to Local (R2L), user to root (U2R), and brute force. Bare in mind that these attacks can be executed using any of network protocols and services such as HTTP, FTP, ICMP, TCP, UDP, and SMTP. As the consequence of that, these attacks are becoming more prevalent and considered to be the most serious threat of network security. Notably, the most effective way to prevent such attacks is using a network-based intrusion detection system (NIDS), which can monitor the physical and software infrastructure of the system and search for any abnormal behavior.[2]

In network security literature, there are two common approaches to NIDSs, ie, the signature-based detection and the anomaly-based detection. Signature-based detection or also called misuse detection is worth to use when attack signature is already known. On the other hand, anomaly-based detection can be used for either known or unknown attacks. The main idea behind using NIDS is traffic identification within three consecutive steps. First, NIDS captures the traffic that passed through the network. Next, it processes the captured traffic in such a way that it generates traffic records. Each of these records consists of several useful features that are extracted directly from the captured traffic. Then, by using a previously trained data mining algorithm, it classifies the tested traffic record to either normal or intrusive activity. Practically, when a traffic record is classified as an abnormal, NIDS raises the red flag and an alarm that the system is underwent to a possible attack. There are many network intrusion detection algorithms used, but among of them, machine learning models are the most commonly used approaches due to their ability to learn from data and then distinguish between normal and malicious users.[3]

Despite the popularity of using traditional machine learning methods for classification tasks, these have many deficiencies that need to be addressed, such as the perspective of full features representation, the complexity of the problem, and the limitation to static classification applications.[4] In 2006, a new concept of representation learning called deep learning has put forward. Deep learning is considered as a class of machine learning techniques that have, in hierarchical

architectures, many layers of information processing stages for pattern recognition or classification. Rather than it overcomes the former deficiencies of traditional machine learning methods, it achieves recently great success in many research fields. Because of deep learning success and stability, it has been actively used in a wide range of applications nowadays such like computer vision, natural language processing, and cybersecurity systems.[5]

The aims and contributions of this research are two-fold, as follows.

- We performed an empirical study on network intrusion detection as a multiclass classification task. Moreover, we utilized four deep learning models with pretraining phase by leveraging a particle swarm optimization (PSO)–based algorithm to select deep learning models' hyperparameters automatically. We validated the effectiveness of the used models by using four common IDS datasets. Based on the findings, this approach enhanced deep learning models' detection rate (DR) by 1% to 5% as well as decreasing false alarm rate by 1% to 3% from the corresponding values of deep learning models without pretraining phase.
- We presented a comprehensive analysis of the experimental results by using 22 evaluation metrics, Friedman statistical test, and two ranking methods. Some of the used metrics are widely used in the previous studies that focused on network intrusion detection subject, so it would be easy for readers to compare results. The others are common metrics when doing a multiclass classification task. Hence, we used all these metrics to give further analysis and complete view of deep learning models' performance when using our approach.

The rest of this paper is organized as follows. Section 2 describes the NIDS datasets and their characteristics in detail. The literature review is introduced in Section 3. Then, Section 4 shows how our experiments are established and which models are used. In Section 5, we present briefly a list of the used evaluation metrics and their formulas. The performance of the used models is analyzed in Section 6. Finally, we draw conclusions in Section 7.

2 | NIDS DATASETS

It is beyond all doubt that the key component of building intelligent NIDS is the existence and availability of an effective dataset. A dataset with enough amount of quality data, which mimics the real time, is required to train and test the NIDS built-in model.[6] Hence, a quite number of NIDS datasets have been created since 1998. Gharib et al[7] reported 11 NIDS datasets starting from 1998 to 2016, namely, DARPA,[8] KDD CUP 99,[9] DEFCON,[10] CAIDA,[11] LBNL,[12] CDX,[13] Kyoto,[14] Twente,[15] UMASS,[16] ISCX2012,[17] and ADFA.[18] Furthermore, they proposed an evaluation framework for NIDS datasets that reflects the characteristics of a valid dataset and consists of 11 characteristics, namely, complete network configuration, complete traffic, labeled dataset, complete interaction, complete capture, available protocols, attack diversity, anonymity, heterogeneity, feature set, and metadata.[7]

In this paper, we selected four different NIDS datasets to assess the performance of the used models. Despite KDD CUP 99 and NSL-KDD datasets being relatively old, they are still considered as well-known benchmarks and are extensively used in network intrusion detection field. For the sake of diversity and exploring up-to-date NIDS datasets, CIDDS and CICIDS2017 datasets are involved in this study. The following four sections describe the used NIDS datasets. Table 1 summarizes the main characteristics of the used datasets in terms of format of data, number of protocols, attacks, features, etc. It is worthy to say that the number of samples in the training and test sets of the KDD CUP 99 in Table 1 is only 10% of the KDD CUP 99 dataset. Moreover, in the

TABLE 1 Datasets and their main characteristics

Characteristics		Datasets			
		KDD CUP 99	NSL-KDD	CIDDS	CICIDS2017
Year		1999	2009	2017	2017
Raw format		tcpdump	tcpdump	NetFlow	pcap
No. of protocols		6	6	5	5
No. of attacks		38	38	92	20
Attack categories		4	4	4	6
No. of features		41	41	12	80
No. of labeled classes		5	5	5	7
Distribution of the training set	Normal	97 278	67 343	160 000	21 428
	Attacks	396 743	58 630	640 000	128 568
	Total	494 021	125 973	800 000	149 996
Distribution of the test set	Normal	60 593	9711	160 000	21 428
	Attacks	231 707	12 833	640 000	128 568
	Total	292 300	22 544	800 000	149 996

fourth and fifth columns of Table 1, the number of samples in the training and test sets is only 5% and 10% of the CIDDS and CICIDS2017 datasets, respectively. Finally, the details of how we selected the subsets are presented in Sections 2.3 and 2.4 for CIDDS and CICIDS2017 datasets, respectively.

2.1 | KDD cup 99

It is first started when MIT Lincoln Laboratory had constructed DARPA dataset to be used in network security analysis in 1998. In short time, this dataset was used in various research works and one of them stated that DARPA dataset is insufficient in network intrusion detection due to its oldness and limitation to represent real-world network traffic.[19] In order to overcome these serious limitations, an updated version of DARPA is created in 1999 by processing tcpdump portion, namely, KDD CUP 1999 or simply KDD CUP 99. This dataset has been widely used in the field of network intrusion detection and is publicly available on the Internet.[20] The full dataset consists of about five million connection records in the training set and around two million connection records in the test set. Practically, only 10% of KDD CUP 99 dataset is using in both training and test. As a result, there are 494 021 samples in training data as well as 292 300 samples in test data.

Every sample is labeled to either normal or an attack record. There are a total of 38 types of attacks are included in the 10% of the dataset. In contrast to the test set where samples from all types of attacks appeared, only samples from 24 types of attacks are appeared in the training set. This is to evaluate the effectiveness of the tested model to detect novel attacks that are not appearing in the training set. Moreover, to improve DR, similar attacks are combined together into a single category which leads to form four major attacks categories, namely, DoS, Probe, R2L, and U2R. According to that, there are five classes available in the KDD CUP 99 dataset, which are Normal, DoS, Probe, R2L, and U2R. The distribution of samples into each class in the training set is as follows: Normal (97 278), DoS (391 458), Probe (4107), R2L (1126), and U2R (52). On the other hand, the distribution of samples in the test set is as follows: Normal (60 593), DoS (223 298), Probe (2377), R2L (5993), and U2R (39). Notably, the distribution of the 10% of KDD CUP 99 dataset is an imbalance in both training and test sets. Wherefore, synthetic minority oversampling technique (SMOTE), which is a very popular technique, has been applied to deal with this issue.[21]

The 10% of KDD CUP 99 has six different network protocols and services such as SMTP, HTTP, FTP, Telnet, ICMP, and SNMP. Finally, it contains 41 features that can be divided into

three types, namely, basic features, traffic-based features, and content-based features. The data types of these features are nominal (3), binary (4), and continuous (34). Nevertheless, the 10% of KDD CUP 99 has inherent drawbacks in which the most samples in both training and test sets are redundant.[22]

2.2 | NSL-KDD

The inherent problems of the KDD CUP 99 necessitate the need for a new NIDS dataset. Thus, Tavallaee et al proposed in 2009 an improved and reduced version of the 10% of KDD CUP 99 dataset, namely, NSL-KDD.[22] They solved the drawbacks of the 10% of KDD CUP 99 in two ways. First, they eliminated all the redundant records from both training and test sets. Second, they partitioned the records into various difficulty level, then they selected records from each difficulty level inversely proportional to the percentage of records in the original 10% of KDD CUP 99 dataset. As a result of that, the NSL-KDD has a reasonable number of records in both training and test sets and makes it affordable to run the experiments on the complete set. In spite of being quite out-of-date to represent the real networks, it still considered as a benchmark and extensively used in network intrusion detection studies. In addition to that, NSL-KDD dataset is publicly available on the Internet.[23]

The NSL-KDD dataset has a total of 125 973 connection samples in the training set, whereas it has 22 544 connection samples in its test set. The distribution of samples into each class in its training set is as follows: Normal (67 343), DoS (45 927), Probe (11 656), R2L (995), and U2R (52). On the other hand, the distribution of samples in the test set is as follows: Normal (9711), DoS (7458), Probe (2421), R2L (2887), and U2R (67).

2.3 | CIDDS

Coburg intrusion detection dataset (CIDDS) is an up-to-date and publicly available NIDS dataset, which contains a realistic data that can be used for network intrusion detection mission.[24] There are two versions of this dataset that are created in 2017, namely, CIDDS-001[25] and CIDDS-002.[26] In order to create CIDDS dataset, a small business environment was emulated using OpenStack. This networking environment has included several clients and typical servers. They emulated the normal behavior of clients by using Python scripts. In the CIDDS-001, they executed a total of 92 various attacks from inside and outside the networking environment. These executed attacks can be grouped into four major attack categories, namely, DoS, Brute Force, Port Scan, and Ping Scan. Meanwhile, in the CIDDS-002, they executed a total of 43 port scan attacks only from inside of the networking environment. They captured NetFlow data over a period of four weeks for the CIDDS-001 and two weeks for the CIDDS-002. The CIDDS-001 contains about 32 million traffic records using five different network protocols; meanwhile, the CIDDS-002 contains around 16 million flows. Finally, the CIDDS dataset contains 12 features with anonymized public IP addresses as well as a class label to identify the particular traffic record to one of five possible classes.

It was reported that, "*We captured approximately 32 million flows over a period of four weeks and categorized them into five classes.*"[25] Unlike KDD CUP 99 and NSL-KDD datasets in which there are specified number of samples for both training and testing, CIDDS-001 dataset is very huge dataset that has 32 million samples in different files. In addition to that, in CIDDS dataset,

there are no specified training or test sets to be used in the experiments. According to that, we selected 10 subsets (5% of CIDDS) from CIDDS-001 dataset for training and testing to reduce the training and testing time reasonably because, when using the full size of CIDDS-001 dataset, the training and test times would be 10 to 14 hours, which is too long. The 10 subsets are selected randomly by using the sampling without replacement technique to ensure that, once an object is selected, it removed from the population. Moreover, the 10 subsets are equivalent in size and each of them contains 160 000 samples for training and testing. For the sake of ensuring the diversity of traffic records and avoiding overfitting, we have implemented a balanced training and test sets for each subset. Regarding the training set for each subset, it has 80 000 samples, which are evenly distributed as follows: Normal (16 000), DoS (16 000), brute force (16 000), port scan (16 000), and ping scan (16 000). Then, the same process is done again when creating the test set, but with a total of 80 000 samples, which do not exist in the training data. The samples distribution in the test set is as same as the samples distribution in the training set. Furthermore, some types of attacks are included only in the test set rather than in the training set to examine the ability of the used models to classify them correctly. According to this procedure, we believe that both training and test data are balanced and the used models will not bias to any class during the training phase.

2.4 | CICIDS2017

Canadian Institute for Cybersecurity Intrusion Detection System (CICIDS2017) is a modern anomaly-based NIDS dataset proposed in 2017 and publicly available on the Internet upon a request from its owners.[27] The purpose behind the creation of this dataset is to release a reliable and up-to-date dataset to help researchers to evaluate their models properly. Moreover, it is reported that this dataset overcomes all shortcomings of other existed NIDS datasets.[28] The environment infrastructure consists of two separate networks, which are attack-network and victim-network. The victim-network is used to provide the benign behavior by using B-Profile system.[29] On the other hand, the attack-network is utilized for executing the attack flows. Both of them are equipped with the necessary network devices and PCs running different operating systems. Moreover, they used CICFlowMeter to analysis the captured pcap data over five working days. The traffic records in this dataset are based on HTTP, HTTPS, FTP, SSH, and email protocols. In addition to that, the attack flows consist of a total of 20 attacks and are split into six major categories, namely, Brute Force, Heartbleed, Botnet, DoS, Web attack, and Infiltration. Finally, the CICIDS2017 dataset contains 80 features as well as a class label to identify the particular traffic record to one of seven possible classes.

CICIDS2017 dataset has approximately three million network flows.[27] Because of this huge amount of traffic records and for the same reasons explained earlier in the second paragraph in Section 2.3, we selected 10 subsets (10% of CICIDS2017) from the original CICIDS2017 dataset for training and testing. Moreover, we followed the same procedure that is mentioned earlier in Section 2.3 when selecting the 10 subsets randomly. Practically, each subset contains around 30 thousand samples, which are divided equally to training and test sets. Regarding the training set for each subset, it has approximately 15 thousand samples, which are evenly distributed as follows: Normal (2142), Brute Force (2142), Heartbleed (2142), Botnet (2142), DoS (2142), Web attack (2142), and Infiltration (2142). On the other hand, there are a total of different 15 000 samples in the test set, which are evenly distributed as same as in the training set. Table 2 shows how the used datasets comply with the 11 characteristics of the evaluation framework, which is proposed by Gharib et al.[7]

TABLE 2 Comparison between the used datasets based on the evaluation framework

Characteristics	KDD CUP 99	NSL-KDD	CIDDS	CICIDS2017
Complete Network Configuration	√	√	√	√
Complete Traffic	X	X	√	√
Labeled Dataset	√	√	√	√
Complete Interaction	√	√	√	√
Complete Capture	√	√	√	√
Available Protocols	√	√	√	√
Attack Diversity	√	√	√	√
Anonymity	X	X	X	√
Heterogeneity	X	X	√	√
Feature Set	√	√	√	√
Metadata	√	√	√	√

3 | LITERATURE REVIEW

In the last decade, dozens of research works focused on solving network intrusion detection problem. Some of them used traditional (shallow) machine learning models, and the other adopted deep learning models recently. To start with, there are many studies that have addressed techniques and challenges of using traditional machine learning algorithms in intrusion detection systems.[30-32] Natessan et al proposed a three-stage filter using enhanced adaptive boosting (AdaBoost) along with decision tree and naive Bayes classifiers to detect network attacks on KDD CUP 99 dataset.[33] They acquired DR with 85.78% and false alarm rate (FAR) with 1.97%. Dhanabal et al introduced a study to analyze the effectiveness of various machine learning algorithms in WEKA for network intrusion detection on six features of NSL-KDD dataset.[34] Similar to the former study, Meena et al also introduced a review paper on various classification algorithms in WEKA to detect anomalies on both KDD CUP 99 and NSL-KDD datasets.[35] Ant tree miner classifier, which uses decision tree and ant colony optimization, is used for network intrusion using NSL-KDD dataset (DR = 57.05%).[36] Depren et al proposed an intelligent NIDS for both anomaly and misuse detection on 10% of KDD CUP 99 dataset (DR = 99.9%, FAR = 1.25%).[37] A hybrid intelligent approach that has two phases, the first filters the tainting data and the second detects the intrusions on the NSL-KDD dataset, is introduced and gained good results.[38] Zhou[39] proposed an efficient distance computing method for network intrusion detection on the KDD CUP 99 dataset (DR = 99.54%, FAR = 1.49%).

Alternatively, there are many review articles point out the success of using deep learning techniques in solving network intrusion detection problem.[40,41] One of the common deep learning models is an AutoEncoder (AE). Aminanto et al introduced an artificial neural network (ANN) for feature selection using a single hidden layer stack AE as a classifier with two encoder layers, which are applied on the KDD CUP 99 dataset (DR = 99.9%).[42] Niyaz et al proposed a self-taught learning model in two stages, the first is sparse AE for unsupervised feature learning and the second is softmax regression classifier trained on the derived training data.[43] Their model is applied on the NSL-KDD dataset, and they got accuracy greater than 98%. Yu et al introduced a stacked denoising autoencoder model for session-based network intrusion on the ISCX2012 dataset (DR = 99.39% for binary classification).[44] Shone et al proposed a novel stacked nonsymmetric deep autoencoder

classifier for network intrusion detection on both KDD CUP 99 (DR = 97.85%, FAR = 2.15%) and NSL-KDD (DR = 85.42%, FAR = 14.58%) datasets.[45]

Furthermore, deep neural network (DNN) is a deep structural ANN model and widely used in the field of network intrusion detection. A comparative study of using DNN in network intrusion detection on the KDD CUP 99 dataset is introduced and obtained accuracy equals 99.9%.[46] Kim et al presented a method of detecting network attacks using a DNN model on refined data of the KDD CUP 99 dataset (DR = 99%, FAR = 0.08%).[47] In the study of Potluri and Diedrich,[48] an accelerated DNN model is used along with AEs and softmax layer to perform the fine tuning with supervised learning. Their accelerated DNN model is evaluated using NSL-KDD dataset with 97.5% that is DR and 3.5% that is FAR.

Moreover, recurrent neural network (RNN) is another recent deep architecture model that is widely used in the detection of abnormalities of network traffics. Kim et al introduced a long short-term memory recurrent neural network (LSTM-RNN) classifier for network intrusion detection on the KDD CUP 99 dataset and got DR = 98.88% and FAR = 10.04%.[49] Yin et al[50] proposed an RNN-based intrusion detection system and applied it on the NSL-KDD dataset (DR = 72.95%, FAR = 3.44%). Ponkarthika et al explored intrusion detection system based on LSTM-RNN architecture model.[51] They trained and tested their model over the KDD CUP 99 dataset with accuracy reached 83%. In the research study of Kim et al,[52] LSTM-RNN-based ensemble method is presented and evaluated using the ADFA dataset with achieved DR = 90% and FAR = 16%.

In addition to that, deep belief network (DBN), which is a stacked version of layer-to-layer restricted Boltzmann machine (RBM), intensively utilized in intrusion detection tasks in the past decade as a well-known deep learning model. Fiore et al adopted a discriminative RBM-based model for anomaly detection on 10% of KDD CUP 99 dataset.[53] Gao et al[54] proposed a network intrusion detection model based on DBN and demonstrated the experiment over the KDD CUP 99 dataset (DR = 92.33%, FAR = 0.76%). Alom et al explored the ability of DBN model to detect anomalies on 40% of NSL-KDD dataset and achieved detection accuracy with 97.5%.[55] In the study of Liu and Zhang,[56] the extreme learning machine (ELM) is applied into the learning process of the DBN model, and then evaluated using the NSL-KDD dataset (DR = 91.8%). Alrawashdeh et al introduced a deep learning approach using both RBM and DBN for online anomaly intrusion detection system on the 10% of KDD CUP 99 with 97.9% is DR and 2.47% is FAR.[57]

Ludwig[58] presented an ensemble deep learning model, which comprises AE, DBN, DNN, and ELM methods. He used the NSL-KDD dataset to measure DR (97.95%) and FAR (14.72%) metrics of the implemented model. A hybrid scheme for intrusion detection that combines DBN and support vector machine (SVM) is proposed by Salama et al[59] and performed tests on the NSL-KDD dataset (Accuracy = 92.84% when only 40% of the training samples is considered). In the study of Menon and Radhika,[60] a combined DBN-SVM model is introduced for intrusion detection with a high DR. Alom et al explored the effectiveness of using ELM and regularized ELM (RELM) methods in network intrusion detection on the NSL-KDD dataset.[61] They had a testing accuracy of 98.20% and 98.26% for ELM and RELM, respectively, after reducing the data dimensions from 41 to 9 essential features with 40% training data. Li et al[62] presented a hybrid intrusion detection scheme, which is based on AE and DBN. They achieved a DR equals 92.20% and a FAR equals 1.58% when 10% of KDD CUP 99 dataset is adopted. Qu et al proposed a DBN-based model for network intrusion detection on the NSL-KDD dataset with accuracy equals 95.25%.[63]

Finally, according to the above-reviewed literature, we can conclude the following three issues. (i) The vast majority of the previous studies extremely focused on binary classification and their results were presented in the form of binary classification rather than multiclass classification,

which the latter is harder. (ii) Some of the previous studies manipulated the number of features of the KDD CUP 99 or NSL-KDD datasets by either increasing the features up to 122 or taking a subset of them. (iii) Some of the previous studies used only a subset of samples from the NSL-KDD or KDD CUP 99 datasets in their experiments. Therefore, to our knowledge, developing a multiclass classification deep learning approach for network intrusion detection that applies to the full features and size of 10% of KDD CUP 99 and NSL-KDD datasets with high accuracy and DR as well as a low FAR value is still a big challenge.

4 | EXPERIMENTAL SETUP AND MODELS

We have performed four empirical experiments using the NIDS datasets we described in Section 2. Each one is deemed to be a multiclass classification task where, beside the normal class, there are multiple *n* attack classes existed. Moreover, in each experiment, the particular dataset is exploited along with six different models to discover the effectiveness of these models to detect various intrusive attacks, which are included in that dataset. Two out of these six models are machine learning (shallow) models. In contrast, the remaining four models are deep learning models, which belong to three different deep learning techniques. In the following subsections, we have explained the methodology of performing our empirical experiments as well as the description of each model.

4.1 | Shallow learning models

We preferred to use the Azure Machine Learning (AML) studio[64] to host our experiments on the datasets using two of its built-in multiclass models. The AML is a cloud-based machine learning suite developed by Microsoft and has many powerful advantages such as having a web-based working environment, an interactive and collaborative design, and a high capacity computing power for long running experiments.[65] The two selected multiclass shallow learning models are decision forest (DF) and decision jungle (DJ), which are known to give high accuracy levels on several domains. The DF is a classifier that learns multiple decision trees in such a way that the final decision is the combination of all predictions of its decision trees.[66] Building such a forest is worthwhile to compensate for the mistakes of a single tree by the other trees. On the other hand, DJ is an extensive form of decision forests in which uses rooted decision directed acyclic graphs.[67] In addition to that, the DJ allows tree branches to merge, which yields to reduce memory usage. The parameter settings we prefer for both models are shown in Table 3. In all experiments, for both DF and DJ models, we fixed the threshold at 0.5.

Our methodology in the experiments for shallow learning models is as follows. First, we uploaded all files of the datasets into the AML workspace. Then, we performed the data preprocessing on the datasets by applying data numericalization and normalization phases. The first constraint of our experiments; most of the machine learning models can use only numerical values for training and testing. Therefore, it is necessary to convert all nonnumerical values to numerical values by performing a data numericalization process. Indeed, in the literature, there are two methods to perform the data numericalization. The first is called "one-hot encoding," which gives each type of the nominal attribute a different binary vector. For instance, in KDD CUP 99 and NSL-KDD datasets, there are three nominal features, namely, "protocol_type," "service," and "flag" features. For "protocol_type" feature, there are three types of attributes, ie, "tcp," "udp," and "icmp," and its numeric values are encoded as binary vectors (1,0,0), (0,1,0), and (0,0,1). Similarly, the feature "service" has 70 types of attributes, and the feature "flag" has

TABLE 3 The parameter settings used in decision forest and decision jungle models

Decision Forest		Decision Jungle	
Setting	**Value**	**Setting**	**Value**
Resampling method	Bagging	Resampling method	Bagging
Trainer mode	Single Parameter	Trainer mode	Single Parameter
Number of decision trees	8	Number of decision DAGs	8
Maximum depth of the decision trees	32	Maximum depth of the decision DAGs	32
Number of random splits per node	128	Maximum width of the decision DAGs	128
Minimum number of samples per leaf node	1	Number of optimization steps per decision DAG layer	2048
Allow unknown categorical levels	Yes	Allow unknown categorical levels	Yes

11 types of attributes. Continuing in this way, 41-dimensional features map into 122-dimensional features after transformation. The second method is what we used in our study in such a way that, for each nominal feature, its values are ordered alphabetically. After that, the ordered nominal values are converted to numerical values by assigning specific values to each variable ranged in [1, length of the list] (eg, "icmp" = 1, "tcp" = 2 and "udp" = 3).

Compared with one-hot encoding method, we prefer to use the second method because it has many advantages. The second method does not increase the number of features because every transformed nominal feature is represented by one value. In contrast, one-hot encoding method increases the number of features because every transformed nominal feature is represented by a binary vector, which its length depends on number of the nominal feature's values. As a result, the architecture of the models when using the second method will be simpler than when using one-hot encoding method that because the model's inputs will be less. Thus, the second method will decrease the time needed to train and test the model.

The data normalization phase has taken place when all numeric features in the dataset (including the transformed nominal features) are mapped into [0,1] linearly by using Min-Max transformation formula in (1), ie,

$$x_{i_{=\frac{x_i-\text{Min}}{\text{Max}-\text{Min}}}}, \tag{1}$$

where x_i is the numeric feature value of the ith sample, Min and Max are the minimum and maximum values of every numeric feature, respectively. Then, DF and DJ models are trained in parallel on the training set of the particular dataset by using Train module. After that, the trained models are scored in parallel on the test set of the particular dataset by using the Score module. Finally, an Evaluation module is put forward to extract the classification outcomes. A diagrammatic representation of the processes in these experiments is shown in Figure 1.

4.2 | Deep learning models

Although using deep learning models may increase the training time, in general, these models exceed the traditional machine learning algorithms in terms of performance. Above all, the main concern when using one of these deep learning models is how to adjust its hyperparameters efficiently. The hyperparameters of a deep learning model are the set of vital settings that in such manner it controls the architecture and performance of that model in the underlying task. Practically, the problem is that the prior knowledge of these hyperparameters is required, that is, the

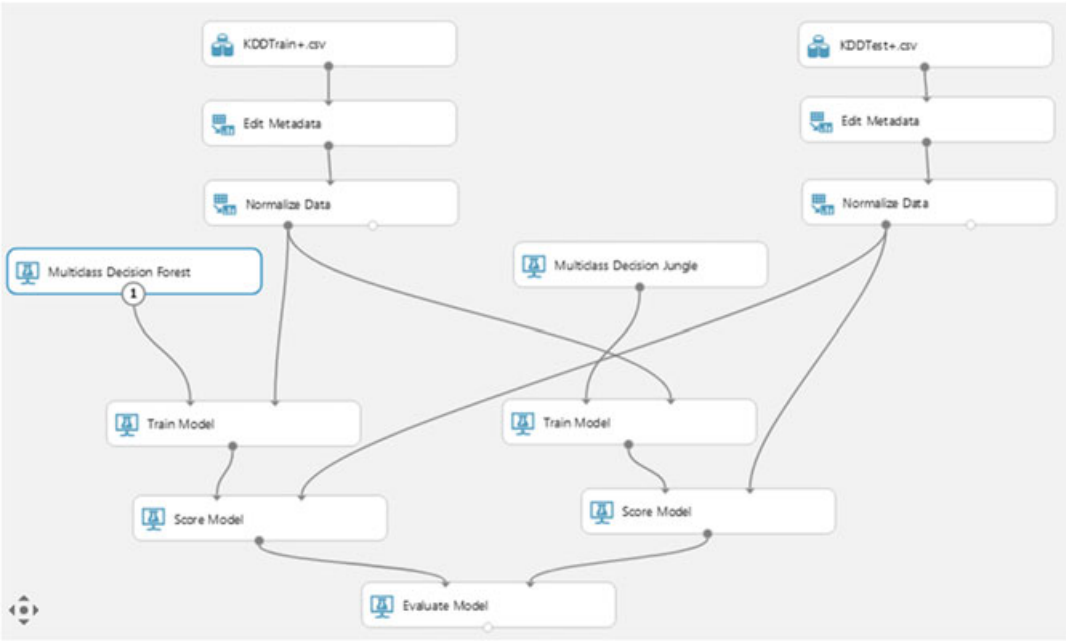

FIGURE 1 The experimental setup diagram of the shallow learning models in AML [Color figure can be viewed at wileyonlinelibrary.com]

values of these hyperparameters must be set just before the learning process begins. To solve this problem, several techniques are utilized, but, among them, using the evolutionary algorithms for an automatic selection of hyperparameters is recently widely used.

Elmasry et al[68] proposed a novel PSO-based algorithm that outputs the optimized hyperparameters vector of the given deep learning model. The former PSO-based algorithm seeks to maximize the accuracy over the given training and validation sets. This approach consists of four consecutive phases, which are preprocessing, initialization, evolution, and finishing phases. It begins with the preprocessing phase where the initial parameters of PSO algorithm are set as well as the user defines a list of the desired hyperparameters and their default domains. Then, in the initialization phase, the initial position and velocity vectors of the swarm for the first iteration are created depending on the previous phase. After that, the algorithm enters the evolution phase by executing a loop and updates the position and velocity vectors of the particles. This search process continues until one of the stopping criteria is satisfied. Finally, the PSO-based algorithm outputs the optimized hyperparameters vector and terminates in the finishing phase.

We used the aforementioned PSO-based algorithm for selecting the optimal hyperparameters of each deep learning model in our experiments. At the preprocessing phase of the algorithm, we defined 12 hyperparameters, namely, learning rate, decay, momentum, number of epochs, batch size, optimizer, initialization function, number of hidden layers, layer type, dropout rate, activation function, and number of neurons of the hidden layer. It is worthy to say that the first eight of them are global parameters, that is, they are fixed in the model. In contrast, the last four parameters are layer-based, which means they vary from layer to layer. Moreover, the default domain for each hyperparameter is set identically to the domains illustrated previously in the study of Elmasry et al.[68] Table 4 shows the values of major PSO parameters, which are gained

TABLE 4 The particle swarm optimization (PSO) parameters recommended domains and selected values

PSO Parameter	Domain	Selected value
Swarm size	[5,20]	20
Minimum velocity	{0,1}	0
Maximum velocity	{0,1}	1
Acceleration coefficients	[1,5]	2
Inertia weight constant	[0.4,0.9]	0.9
Maximum number of iterations	[30,50]	30
Evolution threshold	[0.001,0.0001]	0.0001

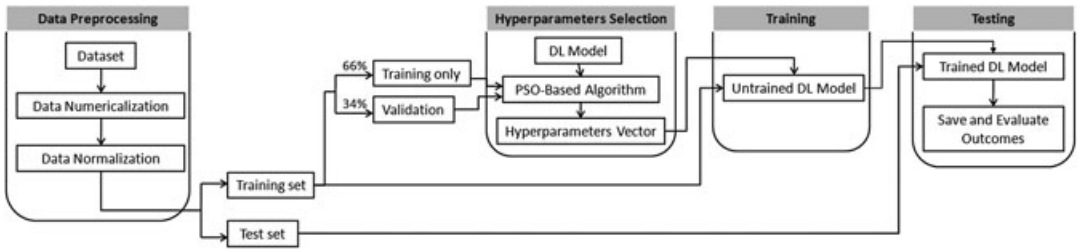

FIGURE 2 The methodology flowchart of the deep learning experiments. PSO, particle swarm optimization

by performing a grid search for each parameter in its predefined domain. These recommended domains are introduced in many theoretical and empirical studies.[69,70]

Our flow of execution in the experiments of deep learning models is as follows. First, a data preprocessing phase is put forward in the same manner as we mentioned in Section 4.1. Next, the training set of the particular dataset is split into two separate parts using a hold-out sampling with 66% training only data and 34% validation or unseen data. Afterwards, the obtained parts, as well as the type of the deep learning model we want to use, are delivered to the PSO-based algorithm for finding the optimal hyperparameters vector of that model. Then, we constructed the deep learning model and tuned it by the optimal hyperparameters. The resulting model is trained on the full training set of the particular dataset. Subsequently, the trained model is tested on the test set of the particular dataset. Finally, the classification outcomes are stored for further processing later. Figure 2 depicts the flowchart of the methodology in the experiments of the deep learning models.

Notably, we exploited four well-known deep learning models, namely, deep neural networks (DNNs), long short-term memory-recurrent neural networks (LSTM-RNNs), gated recurrent unit-recurrent neural networks (GRU-RNNs), and DBNs, to accomplish the network intrusion detection tasks on the used datasets. Figure 3 shows the typical structure of the DNN, an LSTM cell, a GRU unit, the Boltzmann machine (BM), the restricted BM (RBM), and the DBN. After finishing all deep learning experiments, we listed in Table 5 the values of global hyperparameters associated for deep learning models on the corresponding datasets.

4.2.1 | Deep neural networks

The DNN merely models the ANN but with many hidden layers.[71] As shown in Figure 3A, the DNN typically consists of an input layer, multiple hidden layers, and an output layer. The hidden

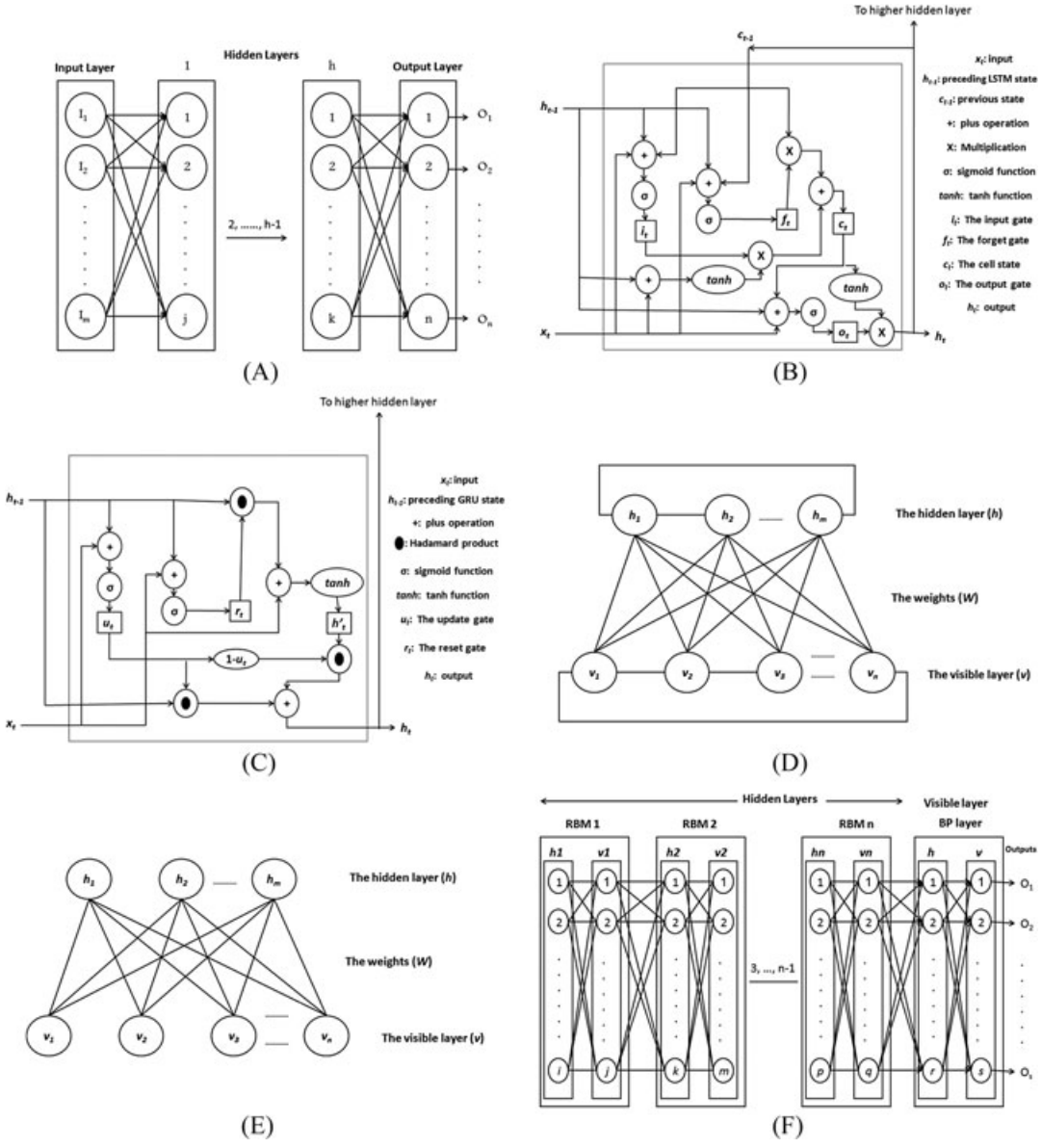

FIGURE 3 The basic structure of the (A) DNN, (B) LSTM cell, (C) GRU unit, (D) BM, (E) RBM, and (F) DBN. BM, Boltzmann machine; DBN, deep belief network; DNN, deep neural network; GRU, gated recurrent unit; LSTM, long short-term memory; RBM, restricted Boltzmann machine

layers deemed to do the most important work in the DNN. Every layer in the DNN consists of one or more artificial neuron in such a way that these neurons are fully-connected from layer to layer. Furthermore, the information is processed and propagated through the DNN in a feed-forward manner, ie, from the input layer to the output layer via the hidden layers. In this study, after applying the PSO-based algorithm, the resulting architectures of the DNN model regarding each dataset are shown in Figure 4.

TABLE 5 The resulting global hyperparameters values

Dataset		Global Hyperparameter							
		Learning rate	Decay	Momentum	Number of epochs	Batch size	Optimizer	Initialization function	Dropout rate
KDD CUP 99	DNN	0.1	0.01	0.9	100	250	RMSprop	Zero	0.5
	RNN	0.8	0.01	0.6	90	300	RMSprop	Lecun uniform	0.4
	DBN	0.01	0.001	0.1	10	450	Nadam	Lecun uniform	-
NSL-KDD	DNN	0.05	0.009	0.55	55	350	SGD	Normal	0.4
	RNN	0.03	0.008	0.5	35	350	Adagrad	Zero	0.3
	DBN	0.01	0.001	0.1	20	500	Adamax	He normal	-
CIDDS	DNN	0.03	0.007	0.6	35	450	Adagrad	Glorot normal	0.2
	RNN	0.02	0.003	0.4	35	450	Nadam	Uniform	0.2
	DBN	0.01	0.001	0.1	25	400	Adam	Glorot uniform	-
CICIDS2017	DNN	0.02	0.005	0.3	15	500	SGD	Uniform	0.2
	RNN	0.02	0.002	0.2	20	500	Adamax	He Normal	0.1
	DBN	0.01	0.001	0.1	30	550	Adam	He uniform	-

Abbreviations: DBN, deep belief network; DNN, deep neural network; RNN, recurrent neural network.

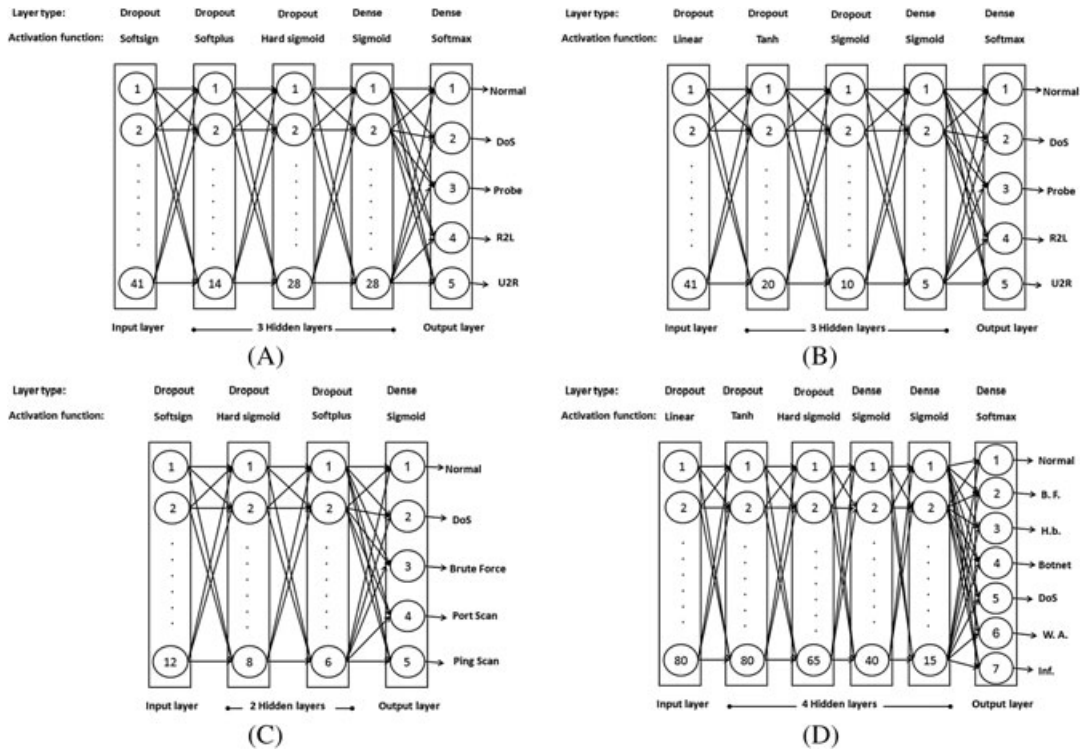

FIGURE 4 The deep neural network architecture regarding datasets (A) KDD CUP 99, (B) NSL-KDD, (C) CIDDS, and (D) CICIDS2017. BF, brute force; DoS, denial of service; Hb, heartbleed; Inf, infiltration; R2L, remote to local; U2R, user to remote; WA, web attack

4.2.2 | Recurrent neural networks

Unlike traditional ANNs, each neuron in any of the hidden layers of the RNN has additional connections from its output to itself, which is so-called self-recurrent, as well as to its adjacent neuron at the same hidden layer. Thus, the information circulates in the network, which practically makes the hidden layers as the storage unit of the whole network. However, the conventional RNN structure suffers from a bunch of serious problems known as gradient vanishing and exploding.[72] This inherent problems often appear when the RNN is trained using the back-propagation technique and limit the use of the RNN to be only in short-term memory tasks. In order to solve these problems, an LSTM structure is proposed by Hochreiter and Schmidhuber.[73] The LSTM uses a memory cell that is composed of four parts, namely, input gate, neuron with a self-recurrent connection, forget gate, and output gate. Meanwhile, the main goal of using a neuron with a self-recurrent is to record information, the aim of using three gates is to control the flow of information from or into the memory cell. Figure 3B shows the structure of an LSTM memory cell. It is reported that the LSTM-RNN model can be obtained easily by replacing every neuron in the RNN's hidden layers to an LSTM memory cell.[49]

Alternatively, Cho et al[74] introduced a new sophisticated unit, which is called GRU, that implements a gating mechanism to overcome the gradient vanishing and exploding in the conventional RNN. The major distinction between LSTM and GRU is that GRU has only two gates (update and reset) instead of three in LSTM. Hence, the GRU just exposes the full hidden content without any control. Figure 3C shows the structure of a GRU unit. Intuitively, the function of the update

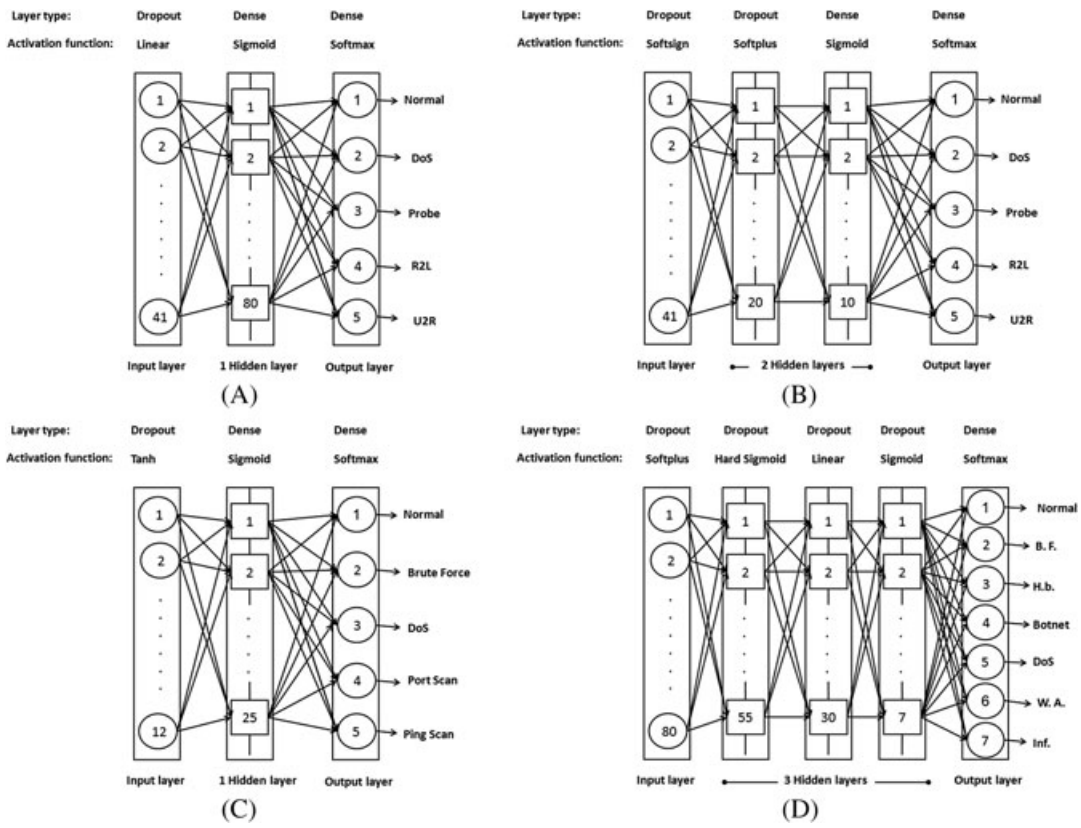

FIGURE 5 The LSTM-RNN/GRU-RNN architecture regarding datasets. A, KDD CUP 99; B, NSL-KDD; C, CIDDS; D, CICIDS2017. BF, brute force; DoS, denial of service; GRU, gated recurrent unit; Hb, heartbleed; Inf., infiltration; LSTM, long short-term memory; RNN, recurrent neural network; R2L, remote to local; U2R, user to remote; WA, web attack

gate in the GRU is to determine how much of the previous information needs to be kept around. Conversely, the reset gate is responsible for deciding how much of the previous information to forget. Obviously, the GRU-RNN model can be obtained by replacing each neuron in the hidden layers of the RNN by a GRU unit.[75] Figure 5 depicts the resulting architectures of the gated RNN model regarding each dataset after applying the PSO-based algorithm. As seen, the architecture of both LSTM-RNN and GRU-RNN models is identical for the same dataset.

4.2.3 | Deep belief networks

The BM is an energy-based neural network model, which consists of only two cascaded layers, namely, the visible layer (v) and the hidden layer (h).[40] Figure 3D shows the basic structure of a BM. Indeed, the BM is a particular form of log-linear Markov random field (MRF), and all its neurons are binary that outputs either 0 or 1. Regarding BM's structure, the neurons are connected by undirected connections between v and h layers as well as between neurons in the same layer. Recently, a customized version of BM is proposed and named RBM, which is considered as a kind of stochastic generative learning model.[76] The RBM is nothing more than a BM without visible-to-visible and hidden-to-hidden connections, that is, the whole connection is only between the visual layer and the hidden layer. The basic structure of a RBM is shown in Figure 3E.

In 2006, Hinton et al[77] introduced a generative probabilistic neural network so-called DBN. Figure 3F depicts the typical structure of the DBN. From a structural point of view, the DBN is a deep classifier that combines several stacked RBMs to a layer of back propagation (BP)[78] neural network. Meanwhile, the stacked RBMs are deemed to be the hidden layers of DBN, the BP layer is the visible layer. In addition to that, the connectivity between all hidden layers in the DBN is undirected connections. In contrast, the last RBM is connected to the visible layer by directed weights. In the open literature, the traditional DBN has two consecutive procedures, which are pretraining and fine-tuning. The pretraining procedure takes place by training all the hidden layers (RBMs) in layer-by-layer manner, ie, it trains only one layer at a time with the output of the higher layer being used as the input of the lower layer. To achieve that, a greedy layer-wise unsupervised training algorithm[79] is used along with unlabeled training data. Then, the parameters of whole DBN are fine-tuned by using BP learning algorithm along with the labeled training data. Recently, the DBN has attracted much attention and used in many data mining applications. Some of these applications use the DBN as an unsupervised feature extraction and feature reduction model. In this case, the DBN has only the stacked RBMs without any BP layer. On the other hand, some applications use the DBN for classification tasks, and in that case, the DBN consists of

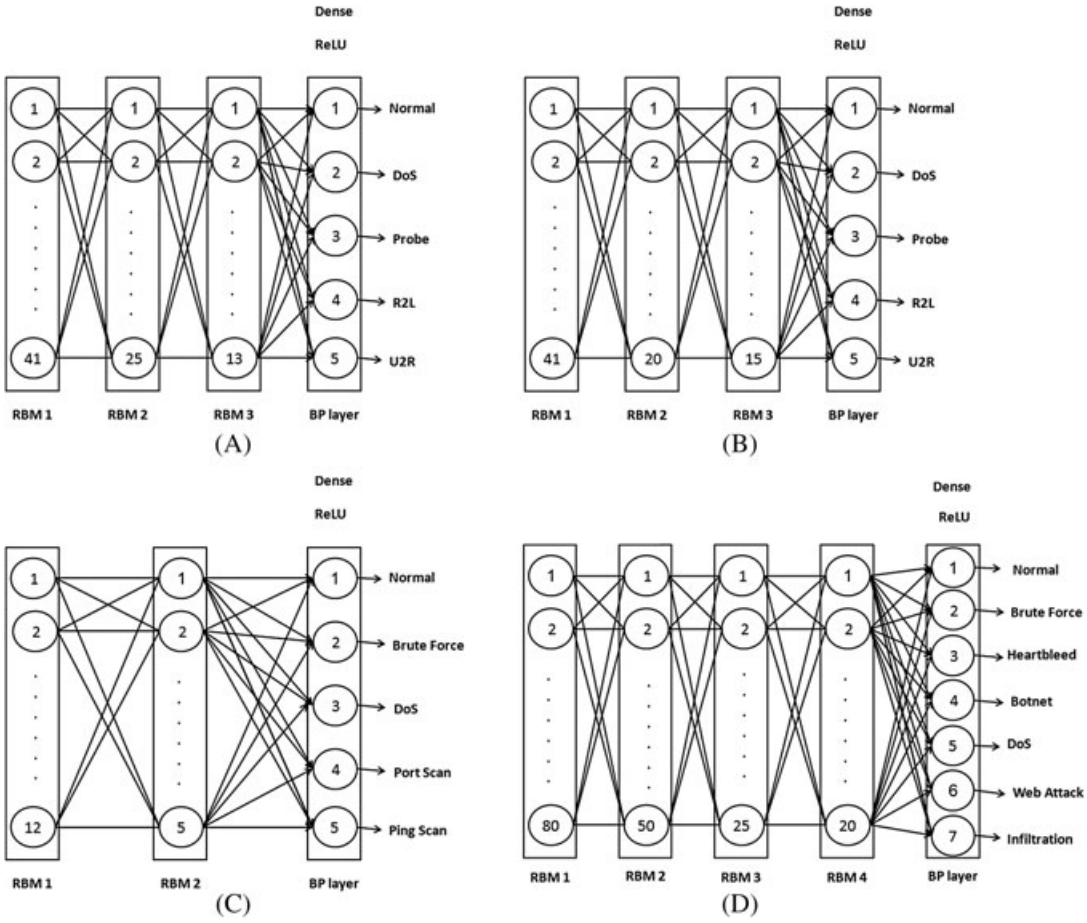

FIGURE 6 The deep belief network architecture regarding datasets. A, KDD CUP 99; B,NSL-KDD; C,CIDDS; D, CICIDS2017. BF, brute force; BP, back propagation; DoS, denial of service; Hb, heartbleed; Inf, infiltration; RBM, restricted Boltzmann machine; R2L, remote to local; U2R, user to remote; WA, web attack

several stacked RBMs along with a BP layer.[59] In this study, we used the DBN as a classifier on each dataset. After applying the PSO-based algorithm, the resulting architectures of the DBN model regarding each dataset are shown in Figure 6.

The term DBN^i, where i denotes the number of the RBM layers, is used to describe the structure of a DBN. Based on our results, we got DBN^3 (41–25–13-5), DBN^3 (41–20–15–5), DBN^2 (12–5-5), and DBN^4 (80–50–25-20-7) for the KDD KUP 99, NSL-KDD, CIDDS, and CICIDS2017 datasets, respectively. Regarding the iterations number, we also got 200, 300, 350, and 500 iteration numbers for the DBN models of the KDD KUP 99, NSL-KDD, CIDDS, and CICIDS2017 datasets, respectively.

5 | EVALUATION METRICS

In this paper, we have performed a multiclass network intrusion classification test, where each dataset has a mixture of normal (negatives) and various attacks (positives) samples. As indicated in Table 1, the number of labeled classes in each dataset varies. Therefore, when each model is applied to the particular dataset, a multiclass confusion matrix[80] is created to visualize the performance of the model. This confusion matrix maintains information about actual and predicted classes. Four main outcomes can be extracted from the confusion matrix, namely, true positives (*TPs*), true negatives (*TNs*), false positives (*FPs*), and false negatives (*FNs*).

Unlike binary classification schemes, the four outcomes have slightly different meaning in multiclass classification tasks. To start with, *TN* is the number of correct predictions of normal samples. On the other hand, *FP* is the sum of all normal samples that wrongly classified to any of the attack classes. *FP* can be calculated according to (2), where N is the number of attack classes, and FP_i is the number of normal samples wrongly classified to ith attack class. *TP* is the sum of all attack samples that truly labeled to their proper attack class, as calculated using (3), where TP_i is the number of accurate predictions of ith attack class. Finally, *FN* is the sum of all attack samples that incorrectly classified to the normal class. *FN* can be calculated according to (4), where FN_i is the number of the ith attack class samples misclassified to normal, ie,

$$FP = \sum_{i=1}^{N} FP_i \tag{2}$$

$$TP = \sum_{i=1}^{N} TP_i \tag{3}$$

$$FN = \sum_{i=1}^{N} FN_i. \tag{4}$$

Then, the four outcomes are exploited to compute 22 evaluation metrics, which will enable us to evaluate models performance on datasets. It is worthy to mention that some equations are modulated to adapt to the terminology definition of multiclass NIDS systems described earlier. The list of the used evaluation metrics definition and their corresponding equations are as follows.

- Accuracy shows the rate of true predictions for all test set, ie,

$$Accuracy = \frac{TP + TN}{Test\ set\ size}. \tag{5}$$

- Precision is the classifier's exactness, ie, the rate of correctly labeled attacks from all samples in the test set that were classified as attacks, ie,

$$Precision = \frac{TP}{TP + FP}. \tag{6}$$

- Recall is the classifier's completeness, ie, the rate of correctly labeled attacks for all attack samples that is presented in the test set. It is also known as Hit, true positive rate (TPR), detection rate (DR), or Sensitivity, ie,

$$Recall = \frac{TP}{Actual\ attacks\ size}. \tag{7}$$

- F1-Score can be considered as the harmonic mean of both precision (P) and recall (R) metrics. It is also known as the F1 metric, ie,

$$F1\_Score = \frac{2 \times P \times R}{P + R}. \tag{8}$$

- False alarm rate (FAR) shows the rate of normal samples that was incorrectly classified to any of the attack classes for all normal samples that presented in the test set. It is also called false positive rate (FPR), ie,

$$False\ Alarm\ Rate = \frac{FP}{FP + TN}. \tag{9}$$

- Specificity gives information about the rate of normal samples that was correctly classified for all normal samples that are presented in the test set. It is also called true negative rate (TNR), ie,.

$$Specificity = \frac{TN}{TN + FP}. \tag{10}$$

- False negative rate (FNR) is the complement of recall, ie, shows the rate of attack samples that were misclassified as a normal class from all attack samples in the test set. It is also called Miss, ie,

$$False\ Negative\ Rate = \frac{FN}{Actual\ attacks\ size}. \tag{11}$$

- Negative precision is the rate of correctly classified normal samples over all samples in the test set that were classified as the normal class, ie,

$$Negative\ Precision = \frac{TN}{TN + FN}. \tag{12}$$

- Error rate gives information about the rate of false predictions for all test set, ie,

$$Error\ Rate = \frac{FP + FN}{Test\ set\ size}. \tag{13}$$

- Efficiency is a proportion of recall to FAR, ie,

$$Efficiency = \frac{Recall}{FAR}. \tag{14}$$

- Area under the curve (AUC) is considered the classifier's ability to avoid false classification.[81] It is also referred to as balanced accuracy, ie,

$$AUC = \frac{1}{2} \times (Recall + Specificity). \tag{15}$$

- Odds-ratio is a metric that gives information about the accuracy of a classifier taking into consideration all main outcomes, ie,

$$Odds\_Ratio = \frac{TP \times TN}{FP \times FN}. \tag{16}$$

- Kappa (K) compares an Observed Accuracy with an Expected Accuracy (random chance). If kappa <0.4 indicating poor agreement. However, kappa is sensitive to the sample size and it is unreliable if one class dominates. It is also known as Cohen's Kappa, ie,

$$Kappa = \frac{(TP + TN) - \lambda}{Test\ set\ size - \lambda}, \tag{17}$$

$$\text{where } \lambda = \frac{(TP + FN) \times (TP + FP) + (FP + TN) \times (FN + TN)}{Test\ set\ size}. \tag{18}$$

- Bayesian DR (BDR) is based on base-rate fallacy problem, which is addressed by Axelsson.[82] Base-rate fallacy is one of the basis of the Bayesian statistics. It occurs when people do not take the basic rate of incidence (base-rate) into their account when solving problems in probabilities. Unlike recall metric, BDR determines the rate of correctly labeled attack samples for all test set taking into consideration the base-rate of attack classes. Mathematically, let I and I^* denote an intrusive and a normal behavior, respectively. Furthermore, let A and A^* denote the predicted attack and normal behavior, respectively. Then, BDR can be computed as the probability $P(I|A)$ according to (19) (see the work of Axelsson[82]), ie,

$$Bayesian\ Detection\ Rate = P(I|A) = \frac{P(I) \times P(A|I)}{P(I) \times P(A|I) + P(I^*) \times P(A|I^*)}, \tag{19}$$

where $P(I)$ is the rate of the attack samples in the test set, $P(A|I)$ is the recall, $P(I^*)$ is the rate of the normal samples in the test set, and $P(A|I^*)$ is the FAR.

- Bayesian true negative rate (BTNR) is also based on the problem of base-rate fallacy. It gives information about the rate of correctly classified normal samples for all test set such that the predicted normal behavior indicates really a normal connection.[82] Mathematically, let I and I^* denote an intrusive and a normal behavior, respectively. Moreover, let A and A^* denote the predicted attack and normal behavior, respectively. Then, BTNR can be computed as the probability $P(I^*|A^*)$ according to (20) (see the work of Axelsson[82]), ie,

$$Bayesian\ True\ Negative\ Rate = P(I^*|A^*) = \frac{P(I^*) \times P(A^*|I^*)}{P(I^*) \times P(A^*|I^*) + P(I) \times P(A^*|I)}, \tag{20}$$

where $P(I^*)$ is the rate of the normal samples in the test set, $P(A^*|I^*)$ is the Specificity, $P(I)$ is the rate of the attack samples in the test set, and $P(A^*|I)$ is the FNR.

- Geometric Mean (g-mean) combines the Specificity and Recall metrics at one specific threshold where both the errors are considered equal. It has been previously used by several research works for evaluating the performance of classifiers on imbalance dataset.[83] Indeed, it has two different versions. Meanwhile, g_mean$_1$ focuses on both the positive and the negative classes,[84] g_mean$_2$ focuses solely on the positive classes, ie,[85]

$$g\_mean_1 = \sqrt{Recall \times Specificity} \tag{21}$$

$$g\_mean_2 = \sqrt{Recall \times Precision}\ . \tag{22}$$

- Matthews correlation coefficient (MCC) is a performance metric that takes into account all the cells of the confusion matrix in its equation. It is deemed to be a balanced measure, which can be used with imbalance datasets, ie, even if the classes are of very different sizes.[86] In addition, MCC has a range of −1 to 1, where −1 indicates a completely wrong classifier, whereas 1 indicates a completely correct classifier. As calculated using (23) (see the work of Matthews[87]), ie,

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FN) \times (TP + FP) \times (TN + FP) \times (TN + FN)}}. \tag{23}$$

- Missed rate (MR) is a performance metric for a multiclass classifier that was proposed by Elhamahmy et al.[88] They defined a new outcome that can be extracted from a multiclass confusion matrix, namely, misclassification of attack class (MC). MC determines the number of the particular attack class samples that are wrongly labeled to any another attack class. In this case, these incorrectly predicted samples could not belong to any of the four main outcomes. MR can be computed using (24), ie,

$$MR = \frac{FN + \sum_{i=1}^{N} MC_i}{Actual\ attacks\ size}, \tag{24}$$

where MC$_i$ is the MC of the *i*th attack class.

- Wrong rate (WR) is also a performance metric for a multiclass classifier and based on MC outcome.[88] It is the proportional fraction of wrongly predicted attack samples to all samples in the test set that were classified as attacks and can be computed according to (25), ie,

$$WR = \frac{FP + \sum_{i=1}^{N} MC_i}{TP + FP + \sum_{i=1}^{N} MC_i}. \tag{25}$$

- F-score per cost (FPC) is a new metric for a multiclass classifier and based on F1-Score, MR, and WR metrics.[88] FPC value varies from 0 to 1, where 0 indicates a completely wrong classifier. Otherwise, when it equals 1, it indicates an ideal classifier, ie,

$$FPC = \frac{F1 - Score}{\sqrt{(F1 - Score)^2 + (Cost)^2}} \tag{26}$$

$$\text{where } Cost = \sqrt{(MR)^2 + (WR)^2}. \tag{27}$$

- Training time is the time elapsed for completing the training phase of the model.
- Testing time is the time elapsed for accomplishing the testing phase of the model.

6 | RESULTS AND DISCUSSION

The PSO-based algorithm was implemented using Python programming language version 3.6.4[89] with NumPy library.[90] In addition to that, we preferred Keras open source neural network library[91,92] over TensorFlow machine learning framework version 1.6[93,94] with CUDA integrated backend version 9.0[95] and cuDNN version 7.0[96] to construct, train, and test all the deep learning models. Regarding the hardware environment, all the experiments were performed on a computer that has the following configurations: Intel Core i7 CPU (3.8 GHz, 16 MB Cache), 16 GB of RAM, and the Windows 10 operating system (64-bit mode). Moreover, the GPU-accelerated computing with NVIDIA Tesla K20 GPU 5 GB GDDR5 is also utilized to accelerate the computations to improve efficiency. In the following three sections, we present our experimental results and discuss them via three different analyses, such as performance analysis, Friedman statistical test, and ranking methods.

6.1 | Performance analysis

The effectiveness of a model to detect network intrusions depends on its profile of evaluation metrics. The higher values in Accuracy, Precision, Recall, and F1-Score, as well as the lower values of FNR and FAR indicate an efficient classifier. The ideal classifier has Accuracy and Recall values reach 1, as well as FNR and FAR values reach 0. Tables 6 to 9 present the values of the evaluation metrics for KDD CUP 99-experiment, NSL-KDD-experiment, CIDDS-experiment, and CICIDS2017-experiment, respectively. In fact, all values are presented in the percentage format except Efficiency, Odds-Ratio, Training time, and Testing time metrics. Furthermore, the bold values represent the best results among the same dataset. In order to compare deep learning models' performance with pretraining phase with their performance without pretraining phase, we also included for each deep learning model its results with pretraining (w/) and without pretraining (w/o). In addition to that, we carried out the experiments of CIDDS and CICIDS2017 datasets over 10 subsets. Then, to prove that the results are not biased, we reported the results in Table 8 and Table 9 for each metric in the form of the average value.

However, the accuracy values of KDD CUP 99 dataset for all models are deemed to be very high, as a matter of fact, the accuracy value is only reflecting the underlying class distribution in the dataset. This condition is also known as the accuracy paradox where the accuracy value does not reflect the exact performance of the model. This misleading assessment happened due to the fact that KDD CUP 99 dataset has inherent problems, especially, it has a numerous number of redundant records. It was reported that 78% and 75% of the samples are duplicated in the train and test set of the KDD CUP 99 dataset, respectively.[22] Hence, the learning model is trained to be biased toward the more frequent samples, which yield to DR between 86% and 98% even when using a simple classification algorithm. Thus, the KDD CUP 99 is not the best choice for NIDS despite it being extensively used in the literature.

In like manner, the NSL-KDD dataset is considered to be the improved version of the KDD CUP 99 dataset, where all the raised drawbacks are resolved. The results of the NSL-KDD for all models are more reasonable than those corresponding values of KDD CUP 99 dataset. Nevertheless, the NSL-KDD dataset has an imbalanced structure. The imbalanced dataset typically refers

TABLE 6 The results of KDD CUP 99-experiment

Metric	DF	DJ	DNN		LSTM-RNN		GRU-RNN		DBN	
			w/	w/o	w/	w/o	w/	w/o	w/	w/o
Accuracy	86.27%	87.81%	92.18%	90.67%	95.2%	91.36%	94.18%	90.64%	**98.63%**	91.36%
Precision	96.28%	97.3%	97.79%	97.41%	98.53%	96.57%	98.24%	96.33%	**99.48%**	95.35%
Recall	86%	87.03%	92.22%	90.64%	95.36%	92.37%	94.35%	91.68%	**98.8%**	93.67%
F1-Score	90.85%	91.88%	94.92%	93.90%	96.92%	94.43%	96.25%	93.95%	**99.14%**	94.50%
FAR	12.69%	9.22%	7.98%	9.23%	5.44%	8.53%	6.48%	9.36%	**1.99%**	4.48%
Specificity	87.31%	90.78%	92.02%	90.77%	94.56%	87.47%	93.52%	86.64%	**98.01%**	82.52%
FNR	5.95%	5.73%	3.01%	3.01%	1.76%	1.76%	1.88%	1.88%	**0.42%**	0.42%
Negative Precision	79.32%	80.56%	88.89%	88.75%	93.36%	92.86%	92.85%	92.33%	**98.4%**	98.10%
Error Rate	7.35%	6.45%	4.04%	4.30%	2.52%	3.99%	2.84%	4.26%	**0.74%**	1.96%
Efficiency	6.77	9.44	11.56	9.82	17.54	10.37	14.56	12.86	**49.64**	35.36
AUC	86.65%	88.91%	92.12%	90.71%	94.96%	89.92%	93.93%	89.16%	**98.4%**	88.09%
Odds-Ratio	99.37	149.57	353.55	296.32	943.66	566.80	723.02	515.85	**11645.4**	1058.30
Kappa	68.55%	71.66%	80.01%	77.16%	86.87%	78.05%	84.51%	76.60%	**95.96%**	77.44%
BDR	96.28%	97.30%	97.79%	97.41%	98.53%	96.57%	98.24%	96.33%	**99.48%**	95.35%
BTNR	79.32%	80.56%	88.89%	88.75%	93.36%	92.86%	92.85%	92.33%	**98.4%**	98.10%
g_mean1	86.65%	88.89%	92.12%	90.71%	94.96%	89.89%	93.93%	89.13%	**98.4%**	87.92%
g_mean2	91%	92.03%	94.96%	93.96%	96.93%	94.45%	96.27%	93.98%	**99.14%**	94.50%
MCC	78.17%	81.16%	87.76%	86.86%	92.32%	87.50%	91.32%	86.62%	**97.73%**	87.58%
MR	14%	12.97%	7.78%	9.36%	4.64%	7.63%	5.65%	8.32%	**1.2%**	6.33%
WR	11.67%	9.98%	6.92%	8.81%	4.32%	9.01%	5.47%	9.77%	**1.3%**	6.07%
FPC	98.05%	98.45%	99.40%	99.08%	99.79%	99.23%	99.67%	99.08%	**99.98%**	99.22%
Training time (sec)	**283**	321	570	628	672	699	609	664	806	854
Testing time (sec)	**202**	223	469	525	511	576	499	518	755	800

Abbreviations: AUC, area under the curve; BDR, Bayesian detection rate; BTNR, Bayesian true negative rate; DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; FAR, false alarm rate; FNR, false negative rate; FPC, F-score per cost; GRU, gated recurrent unit; LSTM, long short-term memory; MCC, Matthews correlation coefficient; MR, missed rate; RNN, recurrent neural network; WR, wrong rate.

to a problem with classification problems where the entire classes are not represented equally. In the case of the NSL-KDD dataset, the test set distribution is as follows: 43%, 33%, 10.7%, 13%, and 0.3% for Normal, DoS, Probe, R2L, and U2R classes, respectively. Accordingly, the learning models tend to classes with the vast majority of samples rather than with ones of the small minority. This leads to that the NSL-KDD dataset is more suitable to binary classification, where there is a small difference between Normal (43%) and Attack (57%) classes, rather than to multiclass classification.

In addition to that, we have implemented subsets of the CIDDS and CICIDS2017 datasets to be balanced datasets in our study, as mentioned in Section 2. As we expected, this balanced structure enhanced the results of all models compared to the results of the NSL-KDD dataset. However, even though the CIDDS and CICIDS2017 datasets are modern form of NIDS datasets, the CICIDS2017 is more reliable than CIDDS dataset. This can be explained by the fact that the CIDDS dataset has a relatively small number of features (only 12), and the CICIDS2017 has more various attack categories than in the CIDDS dataset.

At a glance, we observed that the deep learning models outperformed the shallow learning models in terms of all metrics and for all datasets. This was possible due to the impact of using

TABLE 7 The results of NSL-KDD-experiment

Metric	DF	DJ	DNN		LSTM-RNN		GRU-RNN		DBN	
			w/	w/o	w/	w/o	w/	w/o	w/	w/o
Accuracy	73.35%	73.38%	86.53%	84.95%	91.16%	89.61%	90.63%	88.52%	**93.78%**	91.54%
Precision	94.54%	96.44%	97.73%	95.73%	98.56%	96.47%	98.11%	95.92%	**98.71%**	96.14%
Recall	56.45%	55.27%	78.15%	76.22%	85.72%	81.67%	85.19%	80.71%	**90.25%**	88.87%
F1-Score	70.69%	70.27%	86.85%	85.25%	91.69%	90.29%	91.19%	88.39%	**94.29%**	92.36%
FAR	4.3%	2.7%	2.4%	3.1%	1.66%	2.4%	2.17%	2.98%	**1.55%**	1.93%
Specificity	95.7%	97.3%	97.6%	95.77%	98.34%	97.92%	97.83%	96.39%	**98.45%**	96.27
FNR	40.82%	40.81%	19.32%	18.92%	11.77%	11.93%	12.13%	12.81	**7.67%**	10.31%
Negative Precision	63.95%	64.34%	79.27%	79.51%	86.35%	84.53%	85.92%	83.76%	**90.67%**	87.57%
Error Rate	25.09%	24.39%	12.03%	13.65%	7.41%	8.51%	7.84%	10.52%	**5.03%**	7.61%
Efficiency	13.11	20.49	32.57	30.71	51.7	47.9	39.21	37.92	**58.04**	50.39
AUC	76.07%	76.29%	87.88%	85.26%	92.03%	90.01%	91.51%	87.96%	**94.35%**	92.37%
Odds-Ratio	30.74	48.85	164.57	151.36	432.11	419.34	316.12	300.7	**745.19**	683.79
Kappa	50.11%	50.78%	74.04%	72.83%	82.79%	80.97%	81.81%	77.34%	**87.77%**	85.22%
BDR	94.54%	96.44%	97.73%	97.18%	98.56%	96.65%	98.11%	95.83%	**98.71%**	96.73%
BTNR	63.95%	64.34%	79.27%	77.87%	86.35%	84.15%	85.92%	83.47%	**90.67%**	88.89%
g_mean1	73.5%	73.34%	87.34%	85.36%	91.81%	90.18%	91.29%	88.31%	**94.26%**	92.71%
g_mean2	73.05%	73.01%	87.39%	84.22%	91.91%	90.65%	91.42%	89.13%	**94.39%**	93.67%
MCC	56.06%	57.73%	77.39%	75.39%	85.59%	84.44%	84.69%	82.85%	**90%**	88.57%
MR	43.55%	44.73%	21.85%	25.92%	14.28%	16.71%	14.81%	17.93%	**9.75%**	10.29%
WR	9.59%	9.74%	5.27%	7.82%	4.21%	6.36%	4.83%	6.9%	**3.48%**	4.44
FPC	84.58%	83.79%	96.81%	93.81%	98.71%	97.49%	98.57%	96.71%	**99.4%**	98.19%
Training time (sec)	**72**	89	370	428	472	499	409	464	606	654
Testing time (sec)	**41**	53	269	325	311	376	299	318	555	610

Abbreviations: AUC, area under the curve; BDR, Bayesian detection rate; BTNR, Bayesian true negative rate; DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; FAR, false alarm rate; FNR, false negative rate; FPC, F-score per cost; GRU, gated recurrent unit; LSTM, long short-term memory; MCC, Matthews correlation coefficient; MR, missed rate; RNN, recurrent neural network; WR, wrong rate.

the PSO-based algorithm in the pretraining phase of the deep learning models. As described in Section 4, the PSO-based algorithm attempts to select the optimal hyperparameters of the model that maximizes the accuracy on the given dataset. According to formula (5), as the accuracy value increases, the sum of *TP* and *TN* in the numerator will be increased significantly as the denominator is constant. Therefore, this yields to definitely increase the value of classification metrics such as recall, as well as to decrease the value of misclassification metrics such as FAR. Our findings confirmed that the deep learning models increased the recall metric by 5% to 20% and decreased the FAR metric by 2% to 10% from the corresponding values of the shallow learning models on the same dataset.

By taking an inspective look to the results, we noticed clearly the stability of deep learning models in such a way that they enhance network intrusion detection from a dataset to another in a consistent pattern. However, the gated-RNN models performed superior than the DNN in terms of all evaluation metrics regarding all datasets. The fact behind this culminate of the results that, instead of using the traditional artificial neurons in all hidden layers, the LSTM-RNN and the GRU-RNN models use LSTM memory cells and GRU units, respectively. Moreover, the gated-RNN models have extra hidden-to-hidden connections, that is, among the processing

TABLE 8 The results of CIDDS-experiment

Metric	DF	DJ	DNN		LSTM-RNN		GRU-RNN		DBN	
			w/	w/o	w/	w/o	w/	w/o	w/	w/o
Accuracy	68.08%	73.92%	83.3%	82.95%	89.28%	85.36%	92.84%	90.64%	**94.66%**	91.36%
Precision	97.1%	97.8%	98.83%	97.73%	99.37%	98.57%	99.62%	98.33%	**99.71%**	99.35%
Recall	61.95%	68.95%	80.08%	76.22%	87.15%	82.37%	91.4%	90.68%	**93.6%**	92.67%
F1-Score	75.64%	80.88%	88.47%	85.25%	92.86%	90.43%	95.33%	93.95%	**96.56%**	94.50%
FAR	7.4%	6.2%	3.8%	5.1%	2.2%	3.53%	1.4%	2.36%	**1.1%**	1.48%
Specificity	92.6%	93.8%	96.2%	95.77%	97.8%	96.47%	98.6%	96.64%	**98.9%**	97.52%
FNR	16.2%	12.9%	7.45%	9.92%	4.63%	6.76%	2.63%	3.88%	**1.75%**	2.42%
Negative Precision	58.83%	64.51%	76.35%	74.51%	84.09%	82.86%	90.38%	87.33%	**93.39%**	91.10%
Error Rate	14.44%	11.56%	6.72%	9.65%	4.14%	5.99%	2.38%	3.26%	**1.62%**	2.96%
Efficiency	8.37	11.12	21.07	19.71	39.61	30.37	65.29	60.86	**85.09**	71.36
AUC	77.28%	81.38%	88.14%	85.26%	92.48%	90.92%	95%	92.16%	**96.25%**	94.09%
Odds-Ratio	47.85	80.86	272.1	171.36	837.67	566.80	2452.26	1515.85	**4808.85**	3058.30
Kappa	48.34%	54.45%	66.31%	62.83%	75.55%	71.05%	82.29%	76.60%	**86.12%**	77.44%
BDR	97.1%	97.8%	98.83%	97.18%	99.37%	98.57%	99.62%	98.33%	**99.71%**	99.35%
BTNR	58.83%	64.51%	76.35%	75.87%	84.09%	82.86%	90.38%	90.33%	**93.39%**	92.10%
g_mean1	75.74%	80.42%	87.77%	85.36%	92.32%	89.89%	94.93%	90.13%	**96.21%**	94.92%
g_mean2	77.56%	82.12%	88.96%	84.22%	93.06%	92.45%	95.42%	93.98%	**96.61%**	94.50%
MCC	63.4%	69.73%	81.19%	75.39%	87.99%	87.50%	92.86%	89.62%	**95.06%**	93.58%
MR	38.05%	31.05%	19.93%	25.92%	12.85%	15.63%	8.6%	10.32%	**6.4%**	8.33%
WR	27.67%	22.22%	14.36%	18.82%	9.15%	11.01%	6.47%	9.77%	**5%**	6.07%
FPC	84.91%	90.43%	96.36%	93.81%	98.59%	98.23%	99.37%	99.08%	**99.65%**	99.22%
Training time (sec)	**1083**	1121	1307	1362	1491	1448	1475	1489	1665	1671
Testing time (sec)	**1002**	1023	1226	1202	1290	1361	1321	1367	1582	1654

Abbreviations: AUC, area under the curve; BDR, Bayesian detection rate; BTNR, Bayesian true negative rate; DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; FAR, false alarm rate; FNR, false negative rate; FPC, F-score per cost; GRU, gated recurrent unit; LSTM, long short-term memory; MCC, Matthews correlation coefficient; MR, missed rate; RNN, recurrent neural network; WR, wrong rate.

TABLE 9 The results of CICIDS2017-experiment

Metric	DF	DJ	DNN		LSTM-RNN		GRU-RNN		DBN	
			w/	w/o	w/	w/o	w/	w/o	w/	w/o
Accuracy	78.39%	80.6%	83.34%	81.67%	89.09%	88.61%	93%	91.52%	**95.60%**	93.54%
Precision	98.91%	99.11%	99.37%	97.41%	99.64%	98.47%	99.77%	98.92%	**99.86%**	99.14%
Recall	75.62%	78.07%	81.08%	80.64%	87.58%	85.67%	92.05%	89.71%	**95%**	93.87%
F1-Score	85.71%	87.34%	89.3%	85.90%	93.22%	90.29%	95.75%	92.39%	**97.37%**	96.36%
FAR	5%	4.2%	3.1%	3.93%	1.9%	2.4%	1.3%	2.03%	**0.8%**	1.11%
Specificity	95%	95.8%	96.9%	92.77%	98.1%	97.92%	98.7%	98.39%	**99.2%**	98.72
FNR	4.53%	3.73%	2.95%	3.01%	1.63%	2.93%	0.85%	1.21	**0.55%**	0.91%
Negative Precision	77.74%	81.05%	84.55%	83.75%	90.92%	86.53%	95.09%	89.76%	**96.78%**	95.57%
Error Rate	4.6%	3.8%	2.97%	3.01%	1.67%	2.51%	0.91%	1.52%	**0.59%**	0.98%
Efficiency	15.12	18.59	26.16	26.82	46.1	40.9	70.81	56.92	**118.75**	106.39
AUC	85.31%	86.93%	88.99%	87.71%	92.84%	90.01%	95.38%	93.96%	**97.1%**	96.37%
Odds-Ratio	316.92	476.96	859.16	696.32	2768.61	419.34	8222.02	6972.7	**21418.18**	20577.79
Kappa	58.82%	61.39%	64.77%	62.16%	72.84%	70.97%	79.95%	77.34%	**85.86%**	85.22%
BDR	98.91%	99.11%	99.37%	96.41%	99.64%	97.65%	99.77%	98.83%	**99.86%**	99.73%
BTNR	77.74%	81.05%	84.55%	83.75%	90.92%	88.15%	95.09%	91.47%	**96.78%**	85.89%
g_mean1	84.76%	86.48%	88.64%	86.71%	92.69%	90.18%	95.32%	92.31%	**97.08%**	96.71%
g_mean2	86.48%	87.96%	89.76%	87.96%	93.42%	90.65%	95.83%	94.13%	**97.4%**	96.67%
MCC	82.75%	85.52%	88.53%	86.86%	93.37%	89.44%	96.31%	94.85%	**97.63%**	93.57%
MR	24.38%	21.93%	18.92%	19.36%	12.42%	16.71%	7.95%	10.93%	**5%**	6.29%
WR	21.48%	19.49%	16.89%	18.81%	11.25%	15.36%	7.36%	11.9%	**4.6%**	4.44
FPC	93.51%	94.79%	96.2%	95.08%	98.42%	97.49%	99.37%	97.71%	**99.76%**	98.19%
Training time (sec)	**538**	612	870	826	919	984	977	999	1156	1189
Testing time (sec)	**520**	532	762	720	799	816	821	867	1028	1110

Abbreviations: AUC, area under the curve; BDR, Bayesian detection rate; BTNR, Bayesian true negative rate; DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; FAR, false alarm rate; FNR, false negative rate; FPC, F-score per cost; GRU, gated recurrent unit; LSTM, long short-term memory; MCC, Matthews correlation coefficient; MR, missed rate; RNN, recurrent neural network; WR, wrong rate.

elements in the same hidden layer. Hence, these built-in characteristics of the gated-RNN models, which do not exist in the DNN, enable the classifier to memorize the previous states, explore the dependencies among them, and finally use them along with current inputs to predict the output. Meanwhile, the difference between the performance of the gated-RNN and the DNN models on all datasets is significant for recall metric, which is between 2% and 10%, but it is small for FAR metric, which is between 1% and 2% in all cases.

In spite of the LSTM cell having four gates rather than having two gates like GRU, GRU-RNN is relatively new and its performance is on par with the LSTM-RNN. In the experiments, the LSTM-RNN model worked better in the KDD CUP 99 and NSL-KDD datasets. Meanwhile, the GRU-RNN model performed superior in both CIDDS and CICIDS2017 datasets. Giving the importance to the former findings, the GRU-RNNs tend to do better than the LSTM-RNN in the case of small datasets, and vice versa for the LSTM-RNN in the larger datasets.

Likewise, other than deep learning models, the DBN also enhanced the detection of malicious connection points in the datasets. In addition to that, the results merely indicate that the DBN model is superior to the DNN, LSTM-RNN, and GRU-RNN models in terms of all used evaluation metrics on all datasets. This is due to the nature of DBN's deep structure, where it consists of several RBMs followed by a fully-connected BP layer. The RBMs try to learn the useful features from the inputs by using their hidden and visible layers. During the learning process, the RBMs are pretrained in an unsupervised manner. Then, the shorten extracted features are fed to a BP layer, which is fine-tuned with the whole network by using BP in a supervised manner. Thus, this functionality makes the DBN a robust classifier that able to deeply understand the underlying task. The DBN models produced significant results on all datasets such that recall between 90.25% and 98.8% and FAR between 0.8% and 1.99%. However, of that, DNN, LSTM-RNN, and GRU-RNN models also performed very well in network intrusion detection on the datasets.

Regarding the Bayesian metrics, all deep learning models produced high scores in most cases, which indicates that the confidence of the predicated behaviors is very high. Indeed, this absolutely depends on the structure of the dataset, that is, the BDR metric will increase as much as the number of positive samples in the test set as well as recall values are larger. In contrast, BTNR metric will increase as much as the number of negative samples in the test set is larger and FAR value is smaller. Despite KDD CUP 99 and NSL-KDD datasets being imbalanced, all deep learning models got high g-mean and MCC percentages. Finally, in general, all deep learning models recorded high scores in metrics that measure the true classifications as well as low values in metrics that measure error or misclassifications regarding all datasets.

For the sake of brevity and space limitation, we selected only the standard classification metrics to be presented visually in Figure 7. Figures 7A, 7B, 7C, and 7D show Accuracy, Recall, F1-Score, and FAR percentages of all models in each dataset, respectively. Moreover, Figures 8A, 8B, 8C, and 8D depict the Detection Rate of each class in the KDD CUP 99, NSL-KDD, CIDDS, and CICIDS2017 datasets, respectively. Figures 7 and 8 can give us a visual comparison of the performance of all models on each dataset.

6.2 | Friedman test

In order to give a further inspection of our results, we also performed a well-known statistical test, namely, Friedman test. The Friedman test is a nonparametric test for finding out the differences between three or more repeated treatments.[97] Nonparametric test means that the test does not assume that your data comes from a particular distribution. In this study, we have four repeated treatments ($k = 4$) each for one of the used datasets, and six subjects ($Z = 6$) in every treatment

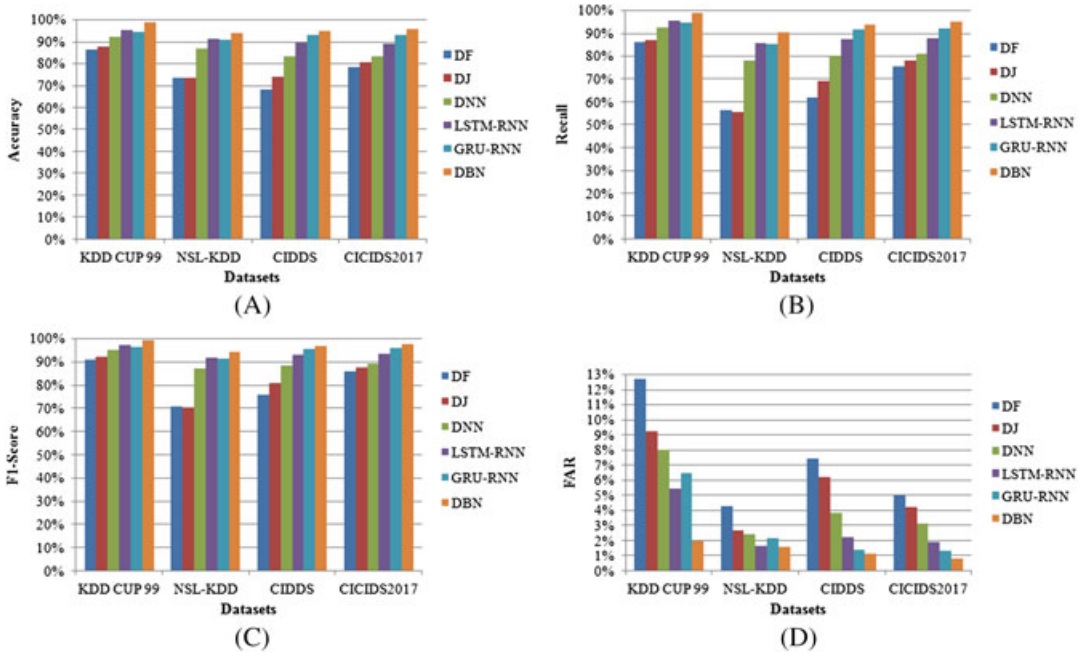

FIGURE 7 Evaluation metrics comparison between models on datasets. A, Accuracy; B, Recall; C, F1-Score; D, False alarm rate. DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; FAR, false alarm rate; GRU, gated recurrent unit; LSTM, long short-term memory; RNN, recurrent neural network [Color figure can be viewed at wileyonlinelibrary.com]

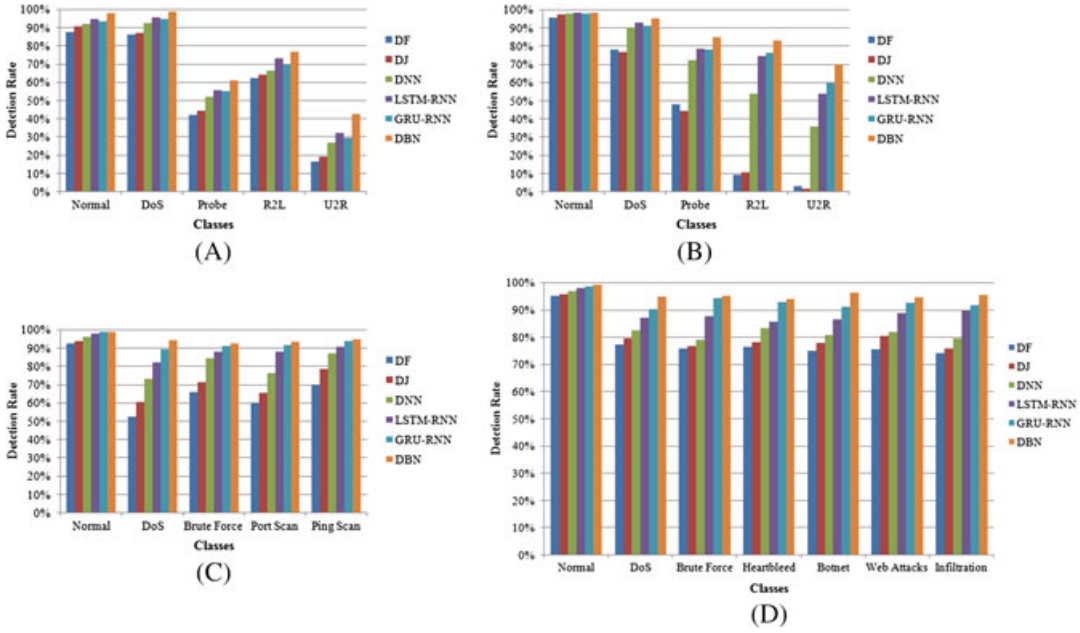

FIGURE 8 Detection rate of models for each class in the dataset. A, KDD CUP 99; B, NSL-KDD; C, CIDDS; D, CICIDS2017. DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; DoS, denial of service; FAR, false alarm rate; GRU, gated recurrent unit; LSTM, long short-term memory; RNN, recurrent neural network; U2R, user to root [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 10 The results of the Friedman test for each outcome

Measurements	FS	FC	P-value	α
TP	18	7.6	0.00044	0.05
FP	18	7.6	0.00044	0.05
TN	18	7.6	0.00044	0.05
FN	17	7.6	0.00071	0.05

Abbreviations: FC, critical Friedman test value; FN, false negative; FP, false positive; FS, calculated Friedman test statistic; TN, true negative; TP, true positive.

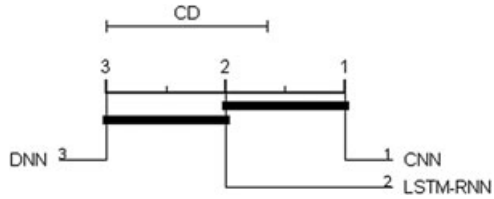

FIGURE 9 The critical difference diagram of the used models on all datasets. CD, critical difference; DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; GRU, gated recurrent unit; LSTM, long short-term memory; RNN, recurrent neural network

that each subject is related to one of the used models. The null hypothesis of the Friedman test is assuming that all treatments have identical effects, that is, there are no differences between the treatments. Mathematically, we can reject the null hypothesis if and only if the calculated Friedman test statistic (*FS*) is larger than the critical Friedman test value (*FC*). Table 10 presents the results of the Friedman test for *TP*, *FP*, *TN*, and *FN* measurements. In all tests, we selected the significance level (α) equals 0.05 because it is fairly common.

According to the results in Table 10, we rejected the null hypothesis of the Friedman test in all cases, because $FS > FC$ and the calculated probability (*P-value*) is less than the selected significance level. Therefore, it can be concluded that the scores of the used models for each measurement are significantly different from each other. In order to interpret the results of the Friedman test visually, we also plotted the critical difference (CD) diagram.[98] Figure 9 shows the CD diagram of the used models on all datasets. Finally, we got the CD value equals 3.7702.

6.3 | Ranking methods

The ranking methods are often utilized to evaluate the performance of different machine learning algorithms as well as to show the trade-off between them to choose the optimal classifier. The main merits of the ranking methods are enabling visualization or summarization performance over the classifier's full operating range and highlighting the information related to skew of the data.[85] In this paper, we selected two widely used ranking methods, namely, receiver operating characteristic (ROC) curves and precision-recall (PR) curves. The major difference between them is that PR curves give a more informative picture of the model's performance when dealing with imbalanced datasets.[99] In other words, PR curves are better for class imbalance problem. Otherwise, ROC curves are better for balanced datasets.

At the outset, an ROC curve is a plot of the true positive rate (or Recall) as a function of the false positive rate (or FAR) of a classifier.[100] The diagonal line of ROC is considered to be the reference line, which means 50% of the performance is achieved. The top-left corner of ROC refers to the best performance with 100%. In this study, we have two balanced datasets, namely, CIDDS and CICIDS2017 datasets. Hence, we have performed the ROC curves analysis of the average performance of each of the used models on the CIDDS and CICIDS2017 datasets. The resulting ROC curves are plotted in Figure 10.

The area under the ROC curve (AUC-ROC or simply AUROC) is a well-known measure to compare quantitatively between various ROC curves.[101] The AUROC value summarizes the corresponding ROC curve into a single value between 0 and 1. The perfect classifier will have AUROC value equals 1. Table 11 presents AUROC values of ROC curves, which are plotted in Figure 10.

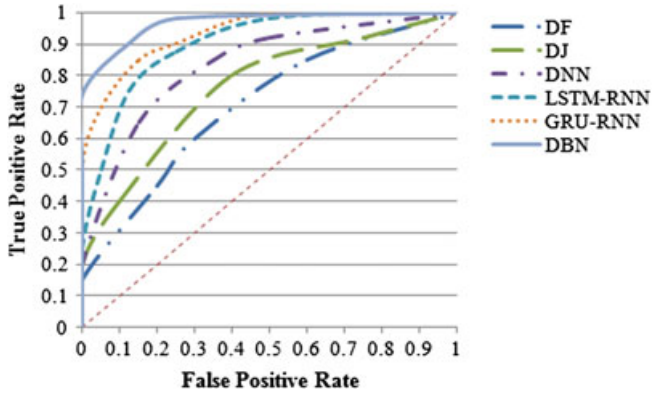

FIGURE 10 Receiver operating characteristic curves of the average performance of the used models over the balanced datasets. DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; GRU, gated recurrent unit; LSTM, long short-term memory; RNN, recurrent neural network [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 11 AUROC values of ROC curves

Model	AUROC
DF	0.813
DJ	0.8416
DNN	0.8857
LSTM-RNN	0.9266
GRU-RNN	0.947
DBN	0.969

Abbreviations: AUROC, area under the receiver operating characteristic curve; DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; GRU; gated recurrent unit; LSTM, long short-term memory; ROC, receiver operating characteristic; RNN, recurrent neural network.

A PR curve is a plot of the precision on *Y*-axis versus the recall on *X*-axis. The goal is to have a model be at the upper-right corner, which is basically getting only the true positives with no false positives and no false negatives.[99] In this study, we have two imbalanced datasets, namely, KDD CUP 99 and NSL-KDD datasets. Thus, we have performed the PR curves analysis of the average performance of each of the used models on the KDD CUP 99 and NSL-KDD datasets. Figure 11 depicts the resulting PR curves.

Furthermore, the area under the PR curve (AUC-PR or simply AUPR) is just the area under the PR curve of a model.[102] Obviously, the AUPR value is in the range of [0,1] and the ideal classifier has AUPR value equals 1. Table 12 presents AUPR values of PR curves, which are plotted in Figure 11. Moreover, ROC and PR curves show that models in the order DBN, LSTM-RNN, GRU-RNN, DNN, DJ, and DF have the effective network intrusion detection performance over all datasets. However, all used models still have a pretty good fit.

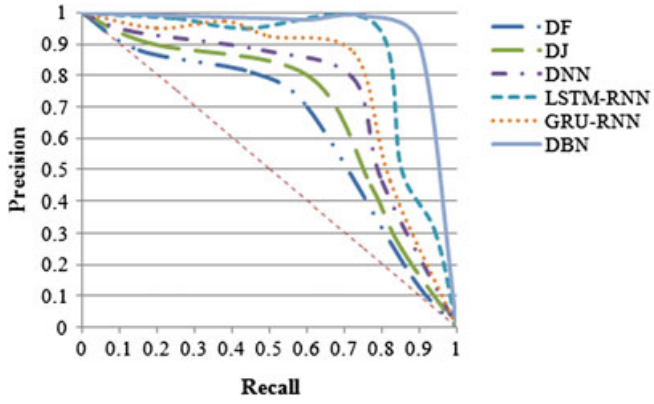

FIGURE 11 Precision-recall curves of the average performance of the used models over the imbalanced datasets. DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; GRU, gated recurrent unit; LSTM, long short-term memory; RNN, recurrent neural network [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 12 AUPR values of PR curves

Model	AUPR
DF	0.8136
DJ	0.826
DNN	0.901
LSTM-RNN	0.945
GRU-RNN	0.9272
DBN	0.9638

Abbreviations: AUPR, area under the precision-recall curve; DBN, deep belief network; DF, decision forest; DJ, decision jungle; DNN, deep neural network; GRU; gated recurrent unit; LSTM, long short-term memory; PR, precision-recall; RNN, recurrent neural network.

7 | CONCLUSIONS

Network intrusion detection is the cornerstone of the cybersecurity domain. Although dozens of studies have been researched in network intrusion detection recently, the existence of a deep study in that subject is seldom. In this paper, we have presented an extensive empirical study for network intrusion detection using the following deep learning models: DNN, LSTM-RNN, GRU-RNN, and DBN. Our motivation was to create an efficient NIDS in perspective of multi-class classification, that is, to distinguish between normal and various malicious connections. To achieve this goal, we have exploited a PSO-based algorithm for automatic selection of optimal hyperparameters of the model. The former process is accomplished prior to the training phase of the models. In addition to that, we have selected two shallow learning methods, namely, decision forest and decision jungle, to be compared with deep learning models. The shallow learning models were exposed directly to datasets without the pretraining phase. In order to examine the models in sundry environments, we utilized four NIDS datasets, namely, KDD CUP 99, NSL-KDD, CIDDS, and CICIDS2017. The first two of them are considered as benchmarks, whereas the latter two are the most up-to-date reliable NIDS datasets. Furthermore, at the end of our experiments, we have employed 22 evaluation metrics to assess the performance of the used models on each of the datasets. To give a clearer view about the performance of the models, we have analyzed the experimental results using performance analysis, Friedman statistical test, and two ranking methods, namely, ROC and PR curves. Our results have showed that the used deep learning models performed achievement in network intrusion detection as well as outperformed the performance of the used shallow learning methods in terms of all evaluation metrics. Moreover, according to our results, the DBN model is superior to other deep learning models. However, the results analyses have proved the effectiveness of all deep learning models in network intrusion detection in such a way that they enhanced the accuracy and DR by 5% to 10% as well as decreased the FAR by 1% to 5% in most cases.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this paper.

FUNDING STATEMENT

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

ORCID

Wisam Elmasry https://orcid.org/0000-0002-0234-4099

REFERENCES

1. Mahmood T, Afzal U. Security analytics: big data analytics for cybersecurity: a review of trends, techniques and tools. In: Proceedings of the 2013 2nd National Conference on Information Assurance (NCIA); 2013; Rawalpindi, Pakistan.
2. Tang TA, Mhamdi L, McLernon D, Zaidi SAR, Ghogho M. Deep learning approach for network intrusion detection in software defined networking. In: Proceedings of the 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM); 2016; Fez, Morocco.

3. Dong B, Wang X. Comparison deep learning method to traditional methods using for network intrusion detection. In: Proceedings of the 2016 8th IEEE International Conference on Communication Software and Networks (ICCSN); 2016; Beijing, China.
4. Deng L. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Trans Signal Inf Process*. 2014;3.
5. Du X, Cai Y, Wang S, Zhang L. Overview of deep learning. In: Proceedings of the 31st Youth Academic Annual Conference of Chinese Association of Automation; 2016; Wuhan, China.
6. Ghorbani AA, Lu W, Tavallaee M. *Network Intrusion Detection and Prevention: Concepts and Techniques*; vol 47. Berlin, Germany: Springer Science and Business Media; 2009.
7. Gharib A, Sharafaldin I, Lashkari AH, Ghorbani AA. An evaluation framework for intrusion detection dataset. In: Proceedings of the 2016 International Conference on Information Science and Security (ICISS); 2016; Pattaya, Thailand.
8. MIT Lincoln Laboratory. DARPA intrusion detection evaluation dataset. https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-data-set
9. Stolfo SJ, Fan W, Lee W, Prodromidies A, Chan PK. Cost-based modeling for fraud and intrusion detection: Results from the JAM project. Paper presented at: DARPA Information Survivability Conference and Exposition; 2000; Hilton Head, SC.
10. T. S. Group. Defcon 8, 10 and 11. 2000. http://cctf.shmoo.com/
11. CAIDA data set OC48 link a (San Jose, CA). 2002. https://www.caida.org/data/passive/passive_oc48_dataset.xml
12. Nechaev B, Allman M. Lawrence Berkeley National Laboratory (LBNL)/ICSI Enterprise Tracing Project; 2004.
13. Sangster B, O'Connor TJ, Cook T, et al. Toward instrumenting network warfare competitions to generate labeled datasets. In: Proceedings of the 2nd conference on Cyber security experimentation and test (CSET'09); 2009; Montreal, Canada.
14. Song J, Takakura H, Okabe Y, Eto M, Inoue D, Nakao K. Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation. In: Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security; 2011; Salzburg, Austria.
15. Sperotto A, Sadre R, Van Vliet F, Pras A. A labeled data set for flow-based intrusion detection. In: Proceedings of the International Workshop on IP Operations and Management; 2009; Venice, Italy.
16. University of Massachusetts Amherst. Optimistic tcp acking. 2011. http://traces.cs.umass.edu/
17. Shiravi A, Shiravi H, Tavallaee M, Ghorbani AA. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput Secur*. 2012;31(3):357-374.
18. Creech G, Hu J. Generation of a new IDS test dataset: time to retire the KDD collection. In: Proceedings of the 2013 IEEE Wireless Communications and Networking Conference (WCNC); 2013; Shanghai, China.
19. McHugh J. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by Lincoln laboratory. *ACM Trans Inf Syst Secur*. 2000;3(4):262-294.
20. UCI Machine Learning Repository. KDD CUP 1999 data set. https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data
21. Adil SH, Ali SSA, Raza K, Hussaan AM. An improved intrusion detection approach using synthetic minority over-sampling technique and deep belief network. In: *New Trends in Software Methodologies, Tools and Techniques: Proceedings of the Thirteenth SoMeT_14*. Amsterdam, The Netherlands: IOS Press; 2014;94-102.
22. Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the KDD CUP 99 data set. In: Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications; 2009; Ottawa, Canada.
23. NSL-KDD dataset. https://github.com/defcom17/NSL_KDD
24. CIDDS-Coburg intrusion detection data sets. https://www.hs-coburg.de/index.php?id=927
25. Ring M, Wunderlich S, Grüdl D, Landes D, Hotho A. Flow-based benchmark data sets for intrusion detection. In: Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS); 2017; Dublin, Ireland.
26. Ring M, Wunderlich S, Grüdl D, Landes D, Hotho A. Creation of flow-based data sets for intrusion detection. *J Inf Warf*. 2017;16(4):41-54.
27. Canadian Institute for Cybersecurity - IDS. 2017. http://www.unb.ca/cic/datasets/ids-2017.html

28. Sharafaldin I, Lashkari AH, Ghorbani AA. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP); 2018; Funchal, Purtogal.
29. Sharafaldin I, Gharib A, Lashkari AH, Ghorbani AA. Towards a reliable intrusion detection benchmark dataset. *Software Networking*. 2017;2017(1):177-200.
30. Sommer R, Paxson V. Outside the closed world: on using machine learning for network intrusion detection. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP); 2010; Berkeley/Oakland, CA.
31. Pietraszek T, Tanner A. Data mining and machine learning—towards reducing false positives in intrusion detection. *Inf Secur Tech Rep*. 2005;10(3):169-183.
32. García-Teodoro P, Díaz-Verdejo J, Maciá-Fernández G, Vázquez E. Anomaly-based network intrusion detection: techniques, systems and challenges. *Comput Secur*. 2009;28(1-2):18-28.
33. Natesan P, Balasubramanie P. Multi stage filter using enhanced adaboost for network intrusion detection. *Int J Netw Secur Appl*. 2012;4(3):121.
34. Dhanabal L, Shantharajah SP. A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. *Int J Adv Res Comput Commun Eng*. 2013;4(6):446-452.
35. Meena G, Choudhary RR. A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA. In: Proceedings of the 2017 International Conference on Computer, Communications and Electronics (Comptelix); 2017; Jaipur, India.
36. Botes FH, Leenen L, De La Harpe R. Ant colony induced decision trees for intrusion detection. In: Proceedings of the ECCWS 2017 16th European Conference on Cyber Warfare and Security; 2017; Dublin, Ireland. p. 53.
37. Depren O, Topallar M, Anarim E, Ciliz MK. An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Syst Appl*. 2005;29(4):713-722.
38. Panda M, Abraham A, Patra MR. A hybrid intelligent approach for network intrusion detection. *Procedia Engineering*. 2012;30:1-9.
39. Zhou G-H. An effective distance-computing method for network anomaly detection. In: Proceedings of the International Conference on Security Technology; 2011; Jeju Island, South Korea. 177-182.
40. Aminanto E, Kim K. Deep learning in intrusion detection system: an overview. In: Proceedings of the 2016 International Research Conference on Engineering and Technology (2016 IRCET), Higher Education Forum; 2016; Kuta, Indonesia.
41. Vani R. Towards efficient intrusion detection using deep learning techniques: a review. *Int J Adv Res Comput Commun Eng*. 2017;6(10):375-384.
42. Aminanto ME, Kim K. Deep learning-based feature selection for intrusion detection system in transport layer. In: Proceedings of the Summer Conference of Korea Information Security Society (CISC-S'16); 2016; Seoul, South Korea.
43. Javaid A, Niyaz Q, Sun W, Alam M. A deep learning approach for network intrusion detection system. In: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS); 2016; New York, NY.
44. Yu Y, Long J, Cai Z. Session-based network intrusion detection using a deep learning architecture. In: *Modeling Decisions for Artificial Intelligence*. Berlin, Germany: Springer; 2017:144-155.
45. Shone N, Ngoc TN, Phai VD, Shi Q. A deep learning approach to network intrusion detection. *IEEE Trans Emerg Top Comput Intell*. 2018;2(1):41-50.
46. Roy SS, Mallik A, Gulati R, Obaidat MS, Krishna PV. A deep learning based artificial neural network approach for intrusion detection. In: Proceedings of the International Conference on Mathematics and Computing; 2017; Haldia, India.
47. Kim J, Shin N, Jo SY, Kim SH. Method of intrusion detection using deep neural network. In: Proceedings of the 2017 IEEE International Conference on Big Data and Smart Computing (BigComp); 2017; Jeju, South Korea.
48. Potluri S, Diedrich C. Accelerated deep neural networks for enhanced intrusion detection system. In: Proceedings of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA); 2016; Berlin, Germany.
49. Kim J, Kim J, Thu HLT, Kim H. Long short term memory recurrent neural network classifier for intrusion detection. In: Proceedings of the 2016 International Conference on Platform Technology and Service (PlatCon); 2016; Jeju, South Korea.

50. Yin C, Zhu Y, Fei J, He X. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*. 2017;5:21954-21961.
51. Ponkarthika M, Saraswathy VR. Network intrusion detection using deep neural networks. *Asian J Appl Sci Technol*. 2018;2(2):665-673.
52. Kim G, Yi H, Lee J, Paek Y, Yoon S. LSTM-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems. 2016. arXiv preprint arXiv:1611.01726.
53. Fiore U, Palmieri F, Castiglione A, De Santis A. Network anomaly detection with the restricted Boltzmann machine. *Neurocomputing*. 2013;122:13-23.
54. Gao N, Gao L, Gao Q, Wang H. An intrusion detection model based on deep belief networks. In: Proceedings of the 2014 Second International Conference on Advanced Cloud and Big Data (CBD); Huangshan, China.
55. Alom Z, Bontupalli VR, Taha TM. Intrusion detection using deep belief networks. In: Proceedings of the 2015 National Aerospace and Electronics Conference (NAECON); 2015; Dayton, OH.
56. Liu Y, Zhang X. Intrusion detection based on IDBM. In: Proceedings of the 2016 IEEE 14th International Conference on Dependable, Autonomic and Secure Computing, 14th Intl Conference on Pervasive Intelligence and Computing, 2nd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress; 2016; Auckland, New Zealand.
57. Alrawashdeh K, Purdy C. Toward an online anomaly intrusion detection system based on deep learning. In: Proceedings of the 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA); 2016; Anaheim, CA.
58. Ludwig SA. Intrusion detection of multiple attack classes using a deep neural net ensemble. In: Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (SSCI); 2017; Honolulu, HI.
59. Salama MA, Eid HF, Ramadan RA, Darwish A, Hassanien AE. Hybrid intelligent intrusion detection scheme. In: *Soft Computing in Industrial Applications*. Berlin, Germany: Springer; 2011:293-303.
60. Menon DM, Radhika N. A secure deep belief network architecture for intrusion detection in smart grid home area network. *IIOAB Journal*. 2016;7:479-483.
61. Alom Z, Bontupalli VR, Taha TM. Intrusion detection using deep belief network and extreme learning machine. *Int J Monit Surveillance Technol Res*. 2015;3(2):35-56.
62. Li Y, Ma R, Jiao R. A hybrid malicious code detection method based on deep learning. *Int J Secur Appl*. 2015;9(5):205-216.
63. Qu F, Zhang J, Shao Z, Qi S. An intrusion detection model based on deep belief network. In: Proceedings of the 2017 VI International Conference on Network, Communication and Computing; 2017; Kunming, China.
64. Microsoft Azure Machine Learning Studio. 2018. https://studio.azureml.net/
65. Barga R, Fontama V, Tok WH. Introducing microsoft azure machine learning. In: *Predictive Analytics with Microsoft Azure Machine Learning*. New York, NY: Springer; 2015:21-43.
66. Rokach L. Decision forest: twenty years of research. *Information Fusion*. 2016;27:111-125.
67. Shotton J, Sharp T, Kohli P, Nowozin S, Winn J, Criminisi A. Decision jungles: compact and rich models for classification. *Adv Neural Inf Process Syst*. 2013;234-242.
68. Elmasry W, Akbulut A, Zaim AH. Deep learning approaches for predictive masquerade detection. *Secur Commun Netw*. 2018;2018. Article ID 9327215. https://doi.org/10.1155/2018/9327215
69. Shi Y, Eberhart RC. Parameter selection in particle swarm optimization. In: Proceedings of the International Conference on Evolutionary Programming; 1998; San Diego, CA.
70. Clerc M, Kennedy J. The particle swarm: explosion, stability and convergence in a multidimensional complex space. *IEEE Trans Evol Comput*. 2002;6(1):58-73.
71. Hinton G, Deng L, Yu D, et al. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Process Mag*. 2012;29(6):82-97.
72. Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans Neural Netw*. 1994;5(2):157-166.
73. Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Computation*. 1997;9(8):1735-1780.
74. Cho K, van Merrienboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder–decoder for statistical machine translation. 2014. arXiv preprint arXiv:1406.1078.
75. Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014. arXiv preprint arXiv:1412.3555.

76. Salakhutdinov R, Hinton GE. Deep Boltzmann machines. In: Proceedings of the 12th International Conference on Artificial Intelligence and Statistics; 2009; Clearwater Beach, FL.
77. Hinton GE, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. *Neural Computation*. 2006;18(7):1527-1554.
78. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. *Nature*. 1986;323:533-536.
79. Hinton GE. Training products of experts by minimizing contrastive divergence. *Neural Computation*. 2002;14(8):1771-1800.
80. Swets JA. Measuring the accuracy of diagnostic systems. *Science*. 1988;240(4857):1285-1293.
81. Sokolova M, Lapalme G. A systematic analysis of performance measures for classification tasks. *Inf Process Manag*. 2009;45(4):427-437.
82. Axelsson S. The base-rate fallacy and its implications for the difficulty of intrusion detection. In: Proceedings of the 6th ACM Conference on Computer and Communications Security; 1999; Singapore.
83. Zeng Z, Gao J. Improving SVM classification with imbalance data set. In: Proceedings of the International Conference on Neural Information Processing; 2009; Bangkok, Thailand.
84. Kubat M, Matwin S. Addressing the curse of imbalanced training sets: one-sided selection. In: Proceedings of the 14th International Conference on Machine Learning (ICML), Vol. 97; 1997; Nashville, TN.
85. He H, Ma Y. *Imbalanced Learning: Foundations, Algorithms, and Applications*. Hoboken, NJ; John Wiley & Sons; 2013.
86. Boughorbel S, Jarray F, El-Anbari M. Optimal classifier for imbalanced data using Matthews correlation coefficient metric. *PLOS ONE*. 2017;12(6):e0177678.
87. Matthews BW. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica Biophys Acta Protein Struct*. 1975;405(2):442-451.
88. Elhamahmy ME, Elmahdy HN, Saroit IA. A new approach for evaluating intrusion detection system. *CiiT Int J Artif Intell Syst Mach Learn*. 2010;2(11):290-298.
89. Python. https://www.python.org
90. NumPy. http://www.numpy.org
91. Chollet F. Keras. 2015. https://github.com/fchollet/keras
92. Keras. https://keras.io
93. Abadi M, Agarwal A, Barham P, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016. arXiv preprint arXiv:1603.04467.
94. TensorFlow. https://www.tensorflow.org
95. CUDA- Compute Unified Device Architecture. https://developer.nvidia.com/about-cuda
96. cuDNN- The NVIDIA CUDA Deep Neural Network Library. https://developer.nvidia.com/cudnn
97. Daniel WW. Friedman two-way analysis of variance by ranks. *Appl Nonparametric Stat*. 1990:262-274.
98. Demsar J. Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res*. 2006;7:1-30.
99. Davis J, Goadrich M. The relationship between precision-recall and ROC curves. In: Proceedings of the 23rd International Conference on Machine Learning; 2006; Pittsburgh, PA.
100. Fawcett T. An introduction to ROC analysis. *Pattern Recognit Lett*. 2006;27(8):861-874.
101. Bradley AP. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*. 1997;30(7):1145-1159.
102. Landgrebe TCW, Paclik P, Duin RPW. Precision-recall operating characteristic (P-ROC) curves in imprecise environments. In: Proceedings of the 18th International Conference on Pattern Recognition, Vol. 4; 2006; Hong Kong.

How to cite this article: Elmasry W, Akbulut A, Zaim AH. Empirical study on multiclass classification-based network intrusion detection. *Computational Intelligence*. 2019;1–36. https://doi.org/10.1111/coin.12220