Journal of
Cybersecurity and Privacy

MDPI

Article

A Deep Learning Approach for Network Intrusion Detection Using a Small Features Vector

Humera Ghani *, Bal Virdee and Shahram Salekzamankhani

Centre for Communications Technology, School of Computing and Digital Media, London Metropolitan University, London N7 8DB, UK
* Correspondence: hug0051@my.londonmet.ac.uk

Abstract: With the growth in network usage, there has been a corresponding growth in the nefarious exploitation of this technology. A wide array of techniques is now available that can be used to deal with cyberattacks, and one of them is network intrusion detection. Artificial Intelligence (AI) and Machine Learning (ML) techniques have extensively been employed to identify network anomalies. This paper provides an effective technique to evaluate the classification performance of a deep-learning-based Feedforward Neural Network (FFNN) classifier. A small feature vector is used to detect network traffic anomalies in the UNSW-NB15 and NSL-KDD datasets. The results show that a large feature set can have redundant and unuseful features, and it requires high computation power. The proposed technique exploits a small feature vector and achieves better classification accuracy.

Keywords: deep learning; feedforward neural network; network intrusion detection; UNSW-NB15; NSL-KDD

check for updates

Citation: Ghani, H.; Virdee, B.; Salekzamankhani, S. A Deep Learning Approach for Network Intrusion Detection Using a Small Features Vector. *J. Cybersecur. Priv.* **2023**, *3*, 451–463. https://doi.org/10.3390/jcp3030023

Academic Editor: Feng Wang

Received: 9 June 2023
Revised: 24 July 2023
Accepted: 31 July 2023
Published: 3 August 2023

1. Introduction

The internet has become indispensable for business, commerce, social, and recreational pursuits. With the rapid growth in network usage, its exploitation has correspondingly proliferated. According to a World Economic Forum report, cybersecurity failure is becoming a critical threat to the world [1]. There is a wide array of techniques to deal with cyberattacks, and one of them is identifying anomalies in network traffic. Different approaches are used to detect network traffic anomalies. Two widely used techniques are signature-based and anomaly-based. The signature-based technique detects the known attack signatures, while the anomaly-based technique detects deviations from the normal traffic profile.

Advancements in Artificial Intelligence (AI) and Machine Learning (ML) over the last few years have led to great interest in using these powerful techniques to fight cybercrime [2]. Recently, various techniques have been developed to classify network traffic anomalies, including the use of Decision Tree (DT), Principal Component Analysis (PCA), and the Random Forest (RF) classifier [3], the Long Short-Term Memory—Recurrent Neural Network (LSTM-RNN) classifier [4], and a Deep Neural Network [5]. These techniques have been tested on the UNSW-NB15, KDDCup99, and NSL-KDD datasets. Each of these techniques attempts to reduce the input feature space, as redundant and irrelevant features increase the computational complexity and decrease the classification performance [6].

The authors in [3] employed a nonsymmetric deep autoencoder to compress the input feature space. Their technique reduced the number of input features from 41 to a compressed representation of size 28. The resulting feature set was then passed to the RF classifier, which classified them into five classes consisting of four attack categories and one normal class. In contrast, the research conducted in [4] utilized LSTM and RNN models with an input feature space of size 41. These models also classified the records into five classes, including four attack categories and one normal class. The deep learning architecture employed in [4] compressed the initial 41 input features down to five outputs.

On the other hand, the authors in [5] employed a shallow learning approach, incorporating feature selection and feature extraction techniques to reduce the input feature space. In the feature selection stage, they utilized the DT algorithm to select 32 features from the original 41-dimensional feature space. Subsequently, in the feature extraction stage, they transformed the selected features into principal components. These authors reported the highest classification accuracy when using the RF classifier.

This research explores a distinctive approach in network anomaly detection, diverging from the commonly used deep and shallow learning methodologies. Instead of utilizing the complete feature set and subsequently applying deep or shallow learning methods to reduce the input space, our strategy begins with a smaller input feature space. Deep learning algorithms are then employed to generate a compressed representation, capitalizing on their ability to extract hierarchical and informative features. This departure from the traditional approach aims to enable more efficient processing in the lower-dimensional space. The main objective is to evaluate the classification accuracy of network anomaly detection using a Feedforward Neural Network (FFNN) with a reduced feature vector. Our approach involves utilizing a smaller input space while leveraging the capabilities of an FFNN to generate a compressed representation, thereby minimizing the computational power requirements. To achieve a comprehensive representation of normal and attack traffic, the UNSW-NB15 and NSL-KDD datasets were employed for a thorough analysis of the network behavior.

To the best of the authors' knowledge, no previous study of this nature has been conducted. The results demonstrate that the proposed technique achieves an accuracy of 91.29% and 89.03% on the UNSW-NB15 and NSL-KDD datasets, respectively, for binary classification. The main contributions of this paper include:

- Exploration of the contemporary research to identify a reduced feature vector for evaluating the classification accuracy on the UNSW-NB15 and NSL-KDD datasets.
- Preparation of the dataset for classification, including removing redundancy and inconsistency, as well as transforming and scaling the data to make them suitable for feeding into the classifier.
- Training and testing of the model and evaluation of its performance using the appropriate evaluation metrics.
- Comparison of the obtained results with the contemporary research to assess the effectiveness of the proposed approach.

This paper is organized as follows: Section 2 describes the related work reported recently in the literature. Section 3 discusses the background of the theoretical and conceptual framework of this work. Section 4 elaborates on the proposed methodology. Section 5 presents the results. Section 6 discusses the results and findings of this work, and finally, Section 7 concludes the work presented in this paper and gives recommendations for future work.

2. Related Work

The authors in reference [3] developed a nonsymmetric deep autoencoder (NDAE) to extract features and utilized an RFC for classification purposes. Their NDAE successfully reduced the initial 41 input features to 28. The experiments were conducted on the KDDCup99 and NSL-KDD datasets. In the case of five-class classification, the KDDCup99 dataset yielded an accuracy of 97.8%, while the NSL-KDD dataset achieved 85.4% accuracy. In our research, we also employed the NSL-KDD dataset, but we utilized a deep learning approach for classification. As a result, we were able to attain a higher accuracy using a smaller feature vector.

In [4], the researchers employed the KDDCup99 dataset, which contained 41 input features. They proposed an LSTM-RNN classification model. The experimental process involved two stages, firstly, determining the best values for the hyperparameters and, secondly, assessing the classification performance using these chosen hyperparameters. The authors conducted five-class classifications and achieved an accuracy of 96.9%. This

research successfully leveraged the capabilities of deep learning approaches to achieve a high level of accuracy.

In [5], the authors conducted an evaluation of five distinct datasets utilizing six different classifiers, namely: Naïve Bayes, decision trees, random forests, neural networks (multilayer perceptron), support vector machines (RBF kernel), and l2-logistic regression. The random forests classifier yielded the best performance among the classifiers assessed. The authors only provided the results for this particular classifier. For feature selection, the authors employed the decision tree algorithm, and for feature extraction, they utilized principal component analysis. However, the authors did not specify the final size of the input vector after transformation into principal components. On the UNSW-NB15 dataset, the authors reported a test accuracy of 98.9%. It is worth noting that this research achieved a higher accuracy than our own. Nonetheless, it is important to highlight that the authors ended up with 32 features after performing feature selection, out of a total of 41 features, which is a significantly larger number than in our study, where only nine features were retained.

The authors in [7] employed both the NSL-KDD and UNSW-NB15 datasets in their study. They utilized several models for network intrusion detection, namely support vector machine, multilayer perceptron, restricted Boltzman machine, sparse autoencoder, and wide and deep learning. They used all features. The results revealed that the sparse autoencoder model achieved a better accuracy of 79.3% on the NSL-KDD dataset, while the wide and deep network models attained a higher accuracy of 91.2% on the UNSW-NB15 dataset. Despite using a smaller feature vector in both cases, our technique demonstrated superior performance.

In [8], self-taught and deep learning approaches were based on the NSL-KDD dataset. The approach used consisted of two stages, where feature learning was implemented in the first stage and classification in the second stage. The first stage used unlabeled data, whilst labelled data were used in the classification stage. Sparse autoencoder was applied for feature learning, and softmax regression was employed for classification. This approach achieved 88.39% test accuracy on two-class data. Notably, we achieved an even higher accuracy metric on the same dataset by employing a smaller feature vector.

The authors in [9] utilized all the features available in the UNSW-NB15 and KDD99 datasets for classification purposes. They employed five different classifiers: Naïve Bayes, decision trees, artificial neural networks, logistic regression, and expectation-maximization. Although the authors did not mention the number of classes classified, the inclusion of logistic regression suggests that a two-class classification was performed. On the UNSW-NB15 dataset, they achieved the highest classification accuracy of 85.5% using the decision trees algorithm. However, our research achieved even higher accuracy using a smaller feature vector.

The work in [10] covered flow-based network anomaly detection in a Software Defined Networking (SDN) environment using the NSL-KDD dataset. Six flow features were used from a total of 41 features. Here, a feedforward neural network with three hidden layers was used for classification. The model was referred to as a Deep Neural Network (DNN). They reported 75.75% test accuracy when a 0.001 learning rate was applied. While it is worth noting that our achieved result surpassed theirs, it is important to consider that they employed a smaller number of features in their analysis, which may not have adequately generalized the training set. Furthermore, the sole criterion for selecting these features was that they were flow-based features.

Reference [11] used six different datasets. We discuss this paper with reference to the UNSW-NB and NSL-KDD datasets. The authors designed a multilayer perceptron feedforward network for classification and proposed a Deep Neural Network (DNN) architecture. All the features of the datasets were used for the classification. The authors reported an accuracy of 78.4% on the UNSW-NB15 dataset and 80.1% on the NSL-KDD dataset. It is important to note that these accuracies were lower than those achieved by our approach. We present a comparison table of their results with our results later in Section 6.

In the study [12], all 41 features of the NSL-KDD dataset were utilized. However, after data preprocessing, the number of features increased to 122, resulting in an input vector size of 122. To classify the anomalies, a convolutional neural network with two hidden layers was employed. The classification task was performed on different versions of the NSL-KDD dataset. The highest achieved accuracy of 79.48% was observed on the KDDTest+ dataset, which was lower than the accuracy we achieved using a smaller feature vector in our research. Additionally, the authors proposed an approach to address the issue of data imbalance. Although valuable, this approach was not directly related to our work; therefore, it is not discussed in this study.

Reference [13] conducted feature selection on three datasets: KDDCup99, UNSW-NB15, and NSL-KDD. They employed the fruit fly algorithm, the ant lion optimizer algorithm, and a hybrid version for selecting the most relevant features. The selected features were then evaluated using SVM, KNN, Naïve Bayes, and decision tree classifiers. In our work, we focused on the fruit fly feature selection method and utilized the features identified by this algorithm. Compared to the other two algorithms, the fruit fly algorithm selected the smallest number of features, making it suitable for our study. The authors of reference [13] reported achieving the highest accuracy of 89.0% using the k-nearest neighbor classifier on the NSL-KDD dataset and 90.6% accuracy using the decision tree algorithm on the UNSW-NB15 dataset, both utilizing the features selected by the fruit fly algorithm. However, it should be noted that our research achieved even higher accuracy by employing a deep learning algorithm, whereas the referenced research used shallow learning algorithms.

The authors in reference [14] employed five variants of autoencoders to detect network traffic anomalies in the UNSW-NB15 and NSL-KDD datasets. In this unsupervised approach, the authors leveraged the power of deep learning to learn the compressed representation. On the NSL-KDD test dataset, they achieved an accuracy of 87.9% using the contractive autoencoder, while on the UNSW-NB15 dataset, they achieved an accuracy of 86.6% using the convolutional autoencoder. In both cases, our approach yielded better results than this approach, suggesting that relying solely on deep learning approaches for feature reduction was insufficient. It is better to employ other feature selection techniques to remove unnecessary features before inputting them into the deep learning model.

In reference [15], the authors utilized the NSL-KDD, UNSW-NB15, and CICIDS2017 datasets. In our discussion, we focus on the first two datasets, as they were also used in our own research. They proposed a memory-augmented deep autoencoder to address the overgeneralization issue of autoencoders. On the NSL-KDD dataset, they achieved an accuracy of 89.5%, while on the UNSW-NB15 dataset, they achieved an accuracy of 85.3%. Our proposed model performed better on the UNSW-NB15 dataset, while our research achieved comparable results on the NSL-KDD dataset. This research also relied on the feature compression ability of the deep learning models and did not make any attempt to reduce the number of input features using statistical or machine learning techniques.

Reference [16] tested their proposed method on the UNSW-NB15 dataset. They divided the dataset based on the TCP, UDP, and OTHER protocol categories. For each protocol category, they performed feature selection using the chi-square method, which reduced the number of features by approximately half. For classification, they used a one-dimensional Convolutional Neural Network (CNN). They presented their results for each protocol category separately as well as an overall result. The authors reported an accuracy of 76.3% for the overall categories, which was lower than what we achieved on the same dataset.

In [17], the UNSW-NB15 dataset and variants of the NSL-KDD (KDDTest+ and KDDTest-21) were used. This paper utilized a supervised variational autoencoder, where the latent vector was computed using Wasserstein GAN. It effectively addressed the class imbalance issue in the dataset by synthesizing low-frequency attacks. The proposed approach achieved an accuracy of 93.0% on the UNSW-NB15, 89.3% on the KDDTest+, and 80.3% on the KDDTest-21 datasets. Although the authors did not perform feature reduction

separately, their results demonstrated a comparable accuracy metric to ours due to the removal of the class imbalance issue.

The authors of [18] used the UNSW-NB15 and NSL-KDD datasets for their experiments. Initially, the researchers addressed the issue of class imbalance by generating synthetic records via the application of a Generative Adversarial Network (GAN). Subsequently, they employed a multiclass support vector machine classifier to carry out the classification task. Additionally, the parameters of the classifier were optimized using Bayesian hyperparameter optimization. The authors achieved classification accuracies of 99.5% on the NSL-KDD dataset and 85.38% on the UNSW-NB15 dataset. The reported accuracy in this research outperformed our own findings on the NSL-KDD dataset due to removing the class imbalance issue in the dataset.

3. Background Theoretical and Conceptual Framework

3.1. Artificial Neuron

An artificial neuron is the basic building block with which to construct a neural network. It is simply a computational unit that performs a computation based on other connected units. As shown in the Figure 1, each neuron is connected to the input vector X. This vector is composed of scalar values $x_1, x_2, \ldots, x_d$, where subscript d refers to the size of vector X. An artificial neuron reads the information from vector X and performs a particular computation, which dictates its value. The computation of a neuron can be decomposed into two steps: pre-activation and activation. The pre-activation function is computed by multiplying the weight of each connection $w_1, w_2, \ldots, w_d$ with the corresponding input value; then, the bias term b is added. An activation function defines the output of the neuron g[a(X)]. It is computed over the pre-activation value a(X).

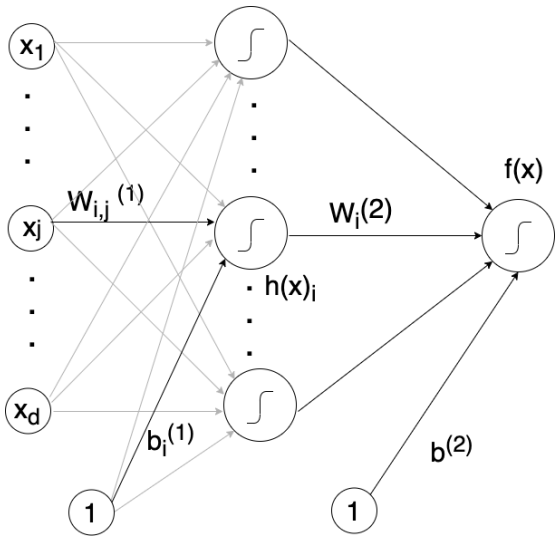

Figure 1. Single-layer neural network.

3.2. Activation Function

Popular choices for a neuron's activation function are the sigmoid, hyperbolic tangent, and rectified linear unit. The sigmoid keeps the neuron's pre-activation between 0 and 1. If the pre-activation value is larger, it saturates towards 1 while if the pre-activation is smaller, it saturates towards 0. The hyperbolic tangent activation function keeps the neuron's pre-activation between -1 and 1, i.e., the function is bounded and can have a positive or negative value. The rectified linear unit (ReLU) activation function is simply the maximum between 0 and the pre-activation value. This function gives a positive result if the pre-activation is positive and 0 if the pre-activation is negative. This activation function is not bounded because the greater pre-activation, the greater the output. It tends to give

neurons with sparse activation values; this means it often gives a neuron output that is exactly zero.

3.3. Single-Layer Neural Network

It is well known that a single artificial neuron cannot model well because it cannot solve nonlinearly separable problems unless the input is transformed into a better representation. This makes it necessary to use several neurons. Figure 1 shows a single layer neural network, where several neurons are used in a hidden layer to transform the inputs into a more conducive representation. This mechanism makes the problem linearly separable. Hence, in a typical single-layer neural network, the first part computes the representation, while the second part classifies it. Each of the *i*th neurons in the hidden layer is able to connect with all the other inputs so that the connection between the ith hidden neuron with the *j*th input can be expressed in matrix $W_{i,j}{}^{(1)}$, where the superscript represents the hidden layer number. The hidden layer bias is represented as $b^{(1)}$, and X represents the input vector. Equation (1) represents the pre-activation computation in a hidden layer.

$$a(X) = b^{(1)} + W^{(1)}X \tag{1}$$

Equation (2) computes the activation $h(X)$ over all the elements of pre-activation $a(X)$.

$$h(X) = g[a(X)] \tag{2}$$

To compute the output activation $f(x)$ in (3), the activation of the hidden units $h^{(1)}$ is multiplied by $W^{(2)}$, which weights the vector between the output and all the hidden units. In Figure 1, $W_i{}^{(2)}$ represents weight of the connection between the output and the ith hidden unit; then, the bias term b is added, and X represents the input vector.

$$f(x) = o\left[b^{(2)} + W^{(2)^T}h^{(1)}(X)\right] \tag{3}$$

3.4. Binary and Multiclass Classification

To perform binary classification, the same choices are made to compute the activation of the output neuron as in the hidden layer neurons. However, if multiclass classification is required, different choices are made as compared to the hidden unit's activation. Figure 2 shows a typical example of a multiclass classification neural network. It has multiple output units, and each unit represents an output class. Each output unit gives the probability of the occurrence of the input class to which it represents. The softmax activation function is used in multiclass classification. In the case of multiple output units, o(a) is used to represent a vector of output values. This vector is normalized, since the denominator is the sum of the numerators, and this ensures that the sum of all the output probabilities is always 1. This function is strictly positive as the exponent of any number is greater than 0. To predict an actual class of an input x, a class with the highest estimated probability from output vector o(a) is given by:

$$f(x) = o(a) = \left[\frac{\exp(a_1)}{\sum_c exp(a_c)} \cdots \frac{\exp(a_c)}{\sum_c exp(a_c)}\right]. \tag{4}$$

Input vector a is the pre-activation value of the final layer of the neural network. Exponential transformation is applied on elements of a. To ensure that the resulting values lie between 0 and 1, a normalization constant c is applied. It is the sum of the exponential values of all the elements of the input vector. Each exponential value is divided by the normalization constant, and each element of the softmax output vector represents the probability of the corresponding class.

3.5. Multilayer Neural Network

Multilayer networks are more powerful than single layer networks. In [7], it is shown that a two-layer network can be trained to approximate any arbitrary function. Figure 2

represents a multilayer neural network. This network transforms an input vector into hidden layers, where each hidden layer is composed of hidden units that compute using linear and then nonlinear transformation. A multilayer neural network can have any number of hidden layers. The total number of hidden layers is represented by (L), while the superscript (k) represents the kth hidden layer. The pre-activation layer can be computed as shown in equation (5), where b represents the bias term, W is the weights matrix, h is the hidden layer activation, and X is the input vector.

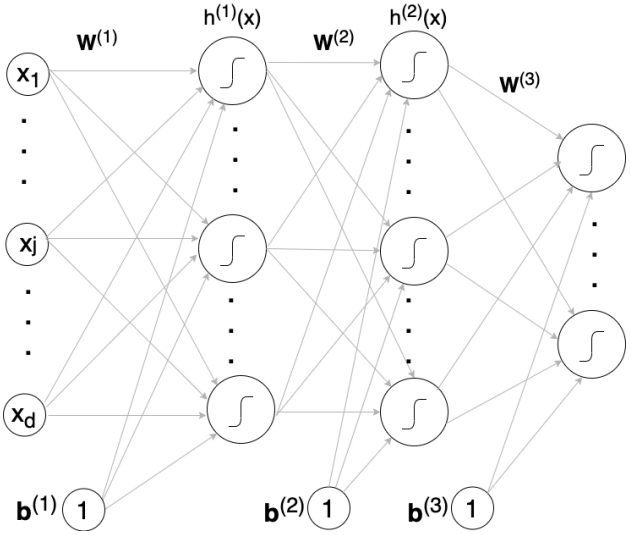

Figure 2. Multilayer neural network.

$$a^{(k)}(X) = b^{(k)} + W^{(k)}h^{(k-1)}(X) \tag{5}$$

The hidden layer activation is computed as:

$$h^{(k)}(X) = g\left[a^{(k)}(X)\right], \tag{6}$$

and the output layer activation is computed as:

$$f(x) = h^{(L+1)}(X) = o\left[a^{(L+1)}\,(X)\right]. \tag{7}$$

The correct magnitudes of these hidden units are not known initially at a given input. A neural network is configured to determine these values by tuning the model's parameters in such a way that the estimated values of the hidden layers eventually converge.

3.6. Forward Pass

A forward pass is completed once the training data are processed through the network. The network output is computed once the computation from each neuron is completed. There is a high chance that this output will not close to a specified target because the weights and biases of the network parameters are randomly initialized.

3.7. Cost Function

The cost function provides quantification of how much the model's estimation varies from the specified target. The aim is to minimize this cost. In the proposed work, the cost function of the Binary Cross Entropy is used for two-class classification.

3.8. Parameter Optimization

Training involves finding the values of parameters so that the neural network can solve a given problem. Consider, for example, a multilayer neural network, shown in Figure 2, with parameters: $W^{(1)}$, $W^{(2)}$, $W^{(3)}$, $b^{(1)}$, $b^{(2)}$, and $b^{(3)}$. The values of these parameters were

determined using the empirical risk minimization approach. These parameters were first initialized randomly before applying in tandem the optimization and backpropagation approaches to optimize them.

3.9. Backpropagation

The backpropagation algorithm performs the following functions to train the neural network; it accepts the input, initializes the weights, computes the output, computes the loss, and backpropagates the error. The network error is computed by taking the difference between the labels and targets. It backpropagates the network error on each parameter based on their contribution on the network loss.

3.10. Optimization Algorithm

The cost function is minimized by the optimizer. This is achieved by tweaking the model's parameters. To perform its job, the optimization algorithm needs to know the step size (η) and the gradient of network parameters. The step size is a hyperparameter; its value is set using a trial-and-error approach, whereas the gradient of the parameters is computed using a backpropagation algorithm. The optimizer backpropagates errors to update network parameters. Many iterations of the optimization are required to complete the training. This is necessary to find the desired values of the network parameters. Popular choices for optimization algorithms are gradient descent, stochastic gradient descent, and Adam. In this work, we used the Adam optimizer, because the learning rate is achieved adaptively.

3.11. Model Evaluation

The model's performance was evaluated to determine the quality of its output results. This step is important because models are designed to predict the class of future unlabeled data points. Evaluation is performed, which is based on the following metrics: accuracy, detection rate (DR), and false positive rate (FPR).

The accuracy is a measure of how often the prediction of the model matches the labels, and the accuracy is calculated using:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{8}$$

The detection rate measures the ratio of normal packets, which the model correctly predicted as normal, to the actual number of normal packets. It is calculated using:

$$Detection\ Rate = \frac{TP}{TP + FN}. \tag{9}$$

The false positive rate measures the ratio of attack packets that the model incorrectly predicted as normal to the actual number of attack packets. It is calculated using:

$$False\ Positive\ Rate = \frac{FP}{FP + TN}, \tag{10}$$

where TP is True Positive, TN is True Negative, FP is False Positive, and FN is False Negative. These terms are defined as follows: TP indicates the model correctly predicted the positive class. TN indicates the model correctly predicted the negative class. FP indicates the model incorrectly predicted the positive class. FN indicates the model incorrectly predicted the negative class.

4. Methodology

4.1. Dataset

The work presented here was based on the UNSW-NB and NSL-KDD datasets. The UNSW-NB15 dataset [19] was created using Ixia's PerfectStorm ONE hardware tool in the

Cyber Range Lab of the Australian Centre for Cyber Security. The tool generated modern normal and attack behaviors. The Tcp dump tool was used to capture 100 GB of raw network traffic. Argus and Bro software were used, and 12 algorithms were written to create 49 features with class labels. In total, 13 features were categorized as basic features, five were flow features, eight were content features, nine were time features, and twelve were additional features. A total of 2,540,044 records were stored in four csv files. Part of this dataset was stored as the training and testing sets with 175,341 and 82,332 records, respectively. This dataset had total 45 features. It contained a mix of numerical, binary, and nominal features. The distribution of the normal and attack packets is shown in Table 1.

Table 1. Distribution of normal and attack packets.

Packet Type	UNSW-NB15		NSL-KDD	
	Training	Testing	Training	Testing
Normal	110,341	45,332	67,342	12,833
Attack	65,000	37,000	58,630	9711
TOTAL	175,341	82,332	125,972	22,544

The NSL-KDD dataset is an improved version of the KDDCup99 dataset [20]. It minimized the duplicate records of the training and testing datasets. It had 43 features, of which 24 were integer type, 15 were float type, and 4 were categorical. Its KDDTrain+ dataset had a total of 125,972 records, out of which 67,342 were normal, and 58,630 were attack records. Its KDDTest+ dataset had a total of 22,544 records, out of which 12,833 were normal records, and 9711 were attack records. It had four attack classes: DoS, Probe, U2R, and R2L.

4.2. Hardware and Software Platform

The work presented here was carried out using a 2.6 GHz 6-Core Intel Core i7. The computational system had 16 GB RAM and the macOS Catalina version 10.15.7 operating system. An Anaconda development environment was used with Python 3.7 version. The Python libraries, pandas, sickit-learn, and matplotlib, were used for writing program code and creating visualizations.

4.3. Feature Selection

We used the most useful features identified in [13]. They applied three different algorithms for feature selection. We used the selected features of the fruit fly algorithm, since this algorithm gave the smallest feature vector. These features are shown in Table 2.

Table 2. Features used in the proposed research.

Dataset	Features Used in the Proposed Research from [13]	
UNSW-NB15	Service	Dload
	Ackdat	Dmeans
	Smeans	res_bdy_len
	ct_state_ttl	is_ftp_login
	ct_src_ltm	
NSL-KDD	Service	Flag
	src_bytes	num_failed_logins
	su_attempted	is_guest_login
	Count	srv_rerror_rate
	srv_diff_host_rate	dst_host_same_src_port_rare
	dst_host_srv_rerror_rate	

4.4. Data Preprocessing

A data preprocessing step was necessary to transform the data into a suitable form for modeling. In this step, two tasks were performed. In the first task, feature scaling was performed to normalize the training and to test the datasets. In the second task, the label vector, with two nominal values, i.e., normal and attack, was encoded with the integer values.

4.5. Model Architecture

We used a feedforward neural network with five hidden layers. Each layer had 50, 30, 15, 5, and 2 neurons, respectively. The nonlinearity was achieved using the ReLU function, while the Adam function was used for optimization. For the UNSW-NB dataset, the maximum iteration value was 50, while for the NSL-KDD, it was 200. This information is shown in Table 3. Our choices of hidden layers, the number of neurons in each layer, the activation function, the maximum iterations, and the optimizer were finalized after the trial of experiments. The block diagram of the model architecture is shown in Figure 3.

Table 3. Model hyperparameters.

Hyperparameters	UNSW-NB15	NSL-KDD
Hidden Layer Sizes	50, 30, 15, 5, 2	50, 30, 15, 5, 2
Learning Rate	Adaptive	Inverse scaling
Activation Function	ReLU	ReLU
Optimizer	Adam	Adam
Maximum Iterations	50	200

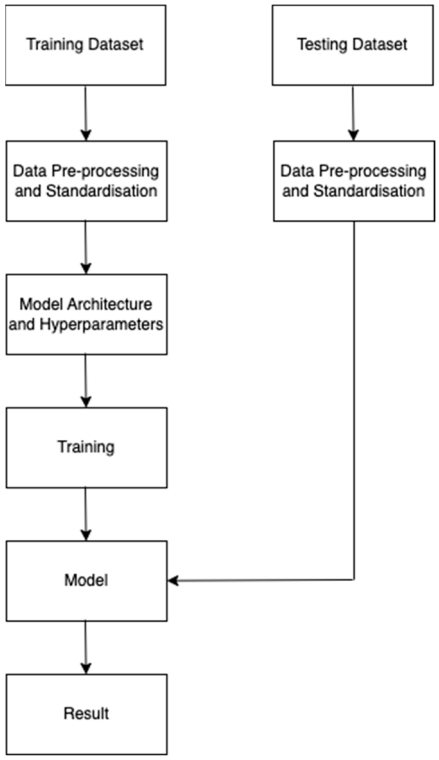

Figure 3. Block diagram of the proposed architecture.

4.6. Training

On the UNSW-NB15 dataset, the model was trained on 50 iterations, whereas on the NSL-KDD dataset, it was trained on 143 iterations, although we set the value of the maximum iteration (max_iter) parameter to be 200. The choice of the hidden layer's size is completely arbitrary, as there is no set rule for this. Using GridSearchCV class, the best options for the learning rate, activation function, and optimizer were identified and utilized.

Initially, each model ran a forward pass, where it randomly initialized the parameters and computed the output. This predicted output was compared with the actual labels thus allowing the loss to be determined. This was followed by running the backpropagation algorithm to compute the gradients. The errors were passed backward. This process was repeated until models were trained.

5. Results

Table 4 shows that by using the feedforward neural network on the reduced feature set of the UNSW-NB15 dataset, we achieved 91.29% accuracy, a 91.38% detection rate, and an 8.79% false positive rate. Using the same classifier on the reduced feature set of the NSL-KDD dataset we achieved 89.03% accuracy, a 95.65% detection rate, and a 17.59% false positive rate. Figures 4 and 5 show the model loss after each iteration. It is clear from Table 4 that the model achieved a better accuracy and false positive rate on the UNSW-NB15 dataset; however, a better detection rate was achieved on the NSL-KDD dataset. On the UNSW-NB15 dataset, the results were more consistent as compared to the NSL-KDD.

Table 4. Evaluation of the proposed model.

Dataset	Accuracy	Detection Rate	False Positive Rate
USNW-NB15	91.29%	91.38%	8.79%
NSL-KDD	89.03%	95.65%	17.59%

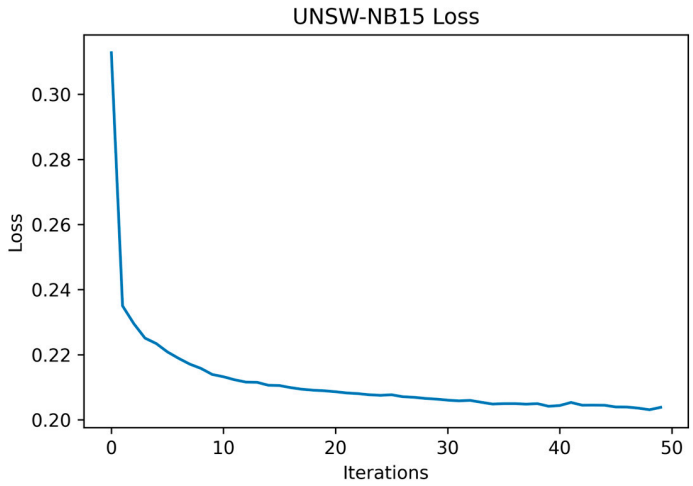

Figure 4. UNSW-NB15 loss.

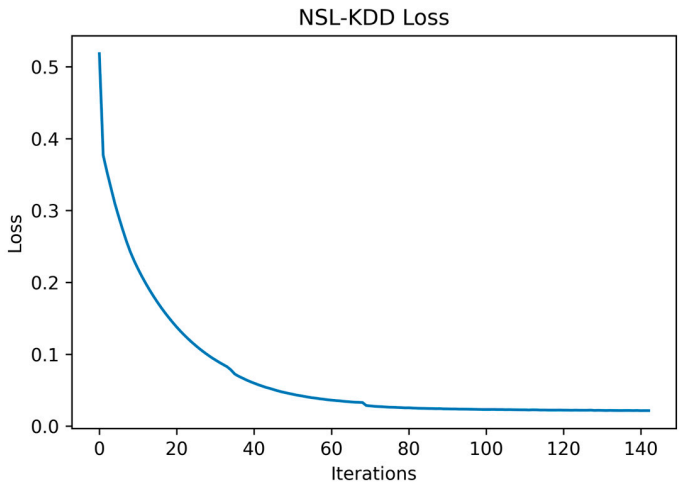

Figure 5. NSL-KDD loss.

6. Discussion

The proposed research utilized the same features as those used in [13]. It is clear from Table 5 that the proposed model, which employed an FFNN classifier, achieved higher accuracy compared to the decision tree (DT) and k-nearest neighbor (KNN) on both datasets. This is due to the inherent feature extraction ability of the FFNN. Conversely, in [11], all features were given to the deep neural network (DNN) classifier resulting in lower accuracy on both datasets. This is due to the absence of any feature selection method that could identify and retain only the important features while removing the unimportant features. This research confirms that all features are not useful, and a model's performance improves, when we use a small feature vector consisting only of features with predictive power. Using a small feature vector also requires less processing power.

Table 5. Comparison of the models.

Research	Dataset	Accuracy	Classifier	Features
[11]	UNSW-NB15	78.4%	DNN	41
	NSL-KDD	80.1%	DNN	41
[13]	UNSW-NB15	90.62%	DT	9
	NSL-KDD	89.02%	KNN	11
Proposed Research	UNSW-NB15	91.29%	FFNN	9
	NSL-KDD	89.03%	FFNN	11

7. Conclusions

This research investigated the classification performance of a deep-learning-based feedforward neural network classifier, when a lightweight feature vector was used to detect network traffic anomalies. The effectiveness of a small feature vector was demonstrated in detecting unsolicited and harmful network intrusion. We used the UNSW-NB15 and NSL-KDD datasets and the feedforward neural network classifier to perform the experimental study. The classification accuracy achieved confirms that the model performed better when a small feature vector was used.

A limitation of this research is that it only used the FFNN for classification. If other deep learning architectures had been utilized, we could have presented a comparative view of their performance and potentially achieved better results.

In the future, we intend to expand this work in the following directions: (i) perform classification using other deep learning algorithms to ascertain the degree of accuracy improvement; (ii) compute the classification accuracy of each attack type separately; (iii) implement this approach with real network traffic and evaluate the performance; (iv) for automated security, integrate this model with the continuous integration pipeline of an organization.

Author Contributions: Conceptualization, H.G. and B.V.; methodology, H.G.; software, H.G.; validation, H.G.; formal analysis, H.G.; investigation, H.G.; resources, H.G.; data curation, H.G.; writing—original draft preparation, H.G.; writing—review and editing—B.V. and S.S.; visualization, H.G.; supervision, S.S.; project administration, H.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are available in a publicly accessible repository. https://ieee-dataport.org/ (accessed on 1 February 2023).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. The Global Risks Report 2022. World Economic Forum. Available online: https://www.weforum.org/reports/global-risks-report-2022 (accessed on 24 September 2022).
2. Dario, A.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; Mané, D. Concrete problems in AI safety. *arXiv* **2016**, arXiv:1606.06565.
3. Nathan, S.; Ngoc, T.N.; Phai, V.D.; Shi, Q. A deep learning approach to network intrusion detection. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 41–50.
4. Jihyun, K.; Kim, J.; Thi Thu, H.L.; Kim, H. Long short term memory recurrent neural network classifier for intrusion detection. In Proceedings of the 2016 International Conference on Platform Technology and Service (PlatCon), Jeju, Republic of Korea, 15–17 February 2016; pp. 1–5.
5. Fares, M.; Zseby, T.; Iglesias, F. Analysis of lightweight feature vectors for attack detection in network traffic. *Appl. Sci.* **2018**, *8*, 2196. [CrossRef]
6. Khalid, S.; Khalil, T.; Nasreen, S. A survey of feature selection and feature extraction techniques in machine learning. In Proceedings of the 2014 Science and Information Conference, London, UK, 27–29 August 2014; pp. 372–378.
7. Jiaqi, Y.; Jin, D.; Lee, C.W.; Liu, P. A comparative study of off-line deep learning based network intrusion detection. In Proceedings of the 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN), Prague, Czech Republic, 3–6 July 2018; pp. 299–304.
8. Ahmad, J.; Niyaz, Q.; Sun, W.; Alam, M. A deep learning approach for network intrusion detection system. In Proceedings of the 9th EAI International Conference on Bio-Inspired Information and Communications Technologies (Formerly BIONETICS), New York, NY, USA, 3–5 December 2015; pp. 21–26.
9. Moustafa, N.; Slay, J. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Inf. Secur. J. A Glob. Perspect.* **2016**, *25*, 18–31. [CrossRef]
10. Tang, T.A.; Mhamdi, L.; McLernon, D.; Zaidi, S.A.R.; Ghogho, M. Deep learning approach for network intrusion detection in software defined networking. In Proceedings of the 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), Fez, Morocco, 26–29 October 2016; pp. 258–263.
11. Vinayakumar, R.; Alazab, M.; Soman, K.P.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access* **2019**, *7*, 41525–41550. [CrossRef]
12. 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. [CrossRef]
13. Samadi Bonab, M.; Ghaffari, A.; Soleimanian Gharehchopogh, F.; Alemi, P. A wrapper-based feature selection for improving performance of intrusion detection systems. *Int. J. Commun. Syst.* **2020**, *33*, e4434. [CrossRef]
14. Thavavel, V.; Binbusayyis, A. Application of deep autoencoder as an one-class classifier for unsupervised network intrusion detection: A comparative evaluation. *PeerJ Comput. Sci.* **2020**, *6*, e327.
15. Min, B.; Yoo, J.; Kim, S.; Shin, D.; Shin, D. Network anomaly detection using memory-augmented deep autoencoder. *IEEE Access* **2021**, *9*, 104695–104706. [CrossRef]
16. Hooshmand, M.K.; Hosahalli, D. Network anomaly detection using deep learning techniques. *CAAI Trans. Intell. Technol.* **2022**, *7*, 228–243. [CrossRef]
17. Yang, Y.; Zheng, K.; Wu, B.; Yang, Y.; Wang, X. Network intrusion detection based on supervised adversarial variational auto-encoder with regularization. *IEEE Access* **2020**, *8*, 42169–42184. [CrossRef]
18. Pandey, A.K.; Singh, P.; Jain, D.; Sharma, A.K.; Jain, A.; Gupta, A. Generative Adversarial Network and Bayesian Optimization in Multi-class Support Vector Machine for Intrusion Detection System. *Int. J. Intell. Eng. Syst.* **2023**, *16*, 110–119.
19. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, ACT, Australia, 10–12 November 2015; pp. 1–6.
20. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A detailed analysis of the KDD CUP 99 data set. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, ON, Canada, 8–10 July 2009; pp. 1–6.