Journal of Information Security and Applications 58 (2021) 102804

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Attack classification of an intrusion detection system using deep learning and hyperparameter optimization

Yesi Novaria Kunang [a,b,e], Siti Nurmaini [b,*], Deris Stiawan [c], Bhakti Yudho Suprapto [d]

[a] Doctoral Engineering Department, Faculty of Engineering, Universitas Sriwijaya, Palembang, Indonesia
[b] Intelligent System Research Group, Faculty of Computer Science, Universitas Sriwijaya, Palembang, Indonesia
[c] Computer Networking & Information Systems, Faculty of Computer Science, Universitas Sriwijaya, Palembang, Indonesia
[d] Electrical Engineering Department, Faculty of Engineering, Universitas Srwijaya, Palembang, Indonesia
[e] Faculty of Computer Science, Universitas Bina Darma, Palembang, Indonesia

ARTICLE INFO

Keywords:
Intrusion detection system
Deep learning
Hyperparameter optimization
Multiclass classification

ABSTRACT

A network intrusion detection system (NIDS) is a solution that mitigates the threat of attacks on a network. The success of a NIDS depends on the success of its algorithm and the performance of its method in recognizing attacks. We propose a deep learning intrusion detection system (IDS) using a pretraining approach with deep autoencoder (PTDAE) combined with a deep neural network (DNN). Models were developed using hyperparameter optimization procedures. This research provides an alternative solution to deep learning structure models through an automatic hyperparameter optimization process that combines grid search and random search techniques. The automated hyperparameter optimization process helps determine the value of hyperparameters and the best categorical hyperparameter configuration to improve detection performance. The proposed model was tested on the NSL-KDD, and CSE-CIC-ID2018 datasets. In the pretraining phase, we present the results of applying our technique to three feature extraction methods: deep autoencoder (DAE), autoencoder (AE), and stack autoencoder (SAE). The best results are obtained for the DAE method. These performance results also successfully outperform previous approaches in terms of performance metrics in multiclass classification.

1. Introduction

The development of Internet of Thing (IoT) network technology has had an impact on the growing number of devices and intelligent applications connected to it [1]. This also increases the risk of cyberattacks [2,3]. Consequently, the development of mechanisms for application security and infrastructure has become an important issue [4,5]. Overcoming the threat of attacks to a network requires a security system that can detect attacks, known as an intrusion detection system (IDS).

Researchers have been developing IDSs that can be used on various heterogeneous IoT network platforms [6]. To minimize false alarms in an IDS, some researchers have used machine learning (ML) [7,8]. Using the ML approach, an IDS more accurately recognize attacks based on the features of each attack [8]. Unfortunately, with the complexity of networks such as the IoT, traditional ML algorithms have limitations when processing an increasing volume of data [7,9–12]. Moreover, ML's feature-learning process performs poorly when attempting to extract meaningful information from big data [13,14].

A predictive analytics solution called deep learning (DL) is required to overcome the disadvantages of ML methods in an IDS with big data. Deep learning involves the development of artificial neural network algorithms that use many hidden layers to extract meaningful information representations from big data. With optimal processing and utilization of a graphics processor, DL demonstrates high performance in the big data domain [13]. It has the advantage of extracting large numbers of data features [11]. With such potential, the DL technique is suitable for attack detection and classification in networks as varied as those in the IoT. Unfortunately, it is not easy to achieve the best DL architecture modeling. The complexity of the IDS model with deep learning necessitates a hyperparameter tuning process from DL architectures to improve system performance [14].

The hyperparameter tuning process becomes a challenge for a network IDS when it is used for a deep structured model such as deep neural networks (DNNs). This process involves many parameter values called hyperparameters, which should be specified when building and training models. The hyperparameters, such as the number of layer

* Corresponding author.
E-mail address: sitinurmaini@gmail.com (S. Nurmaini).

https://doi.org/10.1016/j.jisa.2021.102804

networks, the number of nodes per layer, and the activation function used. Furthermore, the optimization method has additional parameters that must be specified, such as the learning rate value. One method of finding the hyperparameters is manually, where a set of parameters is first tested, and new parameters are used to see whether the performance of the optimization method improves [15–18]. The traditional manual tuning process is time-consuming and requires a domain expert to accelerate the process. This necessitates automating hyperparameter tuning to accelerate the hyperparameter optimization (HPO) process. Automating the HPO process is essential to reducing human effort [14] and increasing the performance of ML and DL algorithms [19].

The HPO process can automatically be performed by dividing the range of each parameter by the same value, and then the computer performs a combination of values known as grid terms [19]. However, the weakness of the grid search is the process of tuning, which is time-consuming when hyperparameters are added, as the number of combinations of parameters increases exponentially [14]. To overcome this, grid search must be combined with random search [14,20]. This combination attempts several random parameter combinations, providing better detection result especially if some hyperparameters offer better performance than other parameters [21,22].

This research provides an alternative solution to selecting DL structure models by automatically performing the HPO process through a combination of grid search and random search techniques. The HPO process searches the expected hyperparameter candidate values most likely to increase the classification of the DL algorithm model in the IDS. The approach involves the regulation of various deep structured models with an autoencoder (AE) and a DNN model and the use of an optimization method to select the numerical hyperparameter values and the best categorical hyperparameter configurations. With uniform probability distributions, the process is repeated to obtain the best IDS model by observing the best detection rate performance in attack classification. The automatic HPO tuning process is done in detail for numeric and categorical parameters, such as kernel initial and various variant ReLU activation functions. This approach has never been performed by other researchers on the DNN deep learning model with DAE features extraction.

2. Related work

This section discusses state of the art in the use of DL in IDSs and the use of HPO in deep structured models.

Deep Learning (DL) has shown more accurate performance than other ML technique on big datasets [23–25]. It can extract complex abstractions of high-level data representations from large volumes of unsupervised data [11,26]. The use of unsupervised or semi-supervised feature learning and hierarchical feature extraction in DL replaces manual feature engineering. With DL's many advantages, many studies have used it to improve classification performance, including detection and classification of attacks against IDSs. Recent studies using DL techniques such as deep belief networks [27], DNNs [28,29], and deep Boltzmann machines [30] have shown significant improvements to classification performance.

Some research has attempted to improve the performance of detection and classification attacks by modifying DL network architectures. Some researchers use AE variants for a model's learning features [16, 31–37]. In [31], a stacked autoencoder (SAE) was used by stacking multiple layers of AEs with variations of two and three hidden layers and by using softmax as a classifier. The research attempted to set the parameters of the number of neurons in the hidden layer and used a variety of different activation functions. In another study [34], a deep autoencoder (DAE) was combined with the deep feed-forward neural network to classify attacks using default hyperparameters. A DAE was combined with Naïve Bayes in another study [35] to detect attacks by varying the number of neurons and layers in the DAE. Non-symmetric deep autoencoders (NDAEs) combined with random forest [36] used varying combinations of neurons and layers in the AE structure to detect attacks. Some studies [31,34–37] have demonstrated increased performance in attack detection and classification. Unfortunately, DL model structures are selected according to the number of layers and the number of neurons in each layer, while in detail performing hyperparameter tuning could improve DL performance [14, 38].

Two studies used the recurrent neural network (RNN) model [18] and feed-forward neural network (FNN) model combined with a convolutional neural network (CNN) [39] for the IDS, respectively. Both studies varied only the number of neurons in the hidden layer and various learning rate values to improve the model's performance. Another study [40] used a restricted Boltzmann machine (RBM) model using hyperparameter tuning of an IDS on the balanced dataset. The RBM training process was conducted on the ISCX dataset with a balanced number of normal and anomaly records. The model then tuned hyperparameters, including learning rate value, epoch, and mini-batch size by comparing the contrastive divergence and persistent contrastive divergence algorithms on the RBM model. The RBM model uses only two layers and may still be evaluated with deep restricted Boltzmann machine (DRBM) techniques to improve performance. Also, the process of tuning hyperparameters was performed manually by individually testing the impact of parameters on the model instead of simultaneously.

Self-taught learning through a DL approach was used for an IDS in another study [16]. The proposed method used feature learning and dimension reduction with a sparse autoencoder (SAE). The SAE model performed hyperparameter tuning for decay, momentum, and learning rate with a manual technique. After the pretraining stage, the next phase was classification using the support vector machine algorithm to detect and classify attacks. The model used the SAE with only one hidden layer. The accuracy in multi-class classification was only 80.48%, and the recall value was only 68.28%.

Recent research in the field of botnet attack detection uses an artificial neural network with HPO [41]. The model was developed using the multilayer perceptron. In the study, it was used to perform a hyperparameter tuning process with a grid search optimization technique. The tuned hyperparameters included the number of nodes in the hidden layer and the alpha value in the L2 regularization parameter. The model was tested on the CSE-CIC-IDS2018 dataset, resulting in improved performance in binary classification compared to performance with the default parameter values. Hyperparameter tuning optimization was limited to only two parameters. Unfortunately, the composition structure of the training and testing data was not explained. Another study conducted detailed experiments on a DNN by selecting a topology (the number of hidden layers and neurons) and varying the learning rate value [42].

Another study [33] combined DAE with DNN to evaluate the performance of the algorithm on the NSL-KDD dataset. The results show acceptable performance in the overall detection of probe, root to local (R2L), denial of service (DoS), and user to root (U2R) attacks, reaching 99.3% for data training. The results showed improved results from previous research on the detection rate and speed of detection. However, the weakness of the presented models was that they only use KDDTrain+ dataset. Models were not tested with NSL-KDD data testing, which has more complex data (some attacks are not present in the training data), making it more realistic. The DL architecture consisted of only one hidden layer, which allowed analyzing the effect of adding layers to the IDS model. However, the hyperparameter values, such as learning rate on the DL architecture, were selected manually rather than tuned automatically.

Some of this research in the field of IDS can be developed further by setting the DL architecture parameter to improve its performance, especially for the DNN architecture. Hyperparameter tuning and optimization of DNNs is a fundamental factor in producing competitive performance results [43]. Hyperparameter optimization itself is an automatic process built on ML and DL models. The method of tuning hyperparameters improves the performance of ML and DL algorithms compared to the default parameters in libraries [19,44]. Related to HPO, [45] used the Bayesian method for adaptive HPO that models the function of loss and performance of multiple datasets. The random search was used in [43], revealing that a random search technique used on a DNN model can reduce errors and execution time. A random search method can improve the performance of HPO algorithms.

The Deep Reinforcement Learning paradigm (DRL) was used in IDS research [46]. The author made some improvements to the classic DRL. They replaced the environment in traditional DRL with a sampling function of training intrusion data that generates rewards based on error detection during the training phase. With Double Deep Q-Network (DDQN) algorithm and some of the adjustments in DRL parameters successfully improve detection results for binary classification in the NSL-KDD dataset and multi-class classification in the AWID dataset.

3. Proposed method and design

This section describes the computational architecture and methods adopted for the proposed framework. The developed deep structured model is a hybrid model using a DAE for pretraining and DNN for the process of attack identification. The flow process model for detection and classifications of attacks is shown in Fig. 1, consisting of three stages. The first stage begins with the process of preparing and preprocessing data. The second stage continues the development of the deep structured learning model with HPO to obtain the best model that produces the best detection performance. The end phase is the evaluation of classifiers to obtain the best model. The hyperparameter values of the DL algorithm significantly affect the outcome. The HPO algorithm was used to process the tuning hyperparameters, such as the number of hidden layers or number of neurons in each layer, learning rate, batch size, activation function, and kernel initializers.

Grid search and random search options were used to optimize the hyperparameter values. Grid search performs a complete search against the full set of hyperparameters. Due to a large number of hyperparameters to be tuned, meaning the number of model combinations of hyperparameters is also large, the automatic HPO process was combined with random search with a reduced probability technique to create combinations of parameters, with no specific order or criteria. This process repeats a random number of grid combinations and records the model that obtains the highest validation rate. Finally, one model with the best hyperparameter valued, producing the best level of detection performance was obtained. To evaluate the proposed model, we used the NSL-KDD and CSE-CIC-IDS2018 datasets.

3.1. Deep learning model

Our proposed network IDS used the pretraining process with DAE as feature extraction and fine-tuning phase using DNN architecture, as shown in Fig. 1. A DAE performs the feature extraction process with encoding and decoding. The bottleneck layer (middle layer) with a smaller dimension for the dataset input feature is a feature representation that has been extracted. The result of the extracted feature in the form of encoding structure, along with the weight and bias values are transferred to the DNN structure for the fine-tuning process.

In the DAE structure, the data going into the input layer are from the result of the preprocessing dataset (X). The DAE outputs produce the same data $\hat{X}$ (resemble X data). The structure and data in the decoding layer are not forwarded to the DNN model. The DNN model trains data binary attack classification and multiclass classification. The hyperparameter tuning process produces the best model by observing the detection rate of the attack classification.

3.1.1. Deep autoencoder model

One focus of this research was to develop an effective IDS by extracting features from raw data to data in a better representation of low-dimensional features. The feature extraction process aims to increase the efficiency and effectiveness of the detection and classification of binary (normal or anomaly) and multiclass classification attacks in the NSL-KDD and CSE-CIC-IDS2018 dataset.

A DAE is an AE with more than one hidden layer (Fig. 1). The addition of hidden layers in a DAE allows the AE to learn more complex patterns of data mathematically. On an AE with a single hidden layer, the process of mapping from layer inputs to hidden layers is the encoding phase. The mapping of the hidden layer to the output layer is the decoding phase. In a DAE with many hidden layers are additional encoder and decoder pairs. In Fig. 1, the DAE structure consists of five hidden layers (composed of three pairs of encoders and decoders). The DAE stage begins with the phase during which the first encoder ($E1$) encodes input X, the second encoder ($E2$) encodes the output from $E1$, and the third encoder ($E3$) encodes the output from $E2$. The encoding phase on the middle layer can be written as $Z = E3(E2(E1(X)))$.

In a single hidden layer, AE vector encoder h is notated as $h = f(W.X + b)$, where W is a weight matrix, b is a bias vector, and X is an input vector. The vector encoding function in forward propagation for hidden layer l becomes Eq. (1)

$$h^{(l+1)} = f(W^{(l)}.h^{(l)} + b^{(l)}), \tag{1}$$

so that the encoding phase of each layer can be written as $E1 = f(W^{(1)}.X+b^{(1)})$; $E2 = f(W^{(2)}).E1+b^{(2)}$; and $Z = E3 = f(W^{(3)}).E2+b^{(3)}$.

The decoding phase is executed in the opposite direction of the encoding phase: The first decoder will be the last to be decoded. The final reconstruction phase of decoding vector output is $\hat{X} = D1(D2(D3(E3(E2(E1(X))))))$. For an AE with a single hidden layer $\hat{X} = f(W^T.h+b')$, the decode function for DAE becomes: $D3 = f((W^{(3)})^T.Z+b^{(3)'})$; $D2 = f((W^{(2)})^T.D3+b^{(2)'})$; $\hat{X} = D1 = f((W^{(1)})^T.D2+b^{(1)'})$, where $f(.)$ is a node activation function used on each layer. The activation function $f(.)$ on neural network neurons is a mathematical operation applied to the output signal used to enable or disable neurons. The activation function maps the output value into the desired range, such as between 0 and 1 or −1 and 1 (depending on the activation function used).

The cost function of the DAE is the distance function between input and reconstructed $\hat{X}$ Cost, also called a loss, can be calculated with mean squared error loss for the activation function:

$$J(w, b, x^i, \hat{x}^i) = \frac{1}{2}\left\|x^i - \hat{x}^i\right\|^2. \tag{2}$$

The input data are normalized to between 0 and 1, and then the reconstruction process on the output layer can be conducted with a nonlinear sigmoid function. For input, a binary number or input with a range between 0 and 1 can be used for binary cross-entropy as a loss function [47,48]. For overall training of m data, $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} J(w,b,x^i,\hat{x}^i)$. The minimal loss value is calculated using the following equation:

$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m}[x^i \log(\hat{x}^i) + (1 - x^i)\log(1 - \hat{x}^i)] \tag{3}$$

Backpropagation updates the weight and bias values of each node in each layer to reduce the cost. The best cost for the most minimal loss value is close to zero. After the AE training process, the data on the layer bottleneck (Z) are a representation of the data in the low-dimensional encoding process. The encoding structure (Z) is forwarded as input to the DNN classifier, called transfer learning. Transfer learning transfers the Z-encoding structure to the AE and the weight and bias values to the classifier.

The feature extraction model with AE is a pre-training process that proceeds to the training and learning process of the classifier model. With the transfer learning process, HPO with an AE affects the learning

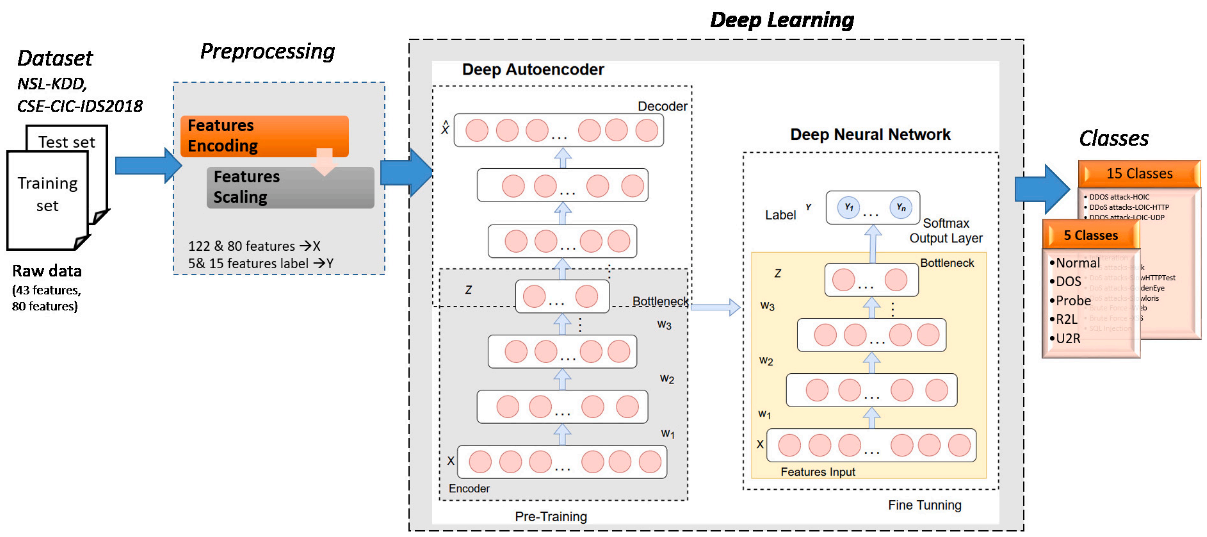


Fig. 1. Proposed deep learning architecture.

process of the classifier model. Optimizing the IDS must also optimize the hyperparameter value of the AE model. Thus, on the model feature extraction with AE, the hyperparameter tuning process was performed for the number of layers, the number of neurons from each layer, activation function, learning rate, kernel initialization, and function loss. Metrics for the AE were obtained by observing the loss value of the AE model.

3.1.2. Deep neural network model

An attack detection model was developed using a DNN algorithm as a classifier. The encoding process results from the automatic feature extraction of the AE model from the training data to the input of classifier, as shown in Fig. 1. Input from the DNN architecture is output Z, generated from the AE process in the form of training input data X. Another layer is added to the DNN result from output y (a label that can consist of five attack classes or binary classes on the NSL-KDD dataset). Then, the process of retraining is conducted using the weight and bias values of the AE as pretraining values, to learn the output y. The output $\hat{y}$ can be written as

$$\hat{y} = f(W^{(l)}.h^{(l)} + b^{(l)}) = f(z^{(l+1)}), \quad (4)$$

where $l + 1$ = the last layer.

As for the AE structure function, $f(.)$ is the activation function. Among other activation functions, the rectified linear unit (ReLU) function has advantages [15,49]. In this study, we evaluated the use of several variants of ReLU activation functions such as SELU, PReLU, ELU, and leaky ReLU for the hidden layer. In the output layer, we used the sigmoid activation function for the binary classifier and softmax activation for the multiclass classifier (five classes of attack).

The initial parameter was specified before training the DNN.$W^{(1)}$, $W^{(2)}$, $W^{(3)}$ and $b^{(1)}$, $b^{(2)}$, $b^{(3)}$ values were obtained from the DAE encoding process. The weight parameter needed to be initialized randomly as a small value [e.g., distributed around zero; $n(0, 0.1)$]. The resulting output $\hat{y}$ approaches the actual value y. For the output nodes, the difference between the network activation $\hat{y}^i$ and the actual target value y^i is calculated. In the hidden unit, the error value is calculated based on the average weight of the error nodes that use h_i^l as inputs. The loss function and activation function, for binary classification, use the binary cross-entropy and sigmoid functions, while for multi-class classification, we use the categorical cross-entropy and softmax functions [50].

3.2. Experimental design

The experiment process ran on a computer platform with specifications including an Intel Core i7-7700HQ at 2.8 GHz, 4 cores, 32 GB of RAM, NVIDIA GTX 1060 with 6 GB GDDR5 graphics memory, and operating system Windows 10. The DL structure was developed using the Python programming language with computation utilizing the TensorFlow-GPU library with Keras neural network library. To evaluate the performance of the HPO tuning process, the Talos library [51] was used by applying a random search technique on a combined hyperparameter list.

To verify the capabilities of the proposed model in this work used two datasets, the NSL-KDD dataset, and the CSE-CIC-IDS2018 dataset. NSL-KDD dataset is selected as a benchmark of several previous studies in IDS. For the second dataset, we use the new real dataset CSE-CIC-IDS2018 dataset, which is closer to the current payload network and represents the detection of more recent attacks. In evaluating the performance of DL with various autoencoder feature extraction methods on both datasets, we use all of the features in network connection record categorizing a class attack into its categories.

3.2.1. NSL-KDD dataset

The NSL-KDD dataset was used to perform a benchmark test of an IDS that was developed over several previous studies. The NSL-KDD dataset itself was recommended by Tavalaee et al. to replace KDD Cup 99 [52]. The dataset comprises the KDDTrain+ training data and KDDTest+ testing data. KDDTrain+ consists of 22 attack types and normal packet data types, while KDDTest+ added 37 attack types grouped into four attacks, as shown in Table 1. NSL-KDD+* is an NSL-KDD dataset without new types of attacks (by eliminating 17 new types of attacks). This dataset consists of 41 features and 1 class label. The composition of attacks, divided into Denial of Service (DoS) attacks, probe attacks, Remote to Local (R2L) attacks, and User to Root (U2R) attacks, is shown in Table 1.

3.2.2. CSE-CIC-IDS2018 dataset

The Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC) develops the CSE-CIC-IDS2018 dataset to meet the needs of the attack detection benchmark dataset that represents traffic composition and attack on the current modern network [53]. This dataset consists of 80 features, including labels. Most of the features are statistical traffic information based on flow, which is degenerated and extracted with CICFlowMeter. Detailed scenario and extracted traffic features process are explained in [53,54].

Table 1
NSL-KDD dataset composition.

Attacks Category	KDDTrain+		KDDTest+		KDDTest+*	
Normal	67,343	53.46%	9711	43.08%	9711	51.67%
DoS	45,927	36.46%	7458	33.08%	5741	30.55%
Probe	11,656	9.25%	2421	10.74%	2199	11.70%
R2L	995	0.79%	2754	12.22%	1106	5.88%
U2R	52	0.04%	200	0.89%	37	0.19%
Total	125,973	100%	22,544	100.00%	18,794	100.00%

The features of this dataset significantly different from the NSL-KDD dataset. Almost the overall features are continuous data except for the dst_ports, protocols, timestamps, and labels.

This dataset consists of six different intrusion scenarios, Brute-Force, Botnet, DoS, Web attacks, DDoS, and infiltration, with a total of 14 types of intrusions. The total number of records in this dataset is 16,232,943. The benign traffic encompasses 13,484,708 records (83.07% of the data), while the remainder is malicious records. In this work, we use 10% data for training data and 2.5% for testing data. With an imbalance of the amount of malicious and benign in the data, then for benign data, only used data about 7.5% of benign data. In the malicious attack data, we used composition based on a comparison of the amount of data. For attack data with a little amount of data (<2000), we fetched entire data. Others vary between 35%–75% depending on the amount of data. So the composition of benign and malicious data is set close to 50:50 for both testing and training data. The structure of the training and testing data used are shown in Table 2.

3.2.3. Data preparation and preprocessing

The preparation and preprocessing stages are shown in the flow process in Fig. 1. This begins with selecting a dataset and converting the categorical data into numerical data, called feature encoding. It continues by transforming feature data within a given scale to obtain high-value features, not dominating features. After preprocessing, data are ready for the training and testing processes.

In the preparation process of the NSL-KDD dataset, the label/class field must be grouped by attack type as in Table 1. For attack detection, the Class column is divided into two classes: normal and anomaly. For classifying attacks, the Class column is mapped into four categories: DoS, probe, R2L, and U2R. In the initial stage of this preprocessing, the dataset consists of 42 features. The features in the NSL-KDD dataset comprise three types: nominal, binary, and numeric. Machine learning and DL algorithms cannot process nominal data directly. The proposed model uses all features, including non-numeric data. For this, the non-numeric data features of the dataset must be converted into numeric data (specifically, the protocol type, service, and flag features).

Feature mapping must be performed to convert categorical data into numeric data. This process is called feature encoding. There are two approaches to mapping categorical values into numerical values: one-hot encoding [55] and ordinal encoding [29]. In One-hot encoding, for each level of the categorical variable are mapped into a dummy variable in binary formed [55,56]. The ordinal encoding uses all features, and data of non-numeric features on the dataset are converted into numeric data. In the initial research, one-hot encoding provided better classification results than ordinal coding [15]. Therefore, this research used one-hot encoding. This coding technique is essential because several studies on IDS used ordinal encoding [29,36], which maps features to 41 features.

There are three nominal features in the NSL-KDD dataset: protocol_type, service, and flag. For example, the protocol_type feature has three categorical values, namely transmission control protocol (TCP), user datagram protocol (UDP), and internet control message protocol (ICMP). With one-hot encoding, three numeric features replace the original protocol_type: protocol_type_tcp, protocol_type_udp, and protocol_type_icmp. The binary value for each new feature shows which type of protocol it is. Of the three new columns generated from protocol_type, only one is worth 1 for each instance. For example, the list of protocol_type features for four instances [TCP, UDP, ICMP, UDP] is [[1, 0, 0, 0], [0, 1, 0, 1], [0, 0, 1, 0]] in the form of one-hot encoding. With one-hot encoding, the service feature is mapped into 70 new features, the flag feature to 11 new features. Thus, 41 features of the dataset are mapped to 122 features, and the multi-class labels are mapped into five label features.

In the CSE-CIC-IDS2018 dataset consisting of 80 features, we use the 79 features by eliminating the timestamp features. Most data types in this dataset are continuous data. Preprocessing data is also done by performing a feature encoding for feature protocol and feature labels. For feature protocol mapped to 3 instances, namely TCP, UDP, and hop-by-hop IPv6 (HOPOPT). Feature encoding generates 80 features as input data and 15 features as multiclass labels.

The feature scaling process will convert the range of values from the full features into a predefined range. The method of scaling features is required for features that have large values, not for dominating other features. Some ways to do feature scaling for ML include standardization, scaling, and normalization. Initial testing was performed using Z-score normalization and min–max scaling with range [0.1] and [−1.1]. Min–max scaling with range [0.1] delivers the best results in the AE process. The next step used min–max scaling, as in (5).

$$x_{i,j}^{norm} = \frac{x_{i,j} - min(x_{:,j})}{max(x_{:,j}) - min(x_{:,j})} \tag{5}$$

where $i = 1, \dots, m$ and m = amount of training or testing data; $j = 1, \dots, n$ and n = number of features; $x_{i,j}$ is variable value x_i of the attribute j; $x_{i:j}$ is the value of the $--j$ feature; $min(x_{:,j})$ is the smallest value of the column feature; and $max(x_{:,j})$ is the largest value of the column feature. Scaling maps features in the range of [0–1].

3.2.4. Metric evaluation

Based on the performance metric ontology [57], confusion matrix helps represent the classification results as true or false values [58,59]. A true positive (TP) value means the IDS has successfully detected an attack. A false positive (FP) means normal behavior is incorrectly classified as an attack by the IDS. A true negative (TN) means normal behavior is successfully labeled as normal by the IDS. A false negative (FN) means an attack has not been detected by the IDS and is classified as normal. The confusion matrix is used to calculate some of the performance metric values.

For performance testing of the network anomaly IDS, most studies have adopted common metrics such as accuracy (ACC), detection rate (DR), false alarm rate (FAR), precision, and F1-score [60–62].

3.2.5. Hyperparameter optimization

Fig. 2 shows the HPO process for obtaining the best result from the proposed DL models using a DNN. The hyperparameter tuning process is indispensable to testing results that show overfitting values, as in the case of the NSL-KDD dataset. The detection rate of data training results in the best value. The hyperparameter values to be evaluated to obtain the best model are the number of neurons and layers, the value of the learning rate, initialization weights, activation function, and loss function. The hyperparameter values used for parameter tuning are shown in Table 3.

The number of neurons in each layer affects the system's complexity. A higher number of neurons tends to increase training time and response time [63]. The number of neurons in each layer in the multilayer model controls capacity. Adding layers helps the learning process due to the complex data representation in the automated feature engineering. The addition of layers improves accuracy, depending on the complexity of the dataset [64,65].

The value of the learning rate α that we used is between 0 and 1, which is less than 1 and higher than the 10^{-6}. The default value for a learning rate of 0.01 for a standard multilayer network [66]. The

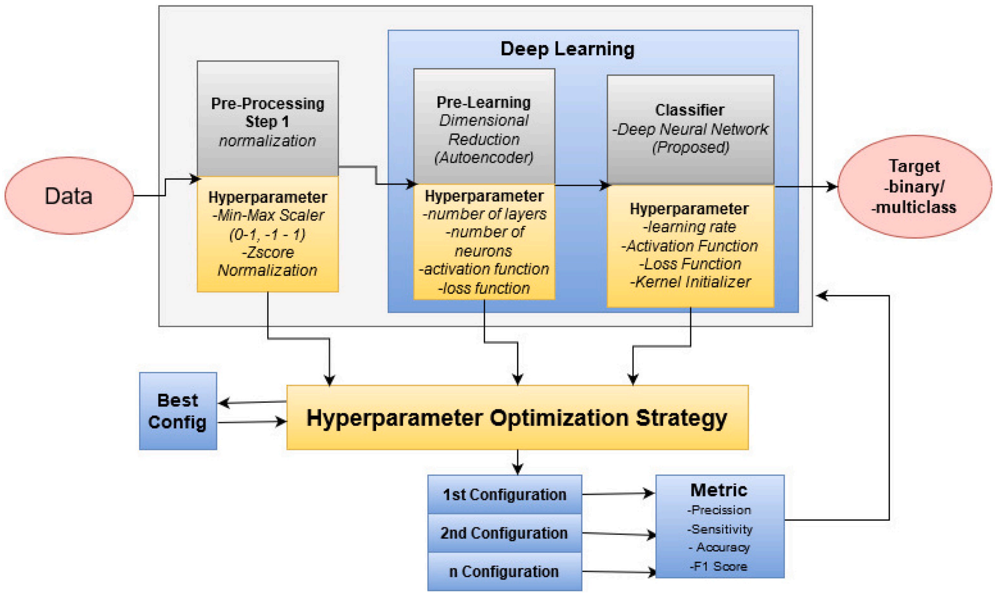

Deep Learning
Pre-Processing Step 1 normalization
Hyperparameter -Min-Max Scaler (0-1, -1 - 1) -Zscore Normalization
Pre-Learning Dimensional Reduction (Autoencoder)
Hyperparameter -number of layers -number of neurons -activation function -loss function
Classifier -Deep Neural Network (Proposed)
Hyperparameter -learning rate -Activation Function -Loss Function -Kernel Initializer
Data
Target -binary/ -multiclass
Best Config
Hyperparameter Optimization Strategy
1st Configuration
2nd Configuration
n Configuration
Metric -Precission -Sensitivity - Accuracy -F1 Score

Fig. 2. Intrusion detection system optimization.

Table 2
CSE-CIC-IDS2018 dataset composition.

Category	Attack Types	Size		Train		Test	
Benign	/	13,484,708	83.070%	803,025	49.590%	201,238	49.709%
DoS	DDOS attack-HOIC	686,012	4.226%	192,098	11.863%	48,006	11.858%
	DDoS attacks-LOIC-HTTP	576,191	3.550%	161,447	9.970%	40,219	9.935%
	DDOS attack-LOIC-UDP	1,730	0.011%	1,362	0.084%	368	0.091%
Botnet	Bot	286,191	1.763%	85,842	5.301%	21,479	5.306%
Brute Force	FTP-BruteForce	193,360	1.191%	58,055	3.585%	14,452	3.570%
	SSH-Bruteforce	187,589	1.156%	56,332	3.479%	14,013	3.461%
Infilteration	Infilteration	161,934	0.998%	48,347	2.986%	11,892	2.937%
DoS	DoS attacks-Hulk	461,912	2.846%	138,459	8.550%	34,758	8.586%
	DoS attacks-SlowHTTPTest	139,890	0.862%	41,974	2.592%	10,484	2.590%
	DoS attacks-GoldenEye	41,508	0.256%	25,008	1.544%	6,123	1.512%
	DoS attacks-Slowloris	10,990	0.068%	6,612	0.408%	1,630	0.403%
Web Attacks	Brute Force -Web	611	0.004%	496	0.031%	115	0.028%
	Brute Force -XSS	230	0.001%	187	0.012%	43	0.011%
	SQL Injection	87	0.001%	71	0.004%	16	0.004%
Total		16,232,943	100%	1,619,315	100%	404,836	100%

learning rate affects the precision of the neural network. The weight and bias initialization values affect how quickly the network converges (reaches the local minimum and global threshold). Some functions create an initialization weight for the hidden unit, resulting in a faster iteration [49]. Usually, weight and bias values are filled with small random numbers, but in our experiment, we use some kernel initializers function as in Table 3.

Table 3
Optimized Hyperparameters.

Hyperparameters	List of Values
Number of hidden layer DAEs	1, 3, 5, 7
Number of hidden layer nodes	100, 90, 80, 75, 70, 60, 50, 40, 35, 30, 25, 20 and 15
Learning rate	0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1
Kernel initialization	Uniform, lecun_uniform, normal, zero, glorot_normal, glorot-uniform, he_normal, he_uniform
Batch size	32, 64, 256
Activation function	ReLU, ELU, SELU, PReLU, Leaky ReLU

4. Results and discussion

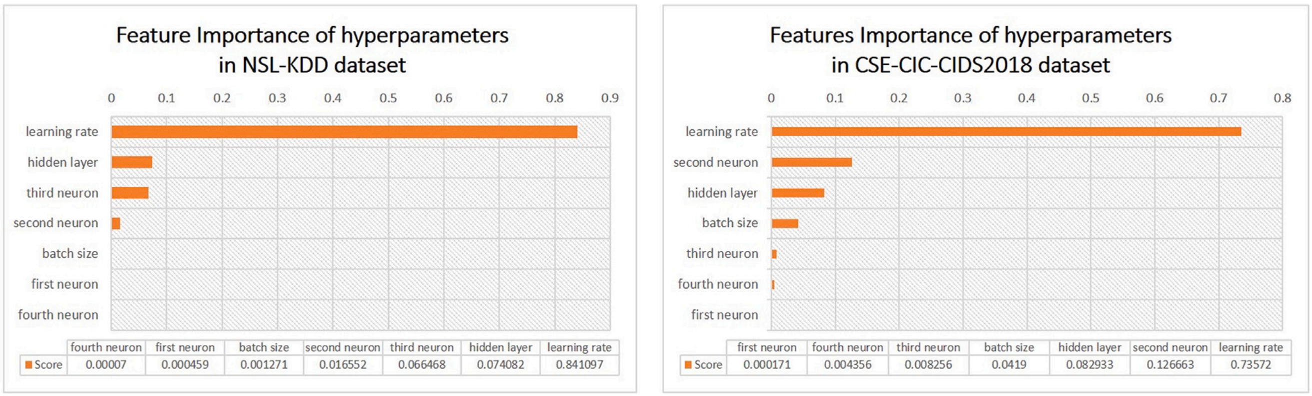
4.1. *Hyperparameter importance*

The automated HPO tuning process for the hyperparameters shown in Table 4 was performed hierarchically using the Talos library based on the number of hidden layers of the DAE model. One of the challenges of the HPO tuning process is testing the excessive number of models resulting from the combination of hyperparameters. Therefore, we used a random search with a permutation technique by reducing the number of sample combinations automatically. The number of models from combinations of hyperparameters is shown in Table 4, with as many as 32,445 models in NSL-KDD. Given the number of models that would be attempted, we used grid downsampling by reducing the model with a uniform random sampling 1952 models (6% of the combinations of all models) in dataset NSL-KDD. The CSE-CIC-IDS2018 dataset generates 23,100 combinations of hyperparameter models. This dataset has a huge dimension, so the training process took a long time. We slightly reduced the number of models that had a large learning rate value. The combination of hyperparameters is reduced to only 5% of the combination become about 1.067 models.

In the HPO process, we used 30 epoch in the DAE pre-training process and 50 epoch for DNN fine-tuning process on the NSL-KDD dataset. The loss function in the DAE process used binary cross-entropy on the NSL-KDD dataset and the categorical cross-entropy for the DNN process. Binary cross-entropy chosen because from 122 features input consist of 90 data binary features and the remaining 32 discrete

Table 4
Number of models from the combination of hyperparameters in both datasets.

Number of DAE hidden layers	Number of DNN hidden layers	Number of hyperparameter combination models		Grid downsampling of hyperparameter combinations models with uniform random sampling	
		NSL-KDD	CSE-CIC-IDS2018	NSL-KDD	CSE-CIC-IDS2018
1 hidden layer	1 layer	2,625	2100	210	206
3 hidden layers	2 layers	5,040	5040	503	253
5 hidden layers	3 layers	11,340	7560	567	338
7 hidden layers	4 layers	13,440	8400	672	270
Total models		32,445 models	23,100 model	1952 models	1067 models



	fourth neuron	first neuron	batch size	second neuron	third neuron	hidden layer	learning rate
▬ Score	0.00007	0.000459	0.001271	0.016552	0.066468	0.074082	0.841097

	first neuron	fourth neuron	third neuron	batch size	hidden layer	second neuron	learning rate
▬ Score	0.000171	0.004356	0.008256	0.0419	0.082933	0.126663	0.73572

Fig. 3. Impact of hyperparameters on the increasing value of the detection rate of the overall model. (a) hyperparameter features impact in the NSL-KDD dataset; (b) hyperparameter features impact in CSE-CIC-IDS2018 dataset.

features that already normalized in the range of 0 and 1. In the CSE-CIC-IDS2018 dataset, the DAE pre-training process uses 20 epoch and 30 epoch on DNN fine-tuning classification process. The loss function used MSE on DAE and categorical cross-entropy on DNN. MSE is used because from 80 input features, 77 features are continuous data. After the best multi-class classification performed, we increase the number of epoch to become 200 epoch on the fine-tuning process to improve the precision and sensitivity, especially in attack classes that are difficult to detect.

In this experiment, we focused on a variant of ReLU activation because in previous research [15], this activation function in hidden layers resulted in the best performance. HPO tuning was performed for the activation functions ReLU, ELU, SELU, PReLU, and Leaky ReLU. For optimization function using Adam optimization. The Adam optimization function tuned only the value of learning rate α, while the decay and parameters $\beta 1$ and $\beta 2$ used the default value because the learning rate was the most crucial hyperparameter to be tuned [64], while the epsilon and momentum parameters do not have a significant impact on performance [64,67].

We analyzed a significant relative impact of each parameter against the increase in the detection rate of the overall model in Fig. 3. Fig. 3(a) describes the feature importance from all of 1952 models on the NSL-KDD dataset measured from the base value. The base value is taken from the base model with a standard parameter, 1 hidden layer for AE with 60 nodes on the hidden layer. The base Model was fine-tuning with DNN and the ReLU as an activation function. The detection rate result from this base model is used as the base value compared to the detection rate of the overall model. Using the RandomForestRegressor of scikit learning library counts the impact of each parameter against the rate detection value of the model. The cost of the mean decrease detection rate for all normalized hyperparameters shows how the parameter effect significantly reduces the detection rate. The features importance becomes a benchmark of how essential parameters are to the performance of the detection model. The more unimportant a variable or parameter is, the less it affects the value of accuracy.

For the NSL-KDD dataset, Figs. 3(a) shows that the learning rate value dramatically affects the performance metric, followed by the number of hidden layers in the deep structured model. The number of neurons in the second and third layers gives the impact though not significant. The second parameter that has a considerable effect is the number of hidden layers. As for the batch size, it does not have a substantial impact on improving the detection sensitivity for the NSL-KDD dataset. The number of batch sizes was either 32, 64, or 256, and Fig. 4(c) shows that this parameter was not as influential as a result of detection performance. The batch size parameter was more impactful on the speed of learning and the stability of the learning process.

Fig. 3(b) shows the impact of hyperparameter features in CSE-CIC-IDS2018 from a total of 1067 tested models against the value of the detection rate on the base model. The initial model used the AE model with one hidden layer and 40 neurons in the middle layer. In DNN classifier used ReLU as an activation function. The result shows the most influential parameters on the increase and decrease in detection rate is the learning rate value. The second factor is quite influential is the number of neurons in the second hidden layer, as well as the number of hidden layers. The impact sequence of hyperparameter's importance on these two datasets is slightly different, due to differences in features in the raw data of each dataset. For NSL-KDD datasets after preprocessing was dominated by binary data type, while the CSE-CIC-IDS2018 dataset was dominated by continuous data type.

In Figs. 4(a), 5(a) shows the box plot chart of the learning rate in NSL-KDD and CSE-CIC-IDS2018 datasets, respectively. In the NSL-KDD dataset, the default value of 0.01 for the learning rate gives higher performance than other costs. While in the CSE-CIC-IDS2018 dataset, a smaller learning rate value provides better detection value. A higher grade of the learning rate, such in both cases, 0.1, gives poor results in performance. The performance tends to oscillate in a model with a higher learning rate. Even in some cases, the loss value is not decreasing at the beginning of the learning phase. The higher the value of learning rate, the lower the accuracy. In reverse, the smaller the learning rate, the higher the accuracy of the network, with as a consequence, the training process was longer.

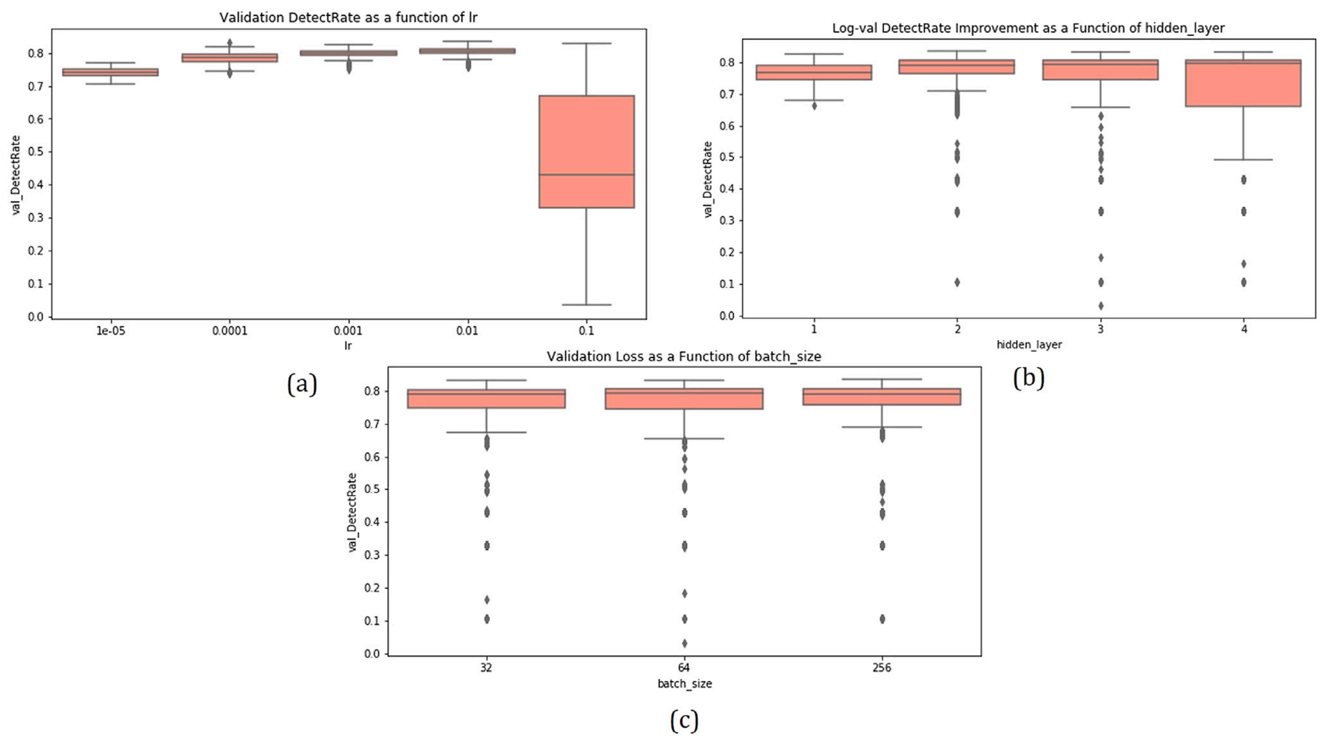


Fig. 4. Effect of each hyperparameter value on rate detection of the overall model in NSL-KDD dataset: (a) effect of the learning rate; (b) effect of the number of hidden layers; (c) effect of batch size amount.

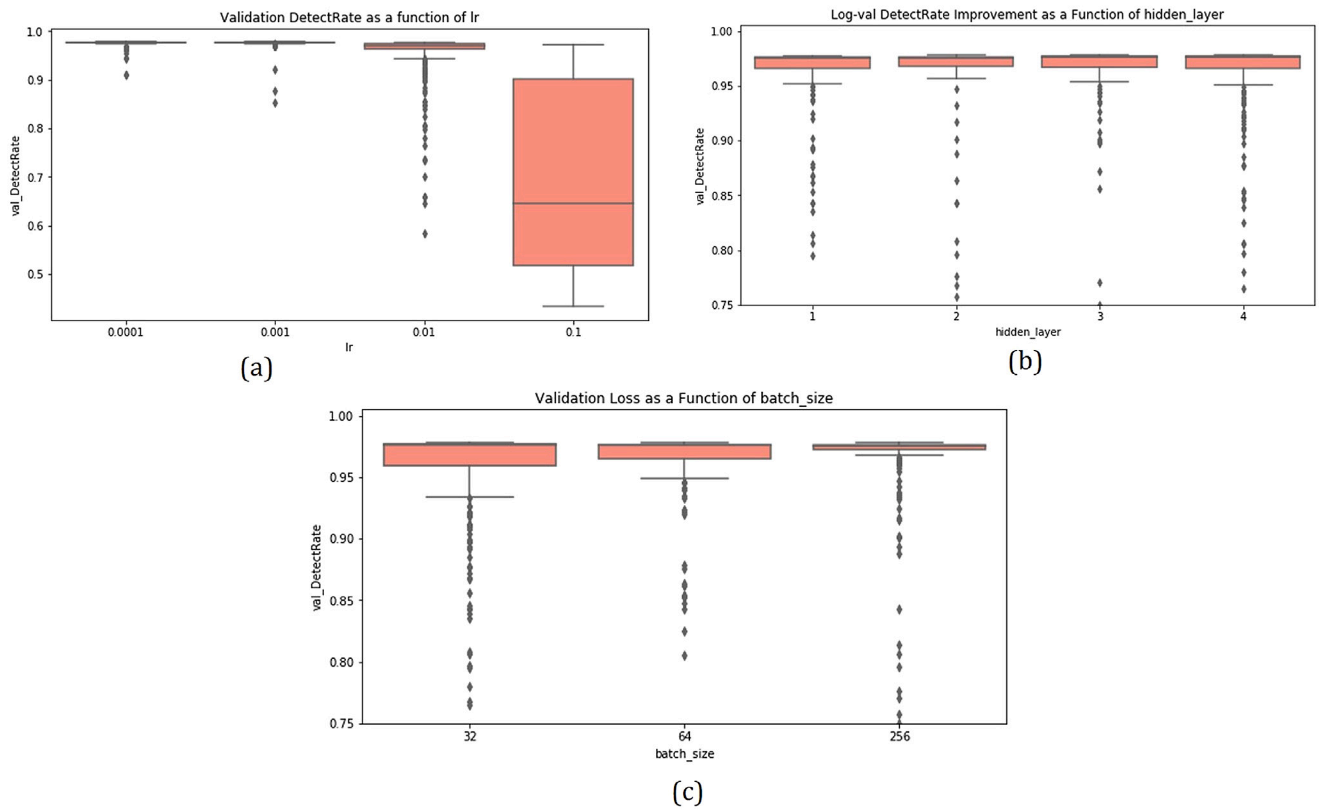


Fig. 5. Effect of each hyperparameter value on rate detection of the overall model in CSE-CIC-IDS2018: (a) effect of the learning rate; (b) effect of the number of hidden layers; (c) effect of batch size amount.

The box plot chart of the number of hidden layers of the overall tested model in both of dataset can be seen in Figs. 4(b) and 5(b). According to the median of performance graph data generated, more layers can improve performance even though the value increases slightly by adding the number of hidden layers to two, three, and four layers. However, even though increasing the number of layers can

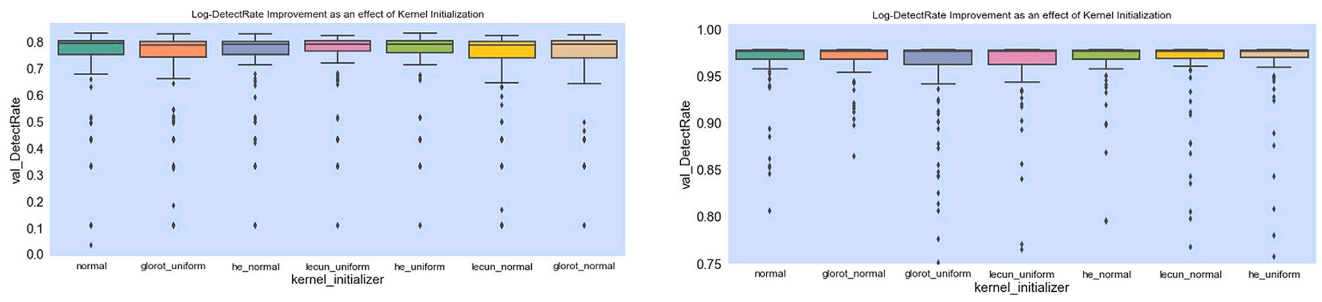


Fig. 6. Impact of kernel initializers against the detection rate of the overall model (a) result in NSL-KDD dataset; (b) result in CSE-CIC-IDS2018 dataset.

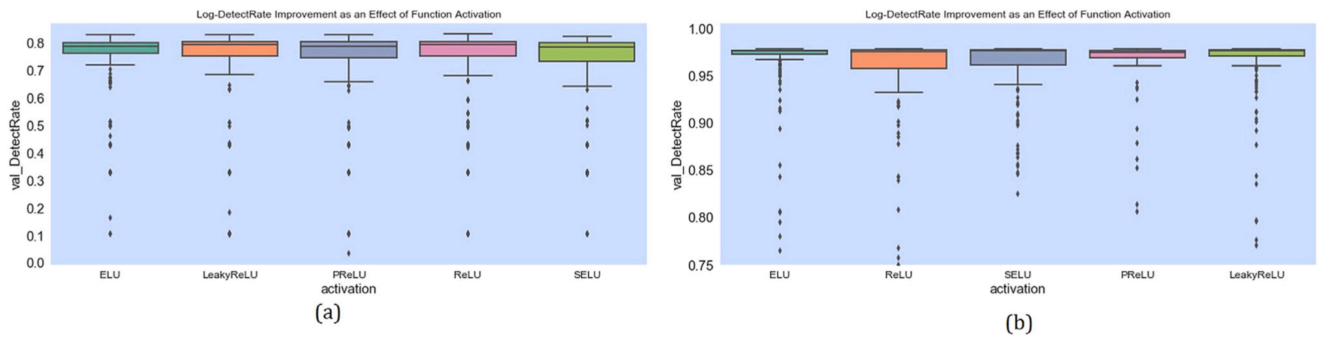


Fig. 7. Impact of activation function against the detection rate of the overall model (a) result in NSL-KDD dataset; (b) result in CSE-CIC-IDS2018 dataset.

improve performance, it does not apply to high learning rate values (0.1), which overall show low detection performance (sensitivity) for all models. The lower quartile data range is also greater at a high learning rate (0.1). The plot shows a correlation between the value of the learning rate and the number of hidden layers. With the addition of hidden layers, the optimization process becomes more complicated. A higher number of hidden layers has a more complex surface error compared to smaller models (the number of hidden layers is lower). Therefore, for more complex models (the number of hidden layers is higher), a lower learning rate is preferable.

The batch size parameter quite contributes to the increased detection rate on the CSE-CIC-IDS2018 dataset, whereas in the The NSL-KDD dataset is not valid. The batch size value 256, in Fig. 5(c), is more robust than the other batch size value. This effect is caused by the amount of data from the CSE-CIC-CIDS2018 dataset is quite large compared to the NSL-KDD dataset, so that a large batch size will give more stable results. Whereas in the NSL-KDD dataset, the batch size only affects the speed of convergence of the model and does not have much effect on the resulting performance.

The categorical parameter, kernel initializer parameter and activation function for both datasets are shown in Figs. 6 and 7. The kernel initializer only affects the convergence speed of the model, while the resulting performance value is almost the same for the entire function. However, on average, the box plot graphs of Fig. 6(a) show that he_uniform and lecun_uniform as kernel initializers function provide a better average value than other functions for NSL-KDD dataset. While in the CSE-CIC-IDS2018 dataset, he_uniform and he_normal kernel initializers provide more stable results. The function of He kernel initializers shows a faster convergent model with an earlier reduction loss in the ReLU activation function and its variant [49].

In the activation function, ELU gives an average value that is more stable than other activation functions in both datasets. In general, function activation ELU provides better performance in classifying. It is possible because ELU is more robust to noise. ELUs have negative values which allow mean activation closer to zero, but with lower computational complexity [68]. However, the ReLU function also shows an average value in the upper quartile for both datasets. In the CSE-CIC-CIDS2018 dataset, the activation Leaky ReLU and PReLU functions appear more stable than ReLU. Leaky ReLU and PReLU have an advantage about the same as the ELU activation function to overcome the "dying ReLU" condition encountered in the ReLU activation function when the gradient value is less than 0. Leaky ReLU and PReLU adaptively learn the parameters of the rectifiers and improves accuracy at a negligible extra computational cost [49].

4.2. Performance comparison

The HPO process was performed by testing 1962 models in NSL-KDD and 1067 models in CSE-CIC-IDS2018. It was combining the number of hidden layers, the number of nodes in each hidden layer, activation function, kernel initialization, and learning rate value. The learning performance evaluation can be shown in the learning curves in Fig. 8. Curves 8(a) and 8(b) exhibit overfitting issues on the NSL-KDD dataset. This case occurs because the model has to learn more than is required for attacks classification in training data. Meanwhile, on CSE-CIC-CIDS2018 in Fig. 8(c) and (d), deep learning models with a combination of DAE for the feature extraction and DNN for classifications show pretty good performance. The plot of learning curves showing between training and testing curves does not happen overfitting. The testing loss curve decrease to the point of stability and have a small gap with the training loss. Although in 150 epoch curves slightly oscillate. It occurred because the model is trained for long.

Tables 5 and 6 show the best deep structured models obtained through HPO for the screening of multiclass attacks, five classes in the NSL-KDD dataset and 15 classes in CSE-CIC-IDS2018 dataset, respectively. Precision, recall, and F1-score parameter metrics are weighted average values, which are the average value of metric weighting by class frequency. The weighted average precision, recall, and F1-score provide better overall performance estimation for imbalanced class data frequencies [69]. The accuracy value in Tables 5 and 6 is the total accuracy value of prediction items from all predicted data. The average accuracy value of each class is less precise for multiclass problems, especially for imbalanced datasets, because the majority class will dominate the results [70,71].

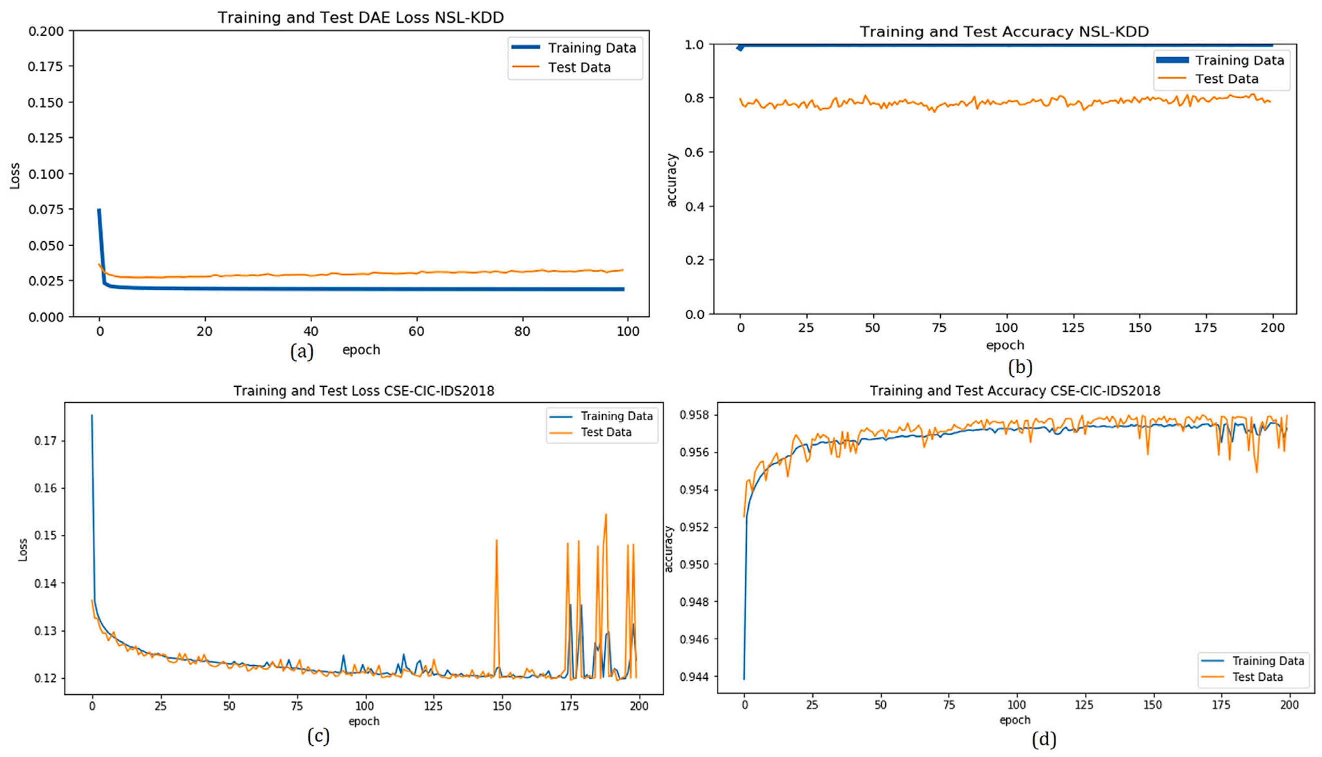


Fig. 8. Evaluation of training and testing data in (a) Optimization learning curves after 100 epoch in NSL-KDD dataset, (b) Performances Learning Curve in NSL-KDD for multi-class classification. (c) Optimization learning curves after 200 epoch in CSE-CIC-IDS2018 dataset, (b) Performances Learning Curve for multi-class classification in CSE-CIC-IDS2018 dataset.

Table 5
The best performance values for the deep structured model with best hyperparameter values on test dataset KDDTest+ in NSL-KDD dataset.

Number of DNN hidden layers	Best hyperparameter					Precision (%)	Recall (%)	Overall Accuracy (%)	F1-Score (%)
	DAE structure	batch size	activation function	kernel initializer	learning rate				
1 layer	122-100–122	64	Leaky ReLU	normal	0.01	84.54	82.35	82.35	81.27
2 layers	122-100-20-100–122	256	ReLU	he_uniform	0.01	86.02	83.33	83.33	82.04
3 layers	122-75-60-30-60-75–122	64	PReLU	he_normal	0.01	84.43	83.05	83.05	82.03
4 layers	122-100-50-30-20-30-50-100-122	32	eLU	normal	0.0001	85.63	83.25	83.25	81.77

The model produced the best performance with the structured model 122-100-20-100-120 for DAE in the NSL-KDD dataset. The activation function on the hidden layer used the ReLU and sigmoid functions in the last layer and binary cross-entropy loss function, whereas the initialization kernel used he_uniform (Table 5). The output of the bottleneck layer became the input to the DNN with the structure 122-100-20-5, using the ReLU function on the hidden layer and softmax on the last layer, kernel initialization using he_uniform, and function loss using categorical cross-entropy and a learning rate is 0.01. Adding the number of layers does not have a significant impact on the detection rate of the IDS model. The ReLU activation function has the best effect on the model with two hidden layers, while the ELU function has the best effect on the model with four hidden layers.

The best learning rate value on models with one, two, and three hidden layers of the DNN is a value of 0.01 in the NSL-KDD dataset. The value of the learning rate is also highly related to the batch size number; a small batch size value is suitable for low learning rate values [47]. A small batch size (32) provides high performance with a small learning rate (0.0001). A large batch size (256) provides the best results with a high learning rate (0.01), as shown in Table 6.

In Table 6, the models on the CSE-CIC-IDS2018 dataset show a slight increase in classifying performance by adding the number of hidden layers to the DL model despite being insignificant. The best learning rate value for a whole model with a different number of hidden layers is 0.001 on this dataset. The Leaky ReLU activation function and lecun_uniform kernel initializer function generate the best classifying levels on the model with 2 and 4 hidden layers. The ReLU activation function provides the best classification level on the 1 hidden layer and 3 hidden layer models. The result in Table 6 also shows a large batch size value of 256 giving higher detection results to the model with more hidden layers.

For multiclass classification, true positive rate (TPR), false-positive rate (FPR) and accuracy are estimated for each class. The detailed results are shown in Table 7 for NSL-KDD and Table 8 for CSE-CIC-IDS2018. The results in Table 7 show the performance of the best models in the KDDTrain+ as training data and KDDTest+ and KDDTest+* as testing data (in KDDTest+*, attack data that were not in the training data were removed). This model was trained using KDDTrain+ data resulting in the weight-average precision value of 99.81%, recall (sensitivity) of 99.81%, accuracy of 99.89%, and F1-score of 99.81%.

Table 6
The best performance values for the deep structured model with best hyperparameter values on testing dataset CSE-CIC-IDS2018 dataset.

Number of DNN hidden layers	Best hyperparameter					Precision (%)	Recall (%)	Overall Accuracy (%)	F1-Score (%)
	DAE structure	batch size	activation function	kernel initializer	learning rate				
1 layer	80-60–80	32	ReLU	lecun_normal	0.001	95.28	95.70	95.70	94.95
2 layer	80-60-35-60–80	32	Leaky ReLU	lecun_uniform	0.001	95.26	95.70	95.70	95.00
3 layer	80-60-30-25-30-60–80	256	Relu	he_normal	0.001	95.38	95.78	95.78	95.08
4 layer	80-70-40-30-25-30-40-70-80	256	Leaky ReLU	lecun_uniform	0.001	95.38	95.79	95.79	95.11

Table 7
Best performance result for the multi-class IDS on the NSL-KDD dataset.

Class	KDDTrain+			KDDTest+			KDDTest+*		
	TPR(%)	FPR(%)	Acc(%)	TPR(%)	FPR(%)	Acc(%)	TPR(%)	FPR(%)	Acc(%)
Normal	99.78	0.16	99.81	96.69	23.69	85.09	96.69	12.71	92.15
DoS	99.98	0.01	99.99	83.94	1.98	93.36	99.36	0.77	99.27
Probe	99.79	0.09	99.90	85.17	2.03	96.60	100.00	1.51	98.58
R2L	95.38	0.03	99.93	38.53	0.05	92.45	48.25	0.02	93.92
U2R	63.46	0.00	99.98	7.00	0.00	99.17	18.92	0.01	99.84

Table 8
Best performance result for the multiclass IDS on the CSE-CIC-IDS2018.

Class	Training Data			Testing data		
	TPR(%)	FPR(%)	Acc(%)	TPR(%)	FPR(%)	Acc(%)
Benign	99.51	4.54	97.47	99.46	4.48	97.48
Bot	99.98	0.00	100.00	99.97	0.00	100.00
Brute Force -Web	68.15	0.01	99.99	60.00	0.00	99.98
Brute Force -XSS	65.24	0.00	99.99	74.42	0.00	99.99
DDOS attack-HOIC	100.00	0.00	100.00	100.00	0.00	100.00
DDOS attack-LOIC-UDP	99.85	0.01	99.99	100.00	0.01	99.99
DDoS attacks-LOIC-HTTP	99.87	0.00	99.99	99.82	0.00	99.98
DoS attacks-GoldenEye	99.97	0.00	100.00	99.93	0.00	100.00
DoS attacks-Hulk	99.99	0.00	100.00	99.99	0.00	100.00
Dos attacks-SlowHTTPTest	51.58	0.43	98.32	51.99	0.43	98.33
DoS attacks-Slowloris	99.98	0.00	100.00	100.00	0.00	100.00
FTP-BruteForce	88.20	1.30	98.32	88.16	1.29	98.33
Infilteration	23.69	0.25	97.48	23.84	0.27	97.50
SQL Injection	42.25	0.00	100.00	43.75	0.00	100.00
SSH-Bruteforce	99.98	0.00	100.00	99.97	0.00	100.00

As for the KDDTest+ dataset, weight-averaged precision value was 86.02%, recall (sensitivity) 83.33%, accuracy 90.09%, and F1-score 82.04%. For the KDDTest+* dataset, the precision, recall, accuracy and F1-score values, respectively, amounted to 92.61%, 91.88%, 94.93%, and 91.01%.

Table 7 also explains the metric values of each attack class to reveal the ability of the model to detect each attack. Root to local and U2R attacks are very difficult to detect. The results for each class of attack shows the U2R class had low TPR or detection results for training (63.46%) and for both data tests (7% and 18.92%). One of the problematic factors making U2R difficult to detect is the lack of data for the U2R class during the learning process, which was only 0.04% of the total training data (Table 1). Details are shown in the confusion matrix in Fig. 9: Due to the low volume of training data from 52 datasets after the learning process, only 33 attacks were recognized, a detection rate of only 63.46%. Sixteen U2R attacks were detected as normal packets (30.7% of 52 data). This shows that the behavior of the data packet of U2R attacks in the network is very close to normal behavior, so it is difficult to detect. User to root attacks such as LoadModule, Perl, ps, SQL, and buffer overflow attacks are deliberately modified to be stealthy to network IDSs. Therefore, during the testing process, it can only recognize 14 U2R attacks from 200 labels with a detection rate of only 7% during data testing. In addition, 81.5% of new attacks (163 in 200 of the testing data) in the class of U2R that do not exist in the training dataset. Its impact resulted, the IDS model failing to generalize U2R attacks.

For the R2L attack class, the TPR for the training data was quite high at 95.38%. However, for the testing data, KDDTest+ and KDDTes+*, TPR were only 38.53% and 48.25%, respectively. In addition to the imbalanced dataset factor (only 0.79% of the total training data), another factor was stealthy R2L attacks. More clearly seen in Fig. 9, the confusion matrix, R2L attacks on both training data and many testing data were detected as normal packages. Even for testing data, approximately 60% of 2,754 of R2L attacks on testing data were identified as normal packages. Another problematic factor in detecting R2L attacks is the many subcategories of new attacks in the testing data for class R2L (namely SendMail, snmpgetattack, snmpguess, worm, xlock, and xsnoop) that are not present in the training data.

The proposed network-anomaly-based IDS model worked very well in detecting DoS and probe attacks with detection rates respectively of 99.98% and 99.79% for training data. For testing data KDDTest+, the detection rate for the two attacks was 83.94% and 85.17%, respectively, slightly reduced due to new subclasses of attack in the testing data that are not in the training data. If the new subclasses of both DoS and probe attacks are eliminated, then the detection rates of the testing process are above 99%. It can be assumed that the proposed model cannot recognize the attack pattern of each class in general. This is because even if the attack is in one class, the attack pattern can be very different from other attacks in its class. The attack subclass will not likely be recognized if the learning or training process is not performed for this attack. For the data of stealthy attacks (attack patterns resembling normal packets), such as U2R and R2L attacks, the anomaly-based IDS models that work by detecting packets in the network are less effective. Each modified attack for a stealthy attack could trick the IDS on the network during the transport, encoding, execution, action, and cleaning processes of the attack. Stealthy attacks avoid unusual behaviors. The goal of a hidden attack is to mimic normal traffic as much as possible. Attacks are usually spread over several sessions and overtime to avoid an abnormal amount of traffic. They often use basic services and prevent unusual commands that could be detected as a normal package. Stealthy attacks more effectively combine the network anomaly-based IDS with an intelligent host-based IDS that works based on the log host [72].

The Overall results of the 15 class attacks classification on the CSE-CIC-IDS2018 dataset show an accurate detection rate. With the addition of the number of hidden layers in DAE and DNN network structure, does not significantly improve performance. For the best test results obtained for models with 4 hidden layers. The highest performance gained by training models with 200 epoch achieved a weight-averaged precision value of 95.38%, recall (sensitivity) of 95.79%, accuracy of 95.79%, and F1-score of 95.11% for testing data. For details on the performance of each attack can be seen in Table 8 and the confusion matrix in Fig. 10.

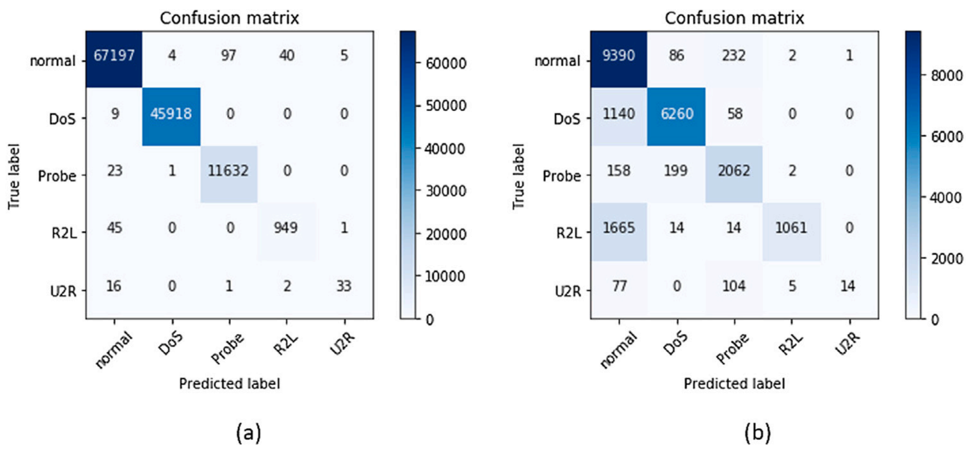


Fig. 9. Confusion matrix for classifying multiclass attacks: (a) training data KDDTrain+; (b) testing data KDDTest+.

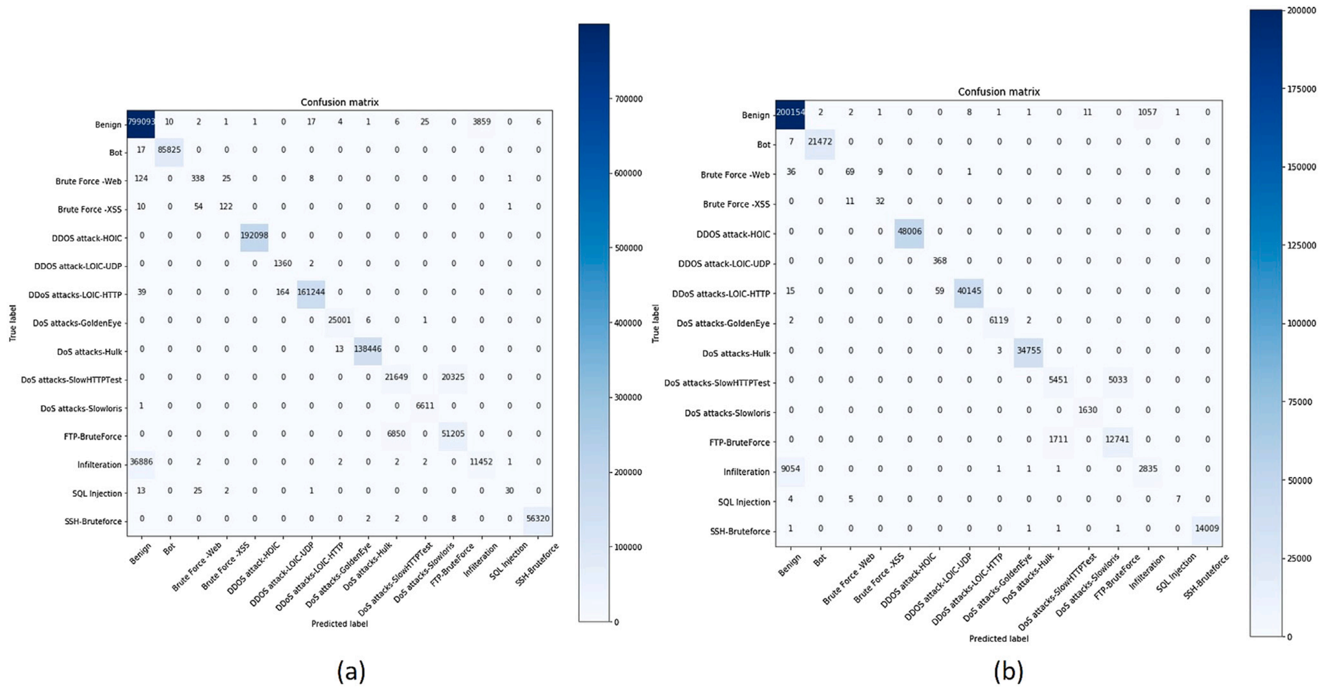


Fig. 10. Confusion matrix for classifying multiclass attacks: (a) training data in CSE-CIC-IDS2018; (b) testing data in CSE-CIC-IDS2018.

The IDS model using deep Autoencoder (PTDAE) combined with a deep neural network (DNN) is very good at detecting attacks with DDoS, DoS, Bot, and Brute Force categories. The detection rate of these types of malicious attacks approaches 100%. For the Infiltration category attack, the proposed IDS model less successfully recognizes the attack, although the training process is upgraded to 200 epochs. The IDS model can only detect 23.69% for this class attack. If viewed on the confusion matrix, most of these infiltration attack packages are known as benign packets. This infiltration attack includes a stealthy attack that utilizes an internal network for illegal access. Hence infiltration attack is hard to detect on the network IDS. Other malicious packets that are difficult to detect on proposed models are web attack categories, including Brute Force Web, brute force-XSS, and SQL injection. For this category of attacks, the detection is quite low $< 70\%$. The main factor of the low detection rate is the lack of training data for this attack classes.

4.3. Comparison with other methods

This section provides a comparison of the DAE feature extraction in the pre-training phase with other algorithms. The alternatives feature extraction models are autoencoder (AE) and stack autoencoder (SAE). The performance result is shown in Table 9 and Fig. 11. In other evaluation models, we use the network structure and best hyperparameter of the DAE + DNN model to construct the SAE + DNN model. In the CSE-CIC-IDS2018 best model has a DAE structure 80-70-40-30-25-30-40-70-80. After the extracted, it will be transferred to DNN with a network structure of 80-70-40-30-25-15. With the same DNN structure that will extract feature with four levels SAE with the structure 80-70-80, 70-40-70, 40-30-40, and 30-25-30. The result will get the same DNN structure that is trained with the value of the same hyperparameter as the PTDAE + DNN model. Our AE + DNN structure use network AE with structure 80-25-80 for the feature extraction. DNN's structure is trained with the addition of a hidden layer number to 80-25-20-20-20-15. The number of hidden layers in the DNN follows the [73] approach.

A comparison of some of the methods of feature extraction demonstrated the PTDAE model combine with DNN, resulting in a better classifying multiclass performance compared to AE + DNN and SAE + DNN for both datasets. The detection rate performances of models using SAE and DAE feature extraction almost identical, especially on

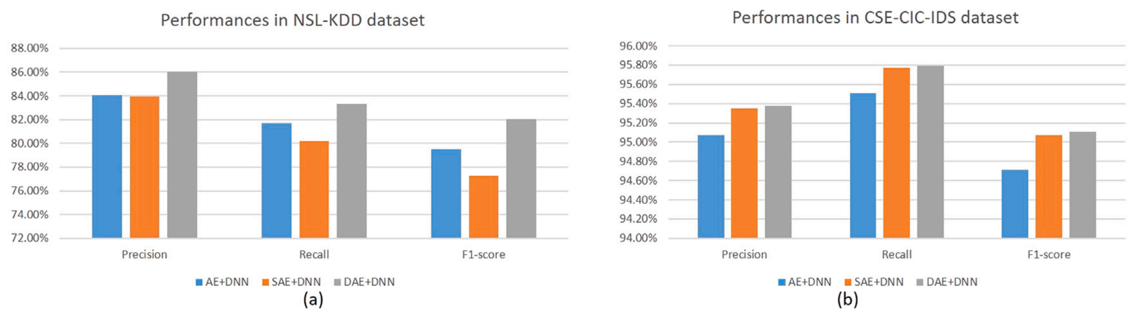


Fig. 11. Grafik Chart multiple techniques autoencoder feature extraction and DNN for multiclass classification.

Table 9
Test result from various techniques autoencoder feature extraction and DNN for multiclass classification.

Algorithm	Precision (%)	Recall (%)	F1-score (%)	Training time (s)	Prediction time (s)
Multi-class Classification in NSL-KDD					
AE+DNN	84.04	81.69	79.49	189.62	0.51
SAE+DNN	83.95	80.17	77.28	469.626	1.104
DAE+DNN	86.02	83.34	82.04	382.48	0.968
Multi-class Classification in CSE-CIC-IDS2018					
AE+DNN	95.07	95.51	94.71	5608.79	11.685
SAE+DNN	95.35	95.77	95.07	4281.99	12.510
DAE+DNN	95.38	95.79	95.11	4162.30	12.582

Table 10
Comparison between the proposed model and state of the art Method NSL-KDDTest.

Method	NSL-KDDTest			
	Precision (%)	Recall (%)	Overall Accuracy (%)	F1-score (%)
Character level +CNN [74]	81.45	79.05	79.05	76.03
Feedforward Neural Network [39]	–	80.30	80.30	–
RNN [18]	83.07	81.29	81.29	79.25
Cond. VAE [76]	81.59	80.10	80.10	79.08
DNN [42]	81.00	78.50	78.50	76.50
RNN with multilayered echo-state machine (ML-ESM) [75]	81.89	80.41	80.41	79.23
Proposed (DAE+DNN)	86.02	83.33	83.33	82.04

the CSE-CIC-IDS2018 dataset that does not have overfitting. However, the training time of SAE with the number of hidden layers for fewer data like in the NSL-KDD dataset takes a little longer.

We also compare the results of our proposed model with recent related studies that used various DL in the NSL-KDD dataset (Table 10). Our proposed model is superior to all models in Table 10 with recall (sensitivity) performance of 83.33% and overall accuracy of 83.33%. Lin et al. [74], who used a CNN for a character-level IDS, achieved only 79.05% for recall and accuracy for the KDDTest+ dataset. Zhu et al. [39] developed network anomaly detection and identification using a feedforward neural network model and CNN model with highest accuracy of 80.3%. Yin et al. [18] used an RNN method with recall rate and accuracy of 81.29%. Likewise, our results are compared to Vinayakumar et al. [42] who used DNN methods, and Tchakoucht and Ezziyyani [75], who developed an IDS with an RNN with multilayered echo-state machine, achieving recall rate and accuracy, respectively, of 78.05% and 80.41%. However, compared to other models, our results are still higher. For the F1-score value, which is the weighted average of precision and recall, our model remains superior to the others with 82.04%.

The enhancing detection sensitivity of data testing in proposed PTDAE combined with DNN models due to the fit selection of hyperparameters and a more fitting structure of the DL model. By performing the feature extraction in input data with DAE and a suitable combination of activation functions, the value of learning rate, as well as a kernel initializer and a fitting network structure can improve the detection results in the test data. However, in the case of the NSL-KDD dataset with a lot of controversies in overfitting problem, it does not mean that the best learning performance in learning will be better performance on data testing. Although models with more number of hidden layers give result in better detection performance during learning, do not apply when testing. So the best model here prioritizes models that provide the most excellent results during testing.

The CSE-CIC-IDS2018 dataset does not provide standard datasets for training and testing. Researchers use training and testing with their own compositions that will significantly affect the overall outcome. It is becoming a significant differentiator when comparing results with other works, for example, when the structure of the unbalanced and challenging prediction class of the original datasets could significantly change using a different test set. But our results are overall good enough with the detection rate of 95.79% for the overall 15 classes without the change of data composition in imbalanced class as Lin et al. [77].

The main problem with the development of models using HPO is the computational cost required to evaluate many models to get the right hyperparameter composition. The Hyperparameter tuning with reinforcement learning (RL) approach with the ability of an agent that can be explored hyperparameter selection policy can be an alternative [78]. Lopez-Martin et al. [46] use RL in the binary classification attack system in the NSL-KDD dataset, demonstrating the excellence of RL that only takes 290.50 s for training and 0.54 s for prediction. In contrast, on our models using the HPO, It takes $>$ 56 h to evaluate learning in the entire of models and time prediction of 0.968 s. However, the HPO approach that we used can further explain the impact of hyperparameter selection when the loss function showing improved performance compared to the RL method such as a puzzled.

Nevertheless, the results of the experiments conducted in models that we proposed have not optimally detected some attacks class, such as Infiltration attack. The lack of detection rate in this class attack reflected the affecting of the raw data features of the dataset itself. The features of raw data in the dataset used are less represents the characteristics of these types of attacks. Adding features such as the prediction feature, which is done by Yu et al. [79], can be considered for IDS performance enhancement for the attack class in future work.

5. Conclusion

As a summary, this research gives some contributions to deep learning in the cybersecurity domain: (1) We proposed IDS model, using DL and a DAE for the pretraining process and fine-tuning with DNN through the process of HPO, improves the results of attack classification in intrusion detection. (2) With the automatic tuning of HPO performed can show the important parameters that have a significant impact on the DL IDS model. (3) The result from the best models produced provides values and functions that can be used as base values or base functions in the development of deep learning models with other approaches and datasets. (4) The optimal feature extraction approach to develop an effective DL IDS.

We provide automatic selecting hyperparameters and DL structure models to increase the classification of DL IDS. Regarding the deep structured model, the selection process of the number of hidden layers, number of neurons, the value of learning rate, kernel initialization, activation function, an optimization technique is complicated. Automatic HPO techniques help determine the most appropriate combination of hyperparameters to obtain the best performance. One of the factors that should be considered in the HPO process is that when a higher number of hyperparameters are optimized simultaneously, the process takes a longer time. The random search in the hyperparameter combination grid accelerates the process of obtaining the best model that provides the best attack detection rate performance. From the HPO process performed on the NSL-KDD dataset and the CSE-CIC-IDS2018 shows the learning rate parameter is an essential parameter that significantly impacts the deep learning IDS performance. Besides, the other influential parameters are the selection of deep learning structures, namely the number of layers and neurons of the DL model.

In addition, we compare several methods in feature extraction with different Autoencoder models (AE, SAE, and DAE) combined with the DNN model. We considering, considering some features: (1) precision, (2) recall, (3) F1-Score, (4) training and (5) prediction times, and using two different intrusion detection datasets (NSL-KDD and CSE-CI-IDS2018) to facilitate the generalization of the results. The best DAE+DNN architecture of the intrusion detection model that we have proposed produces better performance compared to some of the recent related studies using various DL.

In the future, we plan to compare the DL IDS models with other combinations of DL algorithms, various methods of feature extraction, and other datasets. We will also evaluate techniques for improving performance on the imbalanced attack dataset.

CRediT authorship contribution statement

Yesi Novaria Kunang: Software, Visualization, Investigation, Writing - original draft. **Siti Nurmaini:** Supervision, Conceptualization, Methodology, Writing - review & editing. **Deris Stiawan:** Data curation, Validation, Writing - review & editing. **Bhakti Yudho Suprapto:** Validation, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research has received funding from Indonesia Ministry of Research, Technology, and Higher Education (grant agreement 170/SP2H/LT/DRPM/2020). The authors would like to thank data science research group, Universitas Bina Darma and University for support and facilities.

References

[1] Habeeb RAA, Nasaruddin F, Gani A, Hashem IAT, Ahmed E, Imran M. Real-time big data processing for anomaly detection: A survey. Int J Inf Manage 2019;45:289–307.
[2] Díaz López D, Blanco Uribe M, Santiago Cely C, Vega Torres A, Moreno Guataquira N, Morón Castro S, et al. Shielding IoT against cyber-attacks: An event-based approach using SIEM. Wirel Commun Mobile Comput 2018;2018.
[3] Sicari S, Rizzardi A, Miorandi D, Coen-Porisini A. REATO: REActing TO Denial of Service attacks in the Internet of Things. Comput Netw 2018;137:37–48.
[4] Jang J, Jung IY, Park JH. An effective handling of secure data stream in IoT. Appl Soft Comput 2018;68:811–20.
[5] Stiawan D, Idris M, Malik RF, Nurmaini S, Alsharif N, Budiarto R, et al. Investigating brute force attack patterns in IoT network. J Electr Comput Eng 2019;2019.
[6] Fu Y, Yan Z, Cao J, Koné O, Cao X. An automata based intrusion detection method for internet of things. Mob Inf Syst 2017;2017.
[7] Mishra P, Varadharajan V, Tupakula U, Pilli ES. A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Commun Surv Tutor 2018;21(1):686–728.
[8] Masduki BW, Ramli K, Saputra FA, Sugiarto D. Study on implementation of machine learning methods combination for improving attacks detection accuracy on intrusion detection system (IDS). In: 2015 International conference on quality in research. IEEE; 2015, p. 56–64.
[9] Mohammadi M, Al-Fuqaha A, Sorour S, Guizani M. Deep learning for IoT big data and streaming analytics: A survey. IEEE Commun Surv Tutor 2018;20(4):2923–60.
[10] Dong B, Wang X. Comparison deep learning method to traditional methods using for network intrusion detection. In: 2016 8th IEEE international conference on communication software and networks. IEEE; 2016, p. 581–5.
[11] Xie J, Song Z, Li Y, Zhang Y, Yu H, Zhan J, et al. A survey on machine learning-based mobile big data analysis: Challenges and applications. Wirel Commun Mobile Comput 2018;2018.
[12] L'heureux A, Grolinger K, Elyamany HF, Capretz MA. Machine learning with big data: Challenges and approaches. IEEE Access 2017;5:7776–97.
[13] Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E. Deep learning applications and challenges in big data analytics. J Big Data 2015;2(1):1.
[14] Feurer M, Hutter F. Hyperparameter optimization. In: Hutter F, Kotthoff L, Vanschoren J, editors. Automated machine learning: methods, systems, challenges. Springer International Publishing; 2019, p. 3–33. http://dx.doi.org/10.1007/978-3-030-05318-5_1.
[15] Kunang YN, Nurmaini S, Stiawan D, Zarkasi A, Jasmir F. Automatic features extraction using autoencoder in intrusion detection system. In: 2018 international conference on electrical engineering and computer science. IEEE; 2018, p. 219–24.
[16] Al-Qatf M, Lasheng Y, Al-Habib M, Al-Sabahi K. Deep learning approach combining sparse autoencoder with SVM for network intrusion detection. IEEE Access 2018;6:52843–56.
[17] Mohaimenuzzaman M, Abdallah ZS, Kamruzzaman J, Srinivasan B. Effect of hyper-parameter optimization on the deep learning model proposed for distributed attack detection in internet of things environment. 2018, arXiv:1806.07057.
[18] Yin C, Zhu Y, Fei J, He X. A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access 2017;5:21954–61.
[19] Melis G, Dyer C, Blunsom P. On the state of the art of evaluation in neural language models. 2017, arXiv:1707.05589.
[20] Bergstra J, Bengio Y. Random search for hyper-parameter optimization. J Mach Learn Res 2012;13(Feb):281–305.
[21] Bergstra J, Yamins D, Cox DD. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. JMLR; 2013.
[22] Hutter F, Hoos H, Leyton-Brown K. An efficient approach for assessing hyperparameter importance. In: International conference on machine learning. 2014. p. 754–62.
[23] Al-Garadi MA, Mohamed A, Al-Ali A, Du X, Guizani M. A survey of machine and deep learning methods for internet of things (IoT) security. 2018, arXiv:1807.11023.
[24] Xin Y, Kong L, Liu Z, Chen Y, Li Y, Zhu H, et al. Machine learning and deep learning methods for cybersecurity. IEEE Access 2018;6:35365–81.
[25] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature 2015;521(7553):436–44.
[26] Li H, Ota K, Dong M. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. IEEE Netw. 2018;32(1):96–101.
[27] Zhang Y, Li P, Wang X. Intrusion detection for IoT based on improved genetic algorithm and deep belief network. IEEE Access 2019;7:31711–22.
[28] Kang M-J, Kang J-W. Intrusion detection system using deep neural network for in-vehicle network security. PLoS One 2016;11(6):e0155781.
[29] Potluri S, Diedrich C. Accelerated deep neural networks for enhanced intrusion detection system. In: 2016 IEEE 21st international conference on emerging technologies and factory automation. IEEE; 2016, p. 1–8.
[30] Dawoud A, Shahristani S, Raun C. Deep learning and software-defined networks: towards secure Iot architecture. Internet Things 2018;3:82–9.
[31] Thing VL. IEEE 802.11 network anomaly detection and attack classification: A deep learning approach. In: 2017 IEEE wireless communications and networking conference. IEEE; 2017, p. 1–6.
[32] Lopez-Martin M, Carro B, Sanchez-Esguevillas A, Lloret J. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. IEEE Access 2017;5:18042–50.
[33] Rezvy S, Petridis M, Lasebae A, Zebin T. Intrusion detection and classification with autoencoded deep neural network. In: International conference on security for information technology and communications. Springer; 2018, p. 142–56.

[34] Muna A-H, Moustafa N, Sitnikova E. Identification of malicious activities in industrial internet of things based on deep learning models. J Inf Secur Appl 2018;41:1–11. http://dx.doi.org/10.1016/j.jisa.2018.05.002.

[35] Yousefi-Azar M, Varadharajan V, Hamey L, Tupakula U. Autoencoder-based feature learning for cyber security applications. In: 2017 international joint conference on neural networks. IEEE; 2017, p. 3854–61.

[36] 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.

[37] Yu Y, Long J, Cai Z. Session-based network intrusion detection using a deep learning architecture. In: International conference on modeling decisions for artificial intelligence. Springer; 2017, p. 144–55.

[38] Yoo Y. Hyperparameter optimization of deep neural network using univariate dynamic encoding algorithm for searches. Knowl-Based Syst 2019;178:74–83.

[39] Zhu M, Ye K, Xu C-Z. Network anomaly detection and identification based on deep learning methods. In: International conference on cloud computing. Springer; 2018, p. 219–34.

[40] Aldwairi T, Perera D, Novotny MA. An evaluation of the performance of Restricted Boltzmann machines as a model for anomaly network intrusion detection. Comput Netw 2018;144:111–9.

[41] Kanimozhi V, Jacob TP. Artificial intelligence based network intrusion detection with hyper-parameter optimization tuning on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing. In: 2019 international conference on communication and signal processing. IEEE; 2019, p. 0033–6.

[42] Vinayakumar R, Alazab M, Soman K, Poornachandran P, Al-Nemrat A, Venkatraman S. Deep learning approach for intelligent intrusion detection system. IEEE Access 2019;7:41525–50.

[43] Torres J, Gutiérrez-Avilés D, Troncoso A, Martínez-Álvarez F. Random hyper-parameter search-based deep neural network for power consumption forecasting. In: International work-conference on artificial neural networks. Springer; 2019, p. 259–69.

[44] Sanders S, Giraud-Carrier C. Informing the use of hyperparameter optimization through metalearning. In: 2017 IEEE international conference on data mining. IEEE; 2017, p. 1051–6.

[45] Yao C, Cai D, Bu J, Chen G. Pre-training the deep generative models with adaptive hyperparameter optimization. Neurocomputing 2017;247:144–55.

[46] Lopez-Martin M, Carro B, Sanchez-Esguevillas A. Application of deep reinforcement learning to intrusion detection for supervised problems. Expert Syst Appl 2020;141:112963.

[47] Bengio Y. Practical recommendations for gradient-based training of deep architectures. In: Neural networks: Tricks of the trade. Springer; 2012, p. 437–78.

[48] EIbrahim L, Mohamed ZE. Improving error back propagation algorithm by using cross entropy error function and adaptive learning rate. Int J Comput Appl 2017;61(8).

[49] He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. 2015. p. 1026–34.

[50] Kotu V, Deshpande B. Deep learning. In: Data science. Elsevier; 2019, p. 307–42. http://dx.doi.org/10.1016/B978-0-12-814761-0.00010-1.

[51] Talos. Introduction – Talos User Manual. URL https://autonomio.github.io/docs_talos/.

[52] Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the KDD CUP 99 data set. In: 2009 IEEE symposium on computational intelligence for security and defense applications. IEEE; 2009, p. 1–6.

[53] University of New Bruncswick CIfC. A realistic cyber defense dataset (CSE-CIC-IDS2018).

[54] Sharafaldin I, Lashkari AH, Ghorbani AA. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSP. 2018. p. 108–16.

[55] Potdar K, Pardawala TS, Pai CD. A comparative study of categorical variable encoding techniques for neural network classifiers. Int J Comput Appl 2017;175(4):7–9.

[56] Wang Z. Deep learning-based intrusion detection with adversaries. IEEE Access 2018;6:38367–84.

[57] Japkowicz N, Shah M. Evaluating learning algorithms: a classification perspective. Cambridge University Press; 2011.

[58] Kumar G. Evaluation metrics for intrusion detection systems-a study. Int J Comput Sci Mobile Appl 2014;2(11).

[59] Moustafa N, Hu J, Slay J. A holistic review of Network Anomaly Detection Systems: A comprehensive survey. J Netw Comput Appl 2019;128:33–55.

[60] Hodo E, Bellekens X, Hamilton A, Tachtatzis C, Atkinson R. Shallow and deep networks intrusion detection system: A taxonomy and survey. 2017, arXiv:1701.02145.

[61] Kim K, Aminanto ME, Tanuwidjaja HC. Deep learning. In: Network intrusion detection using deep learning. Springer; 2018, p. 27–34.

[62] Hindy H, Brosset D, Bayne E, Seeam A, Tachtatzis C, Atkinson R, et al. A taxonomy and survey of intrusion detection system design techniques, network threats and datasets. 2018, arXiv:1806.03517.

[63] Stathakis D. How many hidden layers and nodes? Int J Remote Sens 2009;30(8):2133–47.

[64] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press; 2016.

[65] Yan B, Han G. Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system. IEEE Access 2018;6:41238–48.

[66] Mendoza H, Klein A, Feurer M, Springenberg JT, Urban M, Burkart M, et al. Towards automatically-tuned deep neural networks. In: Automated machine learning. Springer; 2019, p. 135–49.

[67] Kingma DP, Ba J. Adam: A method for stochastic optimization. 2014, arXiv:1412.6980.

[68] Clevert D-A, Unterthiner T, Hochreiter S. Fast and accurate deep network learning by exponential linear units (elus). 2015, arXiv preprint arXiv:1511.07289.

[69] Jeni LA, Cohn JF, De La Torre F. Facing imbalanced data–recommendations for the use of performance metrics. In: 2013 humaine association conference on affective computing and intelligent interaction. IEEE; 2013, p. 245–51.

[70] Dong Y, Guo H, Zhi W, Fan M. Class imbalance oriented logistic regression. In: 2014 international conference on cyber-enabled distributed computing and knowledge discovery. IEEE; 2014, p. 187–92.

[71] Rahman MM, Davis D. Addressing the class imbalance problem in medical datasets. Int J Mach Learn Comput 2013;3(2):224.

[72] Haider W, Creech G, Xie Y, Hu J. Windows based data sets for evaluation of robustness of host based intrusion detection systems (IDS) to zero-day and stealth attacks. Future Internet 2016;8(3):29.

[73] Shibata K, Ikeda Y. Effect of number of hidden neurons on learning in large-scale layered neural networks. In: 2009 ICCAS-SICE. IEEE; 2009, p. 5008–13.

[74] Lin SZ, Shi Y, Xue Z. Character-level intrusion detection based on convolutional neural networks. In: 2018 international joint conference on neural networks. IEEE; 2018, p. 1–8.

[75] Tchakoucht TA, Ezziyyani M. Multilayered echo-state machine: A novel architecture for efficient intrusion detection. IEEE Access 2018;6:72458–68.

[76] Lopez-Martin M, Carro B, Sanchez-Esguevillas A, Lloret J. Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in iot. Sensors 2017;17(9):1967.

[77] Lin P, Ye K, Xu C-Z. Dynamic network anomaly detection system by using deep learning techniques. In: International conference on cloud computing. Springer; 2019, p. 161–76.

[78] Jomaa HS, Grabocka J, Schmidt-Thieme L. Hyp-rl: Hyperparameter optimization by reinforcement learning. 2019, arXiv preprint arXiv:1906.11527.

[79] Yu W, Wang Y, Song L. A two stage intrusion detection system for industrial control networks based on ethernet/IP. Electronics 2019;8(12):1545.