Article

Not peer-reviewed version

Denial Of Service (DoS) Identification Using Auto Encoder

Mohammad Imtiyaz Gulbarga * and Khaled M. M. Alrantisi

Posted Date: 24 March 2025

doi: 10.20944/preprints202503.1704.v1

Keywords: autoencoder; deep neural networks; network Intrusion detection system; NSL-KDD; DoS

Article

Denial Of Service (DoS) Identification Using Auto Encoder

Mohammad Imtiyaz Gulbarga [1,*] and Khaled M. M. Alrantisi [2]

[1] Computer Network & Engineering, Coordinator Cybersecurity center, Computer Science department at Ala-Too International University

[2] Department of Computer Science and Engineering, Ala-Too International University

* Correspondence: imtiyaz.gulbagra@alatoo.edu.kg

Abstract: In this paper, we propose a deep learning neural network (DNN) using the Autoencoder model for the intrusion detection. An important component of any internet-connected device is a (NIDS). NIDS detects network-based attacks, such as Denial of Service (DoS) attacks was the focus of this research. We identified and analyzed Autoencoder for measuring its accuracy and precision and thus achieved 99% hit / success rate. To evaluate the performance, we utilized the network intrusion dataset Network Security Laboratory - Knowledge Discovery in Databases (NSLKDD).

Keywords: autoencoder; deep neural networks; network Intrusion detection system; NSL-KDD; DoS

1. Introduction

A network surveillance system (NSS), more commonly known as the Intrusion Detection System (IDS), is a system that scrutinizes network data and traffic for any dubious or threatening activity and notifies when such activity is identified. While the primary function of an IDS is anomaly detection and reporting, it can also take protective measures when malicious activity or abnormal traffic is detected, such as blocking traffic from suspicious Internet Protocol (IP) addresses.

An IDS operates differently from an Intrusion Prevention System (IPS), which inspects network packets for potentially harmful traffic with the main objective of averting threats once detected. The concept of IDS dates back to the 1980s [1], where it was observed that a system for monitoring and surveillance of computer security threats could identify the following behaviors: usage outside of standard hours, unusual frequency of access, atypical data reference volume, and irregular patterns in referencing programs or data. The IDS detects and identifies abnormalities in networks (both host-based and network-based) and prevents, detects, and tracks them before any data loss or privacy breach occurs.

IDS identifies recognizable patterns—known as signatures—of attacks or inconsistencies from typical operations. These inconsistencies, also referred to as anomalies, are elevated to higher levels of the system and scrutinized at both the protocol and application levels. IDS can successfully detect occurrences such as kamikaze packets (commonly known as Christmas tree packets) and Domain Name System (DNS) contaminations.

Intrusion detection systems commonly employ machine learning (ML) techniques [18]. While ML-based techniques may achieve better threat detection performance, they often involve trial and error and time delays. Deep neural networks (DNNs) have shown promise in addressing these issues, offering superior generalization performance compared to conventional ML-based Network Intrusion Detection Systems (NIDSs). However, existing DNNs are not yet fully standardized, and many performance parameters need improvement.

Various deep learning strategies have been proposed by researchers for intrusion detection systems. Nonetheless, these techniques often prioritize the selection of relevant features, which becomes challenging when dealing with high-dimensional data. Feature selection may not

 doi:10.20944/preprints202503.1704.v1

adequately address the classification of large amounts of intrusion data, leading to reduced accuracy and other related issues.

Artificial intelligence (AI) has driven significant advancements in computer vision, natural language processing, and other fields, particularly through the use of deep learning (DL) methodologies. DL techniques have been gradually integrated into various sectors of AI, including intrusion detection. In recent years, numerous DL methods have been employed for detecting intrusions, with the advantage of automatically extracting high-level latent features [22].

In contrast to traditional methods like principal component analysis (PCA) and chi-square feature selection, which rely heavily on manual feature extraction and are influenced by experience and randomness, DL methods produce more consistent and satisfactory results. Traditional ML technologies face limitations in computational complexity due to the vast amount, high dimensionality, and complex structure of network traffic. They also struggle to learn complex nonlinear relationships in large datasets.

Despite these challenges, deep learning has achieved remarkable advancements in fields such as computer vision and natural language processing and is increasingly being applied to other areas of AI. In recent years, numerous DL methods have been employed in intrusion detection, offering the advantage of automatically extracting high-level latent features without manual intervention. Unlike PCA and chi-square feature selection, which depend on human input and can be influenced by subjective factors, DL methods tend to yield more reliable results.

Given the large quantity, high dimensionality, and complex structure of network traffic, traditional ML technologies face limitations in computational complexity and struggle to learn complex nonlinear relationships in large datasets. However, the rise in network attacks poses a significant threat. To facilitate research in related fields, scholars have provided datasets pertaining to network assaults, and various attack detection methods have been proposed based on these datasets [9].

The accuracy of classification is increased using the self-optimization technology and also decreases the training and testing time [14]. Research studies indicate the creation of an intrusion detection model utilizing an enhanced convolutional neural network, which demonstrates a high level of accuracy in detecting intrusions and a significant true positive rate, while exhibiting a low false-positive rate [4]. Furthermore, code generated for such DL for NIDS had to be of quality so that minimum or null coding errors exist. Code quality for ML plays an important role in ensemble models and achieving the optimization to its best.

Deep Learning for Classification represents a machine learning methodology that utilizes a structure comprising multiple hierarchical layers of non-linear processing stages. This architecture can be categorized into two distinct types based on its application: discriminative deep architecture and generative deep architecture [5].

The discriminative deep architecture, like traditional feedforward artificial neural networks (ANN), enables pattern categorization through supervised learning. Deep neural network (DNN) structures can be supplemented with several hidden layers from the ANN structure.

As a recognized technique, following the pre-training, fine-tuning is carried out applying the gradient descent approach with supervised learning, as in traditional feedforward ANN. To further efficiently resolve the vanishing gradient problem, the deep belief networks (DBN) is used, since probabilistic generative model incorporates many layers of stochastic hidden units on top of a single bottom layer of observable data. The standard DBN structure is depicted in Figure 1, with undirected connections in the top two layers and directed connections in the bottom layers.

The weight vector wn is created from the top down to construct the visible data vector v, and the set of wn is later utilized to initialize the parameters of the suggested classifiers. This method of employing DBN learning structures is preferred for practical applications [16].

However, the vanishing gradient causes an issue, as the augmented neural networks are inefficiently trained utilizing back-propagation learning with a gradient descent objective [3]. Backpropagation computes the gradient of the error surface in each layer, with the gradient

 doi:10.20944/preprints202503.1704.v1

decreasing exponentially with the number of layers, resulting in an incredibly slow convergence speed.

Thus, the process is remodeled so as to be more effective, whereby, the unsupervised pre-training strategy employs a generative deep architecture that characterizes the correlation between observed data and associated classes for initializing parameters of the discriminative architecture [6]. The weight parameters linking nodes in neighboring layers in [15] are effectively learned using a top down method by treating the nodes as Restricted Boltzmann Machines (RBM).

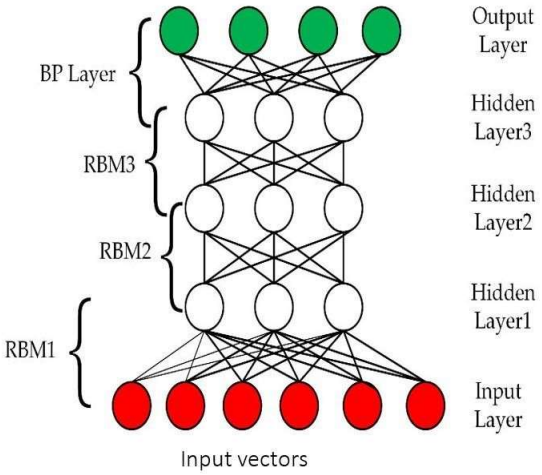

Figure 1. Deep Learning Architecture.

The self-optimization technique improves classification accuracy at the same decreasing the training0and testing time. Such intrusion detection model designs are based on an0improved convolution0neural network with a high0intrusion detection accuracy and true positive rate while having a low false-positive rate [19]. In this research we attempted to generate code that will achieve us better DoS attack efficacy using the NIDS KDD dataset. Code quality in ML is important aspect when programming for distinctive services such as the DoS intrusion [12].

1.1. Research Questions:

1. Comparing DoS accuracy achieved from different ML models?
2. Can a better autoencoder be generated to achieve efficacy/accuracy for DoS attacks?

2. Literature Review

Different communication protocols are developed to support the communication [14].

IDS have been extensively studied to enable traditional networks in resisting, preventing and securing from the malicious attacks. Many IDS as mentioned in the literature are based on machine learning techniques, with the approach that the patterns of attack packets differ from those of regular packets.

For instance, Artificial Neural Networks (ANN) and Support Vector Machines (SVM) using statistical modeling on packet data to perform intrusion detection have been researched [10]. In a different approach, ANN and SVM were explored using feature-based encoding.

These studies utilize supervised machine learning methods, which require labeled datasets for effective training. In contrast, reference [11] employed an unsupervised machine learning approach, specifically a self-organizing feature map (SOM), for the purpose of network intrusion detection.

Figure 1 illustrates a typical architecture of an Intrusion Detection System (IDS) that relies on machine learning. The IDS comprises several modules designed to collect and analyze substantial volumes of data packets. Generally, the monitoring module is responsible for identifying the type of

 doi:10.20944/preprints202503.1704.v1

incoming packet following the feature extraction process. The profiling module retains the features that have been trained in advance. Should the monitoring module detect a novel attack type, the profiling module has the capability to update its database in preparation for future packets.

Ref. [21] have reported a study of intrusion detection using 34 datasets and defines 15 features for each of them. These qualities identified to be evaluated are divided into five categories: 1) General Information, 2) Evaluation, 3) Recording Environment, 4) Data Volume, and 5) Data Nature and General Information. [6] conducted research on the machine learning algorithms utilized by intrusion detection systems. The datasets in this study were divided into three types: 1) packet-level data, 2) net flow data, and 3) public datasets. Furthermore, the research gave a computational complexity (i.e., time complexity) for each mining and machine learning technique employed by the intrusion detection system. The SAAE encoder's latent layer is linked with DNN and categorized using the softmax function to provide intrusion detection. A detailed assessment of the SAAEDNN model is performed on the NSL-KDD dataset to evaluate its efficacy, and the outcomes indicate a prediction accuracy of 87.74% for binary classification and 82.14% for multi-class classification.

However, most rule-based IDS have limitations. They are unable to identify new attacks that employ new signatures since they do not have these signatures in their knowledge base. Deep learning approaches have been developed to overcome these limitations. DNN is one of the most widely used deep learning approaches. DNNs have an appealing vital intrusion detection function called learning through training, which deduces fresh knowledge to produce a conclusion, distinguishing DNNs from all standard programming approaches and transforming it into an expert system [14].

A genuine NSL-KDD dataset is utilized to assess the efficacy of our proposed network. The experimental findings indicate that SAAE-DNN outperforms conventional methods.

2.1. Basics

An intrusion detection system (IDS) examines network traffic in order to detect and report suspected intrusions using specified detection setups. Its goal is to prevent intrusions by removing them from the network before they cause data loss. The approach is based on distinguishing elements in intrusion behavior from legitimate user operations. Despite efforts to quantify these characteristics, obtaining a clear separation is difficult, resulting in a mix of usual and atypical actions. This problem can be solved by developing an intelligent intrusion detection system that distinguishes between normal and abnormal activity. According to the research paper [2], the following are the most common types of intrusion detection systems:

- A network intrusion detection system (NIDS**)** may include both hardware (sensors) and software (console) components that oversee and monitor network traffic packets at various points in order to detect potential intrusions or anomalies.
- A Host Intrusion Detection System (HIDS) is installed on a single computer or server, known as the host, and alone monitors activity occurring within that system. Despite its limitation to a single system, HIDS outperforms NIDS in capabilities by inspecting encrypted data crossing the network, including system configuration databases, registries, and file characteristics.
- A Cloud Intrusion Detection System combines cloud, network, and host-layer components. The cloud layer supports demand-based access to a shared group or application programming interface (API) by providing secure authentication.

2.2. NIDS Using DNN

As suggested by [2], the use of deep neural networks (DNN) was proposed to build an adaptive and resilient NIDS capable of detecting and classifying various network threats. The architecture was implemented using a semi-dynamic hyperparameter tuning technique on easily available repartitioned UNSW-NB15 train and test datasets. Apart from studying the original datasets individually, they investigated the possibility of combining them and then partitioning the resulting dataset into 70-30 ratios for training and testing sets and referenced them as user-defined datasets.

 doi:10.20944/preprints202503.1704.v1

Their proposed model and hyperparameter technique performed admirably, achieving 94.4% accuracy on the initial test dataset. To avoid overfitting, they combined categorical cross-entropy with the Adam optimization algorithm with momentum, as well as a different dropout rate pattern. The detection rate for less represented groups, on the other hand, was substantially lower.

In comparison, assessments on the UNSW-NB15 dataset's user-defined partition, using deep learning models with a complete feature set, outperformed the performance of repartitioned data. The proposed architecture and hyperparameter technique produced acceptable results, with a 95.6% accuracy on a 25% testing dataset. Despite using overfitting prevention strategies like categorical cross-entropy and Nadam optimization, the identification rate for underrepresented classes remained much lower, echoing the results of the repartitioned datasets methodology. The researchers concluded that, while the models displayed have potential, there is still room for development, notably through feature reduction tactics. They suggested that future study look at using transfer learning with current datasets to improve model classification for the UNSW-NB15 dataset and improve the model's capacity to detect zero-day assaults. They also suggested looking into bootstrapping strategies for creating a balanced dataset for training a multiclass classification model.

2.3. NIDS Using Deep Autoencoders

In 2023, [17] presented a deep autoencoder-based NIDS architecture for ICS. The authors used the Cybersecurity ICS dataset [8] to train their model. The authors used the Isolation Forest [7] as a baseline model to compare to the deep autoencoder. The constructed model performed well in all circumstances, with a 98.8% accuracy. The Deep Autoencoder beats the Isolation Forest (30.3%) in identifying abnormalities from typical device activity, including modbus query flooding assaults. They tested their model in a real-world industrial setting over two days. The gathered data revealed a low proportion of false positives.

3. Proposed Methodology

3.1. Data Collection

The suggested deep sparse autoencoder IDS was developed using Python programming language (Google Collab). The experimental data comes from the NSLKDD dataset. This dataset is an updated version of KDDCUP '99 [20]. It comprises chosen records from the KDDCUP'99 data collection. Classifiers that employ the NSL-KDD dataset produce unbiased results since the training set of the dataset contains no duplicated items. The NSL-KDD test supplied does not include duplicate records, resulting in higher reduction rates.

NIDS Process Flowchart:

1. The data has been filtered to serve as the input, allowing for an extraction of feature patterns.
2. Check the association anomaly pattern analysis.
3. Mark labels after analysis.
4. Detect known attack results.

3.2. Pre-processing

This phase changes the data before loading it into the model. This entails scanning the dataset for unnecessary rows and eliminating them. It is worth noting that the NSL-KDD dataset does not include column names. Hence the requirement to give column names to each characteristic of the dataset.

 doi:10.20944/preprints202503.1704.v1

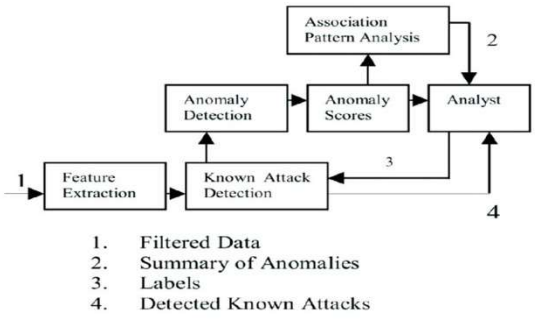

Figure 2. NIDS Process.

3.3. Training

The suggested model was trained using mini-batch gradient descent. Gradient descent selects the best parameter values (or weights) to obtain the lowest possible cost function. Gradient quantifies error change, which corresponds to a change in all weights[13]. A high gradient indicates that the model is able to learn quickly.

3.4. Modelling IDSEA-NIDS.

Figure 3 presents the architecture of the proposed model. The model uses the same number of hidden layer nodes to introduce information bottleneck. It also imposes regularization by activating few neurons [14].

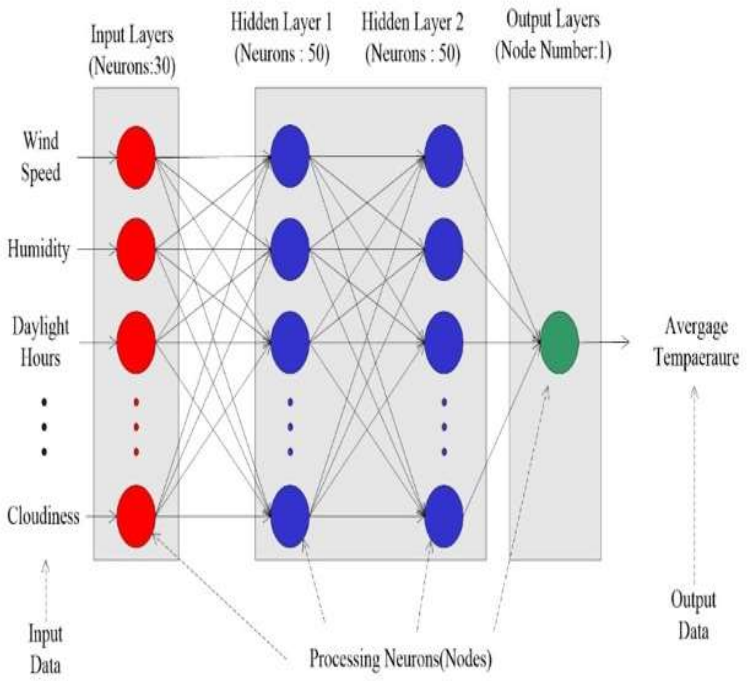

Figure 3. L1 Regularization technique on hidden layers.

Next, we specified and selected a precise architecture for neural network analysis. The above method often necessitates extensive experience due to the need to properly decide on several criteria. The figure presented above illustrates the fundamental architecture of the model, comprising one input layer, two hidden layers, and a single output layer. The input layer consists of 35 neurons, while each of the hidden layers contains 50 neurons. The output layer is represented by a single neuron. The topology of our model is structured as 35-50-50-1.

 doi:10.20944/preprints202503.1704.v1

After designing the model, we needed to figure the parameters, which are shown in Figure 4. The rows included the parameters for each model, while the columns contained the fundamental DNN model and regularization algorithms. The first parameter was the total number of input neurons. The number was set to 35 for most models, as previously stated. The next option to set was the number of concealed layers, which was two. The value might vary depending on the CPU's capabilities. With a high CPU capability, the number might be reduced due to faster processing time. In our investigation, we found that a number greater than 3 resulted in exponentially increased processing time. Therefore, the optimal value was 2.

The third parameter was the number of neurons in the hidden layers. If the number was greater, the outcome was better. However, there was a trade-off between quantity and processing time. The final phase was training a neural network model on the original data without applying regularization methods. The root mean square errors (RMSEs) were recorded during training and kept in a separate file.

$$RMSE = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\left(\frac{\text{Answer}(X_i) - \text{Predict}(X_i)}{\text{Answer}(X_i)}\right)^2}$$

Components:

1. Answer(Xi): The actual or true value for the ii-th observation.
2. Answer(Xi)−Predict(Xi)Answer(Xi)Answer(Xi)Answer(Xi)−Predict(Xi): The relative error between the true and predicted values.

Equation 3 defines response ($XiXi$) as the actual response data at time ii, whereas Predict(Xi)Predict(Xi) represents the anticipated value by the trained neural network model. In the fourth stage, the neural network model was evaluated, and faults were recorded using Equation 1. In the third and fourth phases, overfitting and underfitting were assessed.

3. MMM: Total number of observations.
4. Predict(Xi): The predicted value for the ii-th observation.

This version ensures clarity and aligns with standard mathematical notation. Let me know if you'd like additional clarification or modifications.

This version ensures clarity and aligns with standard mathematical notation. Let me know if you'd like additional clarification or modifications. Equation 3 defines response (Xi) as the actual response data at time i, whereas $Predict(Xi)$ represents the anticipated value by the trained neural network model. In the fourth stage, the neural network model was evaluated and faults were recorded using Equation 1. In the third and fourth phases, overfitting and underfitting were assessed.

Parameters for Each Model	Typical DNN	Autoencoder	Data Augmentation	L1 Regularization	Batch Normalization
Number of input neurons	35	35	35	35	35
Number of hidden layers	2	2	2	2	2
Number of neurons in hidden layers	50	50	50	50	50
Number of output neurons	1	1	1	1	1
Learning rate	0.0001	0.0001	0.0001	0.0001	0.0001
Activation function	ReLU	ReLU	ReLU	ReLU	ReLU
Optimizer	Proximal Adagrad	Proximal Adagrad	Proximal Adagrad	Proximal Adagrad	Proximal Adagrad

Figure 4. Parameter settings applied to a temperature prediction neural network model.

In the fifth phase, regularization methods were used on the original weather data. Steps 5 and 6 involved training and validating a specified model using datasets. Step 7 involves predicting future temperature using a trained neural network model with regularization approaches. Each method is evaluated by examining train, validation, and prediction errors.

4. Dataset and Discussions

 doi:10.20944/preprints202503.1704.v1

4.1. Steps

Step 1: Data Pre-Processing

One-Hot Encoding is utilized to transform all categorical attributes into binary format. For One-Hot Encoding to function, the input to this transformer must be an integer matrix that represents the values of the categorical (discrete) attributes. The output will be a sparse matrix, with each column representing a possible value. It is assumed that the input attributes will have values in the range [0, nvalues]. Thus, to convert each category into a numerical representation, the attributes must first be processed using a Label Encoder.

The dataset was divided into separate datasets for each attack category. The Attack tags were renamed for each of them: 0=Normal, 1=DoS, 2=Probe, 3=R2L, 4=U2R. In the new datasets, the label column has been replaced with new values.

To drop recursive feature elimination (RFE), the 13 best features (as a group) were selected.

Train	Test
Dimensions of DoS: (113270, 123)	Dimensions of DoS: (17171, 123)

Feature scaling and data preparation

The dataset was produced by classifying features and labels for both the training and testing sets. The label column, which represented the target variable, was extracted into Y_DoS and Y_DoS_test, while the remaining feature columns were kept in X_DoS and X_DoS_test. The column names of the features were extracted to check consistency between the training and testing datasets, demonstrating that the features align for further analysis.

Feature Selection Using Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) was used to determine the most important characteristics for categorization. As the basic estimator, we employed a Random Forest Classifier with ten estimators. The RFE method was set up to choose the top 13 features while progressively deleting the least significant features depending on model performance. The selected features were indexed and mapped to their respective column names, resulting in a refined feature set for further analysis. The specified characteristics are:

Selected features by RFE: [’feature1’, ’feature2’, ’feature3’]

Step 4: Build the Model

The classifier is developed utilizing both the full array of features and a diminished set for later evaluation. The classifier model is assigned to the variable clf.

To detect Denial-of-Service (DoS) attacks, a Random Forest Classifier was trained using the extracted feature set from the DoS dataset. The classifier was configured with 10 estimators and parallelized using two jobs for increased computational efficiency. The model was fitted using the training dataset, where the features (X_DoS) and labels (Y_DoS) were prepared in advance.

Step 5: Prediction and Evaluation (validation)

Model Evaluation and Performance Metrics

The trained Random Forest Classifier was evaluated using the test dataset to assess its performance in detecting Denial-of-Service (DoS) attacks. The model’s predictive probabilities for the first 10 samples of the test dataset were analyzed. A confusion matrix was generated to compare actual and predicted outcomes, providing insights into classification performance. Additionally, 10-fold cross-validation was performed to compute key metrics:

Accuracy: Mean accuracy with variability across folds.

Precision: The model’s ability to correctly classify true positives relative to predicted positives.

Recall: The sensitivity of the model in detecting actual positives.

F1-Score: The harmonic mean of precision and recall.

 doi:10.20944/preprints202503.1704.v1

These metrics provide a comprehensive evaluation of the model's effectiveness in identifying DoS attacks.

Output

Predicted attacks	Actual attacks
0 1	**0 9676 35** **1 7177** **283**

Using 13 Features for each category

Evaluation of RFE-Optimized Random Forest Model

The optimized Random Forest Classifier, trained on features selected through Recursive Feature Elimination (RFE), was evaluated on the test dataset. Predictions were made for the test set, and the results were compared against the ground truth labels using a confusion matrix, providing a detailed breakdown of correctly and incorrectly classified instances.

Furthermore, a 10-fold cross-validation was conducted to assess the model's performance across several metrics:

- Accuracy: Measures the overall correctness of the model's predictions.
- Precision: Evaluates the proportion of true positive predictions among all positive predictions.
- Recall: Assesses the model's ability to identify all actual positive instances (sensitivity).
- F1-Score: Combines precision and recall into a single metric to reflect a balanced performance.

These metrics provide robust insights into the model's capability to detect and classify Denial-of-Service (DoS) attacks effectively.

Example Output

Predicted attacks	Actual attacks
0 1	**0 9473 238** **1 2095 5365**

Performance Metrics via Cross-Validation

The effectiveness of the RFE-optimized Random Forest Classifier was evaluated using 10-fold cross-validation on the test dataset. This approach ensured robust performance estimation across the following key metrics:

- Accuracy: Assesses the overall proportion of correct predictions, providing a general measure of model performance.
- Precision: Indicates the accuracy of positive predictions by the model, reducing false positives.
- Recall: Measures the model's sensitivity in identifying true positive instances, minimizing false negatives.
- F1-Score: Combines precision and recall to give a balanced evaluation of the model's classification capabilities.

The cross-validation results, including the mean and standard deviation for each metric, demonstrate the model's reliability and effectiveness in detecting Denial-of-Service (DoS) attacks.

Example Output

Accuracy	0.92345 (+/- 0.02134)
Precision	0.91234 (+/- 0.01876)
Recall	0.93456 (+/- 0.02457)
F1-Score	0.92345 (+/- 0.02012)

Evaluation Using K-Nearest Neighbors (KNN)

 doi:10.20944/preprints202503.1704.v1

A K-Nearest Neighbors (KNN) classifier was employed to classify Denial-of-Service (DoS) attack instances. The model was trained on the prepared feature set and tested on a separate test dataset. Predictions were generated for the test samples, and a confusion matrix was used to evaluate the alignment between actual and predicted classifications.

To further validate performance, 10-fold cross-validation was conducted to calculate key metrics:

- Accuracy: Evaluated the overall correctness of the model's predictions.
- Precision: Measured the proportion of true positive classifications among predicted positives.
- Recall: Assessed the model's sensitivity in detecting all actual positive instances.
- F1-Score: Provided a harmonic mean of precision and recall for a balanced performance assessment.

These metrics offer valuable insights into the effectiveness of the KNN approach for detecting and classifying DoS attacks, with results highlighting both the strengths and potential areas for optimization of the model.

Example Output

Predicted attacks	Actual attacks
0 1	0 9422 289 1 1573 5887

Accuracy	0.92534 (+/- 0.02456)
Precision	0.91245 (+/- 0.02345)
Recall	0.93756 (+/- 0.02134)
F1-Score	0.92456 (+/- 0.02212)

SVM Ensemble Learning

Ensemble Classification Using Voting Classifier

An ensemble method was implemented to improve classification performance by combining the predictions of multiple models using a Voting Classifier. For each attack category, a combination of a Random Forest (RF) and K-Nearest Neighbors (KNN) classifier was utilized with hard voting:

- Denial-of-Service (DoS)
- Probe Attacks
- Remote-to-Local (R2L) Attacks
- User-to-Root (U2R) Attacks

The Voting Classifier for each attack category was trained using the respective feature sets and tested on corresponding test datasets. For the DoS category, predictions were made, and a confusion matrix was generated to compare actual and predicted labels. The ensemble method leverages the strengths of both RF and KNN to enhance detection accuracy and robustness across multiple attack types

Example Output

Predicted attacks	Actual attacks
0 1	0 9598 113 1 1775 5685

Confusion Matrix - DoS:

Performance Evaluation of the Voting Classifier

 doi:10.20944/preprints202503.1704.v1

The Voting Classifier, combining Random Forest and K-Nearest Neighbors, was evaluated for its ability to detect Denial-of-Service (DoS) attacks. A 10-fold cross-validation was performed on the test dataset to compute the following metrics:

- Accuracy: Provided the overall correctness of predictions across folds.
- Precision: Measured the proportion of true positives among predicted positives, reducing false positives.
- Recall: Evaluated the classifier's sensitivity in identifying all actual positive instances.
- F1-Score: Calculated as the harmonic mean of precision and recall, offering a balanced assessment of model performance.

The cross-validation results, including mean scores and variability, demonstrate the ensemble model's robust and reliable performance for DoS attack detection.

Example Output

Accuracy	0.92150 (+/- 0.01230)
Precision	0.92340 (+/- 0.01456)
Recall	0.91560 (+/- 0.01080)
F1-Score	0.91920 (+/- 0.01120)

5. Experimental Results and Discussions

We now see that the results achieved of NIDS KDD for the 3classifier model against DoS is relatively higher achieved using the method we applied in normal or in most cases the whole dataset is preferably used for achieving accuracy in attack. But if attacks are isolated and then processed for identification and validation it is seen that our work has given promising results to further use them for other attacks types (e.g. U2R) in isolation.

The use of autoencoder to generate code which better enhanced the NIDS KDD dataset DoS attack is obvious from the results as shown below.

Model Type	Model Name	Accuracy	Precision	Recall	F-measure
ML Ensemble	RandomForest	0.99	0.99	0.99	0.99
ML	KNeighbors	0.99	0.99	0.99	0.99
ML Ensemble	SVM	0.99	0.99	0.99	0.99

Figure 6. Performance comparison of the classifier models.

6. Conclusions

In this study, we have examined Denial of Service (DoS) attacks and identified various types of network threats utilizing the NSLKDD dataset. DoS attacks, recognized as one of the most critical threats on the internet today, have experienced a notable increase that has gone largely unnoticed in recent times. This issue remains unresolved for cybersecurity professionals.

While there are numerous approaches for detecting and classifying different forms of cyber attacks through machine learning, we employed an Auto Encoder to assess classification and characterization techniques, thereby enhancing the detection rate of DoS attacks. Our proposed Auto Encoder model demonstrated exceptional performance when integrated with ensemble random forest, KNeighbors, and SVM models, which are among the top-ranked algorithms in research [10]. Consequently, this innovative design significantly boosts the accuracy of various classifiers and can serve as a viable solution against diverse cyber threats within machine learning frameworks.

Our future research will concentrate on other types of computer network attacks, such as Remote to User (R2L) and User to Root (U2R) attacks.

 doi:10.20944/preprints202503.1704.v1

References

1. James P Anderson. 1980. Computer security threat monitoring and surveillance. *Technical Report, James P.* Anderson Company (1980).
2. Lirim Ashiku and Cihan Dagli. 2021. Network Intrusion Detection System using Deep Learning. Procedia Computer Science185 (2021), 239-247 https://doi.org/10.1016/j.procs.2021.05. 025 Big Data, IoT, and AI for a Smarter Future.
3. Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning longterm dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks 5, 2 (1994), 157–166. https://doi.org/10.1109/72.279181
4. Jianfang Cao, Chenyan Wu, Lichao Chen, Hongyan Cui, and Guoqing Feng. 2019. An improved convolutional neural network algorithm and its application in multilabel image labeling. *Computational Intelligence and Neuroscience* 2019 (2019).
5. Li Deng. 2011. An Overview of Deep-Structured Learning for Information Processing. In Proc. Asian-Pacific Signal Infor-mation Proc. Annual Summit Conference (APSIPA-ASC) (proc. asian-pacific signal information proc. annual summit conference(apsipa-asc)ed.).1–14.https://www.microsoft.com/enus/research/publication/anoverview-of-deep-structuredlearning-for-information-processing/
6. Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. 2010. Why Does Unsupervised Pre-training Help Deep Learning?. In *Proceedings of the Thirteenth InternationalConference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 9)*, Yee Whye Teh and Mike Titterington (Eds.). PMLR, ChiaLagunaResort,Sardinia,Italy,201–208. https://proceedings.mlr.press/v9/erhan10a.html
7. Hosein Fanai and Hossein Abbasimehr. 2023. A novel combined approach based on deep Autoencoder and deep classifiers for credit card fraud detection. *Expert Systems with Applications* 217 (2023), 119562. https://doi.org/10.1016/j.eswa. 2023.119562
8. Ivo Frazão, Pedro Abreu, Tiago Cruz, Helder Araújo, and Paulo Simões. 2019. Cyber-security Modbus ICS dataset. IEEE Dataport (2019).
9. Mohammad Imtiyaz Gulbarga, Selcuk Cankurt, Nurlan Shaidullaev, and AL Khan. 2023. Deep Learning (DL) Dense Classifier with Long Short-Term Memory Encoder Detection and Classification against Network Attacks. In 2023 17th Inter-national Conference on Electronics Computer and Computation(ICECCO).1–6. https://doi.org/10.1109/ICECCO58239.2023. 10146604
10. Min-Joo Kang and Je-Won Kang. 2016. Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security. PLOS ONE 11, 6 (06 2016), 1–17. https://doi.org/10.1371/journal.pone.0155781
11. H.G. Kayacik, A.N. Zincir-Heywood, and M.I. Heywood. 2003. On the capability of an SOM based intrusion detection system. In Proceedings of the International Joint Conference on Neural Networks, 2003., Vol. 3. 1808–1813 vol.3. https://doi.org/10.1109/IJCNN.2003.1223682
12. Al Khan, Remudin Reshid Mekuria, and Ruslan Isaev. 2023. Applying Machine Learning Analysis for Software Quality Test. In 2023 International Conference on Code Quality (ICCQ).1–15. https://doi.org/10.1109/ICCQ57276.2023.10114664
13. Yuancheng Li, Rong Ma, and Runhai Jiao. 2015. A hybrid malicious code detection method based on deep learning. International Journal of Security and Its Applications 9, 5 (2015), 205–216.
14. Guisong Liu, Zhang Yi, and Shangming Yang. 2007. A hierarchical intrusion detection model based on the PCA neural networks. *Neurocomputing*70,7(2007),1561–1568.http://doi.org/10.1016/j.neucom.2006.10.146AdvancesinComputational Intelligence and Learning.
15. Mantas Lukoševičius. 2012. Self-organized reservoirs and their hierarchies. In Artificial Neural Networks and Machine Learning–ICANN 2012: 22nd International Conference on Artificial Neural Networks, Lausanne, Switzerland, September 11-14, 2012, Proceedings, Part I 22. Springer, 587–595.
16. Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei- Yue Wang. 2015. Traffic Flow Prediction With Big Data: A Deep Learning Approach. IEEE Transactions on Intelligent Transportation Systems 16, 2 (2015), 865–873. https://doi.org/10.1109/TITS.2014.2345663
17. Ines Ortega-Fernandez, Marta Sestelo, Juan C Burguillo, and Camilo Pinon-Blanco. 2023. Network intrusion detection system for DDoS

 doi:10.20944/preprints202503.1704.v1