A Convolutional Neural Network for Improved Anomaly-Based Network Intrusion Detection

Isra Al-Turaiki¹,*,i and Najwa Altwaijry²

Abstract
Cybersecurity protects and recovers computer systems and networks from cyber attacks. The importance of cybersecurity is growing commensurately with people’s increasing reliance on technology. An anomaly detection-based network intrusion detection system is essential to any security framework within a computer network. In this article, we propose two models based on deep learning to address the binary and multiclass classification of network attacks. We use a convolutional neural network architecture for our models. In addition, a hybrid two-step preprocessing approach is proposed to generate meaningful features. The proposed approach combines dimensionality reduction and feature engineering using deep feature synthesis. The performance of our models is evaluated using two benchmark data sets, namely the network security laboratory-knowledge discovery in databases data set and the University of New South Wales Network Based 2015 data set. The performance is compared with similar deep learning approaches in the literature, as well as state-of-the-art classification models. Experimental results show that our models achieve good performance in terms of accuracy and recall, outperforming similar models in the literature.

Keywords: convolutional neural network; cybersecurity; machine learning; network intrusion detection system; NSL-KDD; UNSW-NB15

Introduction
Network intrusion detection systems (NIDS) are essential information security tools. They are used to detect malicious activities in computer networks. An NIDS is designed to monitor traffic traveling through the network. When an attack or a violation is detected, an NIDS raises an alert to notify the network administrator to take the proper action.¹ It is important for organizations to have effective network security measures in place to protect their valuable information, business reputation, and continuity. Combined with other traditional security tools, such as firewalls, access control systems, and antivirus software, NIDS are used to protect information and communication systems against attacks.²

In the literature, NIDS are classified into either signature-based NIDS (SNIDS) or anomaly detection-based NIDS (ADNIDS).³ SNIDS, also called misuse detectors, are used when there exists a list of predefined attacks. They work by checking the traffic against the existing attack list. Research suggests that SNIDS are effective for detecting known attacks and have less false-alarm rates than ADNIDS.³

However, when it is required to detect new types of attacks, SNIDS becomes less effective.³ This problem is solved by switching to ADNIDS, which are able to detect unusual traffic patterns. ADNIDS are more effective at detecting attacks that have not been previously observed. While these systems have a higher false-positive rate than SNIDS, they have gained wide acceptance in the research community on the grounds that they are theoretically capable of detecting new forms of attacks.³

For an NIDS to detect intrusions, it considers network traffic-related features, such as duration, source address, protocol, and flag. It should then detect attacks accurately with a minimum of false alarms. In general, network intrusion detection can be formulated as a
classification problem. An NIDS might classify a connection as normal or as an attack, called binary classification, or alternatively, classify a connection as normal or attack, while specifying the type of attack, called multiclass classification.

Research in intrusion detection systems (IDS) began in the 1980s, and ever since many algorithms have been used to build ADNIDS. Traditional machine learning algorithms such as random forests (RF), self-organized maps, support vector machines (SVM), and artificial neural networks (ANN) have been widely used in developing ADNIDS. However, as data sets are evolving in terms of size and type, traditional machine learning algorithms become increasingly unable to cope with real-world network application environments.1

Despite several decades of research and applications in IDS, there are still many challenges to be addressed. In particular, better detection accuracy, reduced false-positive rates, and the ability to detect unknown attacks are all required.4,5 Recently, researchers have effectively employed deep learning-based methods in a range of applications, including image recognition,6 emotion detection,7 and handwritten-character recognition.8 Deep learning has the ability to identify better representations from raw data,9 compared with traditional machine learning approaches.

The key motivation for this work is to address these challenges. An effective model is a function of both a robust machine learning algorithm and a representative data set with relevant features. We aim to overcome the shallow learning problem via developing a deep learning-based model for ADNIDS, with the objective of classifying novel attacks by examining the structures of normal behavior in network traffic, while improving detection accuracy and reducing the false-positive rate. Feature engineering is the process of creating new and meaningful features from raw ones. The goal of feature engineering is to exploit the most relevant features for improved model performance. We investigate feature engineering compared with a simple two-dimensional (2D) representation of hand-crafted features. In addition, we wish to compare the efficacy of two deep learning approaches, namely fully connected deep neural networks (DNNs) and convolutional neural networks (CNNs), at detecting network attacks, to establish the appropriate approach for ADNIDS.

This research presents the following contributions to the literature: (1) a hybrid two-step preprocessing approach that combines dimensionality reduction and feature engineering using deep feature synthesis (DFS),10 (2) a novel binary classification ADNIDS based on DNNs, (3) a novel multiclass classification ADNIDS based on DNNs for network intrusion detection, and (4) to the best of our knowledge, this is the first work studying the efficacy of skip connections to augment network architecture for anomaly detection, and finally, (5) this article presents a comparison of deep architectures for ADNIDS. We use a CNN architecture, and we evaluate the performance of our proposed models using two benchmark data sets: the network security laboratory-knowledge discovery in databases (NSL-KDD)11 and University of New South Wales Network Based 2015 (UNSW-NB15)12 data sets. We compare our results with similar deep learning approaches and state-of-the-art classification models. Our proposed models achieved high accuracy and precision values, outperforming other models in the literature, and are designed to deal with the binary and multiclass classification problems.

The rest of the article is organized as follows: The Related Work section covers the previous studies in the area of anomaly detection using deep learning and machine learning techniques. In the Methods section, an overview of our proposed models for ADNIDS is presented. In the Results section, experimental settings and performance measures are outlined, and then, we present our performance evaluation. Our conclusions and planning for future work are provided in the Conclusion section. We provide a list of acronyms used in this article in Table 1. The Appendix A1 presents some background information on preliminaries and deep learning methods.

### Table 1. Nomenclature of abbreviations

| Acronym | Definition |
|---------|------------|
| ADNIDS  | Anomaly detection-based intrusion detection system |
| ANN     | Artificial neural networks |
| BFGS    | Broyden–Fletcher–Goldfarb–Shannon |
| CNN     | Convolutional neural network |
| DFS     | Deep feature synthesis |
| DNN     | Deep neural network |
| FC      | Fuzzy clustering |
| GRU     | Gated recurrent unit |
| LSTM    | Long short-term memory |
| MNB     | Modified Naive Bayes |
| NDAE    | Nonsymmetric deep autoencoder |
| NIDS    | Network intrusion detection system |
| NSL-KDD | Network security laboratory |
| PCA     | Principal component analysis |
| ReLU    | Rectified linear unit |
| RF      | Random forests |
| RNN     | Recurrent neural networks |
| SNIDS   | Signature-based network intrusion detection system |
| SOM     | Self-organized maps |
| STL     | Self-taught learning |
| SVM     | Support vector machines |
Related Work
NIDS play an important role in network security by monitoring traffic traveling between all devices on the network. The problem of identifying abnormal network traffic has been widely studied in the literature, and many machine learning algorithms have been used, such as Naive Bayes (NB), ANN, and fuzzy clustering, SVM.

Although traditional machine learning techniques have been widely used to detect network attacks, they still require significant preprocessing. Such algorithms require feature engineering and assume the availability of handcrafted features. However, with fast-paced technological advancements, the size of everyday data sets available to organizations is growing. Thus, shallow learning with traditional machine learning algorithms may not be suitable to deal with real-world environments since it relies on high levels of human involvement in data preparation. In addition, these techniques have the disadvantage of low detection accuracy. Deep learning has emerged recently and demonstrated success in many real-world problems. It has the ability to automatically capture features and correlations in large data sets.

Aldweesh et al. presented a survey on deep learning approaches for anomaly-based intrusion detection systems, including a taxonomy of various IDSs and future research directions. A review of IDS based on deep learning approaches by Ferrag et al. also presented the various data sets used in NIDS and the performance of seven deep learning models under two new real traffic data sets.

A highly scalable and hybrid DNN framework called scale-hybrid-IDS-AlertNet was proposed in the study by Vinayakumar et al. The framework may be used in real time to effectively monitor network traffic to alert system administrators to possible cyber attacks. It was composed of a distributed deep learning model with DNNs for handling and analyzing very large-scale data in real time. The authors tested the framework on various data sets, including NSL-KDD and KDD’99. On NSL-KDD, the best F-Measure for binary classification was 80.7% and 76.5% for multiclass classification.

A DNN model was proposed by Tang et al. to detect anomalies in software-defined networking context. Basically, a simple DNN was used with an input layer, three hidden layers, and an output layer. Training was conducted using the NSL-KDD data set. With six features, the model achieved an accuracy value of 75.75% in the binary classification problem.

A self-taught learning (STL) deep learning model for network intrusion detection was proposed by Javaid et al. The first component of the model was the unsupervised feature learning, in which a sparse autoencoder was used to obtain feature representation from a large unlabeled data set. Then, the second component was an ANN classifier that used Softmax regression classification. Using the NSL-KDD data set, the model obtained accuracy values of 88.39% and 79.10% for two-class and five-class classification, respectively.

Yin et al. proposed a model for intrusion detection using recurrent neural networks (RNNs). RNNs are especially suited to data that are time dependent. The model consisted of forward and back propagation stages. Forward propagation calculates the output values, whereas back propagation passes residuals accumulated to update the weights. The model consisted of 20 hidden nodes, with Sigmoid as the activation function and Softmax as the classification function. The learning rate was set to 0.1, and the number of epochs to 50. Experimental results using the NSL-KDD data set showed the accuracy values were 83.28% and 81.29% for binary and multiclass classification, respectively.

Deep learning and traditional machine learning can be hybridized to improve intrusion detection accuracy. A combination of sparse autoencoder and SVM was proposed by Al-Qatf et al. The sparse autoencoder was used to capture the input training data set, whereas the SVM was used to build the classification model. The model was trained and evaluated using the NSL-KDD data set. The obtained accuracy values were 84.96% and 80.48% for two-class and five-class classification, respectively. These results outperform the performance of traditional methods, such as J48, naive Bayesian, RF, and SVM.

Shone et al. also combined deep learning and traditional machine learning algorithms. A nonsymmetric deep autoencoder (NDAE) was used for unsupervised feature learning. Then, an RF was used for the classification task. Two NDAEs were arranged in a stack, where each NDAE was composed of three hidden layers. In each hidden layer, the number of neurons is the same as the number of features. The KDD’99 data set was used to evaluate the model. Results showed an accuracy of 97.85% for five-class classification.

The effectiveness of several deep learning algorithms in classifying the KDD’99 data set was assessed by Vinayakumar et al. The authors tested CNN and
the combination of CNN with other architectures, such as RNN, long short-term memory (LSTM), and gated recurrent unit. Experimental results showed that the CNN-LSTM outperformed other network structures in multiclass classification, whereas the simple CNN surpasses all other network structures in binary classification. The CNN-LSTM obtained accuracy values of 96.4% and 98.7% for two-class and five-class classification problems, respectively.

Li et al.\(^27\) presented an image conversion method for NSL-KDD data. The preprocessing stage converts various feature attributes into binary vectors, and then, the data are converted into an image. Several experiments were carried out for the binary classification problem using two CNN models: ResNet-50 and GoogLeNet. Using NSL-KDD 'Test', ResNet-50 obtained an accuracy value of 79.14%, whereas GoogLeNet achieved 77.04% accuracy.

Wu et al.\(^28\) built a CNN and an RNN for the classification of the NSL-KDD data set. The authors focused on solving the data imbalance problem by using the cost function-based method. The cost function weight coefficients of each class are set based on the training sample number. The reported accuracies of the deep learning models outperformed traditional machine learning algorithms, such as J48, NB, NB tree, RF, and SVM. However, the accuracy of the CNN model was slightly lower than that of the RNN model.

Altwaijry et al.\(^29\) developed an intrusion detection model using DNN. The proposed model consisted of four hidden fully connected layers and was trained using NSL-KDD data set. The DNN model obtained accuracy values of 84.70% and 77.55% for the two-class and five-class classification problems, respectively. The proposed model outperformed traditional machine learning algorithms, including NB, J48, RF, Bagging, and Adaboost in terms of accuracy and recall.

Although deep learning has demonstrated success in many applications, its performance is not ideal when dealing with small or unbalanced data sets.\(^4\) Tavallaee et al.\(^11\) observed that the KDD’99 data set has redundant records, in both the train and test sets. This implies that reported accuracy values for the majority of models in the literature may not be representative of the performance of the models on real-world data. We believe that most classifier’s performance (on KDD’99) would degrade when tested on the NSL-KDD data set. The performance of published ADNIDS models using balanced and representative data sets is yet to be investigated.

In addition, research efforts in developing ADNIDS models using machine learning or deep learning techniques show that better results are obtained in the binary classification of intrusions compared with multiclass classification.\(^11\) In general, the performance of ADNIDS machine learning models for multiclass classification is not satisfactory. Thus, the ADNIDS problem remains an open problem.

A more recent trend sees ADNIDS researchers studying the applicability of the proposed models to real-world data sets. For this reason, we choose to study the performance of our models on NSL-KDD, an unbiased but old data set, to validate our model with known works, and UNSW-NB15,\(^12\) a more recent data set that is representative of real-world data.

**Methods**

In this study, we use a CNN architecture to build binary and multiclass classification models for ADNIDS. CNNs are one of the best learning algorithms that are capable of understanding complex structures and have shown excellent success in tasks related to image segmentation, object detection, and computer vision applications. The key benefit of CNNs is their power to leverage spatial or temporal correlation in data. CNNs have also been used in the context of intrusion detection for both feature extraction and classification.\(^21\) Fewer parameters are required in a CNN compared with other deep learning architectures. Thus, model complexity is reduced, and the learning process is improved. In the following subsections, we describe the data sets, preprocessing, and the architectures of our proposed models.

**Data set description**

This research is carried out over two data sets: the NSL-KDD data set\(^11\) and the UNSW-NB15 data set.\(^12\) In the next sections, we briefly describe the characteristics of our chosen data sets. A more in-depth treatment of the data sets is available in Appendix A1.

Network security laboratory-knowledge discovery in databases. The NSL-KDD data set is an improved version of the KDD’99 data set.\(^11\) It is a smaller data set that provides better evaluation of classifiers since redundant records are removed.\(^11,30\) Redundant records cause learning classifiers to be biased toward the more frequent records during training, as well as increasing classification accuracy whenever these same records appear in the test set. The training set KDDTrain\(^+\) contains...
125,973 records, and the testing set KDDTest+ contains 22,544 records. The data set simulates the following types of attacks: denial of service (DoS), Probing, user to root attack (U2R), and remote to local attack (R2L). The testing set has some specific attack types that are not present in the training set, thereby allowing realistic intrusion detection system evaluation. Each record in the NSL-KDD data set has 41 features, divided into 3 groups: Basic features are derived from transmission control protocol/internet protocol (TCP-IP) connections, traffic features are collected from window intervals or the number of connections, and content features are taken from the application layer data of connections.

The third model achieved the best performance with regard to classification in both the binary and multiclass classification cases, and as such, we chose it as our basic architecture. Next, we set our hyperparameters on this model. First, we run our model for 10 epochs and monitor the performance as the learning rate is varied in the following set \(0.0001, 0.003, 0.009, 0.001, 0.003\). We set the learning rate to 0.001. Next, we experiment with rectified linear unit (ReLU), Parametric ReLU, and Leaky ReLU with values ranging from \([0.1 \text{ to } 0.3]\), and select Leaky ReLU with \(\alpha = 0.012\). Finally, we test the dropout rate in the following range \([0.1 \text{ to } 0.6]\), and select 0.3 as the best performing rate.

The number of epochs is set to 30; however, we employ early stopping, where we stop training when the performance accuracy on the validation set starts to degrade.

### CNN models

#### Data preprocessing.
Both NSL-KDD and UNSW-NB15 contain nonnumeric data. We preprocess our data in two ways and compare performance based on the different types of preprocessing.

**2D representation.** Here, the network connections are simply transformed into a 2D format suitable for the deep learning architecture. First, we convert the nonnumeric features to numeric features using one-hot encoding. For the NSL-KDD data set, the categorical features are converted into indicator values and then combined with the numerical features to get a total of 121 numeric features. These are represented as an \(11 \times 11 \times 1\) matrix. For the UNSW-NB15 data set, we converted the categorical features into indicator values and then combined with the numerical features to get 196 numeric features. These are represented as a \(14 \times 14 \times 1\) matrix. We normalize features by subtracting the mean and scaling to unit variance.

A sample of our input is visualized in Figure 1a. We refer to the models trained using the 2D representation of the data set as BCNN and MCNN for binary classification and multiclass classification, respectively.

---

#### Determining optimal architecture and hyperparameter settings

To choose the optimal architecture for our CNN model, we ran 3 trials on the following architectures, for 20 epochs each:

1. Three convolution layers with 8, 16, and 32 feature maps, followed by 1 max-pooling layer and 4 fully connected layers.
2. Five convolution layers with 8, 16, 8, 16, and 32 feature maps, followed by 2 max-pooling layers and 4 fully connected layers.
3. Five convolution layers with 8, 16, 8, 16, and 32 feature maps, with skip connections to the second and fourth convolution layers, followed by 2 max-pooling layers and 4 fully connected layers.

---

#### Principal component analysis–deep feature synthesis.

Instead of relying on hand-crafted features, we propose a hybrid two-step preprocessing approach that combines dimensionality reduction and feature engineering. First, the nonnumeric features, or categorical features, are converted into numeric features using nominal integer encoding and then centered around the mean and scaled to unit variance. Then, we use principal component analysis (PCA) on the continuous features, such that 95% of the variance is retained. As
the data sets can contain a number of redundant or highly correlated features, the performance of classification algorithms can suffer. PCA is basically a dimensionality reduction technique that we use to increase interpretability, while minimizing the loss of information.31 PCA has the advantage of being less sensitive to noise compared with other techniques, such as Isomap, locally linear embedding, and Hessian locally linear embedding.32,33 These steps produce a total of 27 and 26 features from the NSL-KDD and UNSW-NB15 data sets, respectively. These features are then converted using addition and multiplication primitives in DFS.10

For the binary classification case, the classes in both data sets are balanced; however, the classes are severely imbalanced in the multiclass case. There are many techniques for oversampling to handle imbalanced data sets. In this study, we use the synthetic minority oversampling technique (SMOTE),34 which is one of the most widely used techniques. The basic idea is to synthesize new points from the minority class. A data point, \( x \), is randomly selected from the minority class in the data set. Then, the \( k \) neighbors of \( x \) are determined, with \( k \) usually set to 5. One of the identified neighboring points, \( y \), is then chosen. A new synthetic record, \( z \), is generated at a randomly selected point between points \( x \) and \( y \) in the feature space. SMOTE has been applied in a variety of applications with demonstrated success.35 The technique has also been shown to be robust and to perform better than simple oversampling. SMOTE is also effective for the reduction of overfitting.

Feature engineering refers to the process of creating new and meaningful features from raw ones. The goal of feature engineering is to select the most relevant features for better model accuracy. In DFS, new deep features are generated by stacking multiple primitives. A feature’s depth is essentially the amount of primitives that are required to create the feature.

Using DFS, 729 and 676 features are automatically produced from the NSL-KDD and UNSW-NB15 data sets, respectively. We select 121 and 196 features, retaining about 99% of the variance, for the NSL-KDD and UNSW-NB15 data sets. A sample of our input is visualized in Figure 1b. Compared with Figure 1a, we observe that there is more interfeature variance, which indicates that redundant features have been eliminated. We refer to the models trained using the PCA-DFS of the data sets as BCNN-DFS and MCNN-DFS for binary and multiclass classification, respectively.

CNN architecture. We propose two CNN models: BCNN and MCNN, where the first model (BCNN) is used for binary classification, and the second model (MCNN) is used for multiclass classification of network attacks.

Input layer. The input layer is either an \( 11 \times 11 \times 1 \) matrix for the NSL-KDD data set, or a \( 14 \times 14 \times 1 \) matrix for the UNSW-NB15 data set, as defined in the previous section. In the rest of this article, we use \( S \) to represent the input image side, that is, the height or width, where \( S = 11 \) or \( S = 14 \), depending on the data set used.

Hidden layers. Our model is composed of a total of five convolutional layers, two pooling layers, and four fully connected layers. Our input image is small, either \( 11 \times 11 \) pixels or \( 14 \times 14 \) pixels, and so a smaller filter size, that is, \( 2 \times 2 \), is more appropriate for this image. As we wish to keep the representational power of our model, we increase the number of feature maps as the network deepens and apply padding in all convolutional layers to overcome the problems of image shrinkage and information loss around the perimeter of the image. We also use batch normalization after each convolutional layer.

The first convolutional layer has as input an \( S \times S \times 1 \) image. We use \( k = 8 \ (2 \times 2 \times 1) \) kernels, zero-padding of 1, and a stride of 1, for a convolutional layer of size \( S \times S \times 8 \). Each activation map \( i \) is calculated as shown in Equation (1), where \( l \) is the current layer, \( B_j^{(l)} \) is a bias matrix, \( k^{(l-1)} \) is the number of kernels used in the previous layer, \( W \) is the current layer kernel matrix, and \( Y^{(l-1)} \) is the output of the previous layer. Our
nonlinearity is the Leaky ReLU function, defined as shown in Equation (2), with $\alpha = 0.12$.

$$Y_i^{(l)} = b_i^{(l)} + \sum_{j=1}^{I_l} w_i^{(l)} Y_j^{(l-1)},$$  \hspace{1cm} (1)

$$\text{Leaky - ReLU}(x) = \begin{cases} 
\alpha x & \text{if } x < 0 \\
\alpha x & \text{if } x \geq 0 
\end{cases}$$  \hspace{1cm} (2)

The second convolutional layer has as input an $S \times S \times 8$, and we use a $k = 16$ kernels for a convolutional layer of size $S \times S \times 16$. We add the input image at this point to the tensor. This step is inspired by the idea of skip connections in residual networks. Skip connections speed-up the learning process and overcome the problem of vanishing gradients. The vanishing gradient problem is encountered when training ANN with gradient-based a serious detriment to deep learning using backpropagation. This is because the neural network’s weights are updated in proportion to the partial derivative of the error function with respect to the current weight during training, and sometimes, if the gradient is very small, the weight value is inadequately updated. In the worst case, this stops the neural network from further training. We noticed the vanishing gradient problem in our model and incorporated skip connections to amplify the input signal and prevent zero gradients. As shown in Figure 2, the input image is added to the tensor at two points in the architecture. This architecture is then repeated, before the fifth and final convolutional layer, which has 32 feature maps.

Next, we have two max-pooling layers that have a $2 \times 2$ window size. The tensor is then flattened and followed by four fully connected layers, with sizes 500, 300, 100, and 20, respectively. All fully connected layers use a 30% dropout rate to reduce overfitting, set experimentally.

Output layer. The output layer is a 5 class Softmax layer (one class for each attack type, plus the normal class). Softmax outputs a probability-like prediction for each character class, see Equation (3), where $N$ is the number of output classes. Our CNN architecture is shown in Figure 2.

Our model incorporates various methods to reduce overfitting. In particular, our model incorporates a dropout layer, where randomly selected activations are set to 0 during training, so that the model becomes less sensitive to specific weights in the network. In addition, our model has weight decay, also called L2 regularization, which reduces overfitting by penalizing model weights. We also incorporate batch normalization, which normalizes the input values of the layer, reducing overfitting and improving gradient flow through the network. Finally, our model incorporates a data augmentation technique, specifically SMOTE, which is also effective for the reduction of overfitting. We report the performance of the final model on a separate unseen test set, which contains new attack types not present in the training set.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}}.$$  \hspace{1cm} (3)

Optimization. In our model, we tested two optimizers: Stochastic Gradient Descent and Adam, and selected Adam as it was found to work better. The loss function used is the categorical cross-entropy loss, which is widely used to calculate the probability that the input belongs to a particular class. It is usually used as the default function for multiclass classification. In our model, we set the learning rate to $lr = 0.001$, set experimentally.

FIG. 2. Proposed CNN model. CNN, convolutional neural network.
Results
Experimental settings
The proposed models were implemented using TensorFlow, an open-source machine learning library, utilizing Keras. Experiments were carried out using GPUs running on the Google Colab environment.

Performance measures
To evaluate the performance of BCNN and MCNN, the following performance measures are calculated: accuracy, precision, detection rate, and F-measure.

Accuracy is the percentage of records classified correctly, and it is calculated using the following equation:

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN},
\]

Precision (P) is the percentage of records correctly classified as anomaly out of the total number of records classified as anomaly. Precision is calculated as follows:

\[
P = \frac{TP}{TP + FP}.
\]

Detection rate (DR), also called True Positive Rate or Recall, is the percentage of records correctly classified as anomaly out of the total number of anomaly records. The detection rate can be calculated as follows:

\[
\text{DR} = \frac{TP}{TP + FN},
\]

F-measure (F) is a measure that combines both precision and detection rate, and it is calculated as follows:

\[
F = \frac{2 \times P \times DR}{P + DR},
\]

where TP (true positive) indicates the number of anomaly records that are identified as anomaly. FP (false positive) is the number of normal records that are identified as anomaly. TN (true negative) is the number of normal records that are identified as normal. FN (false negative) is the number of anomaly records that are identified as normal.

Performance evaluation
Our experiments are designed to evaluate the ability of BCNN and MCNN at detecting network intrusions. Two data sets are used for this purpose: the NSL-KDD data set and the UNSW-NB15 data set. For both data sets, the experimental results are compared with (1) the results of similar approaches in the literature and (2) results of various state-of-the-art classification algorithms. In particular, we compare our models with NB, J48, RF, Bagging, and Adaboost implemented in Waikato Environment for Knowledge Analysis. All models were trained on KDDTrain and tested on KDDTest for the NSL-KDD data set and on the partitioned data set of UNSW-NB15, with 175,341 training records and 82,332 testing records.

We present our results in the following two sections. The NSL-KDD results section presents the results of BCNN and MCNN on the NSL-KDD data set. Binary classification classifies network traffic into: normal or anomaly. Multiclass classification classifies network traffic into five labels: normal, DOS, R2L, U2R, and Probe.

The UNSW-NB15 results section presents the results of BCNN and MCNN on the UNSW-NB15 data set. Binary classification classifies network traffic into: normal or anomaly. Multiclass classification classifies network traffic into 10 labels: normal, Fuzzers, Analysis, Backdoor, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms.

NSL-KDD results
Binary classification. In this section, we present the classification results of BCNN and BCNN-DFS on KDDTest after training using KDDTrain. Table 2 shows the performance measures of BCNN compared with similar approaches in the literature.

We note that BCNN outperformed all models in the literature, achieving an accuracy of 88.81%, while detecting 89% of all attacks, with an F-measure of 89%.

However, with the hybrid feature engineering approach, BCNN-DFS, we find that it outperforms BCNN and all other models in the literature, achieving an accuracy of 90.14% on KDDTest. We note that AlertNet obtains a high recall rate. However, it exhibits a lower precision rate, indicating that it misclassifies a significant proportion of network connections, as shown by its F-measure. We also show our training accuracy, which was 99.50% for BCNN and 99.62% for BCNN-DFS in the bottom row of Table 2.

Next, we compare our work with various state-of-the-art classification algorithms, as shown in Table 2. Both BCNN and BCNN-DFS achieve the higher accuracy and F-measure compared with all other models. Their detection rate outperforms all other models by a large margin, where they are able to detect 89.00% and 90% of all attacks, with an F-measure of 89% and 90%.
Table 2. Performance comparison with several related literature approaches for binary classification on network security laboratorynowledge discovery in databases

| Model       | Accuracy | Precision | Recall | F-measure | Training data sets | Testing data sets |
|-------------|----------|-----------|--------|-----------|--------------------|-------------------|
| STL-IDS²⁴   | 84.96%   | 96.23%    | 76.57% | 85.28%    | KDDTrain⁺         | KDDTest⁺         |
| ANN⁺¹⁶      | 81.20%   | N/A       | N/A    | N/A       | KDDTrain⁺         | KDDTest⁺         |
| DNN⁰        | 88.39%   | 85.44%    | 95.95% | 90.40%    | KDDTrain⁺         | KDDTest⁺         |
| AlertNet⁵   | 80.10%   | 69.20%    | 96.90% | 80.70%    | KDDTrain⁺         | KDDTest⁺         |
| SDN-CDN²³   | 75.75%   | 83.00%    | 75.00% | 74.00%    | KDDTrain⁺         | KDDTest⁺         |
| RNN-IDS²⁰   | 83.28%   | N/A       | 97.09% | N/A       | KDDTrain⁺         | KDDTest⁺         |
| GoogLeNet²⁰ | 77.04%   | 91.66%    | 65.64% | 76.50     | KDDTrain⁺         | KDDTest⁺         |
| ResNet-50²⁷ | 79.14%   | 91.97     | 69.41% | 79.12     | KDDTrain⁺         | KDDTest⁺         |
| TSE-IDS²⁴   | 85.79%   | 88.00%    | 86.80% | 87.4%     | KDDTrain⁺         | KDDTest⁺         |
| BDNN²⁹      | 84.70%   | 79.45%    | 87.00% | 83.05%    | KDDTrain⁺         | KDDTest⁺         |
| BCNN⁰       | 88.81%   | 89.00%    | 89.00% | 89.00%    | KDDTrain⁺         | KDDTest⁺         |
| BCNN-DFS⁰   | 90.14%   | 90.00%    | 90.00% | 90.00%    | KDDTrain⁺         | KDDTest⁺         |
| Naive Bayes | 76.12%   | 92.38%    | 63.27% | 75.10%    | KDDTrain⁺         | KDDTest⁺         |
| J48         | 81.53%   | 97.14%    | 69.61% | 81.10%    | KDDTrain⁺         | KDDTest⁺         |
| Random Forest| 80.45%  | 97.05%    | 67.72% | 79.77%    | KDDTrain⁺         | KDDTest⁺         |
| Bagging     | 82.63%   | 91.87%    | 76.23% | 83.32%    | KDDTrain⁺         | KDDTest⁺         |
| AdaBoost    | 78.44%   | 95.28%    | 65.37% | 77.54%    | KDDTrain⁺         | KDDTest⁺         |
| BCNN⁰       | 99.50%   | 99.45%    | 99.69% | 99.57%    | 80% of KDDTrain⁺  | 20% of KDDTrain⁺ |
| BCNN-DFS⁰   | 99.62%   | 99.7%     | 99.6%  | 99.6%     | 80% of KDDTrain⁺  | 20% of KDDTrain⁺ |

The highest performance measures obtained are shown in bold.

BCNN, binary classification convolutional neural network; BDNN, binary classification deep neural network; TSE-IDS, two-stage classifier ensemble for intelligent anomaly-based intrusion detection system.

Multiclass classification. In this section, we present classification results on KDDTest⁺ after training on KDDTrain⁺. Table 3 presents our results compared with various models from the literature. It should be noted that Shone et al.²⁵ reported training and not testing results, thereby improving their various performance measures. The last row in Table 3 presents our training accuracy for our multiclass classification models.

Our multiclass CNN model, MCNN, achieves an accuracy of 81.1% and is able to detect 81% of all attacks. Its overall F-measure is 80%, outperforming multi classification deep neural network (MDNN),²⁹ and all other models in the literature except for RNN-IDS,²⁰ which it is comparable to. MCNN-DFS achieves a comparable result outperforming MCNN slightly. Table 3 also presents the comparison of our multiclass models.

Table 3. Performance comparisons with several related literature approaches for multiclass classification on NSL-KDD

| Model       | Accuracy | Precision | Recall | F-measure | Training data sets | Testing data sets |
|-------------|----------|-----------|--------|-----------|--------------------|-------------------|
| STL-IDS²⁴   | 80.48%   | 93.92%    | 68.28% | 79.08%    | KDDTrain⁺         | KDDTest⁺         |
| ANN⁺¹⁶      | 79.9%    | N/A       | N/A    | N/A       | KDDTrain⁺         | KDDTest⁺         |
| DNN⁰        | 79.10%   | 83%       | 68%    | 75.76%    | KDDTrain⁺         | KDDTest⁺         |
| AlertNet⁵   | 78.50%   | 81.00%    | 78.50% | 76.50%    | KDDTrain⁺         | KDDTest⁺         |
| RNN-IDS²⁰   | 81.29%   | N/A       | 97.09% | N/A       | KDDTrain⁺         | KDDTest⁺         |
| RNN²⁸       | 81.29%   | N/A       | 69.73% | N/A       | KDDTrain⁺         | KDDTest⁺         |
| CNN²⁸       | 79.48%   | N/A       | 68.66% | N/A       | KDDTrain⁺         | KDDTest⁺         |
| MDNN²⁸      | 77.55%   | 81.23%    | 77.55% | 75.43%    | KDDTrain⁺         | KDDTest⁺         |
| MCNN⁰       | 81.1%    | 83%       | 81%    | 80%       | KDDTrain⁺         | KDDTest⁺         |
| MCNN-DFS⁰   | 81.44%   | 81%       | 84%    | 80%       | KDDTrain⁺         | KDDTest⁺         |
| J48         | 72.73%   | 76.1%     | 72.7%  | 72.6%     | KDDTrain⁺         | KDDTest⁺         |
| Random Forest| 74.99%  | 79.6%     | 75.0%  | 71.1%     | KDDTrain⁺         | KDDTest⁺         |
| Bagging     | 74.83%   | 78.3%     | 74.8%  | 71.6%     | KDDTrain⁺         | KDDTest⁺         |
| AdaBoost    | 66.43%   | N/A       | 66.0%  | N/A       | KDDTrain⁺         | KDDTest⁺         |
| BCNN⁰       | 99.5%    | 99.5%     | 99.5%  | 99.5%     | 80% of KDDTrain⁺  | 20% of KDDTrain⁺ |
| BCNN-DFS⁰   | 99.7%    | 99.5%     | 99.80% | 99.6%     | 80% of KDDTrain⁺  | 20% of KDDTrain⁺ |
| NDAE²⁵      | 85.42%   | 100.00%   | 85.42% | 87.37%    | 90% of KDDTrain⁺  | 10% of KDDTrain⁺ |

The highest performance measures obtained are shown in bold.

MCNN, multi classification convolutional neural network; MDNN, multi classification deep neural network.
with state-of-the-art classification algorithms. MCNN and MCNN-DFS achieve the best results in terms of accuracy, precision, recall, and F-measure, outperforming all the evaluated models.

UNSW-NB15 results

Binary classification. In this section, we present the performance evaluation of BCNN and BCNN-DFS using UNSW-NB15. In addition, for the sake of comparison, we conduct a study on the performance of our previously published model, binary classification deep neural network (BDNN),\textsuperscript{29} using UNSW-NB15. Unfortunately, the number of studies conducted on the full UNSW-NB15 data set is fewer than those using NSL-KDD. Table 4 shows the performance measures of BCNN and BCNN-DFS compared with similar approaches in the literature. The results show that BCNN achieves the highest accuracy and F-measure values, except for two-stage classifier ensemble for intelligent anomaly-based intrusion detection system (TSE-IDS).\textsuperscript{14} In terms of precision, BCNN, BCNN-DFS, and BDNN outperform all the compared state-of-art machine learning algorithms. However, AlertNet\textsuperscript{5} has the best precision value. As for the recall, our models perform better than AlertNet and NB.

Multiclass classification. This section presents the classification results of MCNN, MCNN-DFS, and the previously published model MDNN\textsuperscript{29} using UNSW-NB15 on the multiclass classification problem. Table 5 shows the performance measures of our models compared with similar approaches in the literature. Although the presented numbers reflect degraded performance of multiclass classification compared with binary classification, MCNN outperforms all the compared models on all measures. The performance of MCNN-DFS is comparable to MCNN.

What does the network learn? A case study

In this section, we will present a small case study, designed to showcase what our model learns and

### Table 4. Performance comparison with several related literature approaches for binary classification on UNSW-NB15

| Model           | Accuracy | Precision | Recall | F-measure | Training data sets | Testing data sets |
|-----------------|----------|-----------|--------|-----------|--------------------|-------------------|
| AlertNet\textsuperscript{5} | 78.40%   | 94.40%    | 72.50% | 82.00%    | U-train            | U-test            |
| TSE-IDS\textsuperscript{14}   | 91.27%   | 91.60%    | 91.30% | 91.45%    | U-train            | U-test            |
| BDNN\textsuperscript{29}      | 80.63%   | 86.00%    | 81.00% | 79.00%    | U-train            | U-test            |
| BCNN             | 90.25%   | 91.00%    | 90.00% | 90.45%    | U-train            | U-test            |
| BCNN-DFS         | 89.26%   | 89.00%    | 89.00% | 89.00%    | U-train            | U-test            |
| J48              | 76.95%   | 70.50%    | 99.98% | 82.69%    | U-train            | U-test            |
| Random Forest    | 80.94%   | 74.34%    | 99.84% | 85.23%    | U-train            | U-test            |
| Bagging          | 76.95%   | 70.50%    | 99.98% | 82.69%    | U-train            | U-test            |
| Adaboost         | 78.13%   | 71.63%    | 99.82% | 83.41%    | U-train            | U-test            |
| BDNN\textsuperscript{29}      | 93.21%   | 94%       | 93%    | 93%       | 80% of U-train     | 20% U-train       |
| BCNN             | 94.42%   | 95%       | 94%    | 94%       | 80% of U-train     | 20% U-train       |
| BCNN-DFS         | 95.71%   | 96%       | 96%    | 96%       | 80% of U-train     | 20% U-train       |

The highest performance measures obtained are shown in bold. U-train is UNSW-NB15-training-set, and U-test is UNSW-NB15-testing-set.

### Table 5. Performance comparison with several related literature approaches for multiclass classification on the UNSW-NB15 data set

| Model           | Accuracy | Precision | Recall | F-measure | Training data sets | Testing data sets |
|-----------------|----------|-----------|--------|-----------|--------------------|-------------------|
| AlertNet\textsuperscript{5} | 66.00%   | 62.30%    | 66.00% | 59.60%    | U-train            | U-test            |
| MDNN\textsuperscript{29}     | 62.87%   | 76.00%    | 63.00% | 64.00%    | U-train            | U-test            |
| MCNN             | 69.46%   | 84.00%    | 69.00% | 74.00%    | U-train            | U-test            |
| MCNN-DFS         | 68.52%   | 83.00%    | 69.00% | 73.00%    | U-train            | U-test            |
| Naive Bayes      | 45.22%   | 29.67%    | 38.62% | 33.56%    | U-train            | U-test            |
| J48              | 51.50%   | 28.18%    | 21.48% | 24.38%    | U-train            | U-test            |
| Random Forest    | 68.09%   | 62.51%    | 35.15% | 44.99%    | U-train            | U-test            |
| Bagging          | 51.45%   | 32.85%    | 21.45% | 25.95%    | U-train            | U-test            |
| Adaboost         | 51.50%   | 28.18%    | 21.48% | 24.38%    | U-train            | U-test            |
| MDNN\textsuperscript{29}     | 72.54%   | 73%       | 73%    | 69%       | 80% of U-train     | 20% U-train       |
| MCNN             | 77.27%   | 77%       | 70%    | 69%       | 80% of U-train     | 20% U-train       |
| MCNN-DFS         | 80.51%   | 81%       | 81%    | 81%       | 80% of U-train     | 20% U-train       |

The highest performance measures obtained are shown in bold. U-train is UNSW-NB15-training-set, and U-test is UNSW-NB15-testing-set.
how it performs. For the purposes of this study, we select the UNSW-NB15 data set, in the multiclass classification case, using the MCNN-DFS model. We randomly select 10 samples, such that each is of a different class, from the data set. The samples and their corresponding model prediction are shown in Table 8. Our model’s prediction accuracy on this small sample is 70%, which is to be expected from the results shown in Table 5.

In Table 9, we visually demonstrate what MCNN-DFS learns at each convolutional layer. Each of the layers has multiple filters, and each filter produces an activation map. We show only a subset of these activation maps at each layer. The convolutional network learns this hierarchy of filters. The filters at the earlier layers, in our case Conv-1, learn the low-level features of the data. As we go through the network, in the middle layers, Conv-2 and Conv-3, we learn more complex features, and the last layers, Conv-4 and Conv-5, learn even more complex features and concepts. As learning progresses, we observe that the model captures more complex features at each stage. These features are then input into the fully connected portion of the network to produce the final classification output.

Computational time
In addition to evaluating the classification accuracy of the proposed models, it is important to measure the computational time in real situations of network intrusion detection. The computational complexity of our models for both data sets is shown in Table 10. The table shows both the training and prediction times.

In a real-life setting, our models would be trained offline, and as such, longer training times are acceptable. The second column in Table 10 shows the average training time needed by each model on each data set. In general, binary models complete training faster than multiclass models, and as the data set size increases, so does the training time. However, as the binary models train in a significantly shorter period, we suggest training and deploying the binary model on new data and then training the multiclass model if needed. In the third column, we show average prediction times for each model on each data set. Prediction times are for 100,000 sample network connections on a single machine. Faster times can be achieved if the models are deployed in parallel. As Table 10 shows, all models achieve high-throughput rates.

Discussion
One of the key questions that this research was intended to investigate is how to improve the prediction performance of NIDS systems using deep learning and feature engineering. For the NSL-KDD data set, we observe that the classification accuracy of our proposed multiclass models is lower than that of the binary class models. This observation is consistent with results in the literature. For the multiclass problem, achieving good results on the NSL-KDD data set is difficult. We believe that this is due to the fact that the NSL-KDD data set suffers from the class imbalance problem. For example, the U2R class represents 0.04% of the data set, whereas the R2L class represents 8% of the data set.

We note that the overall performance of the deep learning model with the proposed hybrid feature engineering technique slightly improves prediction accuracy over the NSL-KDD data set. However, the performance on UNSW-NB15 is slightly decreased. This might be attributed to the distribution of the training and testing data sets, which require further investigation. Looking into the confusion matrix of MDNN29 shown in Table 6, we observe that the majority of predictions are placed into four classes only, namely

| Table 6. MDNN29 confusion matrix on UNSW-NB15 |
|-----------------------------------------------|
| Predicted                                      |
| Analysis | Backdoor | DoS | Exploits | Fuzzers | Generic | Normal | Reconnaissance | Shellcode | Worms |
|---------|----------|-----|----------|---------|---------|--------|----------------|------------|-------|
| Actual  |          |     |          |         |         |        |                |            |       |
| Analysis| 0        | 0   | 0        | 499     | 137     | 16     | 25             | 0          | 0     |
| Backdoor| 0        | 0   | 0        | 389     | 172     | 16     | 6              | 0          | 0     |
| DoS     | 0        | 0   | 0        | 3617    | 426     | 29     | 16             | 1          | 0     |
| Exploits| 0        | 0   | 0        | 9475    | 1558    | 52     | 47             | 0          | 0     |
| Fuzzers | 0        | 0   | 0        | 1473    | 4534    | 32     | 23             | 0          | 0     |
| Generic | 0        | 0   | 0        | 1503    | 501     | 16,350 | 517            | 0          | 0     |
| Normal  | 0        | 0   | 0        | 4096    | 11,487  | 9      | 21,408         | 0          | 0     |
| Reconnaissance| 0 | 0   | 0        | 2084    | 1409    | 0      | 1              | 2          | 0     |
| Shellcode | 0 | 0   | 0        | 182     | 195     | 0      | 1              | 0          | 0     |
| Worms   | 0        | 0   | 0        | 37      | 7       | 0      | 0              | 0          | 0     |

DoS, denial of service.
Exploits, Fuzzers, Generic, and Normal. This means that MDNN\textsuperscript{29} was unable to recognize the remaining six classes. As for MCNN, the confusion matrix in Table 7 shows that it has better performance than MDNN.\textsuperscript{29} We note that there are 10,859 normal records classified as Fuzzers. We believe that the random nature of a Fuzzers attack makes it difficult for MCNN to detect a particular pattern. Distinguishing between these two classes could be improved in future work.

Traditional machine learning algorithms have historically been widely used to solve the intrusion detection problem\textsuperscript{4} and are still prevalent today.\textsuperscript{42–45} Although considered shallow learners, they are still being utilized for developing IDS, for example, for the internet of things and big data environments.\textsuperscript{46,47} NB classification assumes conditional independence; however, it is robust to noise. The J48 is an implementation of the C4.5 decision tree algorithm, which provides easy interpretation of the final model. RF, Bagging, and Adaboost have the advantage of being ensemble algorithms, which combine the classification of many weak classifiers. The UNSW-NB15, in particular, as a newer data set, has been used to evaluate fewer approaches in the literature compared with NSK-KDD. This study demonstrated the superiority of deep learning over traditional shallow learning for NIDS systems.

### Table 7. MCNN Confusion matrix on UNSW-NB15

| Predicted | Analysis | Backdoor | DoS | Exploits | Fuzzers | Generic | Normal | Reconnaissance | Shellcode | Worms |
|-----------|----------|----------|-----|----------|---------|---------|--------|----------------|-----------|-------|
| Actual    |          |          |     |          |         |         |        |                |           |       |
| Analysis  | 9        | 241      | 380 | 10       | 2       | 0       | 35     | 0              | 0         | 0     |
| Backdoor  | 10       | 183      | 367 | 9        | 4       | 0       | 0      | 1              | 9         | 0     |
| DoS       | 30       | 2016     | 1261| 536      | 104     | 8       | 25     | 32             | 66        | 11    |
| Exploits  | 54       | 2006     | 1552| 6494     | 267     | 21      | 136    | 355            | 212       | 35    |
| Fuzzers   | 19       | 506      | 745 | 176      | 3897    | 0       | 440    | 39             | 237       | 3     |
| Generic   | 10       | 22       | 122 | 356      | 114     | 18,184  | 11     | 4              | 39        | 9     |
| Normal    | 992      | 1        | 132 | 574      | 10,859  | 4       | 23,955 | 78             | 392       | 13    |
| Reconnaissance | 2 | 267 | 73 | 142 | 39 | 1 | 12 | 2887 | 71 | 2 |
| Shellcode | 0        | 0        | 8   | 18       | 38      | 0       | 6      | 9              | 296       | 3     |
| Worms     | 0        | 0        | 0   | 14       | 4       | 1       | 0      | 1              | 1         | 23    |

### Table 8. MCNN-DFS case study on UNSW-NB15

| Sample | Actual | Predicted | Sample | Actual | Predicted |
|--------|--------|-----------|--------|--------|-----------|
| Analysis | DoS               |          |          |        |           |
| Backdoor | Analysis            | Normal   | Normal   |        |           |
| DoS      | DoS               | Reconn.  | Reconn.  |        |           |
| Exploits | Analysis           | Shellcode| Shellcode|        |           |
| Fuzzers  | Fuzzers           | Worms    | Worms    |        |           |
The complex structures of deep learning models facilitate a better learning process than shallow models. In this study, the performance of the proposed models has been evaluated using two data sets. First, NSL-KDD, a legacy data set that has been widely used to evaluate IDS models. Second, the UNSW-NB15, a modern data set that is representative of real-world data. A number of other modern network intrusion detection data sets are publicly available now, such as CIC-IDS201748 and CIC-IDS2018.49 Thus, more studies, such as the work of Gamage and Samarabandu,50 are required to investigate the performance of deep learning models and baseline models on a wider range of data sets.

AlertNet5 used a traditional deep fully connected neural network architecture, as did CNN,28 which used a traditional architecture with two convolutional layers, followed by two pooling layers. STL-IDS24 incorporated sparse autoencoder, followed by SVM classifiers. Compared with the related work, as shown in Tables 2–4, our proposed models have minimal trade-off between precision and recall. As both measures are essential in the intrusion detection problem, having a system that balances both metrics are an advantage.

To the best of our knowledge, this is the first article incorporating skip connection methodology into CNNs to solve the anomaly detection problem. This has proven to be successful as shown in the Results section where our model outperformed previous models in the literature. This study highlights the benefits of skip connections; however, this line of research is yet to be investigated in other architectures. In addition, we achieve excellent results with a small network size that is trainable in a short period. This enables our network to be deployed for real-life anomaly detection situations. It has high accuracy at detecting anomalies, especially in the binary situation. When it comes to RNNs and other time-based architectures, their usefulness in anomaly detection is yet to be established given current data sets, where the time-based nature of connections is not evident or explicitly stated in the data set.

**Table 9. Class “Analysis” on UNSW-NB15**

| Sample | Conv-1 | Conv-2 | Conv-3 | Conv-4 | Conv-5 |
|--------|--------|--------|--------|--------|--------|

**Table 10. Computational complexity**

|          | Training time | Prediction time (seconds) | Total | Parameters |
|----------|---------------|---------------------------|-------|------------|
|          |               |                           |       | Trainable  | Nontrainable |
| NSL-KDD  |               |                           |       |            |              |
| BCNN     | 00:04:40      | 4.17                      | 586,878 | 586,768  | 110          |
| MCNN     | 00:34:10      | 3.59                      | 1,827,045 | 1,826,935 | 110          |
| UNSW-NB15|               |                           |       |            |              |
| BCNN     | 00:14:54      | 3.82                      | 762,938 | 762,798  | 140          |
| MCNN     | 02:21:41      | 4.02                      | 2,531,210 | 2,531,070 | 140          |

**Conclusions**

NIDS are essential tools for detecting malicious network traffic in today’s computer systems. They are designed to differentiate between previously unseen abnormal network activity and normal patterns. In this article, we presented two novel network intrusion detection models based on deep learning, as well as two
approaches to data preprocessing, a simple preprocessing approach and a hybrid two-step preprocessing approach to generate meaningful features. Our models employ the CNNs paradigm. We design our models for binary and multiclass classification.

Our models were able to achieve excellent performance compared with state-of-the-art classification algorithms. In particular, our models outperform NB, J48, RF, Bagging, and Adaboost in terms of accuracy and recall. In addition, they achieved excellent results compared with similar approaches in the literature. We observe that the classification accuracy of our multiclass models is lower than that of the binary class models.

The development of classifiers for ADNIDS is an essential step toward building a complete intrusion detection framework. The proposed models can be integrated into network systems to detect unusual events, such as new attacks and violations inside an organization’s network. For future work, we plan to use an ensemble to combine the results of our classifiers, to improve predictions. In addition, the neural networks can be combined in a number of different ways to produce the classification result in one step. The proposed models themselves can be improved by increasing the number of hidden layers and neurons, or adding some convolutional layers, using different optimizers and trying new values for the learning rate. Also, methods to balance the various data set classes could potentially improve classification results. We also plan to study various sampling techniques to improve class variability. Finally, additional studies should be conducted to look into distinguishing Fuzzers attacks.

Authors’ Contributions
Conceptualization: N.A. Methodology: I.A. and N.A. Experiments: N.A. Analysis: N.A. and I.A. Article drafting, editing, and final approval: I.A. and N.A.

Data Availability
The NSL-KDD data supporting this study are from previously reported studies and data sets, which have been cited. The processed data are available at https://www.unb.ca/cic/datasets/nsl.html. The UNSW-NB15 data set is publicly available at https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets.

Author Disclosure Statement
No competing financial interests exist.

Funding Information
This research project was supported by a grant from the Research Center of the Female Scientific and Medical Colleges, the Deanship of Scientific Research, King Saud University.

References
1. Kim K, Aminanto ME. Deep learning in intrusion detection perspective: Overview and further challenges. In: 2017 International Workshop on Big Data and Information Security (IWGIS), Jakarta, Indonesia: IEEE, 2017. pp. 5–10.
2. Garcia-Teodoro P, Diaz-Verdejo J, Maciá-Fernández G, et al. Anomaly-based network intrusion detection: Techniques, systems and challenges. Comput Secur. 2009;28:18–28.
3. Aminanto E, Kim K. Deep learning in intrusion detection system: An overview. In: 2016 International Research Conference on Engineering and Technology (2016 IRCET), Ball, Indonesia: Higher Education Forum, 2016.
4. Liu H, Lang B. Machine learning and deep learning methods for intrusion detection systems: A survey. Appl Sci. 2018;8:1398.
5. Vinayakumar R, Alazab M, Soman K, et al. Deep learning approach for intelligent intrusion detection system. IEEE Access. 2019;7:41525–41550.
6. Javid A, Niyaz Q, Sun W, et al. A deep learning approach for network intrusion detection system. In: Suzuki J, Nakano T, Hess H (Eds.), Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), New York, NY: ICST (Institute for Computer Sciences, Social-Informatics), 2016. pp. 21–26.
7. Rong H, Ma T, Cao J, et al. Deep rolling: A novel emotion prediction model for a multiparticipant communication context. Inform Sci. 2019;488:158–180.
8. Altwaijry N, Al-Turaiki I. Arabic handwritting recognition system using convolutional neural network. Neural Comput Appl. 2020;33:2249–2261.
9. Pan Z, Yang C-N, Sheng VS, et al. Machine Learning for Wireless Multimedia Data Security. 2019. DOI: 10.1155/2019/7682306.
10. Kanter JM, Veeramachaneni K. Deep feature synthesis: Towards automating data science endeavors. In: 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), Paris, France: IEEE, 2015. pp. 1–10.
11. Tavallaee M, Bagheri E, Lu W, et al. A detailed analysis of the KDD cup 99 data set. In: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, Canada: IEEE, 2009. pp. 1–6.
12. Moustafa N, Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 Military Communications and Information Systems Conference (MICIS), Canberra, Australia: IEEE, 2015. pp. 1–6.
13. Thaseen IS, Kumar CA. Intrusion detection model using fusion of chi-square feature selection and multi class SVM. J King Saud Univ Comput Inform Sci. 2017;29:462–472.
14. Tama BA, Comuzzi M, Rhee K-H. TSE-IDS: A two-stage classifier ensemble for intelligent anomaly-based intrusion detection system. IEEE Access. 2019;7:94497–94507.
15. Panda M, Patra MR. Network intrusion detection using naive bayes. Int J Comput Sci Netw Secur. 2007;7:258–263.
16. Ingre B, Yadav A. Performance analysis of NSL-KDD dataset using ann. In: 2015 International Conference on Signal Processing and Communication Engineering Systems, Guntur, India: IEEE, 2015. pp. 92–96.
17. Thakumar S, Janosi S, Sung A. Intrusion detection using neural networks and support vector machines. In: Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN’02 (Cat. No. 02CH37290), vol. 2, Honolulu, HI: IEEE, 2002. pp. 1702–1707.
18. Wang G, Hao J, Ma J, et al. A new approach to intrusion detection using artificial neural networks and fuzzy clustering. Expert Syst Appl. 2010; 37:6225–6232.
19. Pasupa K, Sunhem W. A comparison between shallow and deep architecture classifiers on small dataset. In: 2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE), Yogyakarta, Indonesia: IEEE, 2016. pp. 1–6.
20. Yin C, Zhu Y, Fei J, et al. A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access. 2017;5:21954–21961.
21. Aldweesh A, Derhab A, Emam AZ. Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. Knowl Based Syst. 2020;189:105124.
22. Ferrag MA, Maglaras L, Moschyiannis S, et al. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. J Inform Secur Appl. 2020;50:102419.
23. Tang TA, Mhamdi L, McLernon D, et al. Deep learning approach for network intrusion detection in software defined networking. In: 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), Fez, Morocco: IEEE, 2016. pp. 258–263.
24. Al-Qaff M, Lasheng Y, Al-Habib M, et al. Deep learning approach combining sparse autoencoder with SVM for network intrusion detection. IEEE Access. 2018;6:52843–52856.
25. Shone N, Ngoc TN, Phal VD, et al. A deep learning approach to network intrusion detection. IEEE Trans Emerg Topics Comput Intell. 2018;2:41–50.
26. Vinayakumar R, Soman K, Poormachandran P. Applying convolutional neural network for network intrusion detection. In: 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India: IEEE, 2017. pp. 1222–1228.
27. Li Z, Qin Z, Huang K, et al. Intrusion detection using convolutional neural networks for representation learning. In: Liu D, Xie S, Li Y, et al. (Eds.), International Conference on Neural Information Processing, Guangzhou, China: Springer, 2017. pp. 858–866.
28. Wu K, Chen Z, Li W. A novel intrusion detection model for a massive network using convolutional neural networks. IEEE Access. 2018;6:50850–50859.
29. Altwaijry N, Alqahtani A, Al-Turaiki I. A deep learning approach for anomaly-based network intrusion detection. In: Tian Y, Ma T, Khan MK (Eds.), First International Conference on Big Data and Security, Nanjing, China: Springer, 2019.
30. Hamid Y, Balasaraswathi VR, Journaux L, et al. Benchmark datasets for network intrusion detection: A review. J Netw Secur. 2018;20:645–654.
31. Jolliffe IT, Cadima J. Principal component analysis: A review and recent developments. Philos Trans R Soc A Math Phys Eng Sci. 2016;374:20150202.
32. Tsai F. Comparative study of dimensionality reduction techniques for data visualization. J Artif Intell. 2010;3:119–134.
33. Akhbardeh A, Jacobs MA. Comparative analysis of nonlinear dimensionality reduction techniques for breast MRI segmentation. Med Phys. 2012;39:2275–2289.
34. Chawla NV, Bowyer KW, Hall LO, et al. SMOTE: Synthetic minority oversampling technique. J Artif Intell Res. 2002;16:321–357.
35. Alsenan SA, Al-Turaiki IM, Hafez AM. Feature extraction methods in quantitative structure–activity relationship modeling: A comparative study. IEEE Access. 2020;8:78737–78752.
36. Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: A simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15:1929–1958.
37. Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. 2014.
38. TensorFlow. Available online at https://www.tensorflow.org (last accessed April 21, 2019).
39. Home—Keras Documentation. Available online at https://keras.io (last accessed April 21, 2019).
40. Welcome To Colaboratory—Colaboratory. Available online at https://colab.research.google.com/notebooks/welcome.ipynb (last accessed April 21, 2019).
41. Frank E, Hall MA, Witten IH. The WEKA workbench. United States: Morgan Kaufmann. 2016.
42. Yang J, Ye Z, Yan L, et al Modified naive bayes algorithm for network intrusion detection based on artificial bee colony algorithm. In: 2018 IEEE 4th International Symposium on Wireless Systems within the Inter-national Conferences on Intelligent Data Analysis and Advanced Computing Systems (IDACCS-SWS), Lviv, Ukraine: IEEE, 2018. pp. 35–40.
43. Aljawarneh S, Aldawi M, Yassein MB. Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model. J Comput Sci. 2018;25:152–160.
44. Aburomman AA, Reaz MBI. A survey of intrusion detection systems based on ensemble and hybrid classifiers. Comput Secur. 2017;65:135–152.
45. Zhao H, Li M, Zhao H. Artificial intelligence based ensemble approach for intrusion detection systems. J Vis Commun Image Represent. 2019;102736.
46. Othman SM, Ba-Alwi FM, Alsobyhe NT, et al. Intrusion detection model using machine learning algorithm on Big Data environment. J Big Data. 2018;5:34.
47. Verma A, Ranga V. Machine learning based intrusion detection systems for IoT applications. WIREs Commun. 2020;11:2287–2310.
48. Sharafaldin I, Lashkari AH, Ghorbani AA. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISsp, 2018. pp. 108–116.
49. CSE-CIC-IDS2018 on AWS. available at https://www.unb.ca/cic/datasets/ids-2018.html (last accessed February 22, 2021).
50. Gamage S, Samarabandu J. Deep learning methods in network intrusion detection: A survey and an objective comparison. J Netw Comput Appl. 2020;169:101276.
Appendix

Appendix A1.

Network Intrusion Detection
Cybersecurity emerged with the emergence of the internet, with the protection of systems, hardware, software, and data from any cyber attacks as the main goal. Cyber attacks affect many of basic security principles such as confidentiality, integrity, availability, and authentication. Many cyber attacks have been developed over the years, and new ones are developed rapidly. There are two types of attacks: passive and active. In passive attacks, the attacker only views and monitors the content of the system. However, active attacks are where the attacker can alter, modify, and destroy data. Examples of attacks include but are not limited to: malware attacks, phishing attacks, SQL injection attacks, password attacks, eavesdropping, spoofing, DoS attacks, and probing attacks. Various methods have been used to detect these attacks, such as Multivariate Correlation Analysis, Data Mining Algorithms, K Nearest Neighbor, as well as deep learning, which shall be presented next.

Deep Learning
Deep learning has emerged as a field of machine learning with many successful real-world applications. Unlike traditional machine learning algorithms, deep learning does not assume the availability of handcrafted features. Deep learning algorithms are able to automatically capture features using unsupervised or semi-supervised feature learning algorithms. A short review of various methods used for NIDS is presented next.

Fully connected ANN
An ANN is the simplest form of a deep learning architecture. It consists of a feedforward network as the first step. Feedforward networks are the quintessential deep learning model and are of extreme importance to machine learning practitioners, forming the basis of many important commercial applications. It is composed of many neurons, where each neuron is connected to every neuron in the layers preceding or following it. Weights are associated with each connection. Signals in a feedforward network only travel in one direction, starting from the input and moving to output. Then, this is followed by backpropagation, which applies the gradient descent optimization algorithm to minimize the cost function by adjusting the weights. Appendix Figure A1 shows an example of a fully connected neural network.

Convolutional neural network
A CNN is a deep learning architecture that has shown success in computer vision and image processing applications. A CNN usually consists of a number of alternate layers, including convolution, pooling, and fully connected layers.
Convolution layer: convolution refers to the application of $K$ filters (also known as kernels). Filters are used to capture features such as corners and edges in images. A filter is a grid of size $N \times N \times C$, where $N$ is the height and width of the filter, and $C$ is the number of channels in the input image. Convolution is applied such that each filter slides over all pixels in the input image. The pixel value is multiplied by the corresponding value in the filter. The multiplication results are summed to produce a single number. The result of each convolution is called a feature map. Thus, applying $K$ convolutional filters yields a set of $K$ feature maps, each of size $((N - F)/\text{stride}) + 1$, where stride is the number of pixels the filter is moved across in one step. The convolution operation is followed by the application of an activation function. Feature maps are passed to activation functions such as ReLU or Sigmoid function to allow for the classification of linearly separable classes.

Pooling layer: Pooling is an operation that is used to reduce the spatial size of the image. The pooling layer is applied on each feature map independently. The most common type of pooling is max-pooling.

Fully connected layer: A fully connected layer is usually used at the end of the architecture for the purpose of classification. It is composed of many neurons, where each neuron is connected to all neurons in the previous layers. Fully connected layers provide the final nonlinear combinations of features. An example of a convolutional neural network is shown in Appendix Figure A2.

Baseline state-of-art algorithms

NB based on Bayes theorem. NB is a simple probabilistic classification algorithm. It is described as Naive since it assumes conditional independence among class attributes. Research has shown that NB is stable and has comparable performance to other machine learning algorithms. However, conditional independence is not necessarily true in many real-world applications. NB is incapable of handling missing data.

J48 decision trees. A decision tree (DT) is an induction algorithm that generates a tree-like model describing the relationship between attributes and a class label. It works by recursively dividing observations based on the most informative attribute with the highest gain ratio value. DTs are suitable for data sets with categorical data. Continuous data can also be handled when converted to categorical data. In addition to being robust, DT models have the advantage of being easy to interpret, as the model is represented as a set of rules.

A recurrent neural network

An RNN is a neural network that operates in time. At each time step, it accepts an input vector, updates its hidden state via nonlinear activation functions, and uses it to predict its output. RNNs need to keep track of states, which is computationally expensive, and suffer from the exploding and vanishing gradient problem. A LSTM is a class of RNN that is an abstraction for computer memory to help solve these problems. LSTM networks are most suited to classifying, processing, and making predictions based on time series data. A common LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate, all of which cooperate to solve the above problems.
Bagging Bootstrap aggregation. Bagging Bootstrap aggregation is an ensemble machine learning algorithm. Ensemble classifiers are based on combining the predictions of many weak classifiers to improve the prediction performance. In bagging, the training data set is sampled with replacement a number of times, and in each iteration, a classifier is built. To classify an unknown data point, each individual classifier returns its prediction. The bagging classifier counts the votes of each individual classifier, and the new data point is assigned the class with the most votes.

Random forest. RF is an ensemble machine learning method that creates multiple trees through a resampling process called bagging (bootstrap aggregation). Numerous DTs are constructed by resampling using bootstrapping with replacement. Each node of the tree is split using a subset of the attributes that are selected randomly for each tree. Class membership for a new example is predicted as the most commonly predicted class from the (aggregated) decision trees by a simple unweighted majority vote. This method is becoming widely used and has been shown to be rather effective for highly complex multicriteria decision-making problems in a variety of fields. RF is best suited for high-dimensional data sets; it has the ability to deal with missing data and imbalanced data sets.

Adaboost. Adaptive boosting is an ensemble algorithm that is based on boosting. A model is built sequentially, where it is boosted by re-weighting instances based on the performance of the previous model. Adaboost improves the performance of individual weak classifiers, but it is sensitive to noise. It is mainly designed for classification and prediction tasks.

Data sets
Network security laboratory-knowledge discovery in databases

The NSL-KDD data set is an improved version of the KDD’99 data set. It is a smaller data set that provides better evaluation of classifiers since redundant records are removed. Redundant records cause learning classifiers to be biased toward the more frequent records during training, as well as increasing classification accuracy whenever these same records appear in the test set. The training set KDDTrain contains 125,973 records, and the testing set KDDTest contains 22,544 records. In addition, the KDDTest contains 11,850 records, where these records were not classified correctly by all 21 classifiers in. The data set simulates the following types of attacks:

1. DoS: where an attacker attempts to make some resource too busy to handle valid requests or denies legal users access to a machine.
2. Probing Attack: in which an attacker attempts to collect data on a network of computers to find a way around an obstacle to violate the network’s security controls.
3. U2R: an attack in which an attacker gains access as a normal user account on the system, then searches to find any vulnerability to exploit and gain root access to the system.
4. R2L: send packets to a machine via a network, where the attacker does not have an account on that machine, to exploit some breaches to earn local access as a user of that machine.

The data set has both normal and anomaly traffic. Anomaly traffic is in one of four categories: DoS, Probe, R2L, and U2R. The number of instances and attribute names of different attack classes in the NSL-

### Appendix Table A1. Distribution of attacks in the NSL-KDD data set

| Attack | Training set | Testing set | Attribute |
|--------|--------------|-------------|-----------|
| DoS    | 45,927       | 7,458       | back, land, teardrop, neutron, pod, smurf |
|        | 11,656       | 2,421       | ipsweep, nmap, portsweep, satan |
|        | 52           | 200         | loadmodule, buffer-overflow, perl, rootkit |
|        | 995          | 2,754       | fpt-write, guess-passwd, imap, multipath, phf, spy, warezclient, warezmaster, snmpgetattack, http_tunnel, sendmail, named |

The testing set has some specific attack types that are not present in the training set, such attacks are given in bold.

R2L, remote to local attack; U2R, user to root attack.
KDD data set are shown in Appendix Table A1. The testing set has some specific attack types that are not present in the training set, thereby allowing realistic intrusion detection system evaluation. Such attacks are given in bold in Appendix Table A1.

Each record in the NSL-KDD data set has 41 features, divided into 3 groups: Basic features are derived from TCP-IP connections, traffic features are collected from window intervals or the number of connections, and content features are taken from the application layer data of connections. The details of the 41 attributes are shown in Appendix Table A2.

University of New South Wales Network Based 2015

Although the NSL-KDD data set solved the problems of data imbalance and redundancy found in KDD’99, it is an old data set that does not include new types of attacks. The UNSW-NB15 data set is a recent data set created in 2015 by the ACCS. It is being recently used in some studies as it overcomes the limitations of both KDD’99 and NSL-KDD data sets. It contains a combination of normal activity and contemporary synthesized attacks in network traffic.

The data are available in two forms, either as 2 million full connection records with 47 features excluding the class label or a subset composed of a training set of 175,341 records, and a testing set of 82,332 records, and 42 features, excluding the class label. The data set has the following 10 classes:

- **normal**: normal connections.
- **Fuzzers**: an attacker attempts to suspend the network by feeding it randomly generated data.
- **Analysis**: represents different attacks such as port scan, spam, and HTML files penetrations.
- **Backdoor**: an attacker attempts to bypass security mechanisms to access a computer or its data.
- **DoS**: prevents authorized requests from accessing a device due to spurious requests.
- **Exploits**: exploit a known vulnerability in the system.
- **Generic**: a technique that establishes against every block-cipher using a hash function to collision without respect to the configuration of the block-cipher.
- **Reconnaissance**: also defined as probe, simulates attacks that gather network information to avoid them.
- **Shellcode**: the attacker penetrates a slight piece of code starting from a shell to control the target machine.
- **Worms**: attacker replicates itself to spread to other computers. Often, it uses a computer network to spread itself, relying on security failures on the target machine.

The numbers of attacks in the training and testing sets are shown in Appendix Table A3.

Each record in the UNSW-NB15 full connection records data set has 47 features, divided into 5 groups as follows:

1. Flow features: includes the identifier attributes between hosts (e.g., client-to-server or server-to-client).

### Appendix Table A2. Details of the 41 attributes in the NSL-KDD data set

| No. | Feature name               | Type |
|-----|----------------------------|------|
|     | Basic features             |      |
| 1   | Duration                   | Con  |
| 2   | Protocol_type              | Sym  |
| 3   | Service                    | Sym  |
| 4   | Flag                       | Sym  |
| 5   | Src_bytes                  | Con  |
| 6   | Dst_bytes                  | Con  |
| 7   | Land                       | Sym  |
| 8   | wrong_fragment             | Con  |
| 9   | Urgent                     | Con  |
| 10  | Bot                        | Con  |
|     | Content features           |      |
| 11  | Num_failed_logins          | Con  |
| 12  | Logged_in                  | Sym  |
| 13  | Num_compromised            | Con  |
| 14  | Root_shell                 | Con  |
| 15  | Su_attempted               | Con  |
| 16  | Num_root                   | Con  |
| 17  | Num_file_creations         | Con  |
| 18  | Num_shells                 | Con  |
| 19  | Num_access_files           | Con  |
| 20  | Num_outbound_cmds          | Con  |
| 21  | Is_host_login              | Sym  |
| 22  | Is_guest_login             | Sym  |
|     | Traffic features           |      |
| 23  | count                      | Con  |
| 24  | Srv_count                  | Con  |
| 25  | Serror_rate                | Con  |
| 26  | Srv_serror_rate            | Con  |
| 27  | Rerror_rate                | Con  |
| 28  | Srv_rerror_rate            | Con  |
| 29  | Same_srv_rate              | Con  |
| 30  | diff_srv_rate              | Con  |
| 31  | Srv_diff_host_rate         | Con  |
| 32  | Dist_host_count            | Con  |
| 33  | Dist_host_srv_count        | Con  |
| 34  | Dist_host_same_srv_rate    | Con  |
| 35  | Dist_host_diff_srv_rate    | Con  |
| 36  | dst_host_same_src_port_rate| Con  |
| 37  | Dist_host_srv_diff_host_rate| Con  |
| 38  | Dist_host_serror_rate      | Con  |
| 39  | Dist_host_srv_serror_rate  | Con  |
| 40  | Dist_host_rerror_rate      | Con  |
| 41  | Dist_host_srv_rerror_rate  | Con  |

Con, continuous; Sym, symbolic.
2. Basic features: involves the attributes that represent protocols connections.
3. Content features: encapsulates the attributes of TCP/IP; also they contain some attributes of http services.
4. Time features: contains the attributes time, for example, arrival time between packets, start/end packet time, and round trip time of TCP protocol.
5. Additional generated features: this category can be further divided into two groups: general purpose features, where each feature has its own purpose, according to protect the service of protocols, and connection features that are built from the flow of 100 record connections based on the sequential order of the last time feature.

Appendix Table A3. Distribution of attacks in the UNSW-NB15 data set

| Attack class  | Training set | Testing set |
|---------------|--------------|-------------|
| Normal        | 56,000       | 37,000      |
| Fuzzers       | 18,184       | 6062        |
| Analysis      | 2000         | 677         |
| Backdoor      | 1746         | 583         |
| DoS           | 12,264       | 4089        |
| Exploits      | 33,393       | 11,132      |
| Generic       | 40,000       | 18,671      |
| Reconnaissance| 10,491       | 3496        |
| Shellcode     | 1133         | 378         |
| Worms         | 130          | 44          |
| Total         | 175,341      | 82,332      |

Appendix References
A1. Tan C, Sun F, Kong T, et al. A survey on deep transfer learning. In: Kůrková V, Manolopoulos Y, Hammer B, Iliadis L, Maglogiannis I (eds). Artificial Neural Networks and Machine Learning – ICANN 2018. ICANN 2018. Lecture Notes in Computer Science, vol. 11141. Cham: Springer. https://doi.org/10.1007/978-3-030-01424-7_27.
A2. Han J, Kamber M. Data mining: Concepts and techniques. The Morgan Kaufmann Series in Data Management Systems, 3rd ed. United states: Morgan Kaufmann, 2011.
A3. Quinlan JR. Simplifying decision trees. Int J Man Mach Stud. 1987;27:221–234.
A4. Ho TK. The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell. 1998;20:832–844.
A5. Freund O, Schapire R. A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci. 1997;55:119–139.
A6. 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, Ottawa, Ontario, Canada: IEEE, 2009. pp. 1–6.
A7. Hamid Y, Balasaraswathi VR, Journaux L, Sugumaran M. Benchmark datasets for network intrusion detection: A review. IJ Netw Secur. 2018;20:645–654.
A8. Moustafa N, Slay J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 military communications and information systems conference (MICIS), Canberra, ACT, Australia: IEEE, 2015. pp. 1–6.