Received: 17 May 2016 | Revised: 20 February 2020 | Accepted: 4 March 2020

DOI: 10.1002/dac.4401

RESEARCH ARTICLE

WILEY

A deep learning approach with Bayesian optimization and ensemble classifiers for detecting denial of service attacks

Yasin Gormez[1] | **Zafer Aydin**[1] | **Ramazan Karademir**[2] | **Vehbi C. Gungor**[1]

[1]Department of Computer Engineering, Abdullah Gul University, Kayseri, Turkey

[2]Information Technologie, Enterprise Solutions Department, Digiturk, Besiktas, Istanbul, Turkey

Correspondence
Vehbi C. Gungor, Department of Computer Engineering, Abdullah Gul University, Kayseri, Turkey.
Email: cagri.gungor@agu.edu.tr

Summary

Detecting malicious behavior is important for preventing security threats in a computer network. Denial of Service (DoS) is among the popular cyber attacks targeted at web sites of high-profile organizations and can potentially have high economic and time costs. In this paper, several machine learning methods including ensemble models and autoencoder-based deep learning classifiers are compared and tuned using Bayesian optimization. The autoencoder framework enables to extract new features by mapping the original input to a new space. The methods are trained and tested both for binary and multi-class classification on Digiturk and Labris datasets, which were introduced recently for detecting various types of DDoS attacks. The best performing methods are found to be ensembles though deep learning classifiers achieved comparable level of accuracy.

KEYWORDS

autoencoder, deep learning, denial of service attacks, machine learning, network anomaly detection

1 | INTRODUCTION

The huge growth of computer networks and their accessibility via variety of devices including PCs and mobile devices has increased the number of applications running on computer network platforms especially the internet. As a result, the security of computer networks has gained considerable importance. Security vulnerabilities can be both economically costly and technically difficult to resolve by the companies that rely on computer systems.

Computer intrusion is a collection of actions that violate the security of a computer system. To prevent such threats, Intrusion Detection Systems (IDS) have been proposed, which detect anomalies and attacks in a computer network. An IDS typically contains a predictive model (i.e., a classifier) that has the capability of differentiating attacks from normal packets. It can be limited to learn a binary classification problem, which discriminates attacks from normal patterns only or a multi-class problem, which deals with the classification of different attack types and those that do not pose any threat to the network. In all cases, the goal is to develop a classifier that will make accurate predictions on unseen instances.

There are different types of attacks that may target computer networks including denial of service (DoS), man-in-the-middle (MitM), phishing, drive-by, password, SQL injection, cross-site, scripting (XSS), eavesdropping, birthday, and malware.[1] Among those DoS attacks is a popular category in which the goal is to overwhelm the network by sending high volume of traffic so that legitimate users are blocked or disabled. When these attacks are generated by many different resources on the internet, this is called a distributed denial of service (DDoS) attack. Although DoS attacks do not aim to steal information of users, they are frequently used in combination with other attack types to distract

attention.[2] Furthermore, DoS attacks are typically targeted at web sites of high-profile organizations such as governments, which can potentially cause high economic and time costs.[3]

In this paper, several methods have been optimized both for binary and multi-class classification of various DDoS attacks including traditional machine learning algorithms, ensemble methods such as stacking, XGBoost, CatBoost, random forest, deep feature extraction methods, such as auto-encoders, stacked deep auto-encoders, and deep learning classifiers based on auto-encoders. The models are trained and tested on two anomaly detection datasets introduced earlier in Gungor et al. [4] The hyper-parameters of the classification algorithms, including deep learning methods, are fine-tuned using the Bayesian optimization method, which is among the main novelties of this work. In other words, none of the deep learning–based methods in the literature employed Bayesian optimization for hyper-parameter tuning. Furthermore, the Digiturk and Labris datasets have been employed for model training and testing, including a total of five and eight types of DDoS attacks, respectively, which is much higher than the DDoS attacks available in public datasets.

The rest of this paper is organized as follows: Section 2 includes related work on DDoS attacks and addresses the research challenges in anomaly detection for DDoS attacks. Section 3 describes the datasets, accuracy metrics, and classification methods. Section 4 presents experiments and performance evaluation results. Finally, the paper is concluded in Section 5.

2 | RELATED WORK

Various methods have been proposed in the literature for network anomaly detection including standard machine learning classifiers[4–29] and deep learning techniques.[30–47] Muda et al. performed clustering before classification and compared the single classifiers with hybrid classifiers. They showed that the hybrid classifier, which uses Naive Bayes and K-means, obtained the best accuracy with 99.6% on KDDCup99 dataset.[8] Li and Liu compared the non-normalized version of the KDDCup99 dataset with the normalized versions, which are generated using different normalization methods, and they showed that the dataset that is generated using min–max normalization obtained the best accuracy of 99.93%.[9] Ashok et al. obtained 99.90% detection rate using KDDCup99 dataset with the proposed method called k-means Cluster Triangular Area Based Support Vector Machine.[10] Horng et al. obtained 95.72% accuracy using KDDCup99 dataset with the proposed method that uses support vector machines and balanced iterative reducing and clustering using hierarchies.[11] Om and Kundu applied the k-means clustering algorithm followed by the ensemble classifier which uses k-Nearest Neighbor and Naïve Bayes and obtained 99% accuracy on KDDCup99 dataset.[12] Chen and Kim proposed an ensemble method, which uses principal component analysis, decision tree, and Naïve Bayes, and obtained 94.06%, 95.52%, 99.97%, 77.78%, and 77.20% detection rates for attack types Normal, Probe, DOS, U2R, and R2L, respectively, on KDDCup99 dataset.[15] Kim et al. not only obtained 99% detection rate but also reduced the time complexity using NSL-KDD dataset with proposed hybrid intrusion detection method that hierarchically integrates a misuse detection model and an anomaly detection model in a decomposition structure.[16] Kevric et al. developed ensemble classifiers using tree-based algorithms and obtained 89.24% detection rate using NSL-KDD dataset.[19] Gharaee and Hossein developed genetic based support vector machines and obtained 99.05%, 99.95%, 99.06%, 98.25%, and 100% accuracy for attack types Normal, Probe, DOS, U2R, and R2L, respectively, using KDDCup99 dataset.[20] Guha et al. developed an ensemble method that combines artificial neural networks with genetic algorithm and obtained 91.98% accuracy rate on two large datasets.[21] Wheelus et al. showed that the pre-processing of datasets to address class imbalance does indeed provide some benefit using UNSW-NB15 dataset.[23] Pattawaro and Polprasert obtained 86.36% detection rate with the proposed hybrid model that uses combination of feature selection, K-means and XGBoost methods using NSL-KDD dataset.[24] Mirza developed an ensemble method that combines logistic regression, neural networks, and decision trees and obtained 96.14% detection rate using KDDCup99 dataset.[25] Dhaliwal et al. developed several XGBoost models and obtained 98.70% accuracy on NSL-KDD dataset.[26] Dahiya and Srivastava compared two dimension reduction algorithms such as canonical correlation analysis and linear discriminant analysis using several classification algorithms and obtained at most 95.53% accuracy rate on UNSW-NB dataset using canonical correlation analysis with bagging.[27] Verma et al. compared several boosting algorithms using NSL-KDD dataset, and they reached 99.86% accuracy rate using XGBoost with K-means clustering.[28] Alrawashdeh and Khaled implemented deep restricted Boltzmann machine and deep belief network and obtained 97.9% detection rate using KDDCup99 dataset.[32] Kim et al. implemented long short-term memory recurrent neural network using KDDCup99 dataset and obtained 96.93% accuracy.[34] Kaynar et al. obtained 99.42% accuracy with a method that uses stacked autoencoder followed by the softmax classification layer on KDDCup99 dataset.[35] Van et al. implemented a deep belief network on KDDCup99 dataset and

obtained 98% accuracy.[36] Aygun and Yavuz implemented a deep autoencoder on KDDTest+ dataset and obtained 88.65% accuracy.[37] Kim et al. implemented a deep neural network on KDDCup99 dataset and obtained 99% detection rate.[38] Vinayakumar et al. compared deep belief network with the traditional classification algorithms using KDDCup99 and NSL-KDD datasets and showed that the best accuracies are obtained by deep belief networks as 93.3% and 81.7%, respectively.[36] Yousefi-azar et al. obtained 83.34% accuracy using KDDTest+ dataset with a model that employs an autoencoder.[40] Shone et al. proposed nonsymmetric deep autoencoder, and they obtained 97.85% and 85.42 accuracy for KDDCup99 and NSL-KDD datasets, respectively.[42] Zong et al. proposed Deep Autoencoding Gaussian Mixture Model and obtained 93.69%, 47.82% 49.83%, and 93.80% F1-score using KDDCup99, Thyroid, Arrhythmia, and KDD Cup-Rev datasets, respectively.[43] Mirza and Cosan proposed a sequential autoencoder framework using long short-term memory and obtained 95.12 AUC score.[44] Çekmez et al. proposed a new autoencoder model and obtained 91.3% and 85.5% accuracy for KDDTest+ and KDDTest21 datasets, respectively.[45] Zhang et al. obtained 91.97% F1 score with the proposed XGBoost based on stacked sparse autoencoder network using NSL-KDD dataset.[46] Abdulhammed et al. applied sampling methods on CIDDS-001 datasets to generate a balanced dataset and obtained 99.99% accuracy with deep neural networks.[47]

Most of the studies have been conducted using common benchmark datasets such as KDDCup99[48] and UNSW-NB15.[49] However, these benchmarks contain only one class label for DoS attacks. Similarly, in other benchmarks, there are only a limited number of DDoS attacks.[50,51] On the other hand, it is known that many other types of DDoS attacks exist and are produced in the course of time.[51,52] The present work therefore aims to fill this gap by analyzing a rich set of DDoS attacks.

It could be of interest to compare the two related publications of the authors. In Gungor et al., [4] various feature selection and classifier methods are implemented. In the current study, various classifiers and a deep learning method, which are not employed in Gungor et al., [4] have been implemented. Furthermore, while feature selection is not implemented in the present work, hyper-parameter tuning with Bayesian optimization is not implemented in Gungor et al. [4]

A successful anomaly detection system for denial of service attacks must consider the following challenges:

- **IDS must operate in near real-time speeds:** The nature of the current complex and high-speed networking environment makes the attack detection task very difficult. On heavy load conditions, capturing and processing network packets become a tough job. Any packet loss can affect adversely the overall system performance.[53]
- **Enormous attack diversity:** The current diversity of known attacks is very high, and various new types of attacks are emerging continuously. Keeping up with this dynamic environment requires constant work by researchers.[52]
- **Encrypted data:** Encryption of packet payloads constitutes a barrier for deep packet inspection.[54]
- **Evaluation dataset and feature extraction:** KDDCUP'99 dataset has been prepared by MIT Lincoln Labs in order to evaluate and compare intrusion detection methods. However, these data are rather old, and some of its attacks are out of date. In addition, the dataset contains redundant records, which can cause a learning algorithm to get biased towards the more frequent records.[6] One can prepare new data and generate intrusions, but then it will be limited to the known attacks available. It is a challenging task to extract significant features and classify network data manually given that huge amount of network traffic data.

3 | METHODS

3.1 | Datasets

The experiments in this work are conducted on Labris and Digiturk (also known as Ligtv.com.tr) datasets, which are generated by the Labris and Digiturk companies, respectively, by monitoring traffic in several countries. The DDos attacks in Digiturk dataset are generated using hping utility using random IP source generation option and those in Labris dataset in a lab environment by the Labris company. Captured network traffic packet data were transformed and summarized into connection records. The feature construction process includes transformation and summarization steps which are achieved with custom written stored procedures of Microsoft SQL Server. Features can be grouped into three categories as basic features, time-based features, and connection-based features. Basic features contain features that can be extracted easily from packet headers by counting certain properties of packets for the connection. Time-based features are calculated using a time window parameter of 2 s that includes the current connection and

connections that started within 2 s. Connection-based features are calculated using a window parameter of 200 connections that includes the current connection and the previous 200 connections. Finally, the labeling and attack type specification are performed for all records by writing several queries and searching attack characteristics for the dataset. Details of dataset preparation and pre-processing steps can be found in the paper by Gungor et al. [4] Both sets contain a total of 41 input features with some similarities and differences as compared to KDDCup99 dataset. To be precise, the following 14 features in Labris and Digiturk datasets are the same as in KDDCup99: duration, protocol, network_service, src_bytes, dst_bytes, tw_shConnectionCount, tw_shSYNErrorRate, tw_shResetRate, tw_shSameServiceRate, tw_shDiffServiceRate, tw_ssConnectionCount, tw_ssSYNErrorRate, tw_ssResetRate, and tw_ssDiffHostRate. In addition, the following nine features are also the same as KDDCup99 but are computed using a connection-window approach instead of a time-based approach in which the current connection and the past 200 connections are used: cw_shConnectionCount, cw_shSYNErrorRate, cw_shResetRate, cw_shSameServiceRate, cw_shDiffServiceRate, cw_ssConnectionCount, cw_ssSYNErrorRate, cw_ssResetRate, and cw_ssDiffHostRate. The remaining 18 features are not found in the KDDCup99 dataset. The detailed description of all features can be found on the Supplementary document.

A train and a test set are generated from Digiturk and Labris datasets by random sampling. Tables 1 (Ligtv.com.tr) and 2 (labrisnetworks.com) summarize the number of samples and the distribution of class labels in train and test sets. From each train set, 10% of the samples are selected using stratified random sampling as the validation set, and the remaining set of samples are called "train set for optimization." The validation set and train set for optimization are used for optimizing the hyper-parameters of the models using the Bayesian optimization procedure.

The attack types in Digiturk dataset are FIN_attack, FragmentSet, RST_attack, SYN_attack, and SYN_RST. These are explained below.

TABLE 1 Train and test set statistics for Digiturk (Ligtv.com.tr) dataset

	Training set		Test set	
Labels	**# instances**	**%**	**# instances**	**%**
Normal	96,408	52.164	64,957	52.404
FIN_attack	7,254	8.481	4,914	8.628
FragmentSet	19,573	22.885	13,123	23.041
RST_attack	752	0.879	465	0.816
SYN_attack	39,967	46.73	26,684	46.852
SYN_RST	20,864	24.394	13,811	24.249
Total	184,818		123,954	

TABLE 2 Train and test set statistics for Labris dataset

	Training set		Test set	
Labels	**# instances**	**%**	**# instances**	**%**
Normal	77,693	90.839	51,760	90.88
syn_ack_ddos	706	0.825	488	0.857
icmp_ddos	20	0.023	18	0.032
rst_ack_ddos	1,731	2.024	1,126	1.977
rst_ddos	1,135	1.327	755	1.326
fin_ddos	11	0.013	6	0.011
ack_ddos	498	0.582	327	0.574
http_get	1,857	2.171	1,192	2.093
syn_ddos	1,877	2.195	1,282	2.251
Total	85,528		56,954	

- **RST_attack and FIN_attack:** In RST_attack and FIN_attack categories, TCP RST or FIN flag is used to terminate a connection in a three or four way of handshake mechanism of TCP. In this attack, the server receives RST or FIN flood with spoofed IP source addresses or a random sequence number at a very high packet rate. These bogus packets that do not belong to any session cause the server to check its session table for corresponding session thereby resulting in performance degradation by exhausting system resources such as memory or CPU.[55]
- **FragmentSet:** This is a type of IP Fragmentation attack that is based on flooding the network by exploiting the datagram fragmentation mechanisms. IP fragmentation is a communication procedure that breaks down (i.e., partition) messages from one layer of the network into smaller fragments. Every network has a unique limit for the size of the messages that can be transmitted called maximum transmission unit (MTU). If messages (service data unit [SDU]) and meta data added at the link layer exceeds MTU, SDU must be fragmented. IP fragmentation attacks exploit this by using fragmentation protocol within IP as an attack vector. IP fragmentation attacks can have different forms such as UDP/ICMP fragmentation and TCP fragmentation (i.e., teardrop attack). In UDP/ICMP fragmentation, fraudulent UDP or ICMP packets that are larger than MTU are transmitted. In teardrop, TCP/IP reassembly mechanism is prevented from putting together the fragmented data packets.[56,57]
- **SYN_attack:** In SYN flooding, the three-way handshake mechanism of TCP/IP protocol implementation is exploited. According to TCP/IP protocol, connection establishment occurs after three-way handshake mechanism. First, the client sends a SYN packet to the server in order to establish a connection. Then, server responds with a SYN-ACK packet meaning that it accepts the connection request. Finally, the client responds with ACK packet to finish connection establishment. In this state, after a connection is established, client and server can exchange data. Attackers exploit this protocol by sending too many SYN packets to the server causing them to fill up its connection tables. The attacker does not respond to the servers and SYN-ACK packets and server connection table fills up with half-open connections. When the connection table of the server fills up, then no other legitimate users can reach to the server. Normally, a timeout mechanism from TCP clears the half open connections, but the attacker also continues sending fake connection requests and keeping the server busy all the time.[55]
- **SYN_RST:** A combination of SYN and RST attacks are executed.

The attack types in Labris dataset include the following[58]:

- **syn_ack_ddos** is a distributed Denial of Service attack that exploits normal TCP in the form of a three way handshake;
- **icmp_ddos** is known as ping flood, which tries to put servers out of service by request for its response more than the ability of the server;
- **rst_ack_ddos** is known as three-way handshake reset attack, which is used for sending a resetting bit to stop a TCP connection;
- **rst_ddos** is a reset attack but with no handshake (also explained above);
- **fin_ddos** sends a massive amount of TCP packets with fin bit enabled, which makes the server busy dropping the incoming packets;
- **ack_ddos** is a distributed Denial of Service with three-way handshake;
- **http_get** is introduced for attacking web servers;
- **syn_ddos** explained above for Digiturk dataset.

3.2 | Classification methods

3.2.1 | Single classifiers and ensemble methods

The following classification algorithms are implemented using the scikit-library of Python[59]: naïve Bayes,[60] support vector machine,[61] multilayer perceptron,[62] logistic regression,[63] and k-nearest neighbor.[64,65] The multi-layer perceptron contains a single hidden layer with a fixed learning rate schedule. The activation function is selected as ReLU. The weight parameters are initialized using the Le-Cun uniform heuristic[66] and estimated using the Adam optimizer. An RBF kernel is used for the support vector machine. In addition to traditional classifiers, XGBoost,[67] Catboost,[68] random forest,[69] and stacking[70] methods are implemented as ensemble models, which have shown to obtain promising results in recent Kaggle competitions.[71] The stacking ensemble includes the multi-layer perceptron,

logistic regression, k-nearest neighbor, and random forest methods as the base learners and support vector machine as the meta-learner.

3.2.2 | Autoencoder based deep learning classifier

Autoencoder (AE) is a derivate of neural networks that achieved successful results in many fields such as image processing, noise reduction, and dimension reduction.[72] A standard AE has an input, a hidden, and an output layer. The number of neurons in input and output layers are the same and are equal to the number of features in dataset. The number of neurons in the hidden layer of AE can take any integer value, which defines the new number of dimensions. With the increase in popularity of deep learning, deep autoencoder (DAE) models have been developed and employed in many applications recently. A DAE can contain any number of hidden layers with any number of neurons in hidden layers. A standard AE and DAE can be divided into two parts as encoder and decoder. The encoder maps the original dataset into a new space, while the decoder part maps the encoded data back to the original input space with the goal of recovering the original input features. In this paper, an AE and a DAE with three hidden layers are implemented. Figure 1 illustrates the architecture of both models, where each hidden layer consists of a dense layer with batch normalization, activation, and dropout regularization steps.

After training an autoencoder model, the new feature vectors obtained using the encoder section can be processed by subsequent classification algorithms as shown in Figure 2. The deep learning classifier implemented in this work employs a k-NN algorithm as the classifier method after the encoder block of the AE and DAE methods. Note that in Figures 1 and 2, the leftmost blocks of the encoder models represent the input features.

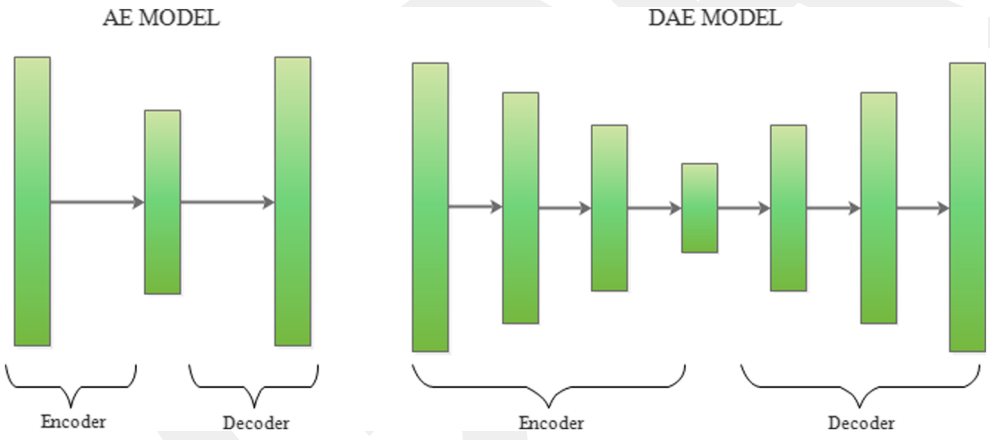

FIGURE 1 Architecture of standard and deep autoencoder models. The length of the bars represents the number of units in input and hidden layers

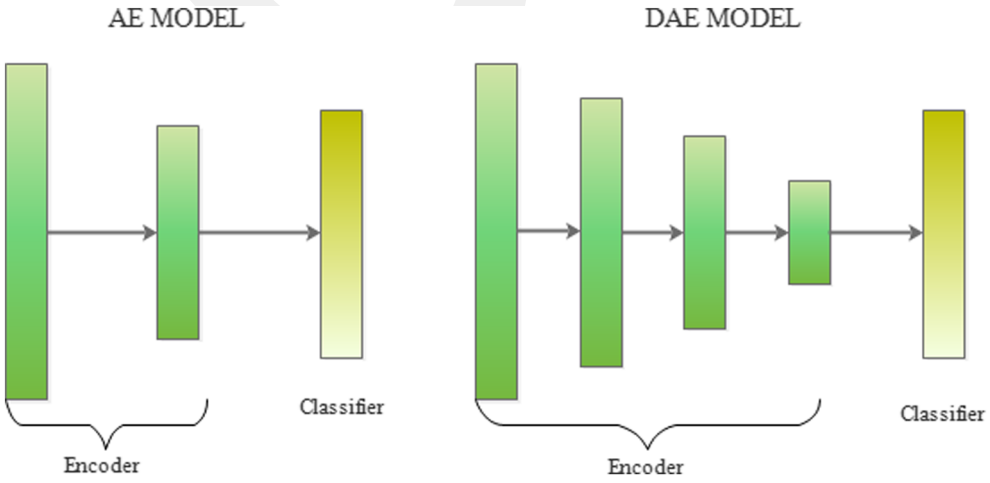

FIGURE 2 Autoencoder-based deep learning classifier. The length of the bars represents the number of units in input and hidden layers

3.2.3 | Hyper-parameter optimization

Hyper-parameter optimization enables to fine tune a model's hyper-parameters before evaluating their performance. To find the optimums, samples of hyper-parameter values are generated using the Bayesian optimization procedure. For each hyper-parameter configuration, prediction models are trained on train set for optimization and tested on validation set (see Section 3.1). The particular parameter configuration that gives the best overall accuracy is selected as the optimum. In the next step, each model is trained using the optimums on the original train set, and predictions are computed on test set to evaluate the accuracy. These steps are summarized in Figure 3.

In this paper, the hyper-parameters of all classifiers are optimized using Bayesian optimization algorithm[73] implemented using Scikit-optimize library of Python.[74] To achieve this, the gp_minimize method of the skopt module is employed using the following command: sr = gp_minimize(run_clf,dimensions = param_grid,acq_func = 'EI', n_calls = 100), where run_clf is the function to minimize, dimensions are set to the parameter grid and are provided as a list containing search space dimensions, acq_func is the function to minimize over the Gaussian prior and is set to negative expected improvement (EI), and n_calls is the number of calls to the function to minimize and is set to 100. All the other parameters of gp_minimize are set to their default values. This method uses a Gaussian process to model the surrogate function and optimizes the expected probability that new trials will improve the current best solution. It assumes the function values to follow a multivariate Gaussian distribution. The covariance of the function values is assigned by a Gaussian process kernel among the parameters. This is followed by selecting the next parameter through the acquisition function over the Gaussian prior.

As compared to grid search, which is a standard technique for hyper-parameter tuning, Bayesian optimization can search in a larger space of hyper-parameters. The method accepts an interval (i.e., minimum and maximum values) for each parameter and can consider any value in that interval. The second advantage is related to the speed. For example, for the same search space and computational resources, Bayesian optimization can take days while grid search may take years to finish.

The following hyper-parameters are optimized in this work: iteration and depth for CatBoost; number of nearest neighbors for k-nearest neighbor; eta, max-depth, and silent for XGBoost; number of neurons, number of epochs, learning rate, and batch size for multi-layer perceptron; number of trees for random forest; C and gamma for support vector machine. The beta1 and beta2 parameters of Adam optimizer are set to 0.95 and 0.99, respectively. The ranges selected for the hyper-parameters that are optimized are tabulated in Table 3.

FIGURE 3 Model fine-tuning, training, and testing steps

TABLE 3 Hyper-parameter ranges of classifiers used for optimization

Classifier	Parameter	Lowest	Highest
k-NN	k	1	100
Random forest	Number of trees	1	200
SVM	Gamma	2^{-20}	2^{20}
MLP, AE, DAE	Learning rate	10^{-8}	10^{-1}
MLP, AE, DAE	Number of hidden units	2	40
MLP, AE, DAE	Batch size	1	1024
MLP, AE, DAE	Number of epochs	1	50
AE, DAE	Dropout rate	0	0.5
CatBoost	Number of iterations	1	40
CatBoost	Depth	1	20
XGBoost	Eta	0.1	1.0
XGBoost	Depth	1	40

3.3 | Accuracy measures

The following accuracy measures are used to evaluate the success of the classifiers: overall accuracy, detection rate (i.e., precision), false alarm rate (i.e., FP-rate), and *F*-measure.[75] In binary classification problem, the samples corresponding to attacks formed the positive class.

4 | RESULTS AND DISCUSSION

4.1 | Classification methods

Tables 4 and 5 include binary classification results for Labris and Digiturk datasets, respectively. Similarly, Tables 6 and 7 contain accuracy measures for multi-class classification on the same datasets. All the rates are given as percentages. The following abbreviations are used for the classifiers: k-nearest neighbor (k-NN), logistic regression (LR), multi-layer perceptron (MLP), naïve Bayes (NB), support vector machine (SVM), random forest (RF), auto encoder (AE), and deep auto encoder (DAE). The best overall accuracy, *F*-measure, detection rate, and false alarm rate are obtained by CatBoost, random forest, and stacking methods in binary classification on Labris dataset achieving perfect separation of attacks from normal traffic. On Digiturk dataset, the best overall accuracy and false alarm rate are obtained by CatBoost, the best F-Measure by stacking and the best detection and false alarm rates

TABLE 4 Binary classification results on Labris dataset

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	99.98	99.96	0.003	99.93
LR	99.63	97.70	0.002	98.00
MLP	99.98	99.86	0.0001	99.99
NB	99.93	99.50	0.050	99.65
SVM	99.77	**100.00**	**0.00**	98.73
CatBoost	**100.00**	**100.00**	**0.00**	**100.00**
XGBoost	99.99	99.99	0.00007	99.99
RF	**100.00**	**100.00**	**0.00**	**100.00**
Stacking	**100.00**	**100.00**	**0.00**	**100.00**
AE + k-NN	99.99	**100.00**	**0.00**	99.98
DAE + k-NN	99.98	**100.00**	**0.00**	99.98

TABLE 5 Binary classification results on Digiturk dataset

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	99.66	99.50	0.004	99.64
LR	98.06	97.58	0.02	97.97
MLP	99.00	98.08	0.017	98.96
NB	98.56	97.50	2.33	98.51
SVM	98.38	99.97	0.0002	98.27
CatBoost	**99.97**	99.50	**0.0001**	99.96
XGBoost	99.92	99.90	0.0009	99.91
RF	99.96	**99.98**	**0.0001**	99.96
Stacking	99.96	99.97	0.0002	**99.99**
AE + k-NN	99.67	99.51	0.004	99.65
DAE + k-NN	99.73	99.63	0.0033	99.71

TABLE 6 Multi-class classification results on Labris dataset

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	98.58	99.95	0.03	91.58
LR	94.96	95.47	0.2	76.92
MLP	97.10	98.71	0.009	81.36
NB	94.36	38.51	5.8	55.43
SVM	98.23	99.99	0.0001	89.93
CatBoost	98.60	**100.00**	**0.00**	91.76
XGBoost	98.57	**100.00**	**0.00**	91.52
RF	98.84	**100.00**	**0.00**	93.23
Stacking	**98.85**	**100.00**	**0.00**	**93.28**
AE + k-NN	98.66	99.88	0.00009	92.13
DAE + k-NN	98.60	99.97	0.00001	91.76

TABLE 7 Multi-class classification results on Digiturk dataset

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	95.40	99.48	0.004	94.90
LR	91.87	96.42	0.029	90.98
MLP	94.90	98.60	0.011	94.41
NB	97.20	94.73	4.83	97.07
SVM	94.31	99.96	0.0002	93.65
CatBoost	99.51	99.96	0.0002	99.49
XGBoost	99.53	99.96	0.0002	99.51
RF	**99.69**	**99.98**	**0.0001**	**99.68**
Stacking	99.67	99.98	**0.0001**	99.65
AE + k-NN	95.65	99.39	0.005	95.25
DAE + k-NN	95.24	99.20	0.006	94.78

TABLE 8 Binary classification results evaluated by 10-fold cross validation experiment on Labris dataset

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	99.98	99.89	0.0001	99.93
LR	99.66	98.44	0.001	98.16
MLP	99.94	99.74	0.0002	99.72
NB	99.91	99.44	0.053	99.59
SVM	99.67	**100**	**0**	98.20
CatBoost	99.99	99.98	0.00001	100
XGBoost	99.99	99.91	0.00009	99.95
RF	**100**	**100**	**0**	**100**
Stacking	**100**	**100**	**0**	**100**
AE + k-NN	99.98	**100.00**	**0.00**	99.98
DAE + k-NN	99.98	**100.00**	**0.00**	99.99

by random forest. The success rates of the other methods are also close to those of the best performing methods. In multi-class classification on Labris dataset, stacking achieved the best performance in all the measures. The best multi-class separation on Digiturk dataset is obtained by random forest, which is able to receive the highest rank

TABLE 9 Binary classification results evaluated by 10-fold cross validation experiment on Digiturk dataset

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	99.63	99.83	0.001	99.84
LR	98.13	97.55	0.022	98.05
MLP	98.93	97.93	0.019	98.89
NB	89.68	82.30	0.195	90.22
SVM	98.04	**99.97**	**0.0002**	95.93
CatBoost	99.84	99.83	0.001	99.84
XGBoost	99.50	99.54	0.004	99.48
RF	**99.96**	**99.97**	**0.0002**	**99.95**
Stacking	**99.96**	**99.97**	**0.0002**	**99.95**
AE + k-NN	99.73	99.60	0.002	99.70
DAE + k-NN	99.72	99.64	0.003	99.70

TABLE 10 Multi-class classification results evaluated by 10-fold cross validation experiment on Labris dataset

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	98.62	99.92	0.00004	91.92
LR	95.14	96.85	0.001	86.84
MLP	94.73	99.12	0.0003	85.78
NB	93.20	72.50	0.016	43.20
SVM	97.88	100	0	92.97
CatBoost	97.75	95.56	0.003	92.90
XGBoost	91.60	99.01	0.0009	90.55
RF	**98.82**	**100**	**0**	**93.31**
Stacking	**98.82**	**100**	**0**	93.30
AE + k-NN	97.60	98.81	0.0001	91.20
DAE + k-NN	97.65	98.90	0.00008	91.30

TABLE 11 Multi-class classification results evaluated by 10-fold cross validation experiment on Digiturk dataset

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	95.28	99.41	0.004	95.12
LR	92.29	97.27	0.02	91.44
MLP	90.52	97.60	0.018	89.21
NB	87.86	89.18	0.093	86.96
SVM	94.25	99.86	0.0007	92.98
CatBoost	95.45	97.21	0.014	95.13
XGBoost	94.84	96.95	0.02	94.45
RF	**99.66**	**99.97**	**0.00002**	**99.32**
Stacking	99.64	**99.97**	**0.00002**	99.29
AE + k-NN	95.60	99.30	0.005	95.20
DAE + k-NN	95.66	99.33	0.0045	95.60

TABLE 12 Binary classification results on Labris dataset without hyper-parameter optimization

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	**99.96**	**99.95**	0.004	**99.92**
MLP	96.66	96.55	0.002	97.01
SVM	99.65	99.92	**0.00001**	98.62
CatBoost	95.66	96.03	0.002	95.73
XGBoost	97.71	97.60	0.0009	96.54
RF	98.96	97.91	0.0003	97.03
Stacking	98.97	99.03	**0.00001**	99.01
AE + k-NN	91.21	92.34	**0.00001**	90.89
DAE + k-NN	79.67	80.12	0.012	75.67

TABLE 13 Binary classification results on Digiturk dataset without hyper-parameter optimization

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	99.58	99.41	0.006	99.58
MLP	97.03	96.58	0.025	96.85
SVM	98.37	**99.97**	0.0002	98.26
CatBoost	94.67	93.89	0.008	93.92
XGBoost	97.65	97.03	0.006	96.83
RF	**99.85**	99.87	**0.0001**	**99.76**
Stacking	99.16	99.13	0.0002	98.87
AE + k-NN	91.23	92.33	0.006	88.26
DAE + k-NN	79.53	79.98	0.015	74.43

TABLE 14 Multi-class classification results on Labris dataset without hyper-parameter optimization

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	98.55	99.91	0.039	91.55
MLP	91.25	92.02	0.05	78.42
SVM	96.15	98.83	0.00023	86.23
CatBoost	89.55	91.07	0.003	81.63
XGBoost	91.23	92.04	0.0007	86.74
RF	**99.01**	**99.98**	**0.000001**	**94.12**
Stacking	96.85	97.14	0.0003	92.32
AE + k-NN	88.21	87.24	0.006	80.12
DAE + k-NN	73.42	74.58	0.021	70.14

in all measures. Tables 4–7 contain accuracy measures on test datasets after performing hyper-parameter optimization on train sets. In addition to those experiments, a 10-fold cross-validation experiment is performed on each train set using the optimum hyper-parameters (Tables 8–11) in which random forest and stacking methods performed the best both in binary and multi-class classification problems and on the two datasets. The level of accuracy obtained on test sets is close to the accuracy measures obtained by cross-validation. This shows that the performance values obtained on test set are robust estimates of the accuracy measures. Furthermore, the accuracies are close to 100%. This shows that DDoS attacks can be predicted with high accuracy both in binary and multi-class classification settings.

TABLE 15 Multi-class classification results on Digiturk dataset without hyper-parameter optimization

Classifier	Accuracy	Detection rate	False alarm rate	F1-measure
k-NN	95.40	**99.48**	0.004	94.90
MLP	88.71	90.03	0.093	86.25
SVM	90.13	96.37	0.001	88.57
CatBoost	89.93	89.88	0.003	83.24
XGBoost	91.23	92.76	0.005	90.23
RF	**98.97**	98.52	**0.0001**	**98.26**
Stacking	97.91	97.92	0.0002	98.03
AE + k-NN	87.25	92.57	0.005	86.67
DAE + k-NN	72.34	73.39	0.036	69.87

Based on these results, the top performing methods are found as ensemble models such as stacking, random forest, and CatBoost followed by XGBoost. The accuracy of autoencoder-based deep learning classifier is also close to that of the ensembles except in multi-class classification on Digiturk dataset. Therefore, extracting new features using deep autoencoders can be an effective strategy for network anomaly detection.

TABLE 16 Optimum hyper-parameters found by Bayesian optimization

Dataset	Class type	Classifier	Optimum hyper-parameters
Labris network	Binary	CatBoost	Iteration = 10, depth = 2
Labris network	Multi-class	CatBoost	Iteration = 35, depth = 14
Digiturk	Binary	CatBoost	Iteration = 23, depth = 12
Digiturk	Multi-class	CatBoost	Iteration = 38, depth = 14
Labris network	Binary	k-NN	K = 5
Labris network	Multi-class	k-NN	K = 5
Digiturk	Binary	k-NN	K = 5
Digiturk	Multi-class	k-NN	K = 3
Labris network	Binary	XGBoost	Eta = 0.1, max depth = 3, silent = 1
Labris network	Multi-class	XGBoost	Eta = 1, max depth = 5, silent = 1
Digiturk	Binary	XGBoost	Eta = 0.9, max depth = 9, silent = 1
Digiturk	Multi-class	XGBoost	Eta = 1, max depth = 9, silent = 1
Labris network	Binary	MLP	beta1 = 0.95, beta2 = 0.99, number of neurons = 22, number of epochs = 12, learning rate = 0.001, batch size = 16
Labris network	Multi-class	MLP	beta1 = 0.95, beta2 = 0.99, number of neurons = 18, number of epochs = 8, learning rate 0.00012, batch size = 32
Digiturk	Binary	MLP	beta1 = 0.95, beta2 = 0.99, number of neurons = 23, number of epochs = 16, learning rate = 0.00032, batch size = 256
Digiturk	Multi-class	MLP	beta1 = 0.95, beta2 = 0.99, number of neurons = 5, number of epochs = 19, learning rate = 0.0000128, batch size = 16
Labris network	Binary	RF	Number of trees = 120
Labris network	Multi-class	RF	Number of trees = 100
Digiturk	Binary	RF	Number of trees = 105
Digiturk	Multi-class	RF	Number of trees = 130
Labris network	Binary	SVM	C = 1, gamma = 0.00781
Labris network	Multi-class	SVM	C = 1, gamma = 0.0039
Digiturk	Binary	SVM	C = 1, gamma = 0.0156
Digiturk	Multi-class	SVM	C = 1, gamma = 0.0039

TABLE 17 Execution times of train test and Bayesian optimization

Classifier	Combined train and test procedures	Bayesian optimization
MLP	55 s	47 min
DAE	3 m 10 s	2 h 13 min
CatBoost	13 s	13 min
XGBoost	1 m 23 s	67 min

Tables 12–15 include the accuracy measures when hyper-parameter optimization is not performed (i.e., default parameters are used for the classifier methods). Based on these results, it can be said that the hyper-parameter optimization improves the accuracy of the classifiers (expect for multi-class classification for Labris dataset) though this improvement is less than 1%. If default hyper-parameters are used, mostly random forest method performed the best. This is expected as random forest is robust to changes in the number of trees parameter. This is the reason for not having a large improvement by hyper-parameter optimization since random forest is among the top performing methods, and it is less sensitive to changes in the number of trees parameter. Nonetheless, the improvement achieved by hyper-parameter optimization could be significant for the network anomaly detection problem due to the costs associated with the damages caused by attacks.

It should be noted that there is class imbalance in train and test sets used in this study. The classifiers in scikit-learn typically use one-vs-rest or one-vs-one for multi-class classification. The most commonly used default strategy is one-vs-rest, but when there is class imbalance, this strategy increases the degree of imbalance further. On the other hand, the one-vs-one technique trains classifiers that are quadratic with respect to the number of class types, which can be costly when the number of classes is high. To handle class imbalance, various techniques can be used such as undersampling, oversampling, assigning weights to data samples belonging to different classes, and techniques such as multi-class roughly balanced bagging.[76]

4.2 | Optimum hyper-parameters

Table 16 includes the optimum hyper-parameters for the classification methods found using the Bayesian optimization method.

4.3 | Execution times

Table 17 includes the running times of combined train and test procedures and Bayesian optimization steps for selected classifiers. As shown in Table 17, the Bayesian optimization takes longer, since it includes many train-test steps. It is also important to note that if grid search is used instead of Bayesian optimization, the hyper-parameter tuning takes 5.5 h for the MLP and approximately 1 day for the DAE method. This shows that Bayesian optimization is faster than the regular grid search optimization though its computational requirement is more than the train-test step.

5 | CONCLUSIONS

In this paper, several machine learning methods have been analyzed for detecting different types of DDoS attacks on Labris and Digiturk datasets introduced recently. These methods include traditional machine learning algorithms, ensemble methods such as stacking, XGBoost, CatBoost, random forest, deep feature extraction methods, such as auto-encoders, stacked deep auto-encoders, and deep learning classifiers based on auto-encoders. The hyper-parameters of the classification algorithms including deep learning methods are fine-tuned using the Bayesian optimization method, which is among the main contributions of this work. Furthermore, the Digiturk and Labris datasets have been employed for model training and testing, including a total of five and eight types of DDoS attacks, respectively, which is much higher than the DDoS attacks available in public datasets. The datasets can be made available through request from the authors. As a future work, other deep learning methods, such as deep

MLP, convolution networks, and different ensemble classifiers will be implemented and compared with the existing classifiers on various benchmark datasets.

ORCID

Yasin Gormez https://orcid.org/0000-0001-8276-2030
Zafer Aydin https://orcid.org/0000-0001-7686-6298
Vehbi C. Gungor https://orcid.org/0000-0003-0803-8372

REFERENCES

1. Top 10 Most Common Types of Cyber Attacks, https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/.
2. DDoS Attack Definitions – DDoSPedia, https://security.radware.com/ddos-knowledge-center/ddospedia/dos-attack.
3. What is a Denial of Service Attack (DoS)?, https://www.paloaltonetworks.com/cyperpedia/what-is-a-denial-of-service-attack-dos.
4. Gungor VC, Aydin Z, Karademir R. Intelligent anomaly detection techniques for denial of service attacks. *International Journal of Computer and Communication Engineering*. 2018;7:20-31.
5. Bellaiche M, Gregoire JC. SYN flooding attack detection based on entropy computing. In *IEEE Global Telecommunications Conference*, Honolulu, HI, USA: IEEE; 2009; pp. 1-6.
6. Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the KDD CUP 99 data set. *IEEE Computational Intelligence for Security and Defense Applications*. 2009;1(6):8-10.
7. Mohammad MN, Sulaiman N, Muhsin OA. A novel intrusion detection system by using intelligent data mining in weka environment. *Procedia Computer Science*. 2011;3:1237-1242.
8. Muda Z, Yassin W, Sulaiman MN, Udzir NI. Intrusion detection based on k-means clustering and naïve Bayes classification, Information Technology in Asia (CITA 11). *7th International Conference*. 2011;1(6):12-13.
9. Liu Z. A method of SVM with normalization in intrusion detection. *Procedia Environmental Sciences*. 2011;11:256-262.
10. Ashok R, Lakshmi AJ, Rani GDV, Kumar MN. Optimized feature selection with k-means clustered triangle SVM for Intrusion Detection. In *IEEE Third International Conference on Advanced Computing (ICoAC)*, Chennai, India: IEEE; 2011; 23-27.
11. Horng SJ, Su MY, Chen YH, et al. A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert Systems with Applications*. 2011;38(1):306-313.
12. Om H, Kundu A. A hybrid system for reducing the false alarm rate of anomaly intrusion detection system. In *IEEE 1st International Conference on Recent Advances in Information Technology (RAIT)*, Dhanbad, India: IEEE; 2012; 131-136.
13. Beitollahi H, Deconinck G. Tackling application-layer DDoS attacks. *Procedia Computer Science*. 2012;10:432-441.
14. Altwaijry H. *Bayesian based intrusion detection system*. Springer Netherlands: IAENG transactions on engineering technologies; 2013: 29-44.
15. Chen ZG, Kim SR. Combining principal component analysis, decision tree and naïve Bayesian algorithm for adaptive intrusion detection, *ACM Proceedings of the 2013 Research in Adaptive and Convergent Systems*. Montreal, QC, Canada: ACM; 2013:312-316.
16. Kim G, Lee S, Kim S. A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications*. 2014;41(4):1690-1700.
17. Kuang F, Xu W, Zhang S. A novel hybrid KPCA and SVM with GA model for intrusion detection. *Applied Soft Computing*. 2014;18: 178-184.
18. Jaswal K, Kumar P, Rawat S. Design and development of a prototype application for intrusion detection using data mining. *IEEE Conference on Reliability, Infocom Technologies and Optimization, Trends and Future Directions*. Noida, India: IEEE; 2015:1-6.
19. Kevric J, Jukic S, Subasi A. An effective combining classifier approach using tree algorithms for network intrusion detection. *Neural Computing and Applications*. 2016;28:1-8.
20. Gharaee H, Hosseinvand H. A New Feature Selection IDS based on Genetic Algorithm and SVM. *IEEE 8th International Symposium on Telecommunications (IST)*. Tehran, Iran: IEEE; 2016:139-144.
21. Guha S, Yau SS, Buduru AB. Attack Detection in Cloud Infrastructures Using Artificial Neural Network with Genetic Feature Selection. *IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. Auckland, New Zealand: IEEE; 2016:414-419.
22. Haider W, Hu J, Slay J, Turnbull BP, Xie Y. Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications*. 2017;87:185-192.
23. Wheelus C, Bou-Harb E, Zhu X. Tackling Class Imbalance in Cyber Security Datasets. *IEEE International Conference on Information Reuse and Integration (IRI)*. Salt Lake City, UT, USA: IEEE; 2018:229-232.
24. Pattaworo A, Polprasert C. Anomaly-Based Network Intrusion Detection System through Feature Selection and Hybrid Machine Learning Technique. *IEEE 16th International Conference on ICT and Knowledge Engineering (ICT&KE)*, Bangkok, Thailand: IEEE; 2018:1-6.
25. Mirza AH. Computer Network Intrusion Detection using various Classifiers and Ensemble Learning. *IEEE 26th Signal Processing and Communications Applications Conference (SIU)*, Izmir, Turkey: IEEE; 2018:1-4.
26. Dhaliwal S, Nahid AA, Abbas R. Effective intrusion detection system using XGBoost. *Information*. 2018;9:1-24.
27. Dahiya P, Srivastava DK. Network intrusion detection in big dataset using spark. *Procedia Computer Science*. 2018;132:253-262.

28. Verma P, Anwar S, Khan S, Mane SB. Network intrusion detection using clustering and gradient boosting. *IEEE 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Bangalore, India: IEEE; 2018:1-7.
29. Abdulhammed R, Musafer H, Alessa A, Faezipour M, Abuzneid A. Features dimensionality reduction approaches for machine learning based network intrusion detection. *Electronics*. 2019;8:1-27.
30. Potluri S, Diedrich C. Accelerated Deep Neural Networks for Enhanced Intrusion Detection System. *IEEE 21st International Conference on Emerging Technologies and Factory Automation*, Berlin, Germany: IEEE; 2016:1-8.
31. Tang TA, Mhamdi L, McLernon D, Zaidi SAR, Ghogho M. Deep Learning Approach for Network Intrusion Detection in Software Defined Networking. *IEEE International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Fez, Morocco: IEEE; 2016:258-263.
32. Alrawashdeh K, Purdy C. Toward an online anomaly intrusion detection system based on deep learning. *IEEE 5th International Conference on Machine Learning and Applications (ICMLA)*, Anaheim, CA, USA: IEEE; 2016:195-200.
33. Javaid A, Niyaz Q, Sun W, Alarm M. A deep learning Approach for Network Intrusion Detection System. *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, New York, NY, USA: ACM; 2016:21-26.
34. Kim J, Kim J, Thu HLT, Kim H. Long Short term memory Recurrent Neural Network Classifier for Intrusion Detection. *IEEE International Conference on Platform Technology and Service (PlatCon)*, Jeju, South Korea: IEEE; 2016:1-5.
35. Kaynar O, Yüksek AG, Görmez Y, Işık YE. Intrusion detection with autoencoder based deep learning machine. *IEEE 25th Signal Processing and Communications Applications Conference (SIU)*, Antalya, Turkey: IEEE; 2017:1-4.
36. Van NT, Thinh TN, Sach LT. An anomaly based network intrusion detection system using deep learning. *IEEE International Conference on System Science and Engineering (ICSSE)*, Ho Chi Minh City, Vietnam: IEEE; 2017:210-214.
37. Aygun RC, Yavuz AG. Network anomaly detection with stochastically improved autoencoder based models. *IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)*, New York, NY, USA: IEEE; 2017:193-198.
38. Kim J, Shin N, Jo SH, Kim SH. Method of Instrusion Detection using Deep Neural Network. *IEEE International Conference on Big Data and Smart Computing (BigComp)*, Jeju, South Korea: IEEE; 2017:313-316.
39. Thing, Vrizlyn LL. IEEE 802.11 Network Anomaly Detection and Attack Classification: A Deep Learning Approach. *IEEE Wireless Communications and Networking Conference (WCNC)*, San Francisco, CA, USA: IEEE; 2017:258–263.
40. Vinayakumar R, Soman KP, Poornachandran P. Evaluating effectiveness of shallow and deep networks to intrusion detection system. *IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Udupi, India: IEEE; 2017:1282-1289.
41. Yousef-Azar M, Varadharajan V, Hamey L, Tupakula U. Autoencoder-based Feature Learning for Cyber Security Applications. *IEEE International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, USA: IEEE; 2017:3854-3861.
42. Shone N, Ngoc TN, Phai VD, Shi Q. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*. 2018;2(1):41-50.
43. Zong B, Song Q, Min MR, Cheng W, Lumezanu C, Cho D, Chen H. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. *The Sixth International Conference on Learning Representations*, Vancouver, BC, CANADA: HSE Publishing House; 2018: 1-19.
44. Mirza AH, Cosan S. Computer network intrusion detection using sequential LSTM neural networks autoencoders. *IEEE 26th Signal Processing and Communications Applications Conference (SIU)*, Izmir, Turkey: IEEE; 2018:1-4.
45. Çekmez U, Erdem Z, Yavuz AG, Sahingöz OK, Buldu A. Network anomaly detection with deep learning. *IEEE 26th Signal Processing and Communications Applications Conference (SIU)*, Izmir, Turkey: IEEE; 2018:1-4.
46. Zhang B, Yu Y, Li J. Network Intrusion Detection Based on Stacked Sparse Autoencoder and Binary Tree Ensemble Method. *IEEE International Conference on Communications Workshops (ICC Workshops)*, Kansas City, MO, USA: IEEE; 2018:1-6.
47. Abdulhammed R, Faezipour M, Abuzneid A, AbuMallouh A. Deep and Machine Learning Approaches for Anomaly-Based Intrusion Detection of Imbalanced Network Traffic. *IEEE Sensors letters*, Piscataway, NJ: IEEE Sensors Council; 2019; 1-4.
48. KDD Cup 99 benchmark data set, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
49. UNSW-NB15 benchmark dataset, https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/.
50. Alkasassbeh M, Al-Naymat G, Hassanat ABA, Almseidin M. Detecting distributed denial of service attacks using data mining techniques. *(IJACSA) International Journal of Advanced Computer Science and Applications*. 2016;7:436-445.
51. Alzahrani S, Hong L. Generation of DDoS attack dataset for effective IDS development and evaluation. *Journal of Information Security*. 2018;9:225-241.
52. Mirkovic J, Reiher P. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communications Review*. 2004;34(2):39-54.
53. Patcha A, Park JM. An overview of anomaly detection techniques: existing solutions and latest technological trends. *Computer Networks*. 2007;51(12):3448-3470.
54. Sperotto A, Schaffrath G, Sadre R, Morariu C, Pras A, Stiller B. An overview of IP flow-based intrusion detection. *IEEE Communications Surveys and Tutorials*. 2010;12(3):343-356.
55. Karademir R. *Intelligent Anomaly Detection Techniques for Denial of Service Attacks*. İstanbul: Master of Science Thesis, Bahçeşehir University; 2015:12-15.
56. IP fragmentation attack, https://en.wikipedia.org/wiki/IP_fragmentation_attack.

57. IP fragmentation attack, https://www.imperva.com/learn/application-security/ip-fragmentation-attack-teardrop/.
58. Mukhandi H. Developing machine learning methods for network anomaly detection. *Master of Science Thesis*, Kayseri: Abdullah Gul University; 2018:2-3.
59. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: machine learning in python. *Journal of Machine Learning Research*. 2011;12: 2825-2830.
60. John GH, Langley P. Estimating Continuous Distributions in Bayesian Classifiers. *Eleventh Conference on Uncertainty in Artificial Intelligence*, San Mateo: Morgan Kaufmann Publishers; 1995:338-345.
61. Cortes C, Vapnik VN. Support vector networks. *Machine Learning*. 1995;20(3):273-297.
62. Bishop CM. *Neural Networks for Pattern Recognition*. USA: Oxford University Press; 1996.
63. le Cessie S, van Houwelingen JC. Ridge estimators in logistic regression. *Applied Statistics*. 1992;41(1):191-201.
64. Aha D, Kibler D. Instance-based learning algorithms. *Machine Learning*. 1991;6:37-66.
65. Bishop CM. *Pattern Recognition and Machine Learning*. Springer-Verlag, Berlin, Heidelberg: Information Science and Statistics; 2006.
66. LeCun Y, Bottou L, Orr GB, Müller KR. *Efficient Backprop*. Neural Networks: Tricks of the Trade; 2012:9-48.
67. Chen T, Guestrin C. *XGBoost: A Scalable Tree Boosting System*. New York, NY, USA: In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16); 2016:785-794.
68. Dorogush AV, Ershov V, Gulin A. CatBoost: gradient boosting with categorical features support. *arXiv*. 2018;1810.11363.
69. Breiman L. *Random Forests Machine Learning*. New York, NY, USA: Springer; 2001; 45, pp. 5-32.
70. Wolpert DH. Stacked generalization. *Neural Networks*. 1992;5:241-259.
71. Kaggle competitions, https://www.kaggle.com/competitions.
72. Hinton GE, Revow M, Dayan P. Recognizing Handwritten Digits Using Mixtures of Linear Models. *7th International Conference on Neural Information Processing Systems*, Cambridge, MA, USA: MIT Press; 1994:1015–1022.
73. Snoek J, Larochelle H, Adams RP. Practical Bayesian optimization of machine learning algorithms. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, eds. *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'12)*. Vol.2 USA: Curran Associates Inc.; 2012:2951-2959.
74. Scikit-optimize, https://scikit-optimize.github.io/.
75. Precision and recall, https://en.wikipedia.org/wiki/Precision_and_recall.
76. Lango M, Stefanowski J. Multi-class and feature selection extensions of roughly balanced bagging for imbalanced data. *Journal of Intelligent Information Systems*. 2018;50(1):97-127.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.

How to cite this article: Gormez Y, Aydin Z, Karademir R, Gungor VC. A deep learning approach with Bayesian optimization and ensemble classifiers for detecting denial of service attacks. *Int J Commun Syst*. 2020;33: e4401. https://doi.org/10.1002/dac.4401