Applied Computer Science 2025, vol. 21, no. 1, pp. 111–125
https://doi.org/10.35784/acs_6667

Submitted: 2024-10-06 | Revised: 2025-01-06 | Accepted: 2025-01-20

Keywords: network intrusion detection, deep learning, CNN, RNN, DBN, autoencoder, NSL-KDD, cybersecurity

Yakub HOSSAIN [1], *Zannatul FERDOUS* [1], *Tanzillah WAHID* [1], *Md. Torikur RAHMAN* [1*], *Uttam Kumar DEY* [1], *Mohammad Amanul ISLAM* [1]

[1] Uttara University, Bangladesh, 2202081012@uttarauniversity.edu.bd, 2201081021@uttarauniversity.edu.bd, tanzillah@uttarauniversity.edu.bd, torikurrahman@gmail.com, uttam@uttarauniversity.edu.bd, amanul.islam@uttarauniversity.edu.bd

[*] Corresponding author: torikurrahman@gmail.com

Enhancing intrusion detection systems: Innovative deep learning approaches using CNN, RNN, DBN and autoencoders for robust network security

Abstract

The increasing sophistication of cyber threats poses significant challenges to network security. This makes effective intrusion detection system (IDS) more important than ever before. Conventional IDS methods, which often rely on signatures or rules it will struggle to keep up with its complex attacks and evolution. This thesis evaluates and analyze the performance of DL algorithms. They include convolutional neural networks (CNN), recurrent neural networks (RNN), deep belief networks (DBN), and Auto-encoder. Using the models, these models are trained and tested only on the NSL-set. KDD data, which is a widely accepted benchmark for evaluating IDS performance. Results show that the proposed deep learning approach significantly outperforms traditional methods, has a higher detection rate, reduce the false positive rate and the ability to identify both known and unknown intrusions. They leverage the strengths of CNN, RNN, DBN, and autoencoders. Doing this research Advances IDS capabilities by providing a robust and adaptable solution to enhance network security.

1. INTRODUCTION

As computer networks expand at an unprecedented rate in the age of technology, companies are achieving levels of connectivity and efficiency previously unheard of. However, there are serious risk factors associated with this connectivity. Cyber-attacks are becoming more frequent and pose serious risks to people, companies, and governments. It might appear in a variety of ways. To prevent complex breaches and protect data integrity, it is crucial to ensure the security of network infrastructure. Network site visitors is scanned by way of an intrusion detection machine (IDS) to look for unusual or suspicious interest which can factor to a safety breach. One drawback of conventional intrusion detection systems (IDS), such signature-based and rule-based approaches, is their inability to identify novel and changing threats. These methods can identify known signatures, but they cannot identify new, unknown threats since they mostly rely on pre-existing patterns or rules to identify intrusions. To prevent complex security threats and data breaches, ensuring the integrity and safety of network infrastructure is crucial. Network traffic is analyzed by an Intrusion Detection System (IDS) to identify unusual or suspicious activities that may indicate a security breach.

Artificial Intelligence (AI) applications, including deep learning, machine learning, image and audio recognition, cybersecurity, and natural language processing, have demonstrated remarkable results. Since deep studying techniques appoint multi-layer neural networks to assemble tricky input representations, they may be perfectly suited for the tough venture of intrusion detection. This thesis examines the application of deep learning techniques, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), autoencoders, and deep belief networks (DBNs), for network intrusion detection. These models are evaluated using the NSL-KDD benchmark dataset, which is widely utilized for assessing IDS performance.

The primary objective of this research is to develop a reliable and adaptable intrusion detection system capable of accurately identifying both known and unknown cyberattacks while minimizing false alarms. By evaluating the effectiveness of various deep learning models to determine the most successful approach for

network intrusion detection, this study provides valuable insights into how deep learning can enhance network security. The findings of this study will contribute to the advancement of intrusion detection systems by offering more robust protection against the continuously evolving landscape of cyber threats.

2. RELATED WORKS

A system has been developed that uses a deep neural network to identify the most potent fraud behaviors, namely in the identification of credit card fraud, according to Fu et al. (2016). Zhang et al. (2018) trained and tested their approach using a neural network and B2C online transaction data from financial institutions. The training and test sets each contained one month's worth of data. The study found that the accuracy rate was 91% and the recall rate was 94%. These results demonstrate a 26% and 2% improvement, respectively, over the earlier work when compared to Fu et al. (2016) research.

Nasr et al. (2018) developed DeepCorr, a convolutional neural network-based intrusion detection system. The technology creates a correlation function that is subsequently applied to detect attempted intrusions. Two layers of convolutional neural networks, one layer of fully connected neural networks, and three layers of fully connected neural networks make up DeepCorr. DeepCorr demonstrated a true positive rate of approximately 80, a learning rate of 0.0001, and a false positive rate of 103 in investigations.

Kültür (2022) explored network intrusion detection using deep learning techniques in her master's thesis at Middle East Technical University. The study aimed to enhance detection accuracy and effectiveness by applying deep learning models, outperforming traditional methods.

Edeh et al. (2021) developed a network intrusion detection system leveraging deep learning techniques. Their study, conducted at the University of Turku, explored the effectiveness of deep learning models in identifying cyber threats, emphasizing improved accuracy and adaptability in intrusion detection.

Shiri et al. (2024) analyzed deep learning models such as CNN, RNN, LSTM, and GRU, comparing their architectures and functionalities. Their study highlighted the strengths and limitations of each model, offering insights into their effectiveness in different applications.

For automatic categorization learning, Yin et al. (2017) enhanced a recurrent neural network with a convolutional neural network. Using the NSL-KDD dataset, the study computed three performance metrics: accuracy, time to first failure, and first failure rate. The authors claim that when 80 hidden nodes are combined with a learning rate of 0.1, anomaly detection performs better. This essay also goes into great detail on the advantages of using a recurrent neural network as an intrusion detection system (ids). A corrected linear unit run has been proposed by Tang et al. (2016) as a method for application networking intrusion detection. Other scientists agreed. The study found that with a limited set of criteria, it is possible to get an 89% recognition rate. The NSL-KDD dataset and the four-evaluation metrics -recall, F-measurement, precision, and accuracy - must be used to assess the network's efficiency. In their study, Jiang et al. suggest a multichannel intrusion detection system based on LSTM (Long Short Term Memory). Using the NSLKDD dataset, the effectiveness of the suggested detection and prevention method is assessed. The accuracy of the long-short-term memory recurrent neural net is 99.94%, the false alarm rate is 9.86%, and the detection rate is 98.94%. Its false alarm rate is 99.23%, while its detection rate is 98.94%.

Shone et al. (2018) proposed a deep learning-based network intrusion detection system using stacked autoencoders and random forests. Their approach improved feature extraction and classification, enhancing detection accuracy while reducing reliance on labeled data.

An intrusion detection system based on the KDD Cup 1999 dataset is presented by Kim et al. (2016). It uses a recurrent neural network with a large short-term memory architecture. The KDD Cup 1999 data set is used to validate the method. There were four attacks and one non-attack in the experiment's output vector, which had 41 characteristics as input. The epoch lasted 500 milliseconds, the batch lasted 50 milliseconds, and the time step lasted 100 milliseconds. The company claims that in all assault scenarios, the attack detection performance is 98.8%.

Ghani et al. (2023) developed a deep learning-based intrusion detection system that uses a compact feature vector to improve efficiency. Their approach reduced computational complexity, ensuring a scalable and effective real-time cybersecurity solution.

Sharafaldin et al. (2019) developed a realistic DDoS attack dataset and taxonomy to enhance the evaluation of intrusion detection systems. Their study focused on creating diverse attack scenarios to improve the accuracy and effectiveness of DDoS detection methods.

The deep auto-encoder was a key tool used by Shone et al., (2018) in their study to find cybersecurity vulnerabilities. To improve classification performance, a non-symmetrically high number of hidden layers is being used instead of deep belief networks. Based on the NSL-KDD and KDD Cup datasets, the study's results quantify performance using KDD Cup 99 accuracy and F-score measurements. This model outperformed the prior study's accuracy, with an average accuracy of 97-85% in the KDD Cup '99 dataset assessment. When tested against the NSL-KDD dataset, the researchers discovered that the proposed model performed 5% better than the deep belief network model, with an accuracy rate of 85.42%. This chapter highlights gaps in existing IDS research. Prior studies, such as Fu et al. (2016) and Y. Zhang et al. (2019), focused on fraud detection, lacking network intrusion relevance. Shone et al., (2018) and Tang et al. (2016) demonstrated CNN potential but struggled with evolving threats. Yin et al. (2017) and Kim et al. (2016) achieved high accuracy using CNN-RNN and LSTM but faced scalability and dataset dependency issues. This study bridges these gaps by combining CNNs, RNNs, Autoencoders, and DBNs to enhance intrusion detection. It addresses evolving threats, reduces false positives, and improves scalability, offering a robust, adaptive IDS solution.

Sama (2022) developed a deep learning-based network intrusion detection system in their doctoral dissertation at Victoria University. The study focused on improving detection accuracy and efficiency by using advanced deep learning models to identify complex network intrusions.

Ahmad et al. (2021) conducted a systematic study on network intrusion detection systems, comparing various machine learning and deep learning techniques. Their research evaluated the strengths, limitations, and effectiveness of these approaches in detecting cyber threats.

CNNs excel at identifying spatial patterns, making them well-suited for analyzing network traffic as structured data that resembles a grid. From basic patterns in lower layers to more complex patterns in higher layers, CNNs effectively extract hierarchical features. By analyzing spatial correlations between network characteristics, CNNs can detect anomalies and indicators of attacks. For instance, they identify spatial dependencies in the data to recognize DoS attacks or probing activities.

RNNs are made to identify relationships and temporal patterns in sequential data. Understanding patterns across time requires their capacity to retain a "memory" of prior inputs. RNNs are skilled at recognising intricate attack scenarios, such as multi-stage invasions or DDoS attacks, by simulating sequential relationships. They analyse network traffic feature sequences in order to identify temporal pattern aberrations.

DBNs are highly effective at identifying subtle security vulnerabilities in data and learning hierarchical representations. They excel in dimensionality reduction and feature extraction, making them valuable for network intrusion detection. In NSL-KDD, DBNs assist in simplifying high-dimensional data so that the system may concentrate on important characteristics. Through the use of layered representations, they are helpful in recognising both known and undiscovered attack types.

By identifying typical data patterns and identifying deviations as possible anomalies, autoencoders specialise in anomaly identification.Autoencoders effectively record typical network traffic patterns and spot anomalous ones. They are useful for identifying abnormalities that indicate intrusions because of their capacity to rebuild input data.

CNNs effectively capture packet topologies and other spatial characteristics in network data. RNNs excel at analyzing sequential data to detect threats that evolve over time. DBNs provide a robust approach to feature learning and classification. Autoencoders primarily aim to identify deviations from learned normal patterns.

3. DEEP LEARNING ALGORITHMS

Deep learning approaches have become more popular as caused by of traditional intrusion detection systems (IDS) becoming less effective against sophisticated digital attacks. Convolutional neural networks (CNNs), recurrent neural networks (RNNs), autoencoders, and deep belief networks (DBNs) are a few examples of models that have shown notable advancements in identifying known and new assaults. RNNs are highly effective at processing sequential data, CNNs excel at recognizing spatial and temporal patterns, and autoencoders detect anomalies by learning typical traffic patterns. By sorting through complex data links, DBNs identify minor security flaws. However, problems like overfitting, the need for large labeled datasets, and processing expenses persist despite these advancements. Present study aims to improve scalability and optimize these models for real-time analysis, confirming deep learning's status as a critical tool for modern network security.

3.1. CNN (Convolutional Neural Network)

Convolutional neural networks, or CNNs, are a specific kind of neural networks that are especially useful for tasks requiring spatial hierarchies and patterns. CNNs are made to analyze grid-like input, such as pictures and sequences. CNNs' capacity to automatically learn from and extract characteristics from raw data has transformed disciplines like computer vision and natural language processing. This section explores CNNs' mathematical underpinnings, operation, and design with a focus on network intrusion detection.

Typically, CNN consists of several types of layers, where each layer serving a distinct purpose feature extraction and transformation.

Convolutional Layers: The convolutional layer is the core component of CNN. The convolutional layer is the core element of CNNs. In order to find local patterns in the input data, this layer applies a collection of filters, also referred to as kernels. Every filter traverses the input, calculating dot products between the filter and certain input areas. The convolution operation can be stated mathematically as follows:

$$\mathrm{Z}_{i,j}^{(l)} = (\mathrm{X} * \mathrm{K})_{i,j} + \mathrm{b} \tag{1}$$

If the bias term is b, the convolutional kernel is K, the input matrix is X, and $\mathrm{Z}_{i,j}^{(l)}$. The result of the convolution operation at point (i, j). A feature map that highlights several facets of the input is the product.

Activation Functions: After the convolution operation, the resulting feature maps are passed through a non-linear activation function. The Rectified Linear Unit (ReLU) is commonly used:

$$ReLU(x) = \max(0, x) \tag{2}$$

ReLU introduces non-linearity to the model, allowing it to learn more complex features.

Pooling Layers: Pooling layers are used to reduce the spatial dimensions of the feature maps, which helps to decrease the computational load and control overfitting. The most common pooling operation is Max Pooling:

$$\mathrm{P}_{i,j}^{(l)} = \max\left(\mathrm{Z}_{i,j}^{(l)}\right) \tag{3}$$

where $\mathrm{P}_{i,j}^{(l)}$ is the pooled output. Pooling layers reduce the resolution of feature maps while retaining the most significant information.

Flattening: After convolution and pooling, the multi-dimensional feature maps are flattened into a one-dimensional vector. This vector is then passed to fully connected layers.

Fully Connected Layers: These layers are standard neural network layers where each neuron is connected to every neuron in the previous layer. They are used to combine the features extracted by the convolutional and pooling layers and make final predictions. The output of a fully connected layer can be expressed as:

$$a = \sigma(Wx + b) \tag{4}$$

where W is the weight matrix, x is the input vector, b is the bias, and σ is the activation function, such as Softmax for classification tasks.

Applying a linear filter to the input and then a non-linear activation is how CNNs perform the convolution operation. The filter weights are intended to capture a variety of properties, including edges, textures, and forms. They are learned during training. The following formula represents this process mathematically:

$$\mathrm{Z}_{i,j}^{(l)} = \mathrm{m}, \mathrm{n} \sum_{m,n} \mathrm{X}_{i+m,j+n}.\mathrm{K}_{m,n} + \mathrm{b} \tag{5}$$

where b is the bias term and K is the kernel applied to the input area X. After that, the activation function is applied to the result to add non-linearity.

Through pooling techniques like Max Pooling, which take the maximum value in each local zone, the dimensionality is reduced:

$$P_{i,j} = \max_{m,1} Z_{i+m,j+n} \tag{6}$$

This reduction in spatial dimensions helps to simplify the feature maps and reduce computation.

In the context of network intrusion detection, CNNs can be applied to analyze and classify network traffic patterns. CNNs are able to automatically learn and extract information like known attack signatures, anomalies, and traffic patterns by considering network traffic as a grid-like series of data. This particular functionality proves to be especially helpful in detecting different kinds of network intrusions, such as DoS assaults, probing activities, and unauthorized access attempts. CNNs are useful for spotting known attack patterns and abnormalities in network traffic because they are excellent at spotting spatial patterns in data. They are able to acquire hierarchical characteristics, in which simpler patterns are identified by lower layers, while more sophisticated attack scenarios are identified by higher layers combining these features. Overall, CNNs are an effective tool for network intrusion detection because of their capacity to automatically identify and identify pertinent aspects from unprocessed network traffic data. This improves the identification and categorization of possible intrusions.

3.2. RNN (Recurrent Neural Network)

Recurrent Neural Networks (RNNs) are a class of neural networks designed to handle sequential data by maintaining a form of memory about previous inputs. This characteristic makes RNNs particularly suitable for tasks involving time-series data, natural language processing, and other sequences where temporal dependencies are crucial. This section provides a comprehensive overview of RNNs, detailing their architecture, mathematical foundations, and specific application in network intrusion detection.

RNNs differ from traditional feedforward neural networks in their ability to maintain state across time steps. The core idea is to introduce feedback loops that allow the network to retain information from previous steps and use it to influence future computations. The primary components of an RNN include:

Recurrent Layer: The fundamental building block of an RNN is the recurrent layer, where the same weights are applied at each time step of the input sequence. At each time step t, the RNN updates its hidden state $h_t$based on the current input x_t and the previous hidden state h_{t-1}. The computation can be expressed as:

$$E(v,h) = -\sum_i b_i v_i - \sum_j c_j v_j - \sum_{i,j} w_{ij} v_i h_j \tag{7}$$

where W_h and U_h are weight matrices, b_h is a bias term, and ϕ is an activation function, typically a non-linear function such as the hyperbolic tangent (tanh) or ReLU.

Output Layer: The output at each time step t is computed from the hidden state h_t using a separate set of weights W_y and bias b_y:

$$y_t = \sigma\left(W_y h_t + b_y\right) \tag{8}$$

where σ is an activation function like Softmax for classification tasks, producing the final output for each time step.

Backpropagation Through Time (BPTT): Training RNNs involve a process known as Backpropagation Through Time, where gradients are computed not just for the current time step but also propagated back through the entire sequence. This allows the network to learn from errors across all time steps.

Mathematical Foundations.

The key mathematical operation in an RNN is the recurrent update of the hidden state. For each time step t, the hidden state h_t is updated based on the previous hidden state h_{t-1} and the current input x_t. The recurrent relationship is given by:

$$h_t = \phi(W_h x_t + U_h h_{t-1} + b_h) \tag{9}$$

where W_h and U_h are weight matrices for the input and recurrent connections, respectively. The output y_t is computed from the hidden state:

$$y_t = \sigma\left(W_y h_t + b_y\right) \tag{10}$$

Training involves optimizing these weight matrices to minimize the error between the predicted outputs and the true labels. The loss function used can be cross-entropy for classification tasks or mean squared error for regression tasks.

Variants of RNNs.

While basic RNNs are effective, they suffer from limitations such as difficulty in learning long-term dependencies due to issues like vanishing and exploding gradients. To address these challenges, several advanced variants of RNNs have been developed:

Long Short-Term Memory (LSTM): LSTM networks introduce memory cells and gating mechanisms to better capture long-term dependencies and mitigate the vanishing gradient problem. The LSTM architecture includes input, forget, and output gates to control the flow of information:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{11}$$

$$f_t = \sigma\big(W_f x_t + U_f h_{t-1} + b_f\big) \tag{12}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{13}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_C h_{t-1} + b_C) \tag{14}$$

$$h_t = o_t \odot \tanh(c_t) \tag{15}$$

where σ denotes the sigmoid activation function, $\odot$denotes element-wise multiplication, and tanh is the hyperbolic tangent function.

Gated Recurrent Units (GRU): GRUs simplify the LSTM architecture by combining the input and forget gates into a single update gate, reducing the complexity of the model:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{16}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{17}$$

$$\tilde{h}_t = \tanh(W_h x_t + W_h(r_t \odot h_{t-1}) + b_h) \tag{18}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{19}$$

where $z_t$is the update gate, r_t is the reset gate, and $\tilde{h}_t$_tis the candidate hidden state.

RNNs are particularly effective for analyzing sequential data, such as network traffic logs, where the temporal order of events is crucial for identifying patterns indicative of intrusions. In network intrusion detection, RNNs can model the sequential dependencies in network traffic data, enabling the detection of complex attack patterns and anomalies. By processing sequences of network traffic features, RNNs can capture temporal relationships and deviations from normal behavior. This capability is essential for identifying sophisticated attacks that may unfold over time, such as Distributed Denial of Service (DDoS) attacks or multi-stage intrusions. In summary, RNNs provide a robust framework for modeling and analyzing sequential data in network intrusion detection, leveraging their ability to capture temporal dependencies and detect anomalies in network traffic patterns. Their advanced variants, such as LSTMs and GRUs, further enhance their effectiveness in handling long-term dependencies and complex attack scenarios.

3.3. DBN (Deep Belief Network)

Deep Belief Networks (DBNs) are a class of deep learning models that are designed to capture complex hierarchical representations of data. They consist of multiple layers of stochastic, latent variables and are capable of learning intricate patterns from high-dimensional data. DBNs are particularly useful in scenarios where feature learning and dimensionality reduction are critical, such as in network intrusion detection. This section delves into the architecture, mathematical foundations, and application of DBNs, emphasizing their role in identifying network attacks.

Architecture.

A Deep Belief Network is composed of multiple layers of Restricted Boltzmann Machines (RBMs) and a final layer of a supervised model, often a softmax classifier. The architecture of DBNs involves the following components:

Restricted Boltzmann Machines (RBMs): Each RBM in a DBN is a two-layer, stochastic neural network consisting of a visible layer and a hidden layer. The visible layer represents the input data, while the hidden layer captures higher-level features. The network is trained to maximize the likelihood of the data, and the connections between the layers are undirected. The key components are:

- Visible Layer: Represents the observed data and has binary or real-valued units.
- Hidden Layer: Represents the features learned from the input data and captures hidden patterns.
- Weights: The connections between the visible and hidden layers are represented by weights, which are learned during training.

The energy function for an RBM is given by:

$$E(v,h) = -\sum_i b_i v_i - \sum_j c_j v_j - \sum_{i,j} w_{ij} v_i h_j \tag{20}$$

where v and h are the visible and hidden units, respectively, b_i and c_j are biases, and w_{ij} are weights.

Stacking RBMs: DBNs are constructed by stacking multiple RBMs in a deep architecture. Each RBM is trained layer-wise, where the output of one RBM serves as the input to the next. This hierarchical stacking allows the DBN to learn increasingly abstract representations of the input data.

Final Classification Layer: After pre-training the RBMs, a final supervised layer is added to perform classification or regression tasks. This layer typically uses a softmax activation function for classification tasks or a linear activation function for regression.

Mathematical Foundations.

The learning process in DBNs involves two main phases: pre-training and fine-tuning.

Pre-training: The pre-training phase involves training each RBM independently using unsupervised learning. The goal is to learn the weights that maximize the likelihood of the data. The contrastive divergence algorithm is commonly used for training RBMs:

$$\Delta w_{ij} = \eta \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right) \tag{21}$$

Where$\langle v_i h_j \rangle_{data}$ is the expected value of the product of visible and hidden units from the data, and $\langle v_i h_j \rangle_{model}$ is the expected value from the model distribution.

Fine-tuning: After pre-training, the entire DBN is fine-tuned using supervised learning. This involves updating the weights of the final classification layer and, optionally, the weights of the RBMs. The fine-tuning phase uses gradient descent to minimize a loss function, such as cross-entropy loss for classification:

$$L = -\sum_i (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \tag{22}$$

where y_i are the true labels, and $\hat{y}_i$ are the predicted probabilities.

Variants of DBNs.

Several variants of DBNs enhance their capabilities and performance:

- Deep Boltzmann Machines (DBMs): DBMs extend RBMs by introducing bidirectional connections between layers, allowing them to capture more complex dependencies between features.
- Stacked Autoencoders: This variant uses autoencoders instead of RBMs to learn hierarchical features, with each autoencoder trained to reconstruct the input data from a compressed representation.

In network intrusion detection, DBNs can be utilized to learn hierarchical representations of network traffic data, enabling the identification of complex attack patterns. The ability to extract and learn from high-level features makes DBNs well-suited for detecting both known and unknown types of intrusions. By employing DBNs, network intrusion detection systems can benefit from improved feature extraction and dimensionality reduction, enhancing their ability to identify subtle anomalies and patterns indicative of malicious activity. The deep hierarchical structure of DBNs allows for more nuanced understanding and classification of network traffic, potentially leading to higher accuracy and robustness in detecting network attacks. Overall, Deep Belief Networks offer a powerful framework for learning hierarchical feature representations in network intrusion

detection. Their ability to model complex patterns and reduce dimensionality makes them a valuable tool in the ongoing effort to enhance cybersecurity through advanced machine learning techniques.

3.4. Auto-encoder

Autoencoders are a type of unsupervised deep learning algorithm used primarily for dimensionality reduction and feature extraction. The central idea behind autoencoders is to learn a compressed representation (encoding) of input data and then attempt to reconstruct the original input from this reduced representation. In the context of network intrusion detection, autoencoders are highly valuable as they can efficiently capture the normal patterns of network traffic and identify deviations that may signal intrusions.

Architecture.

The architecture of an autoencoder consists of two primary components: an encoder and a decoder. The encoder compresses the input into a latent space representation, and the decoder attempts to reconstruct the input from this encoded data. Autoencoders are trained to minimize the difference between the input and its reconstruction, using a loss function to measure reconstruction error. The architecture can be described as follows:

Encoder: The encoder compresses the input data into a lower-dimensional space, extracting the most salient features. It maps the input data XX to a latent space representation Z:

$$Z = f(X) = \sigma(W_e X + b_e) \tag{23}$$

where W_e represents the weights, b_e the biases, and σ is an activation function such as ReLU or sigmoid.

Latent Space: The encoded or compressed representation ZZ is a bottleneck in the network, forcing the model to focus on the most critical features of the input data. This lower-dimensional representation is what makes autoencoders effective for anomaly detection since it discards noise and irrelevant features.

Decoder: The decoder reconstructs the original data from the compressed latent space. It maps ZZ back to the input space, producing a reconstruction X^X^:

$$\hat{X} = g(Z) = \sigma(W_d Z + b_d) \tag{24}$$

where W_d and b_d represent the weights and biases of the decoder, respectively.

The reconstruction error, which measures the difference between the original input XX and its reconstruction X^X^, is minimized during training. The objective is to learn a compressed representation that retains the most important information:

$$L(X, \hat{X}) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X})^2 \tag{25}$$

Variants of Autoencoders.

Several variants of autoencoders have been developed to enhance their performance and versatility, particularly for tasks like anomaly detection in network intrusion systems:

- Denoising Autoencoders: These autoencoders are designed to reconstruct the input from noisy data, improving robustness. During training, random noise is added to the input, but the network is tasked with reconstructing the clean version.
- Sparse Autoencoders: These enforce sparsity in the latent space by penalizing the number of active neurons, ensuring that the network learns a more compressed, meaningful representation.
- Variational Autoencoders (VAEs): VAEs are a probabilistic variant that not only reconstruct the input but also learn the probability distribution of the latent space. VAEs are useful in generating new data samples from the learned distribution and improving anomaly detection by measuring reconstruction probability.

Mathematical Foundations

The learning process of autoencoders involves the optimization of weights and biases to minimize reconstruction error. The training process can be summarized in the following steps:

Encoding Step: The encoder compresses the input data into the latent space representation:

$$Z = \sigma(W_e X + b_e) \tag{26}$$

Decoding Step: The decoder reconstructs the data from the latent representation:

$$\hat{X} = \sigma(W_d Z + b_d) \tag{27}$$

Loss Function: The reconstruction error is minimized during training. A common loss function for autoencoders is the mean squared error (MSE):

$$L = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2 \tag{28}$$

where X_i represents the original input, and $\hat{X}_i$ is the reconstructed output.

Autoencoders are particularly useful in detecting anomalies in network traffic, which is crucial for identifying network intrusions. In network intrusion detection, an autoencoder is trained on normal traffic data. Since the model is optimized to reconstruct normal patterns, any significant deviation in reconstruction error indicates an anomaly, potentially representing a network attack. The ability of autoencoders to learn compressed representations of high-dimensional network traffic makes them effective for detecting both known and unknown intrusions. By capturing the inherent structure of normal traffic patterns, autoencoders can identify subtle irregularities that other algorithms might miss. In summary, autoencoders provide an efficient and powerful method for anomaly detection in network traffic, offering significant advantages in network intrusion detection systems. Their capability to reduce dimensionality while retaining essential features enables them to differentiate between normal and suspicious activities, making them a valuable tool in the field of cybersecurity.

4. METHODOLOGY

This research aims to develop a Network Intrusion Detection System (NIDS) using deep learning algorithms. The NSL-KDD dataset is utilized for evaluation, and the performance of Convolutional Neural Networks (CNNs), Recurrent Neural Network (RNN), Deep Belief Network (DBN) and Autoencoders is then analyzed and compared.

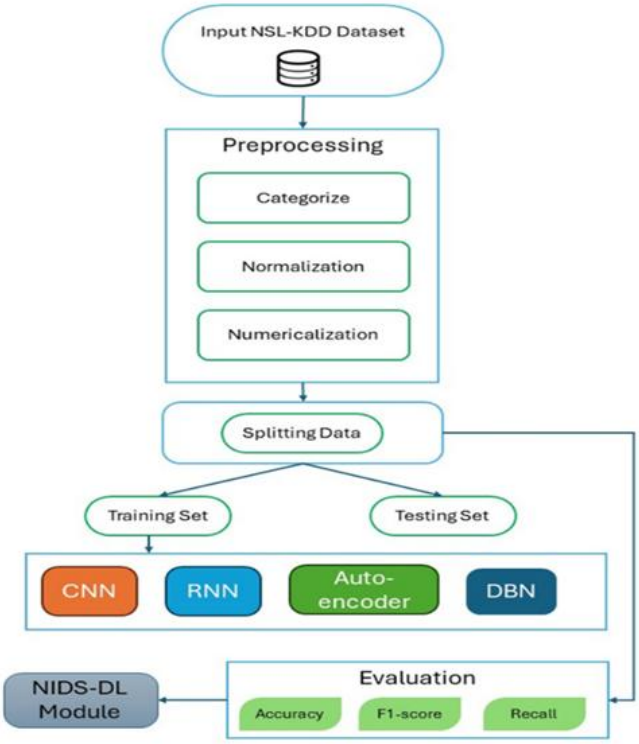

Fig. 1. Methodology for NIDS-DL

4.1. Preprocessing

4.1.1. Dataset preprocessing

The NSL-KDD dataset is used in this research. First, the KDDTrain.txt and KDDTest.txt datasets are combined. Since the dataset is well-structured and contains no null values, data cleaning is not required.

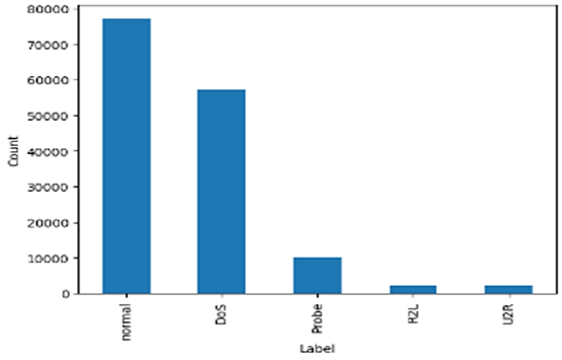

Fig. 2. Categorical level count

Though for better results, we categorize attack levels. And then train and test on dataset on our various models.

Tab. 1. Protocol count

Protocol Type	Count
tcp	121569
udp	17614
icmp	9334

The following steps were employed to preprocess the dataset:

- Drop Columns: Categorize attacks 42 attack level into 5 categories.
- Mapping Attack Subclasses to Main Categories: This section of the code maps certain attack types (sub_label) to more general attack categories (label) in both the training dataset (train_df) and the complete dataset (df_full). By transforming specific attack types into more generic categories using the dict_attack_outcome dictionary, the classification process is made easier, and a more recognizable target variable is provided for modeling.
- Feature and Label Separation with One-Hot Encoding: The features and labels are separated in this section of the code. While the label column in y includes the label, which represents the target variable for prediction, features X are produced by removing the label and sub_label columns from the dataset. The categorical columns in X are then located, and one-hot encoding is used to transform the category values into binary columns. An entirely numerical feature set that may be utilized for machine learning models is produced by encoding, removing the initial category columns from X, and adding the freshly formed binary columns.
- Logarithmic Normalization of Non-Binary Features: This section of the code defines two functions for data normalization: a logarithmic transformation (log(1 + x)) and one utilizing MinMaxScaler. After identifying the columns in X that don't include 0 or 1, it logarithmically normalizes the data in these columns, reducing skewness while maintaining the binary columns' integrity. This makes sure that non-binary characteristics scale correctly.
- Class Weight Calculation for Imbalanced Classes: Although it isn't utilized directly in the next phases, this section of the code sets up a OneHotEncoder to prepare the target labels (y) for machine learning. In order to address the unequal distribution of classes, it then computes class weights. To balance the relevance of each class during model training, weights for each class in the y_train_series are determined using compute_class_weight. These weights, which are helpful for correcting class imbalance in the dataset, are contained in the class_weight_dict that is produced.

- Manual Class Adjustment: In order to solve class imbalance in the dataset, this section of the code constructs a dictionary called class_weight_dict_number, which holds particular class weights. Every key in the dictionary denotes a class label, and the weight allocated to that class is represented by the corresponding value. The class labels that were taken from the earlier class weight computation are stored in the class_name list. In order to enhance the model's performance on underrepresented classes, these weights are employed to modify the relative priority of each class during training.
- K-Fold Cross-Validation Setup: In order to guarantee that every fold preserves the same class distribution as the original dataset, this section of the code sets up Stratified K-Fold Cross-Validation with 10 splits. For reproducibility, it initializes the StratifiedKFold object with shuffles and a random seed. The dataset is divided into training and testing sets for each fold by the code: train_X_new and test_X_new for features, and train_y_new and test_y_new for labels. By employing several train-test splits, this procedure aids in more accurately assessing the model's performance.

Identify the headings.

4.1.2. CNN model

The Convolutional Neural Network (CNN) model begins with an input layer matching the data's feature dimensions. The first convolutional layer applies 32 filters with a 5x5 kernel, followed by ReLU activation to capture important patterns. A max-pooling layer reduces spatial dimensions while retaining essential features. The second convolutional layer, with 64 filters, is followed by another max-pooling layer. The resulting feature maps are flattened into a one-dimensional vector, which passes through two fully connected layers, each with 500 units. To prevent overfitting, a dropout layer with a 0.5 rate is placed between the dense layers. The final output layer is a 5-unit softmax layer for multi-class classification. The model is optimized using Adam with categorical crossentropy loss, and accuracy is used as the evaluation metric.

4.1.3. RNN model

This study uses a Recurrent Neural Network (RNN) model for multi-class classification. It consists of two SimpleRNN layers, each with 50 units. The first SimpleRNN layer is configured to return sequences, preserving temporal relationships across time steps, while the second focuses on capturing sequential patterns. A dropout layer with a 0.2 rate is added to prevent overfitting. The output layer uses a softmax activation function to estimate class probabilities. The model is optimized using Adam with categorical crossentropy loss, and accuracy is used as the evaluation metric.

4.1.4. DBN model

In order to lessen the input complexity, the Deep Belief Network (DBN) version on this code is made from two layered Restricted Boltzmann Machines (RBM). 128 units compose the first RBM layer, and 64 units compose the second layer. After these layers comes a Multi-Layer Perceptron (MLP) classifier with a 128-unit hidden layer for supervised fine-tuning. Data normalization with a StandardScaler is another step in the version pipeline. To ensure a balanced class distribution, the version is educated the usage of stratified five-fold go-validation to assess overall performance on numerous facts splits. Final performance metrics, such as accuracy, remember, and F1-rating, are computed by way of recording cumulative predictions at the side of real values throughout folds. The categorization outcomes of the model are shown through the generation of the confusion matrix.

4.1.5. Auto-encoder model

The auto-encoder model in this study is designed to reduce input data dimensions and extract key features. It starts with an input layer matching the dataset's feature dimensions, followed by dense layers (256 units down to 32) to extract significant features. Each encoding layer uses ReLU activation, batch normalization, dropout, and L2 regularization in the first layer to prevent overfitting. During decoding, the architecture reconstructs the original input, with ReLU activations, except for the final layer, which uses a sigmoid function to ensure accurate reconstruction. The Adam optimizer trains the model using mean squared error (MSE) as the loss function. Early stopping is applied to prevent overfitting. A One-Class SVM is then employed to detect anomalies based on the latent features extracted.

5. EVALUATION & RESULT ANALYSIS

The performance of the Network Intrusion Detection System using deep learning algorithms was evaluated using classification recall, F1 score, and accuracy. The standard criteria for comparing various model setups are true positive rate and false positive rate. Attack detection rate is measured by True Positive Rate, or TPR. Since an anomaly-based IDS typically produces a high false-alarm rate, the False Positive Rate (FPR), widely referred to as the false-alarm rate, is a crucial metric for assessing an IDS. True Positive Rate (TPR) is also called Recall or sensitivity, Recall is described as the ratio of relevant instances that is retrieved. Recall calculation is defined in Equation.

$$\text{Recall} = \frac{TP}{TP+FN} \tag{28}$$

F1-measure is defined as the harmonic mean of the precision and the recall values. The evaluation model uses multi class modified version of F1 (3).

$$\text{F1} = \frac{2*TP}{2*(TP+FP+FN)} \tag{29}$$

Accuracy (Acc) measures the proportion of correct prediction and indicates the proportion of the number of correctly classified data points to total data points (4).

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{30}$$

5.1. Confusion matrix

Confusion matrices are a widely used metric in classification problem solving. For a specific set of test data, the confusion matrix is a matrix which is used to evaluate how well the classification models perform. It can only be determined if the true values of the test data are known. It is also referred to as an error matrix since it displays the errors in the model's performance as a matrix.

True Negative: Model has given prediction Benign, and the actual value was also Benign.

True Positive: The model has predicted Attack, and the actual value was also Attack.

False Negative: The model has predicted Benign, but the actual value was Attack.

False Positive: The model has predicted Attack, but the actual value was Benign.

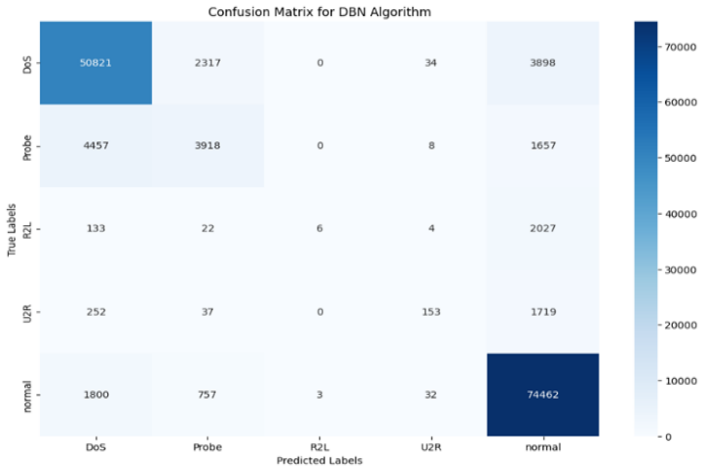

Fig. 2. Confusion matrix for DBN algorithm

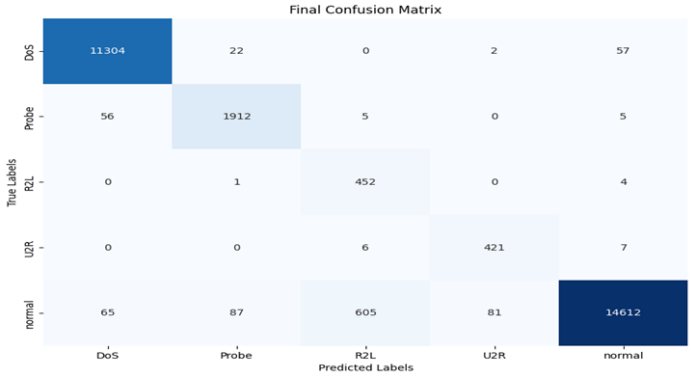

Fig. 3. Confusion matrix for RNN algorithm

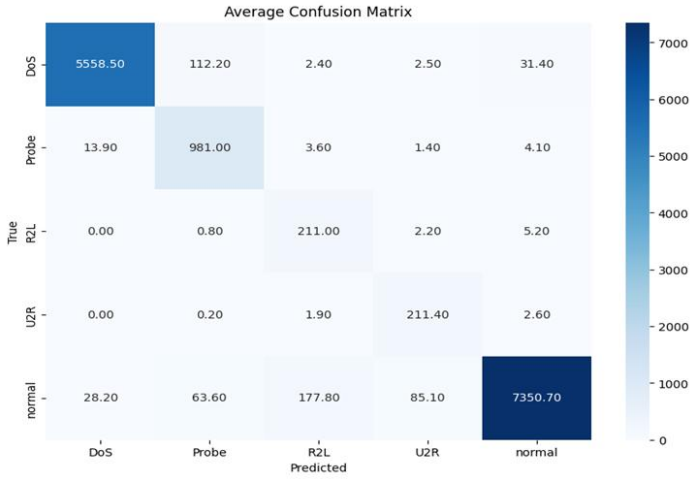

Fig. 4. Confusion matrix for CNN algorithm

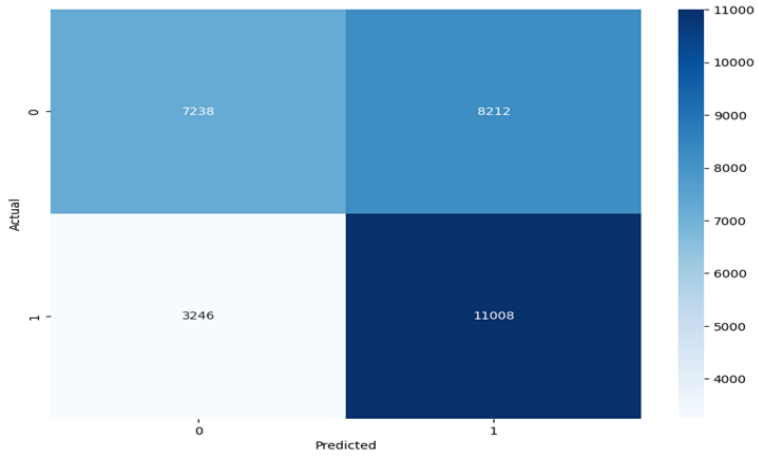

Fig. 5. Confusion matrix for Auto-encoder algorithm

5.2. Result

NIDS (Network Intrusion Detection System) is implemented successfully. The models' strengths differ; autoencoders are more suited for anomaly detection, while CNNs and RNNs perform better in terms of accuracy and recall. DBNs make a substantial contribution to feature representation and dimensionality reduction, despite being marginally less successful in the dataset in question. By tackling changing cyber threats, our collaborative application guarantees a thorough approach to intrusion detection.

The comparison of accuracy, recall and f1-score for different algorithms is shown in the table below.

Tab. 1. Result comparison

	Accuracy	Recall	f1-score
CNN	0.96917	0.96917	0.87766
RNN	0.9662	0.9662	0.9702
DBN	0.8710	0.8710	0.8544
AE	0.946808	0.9463308	0.9446739

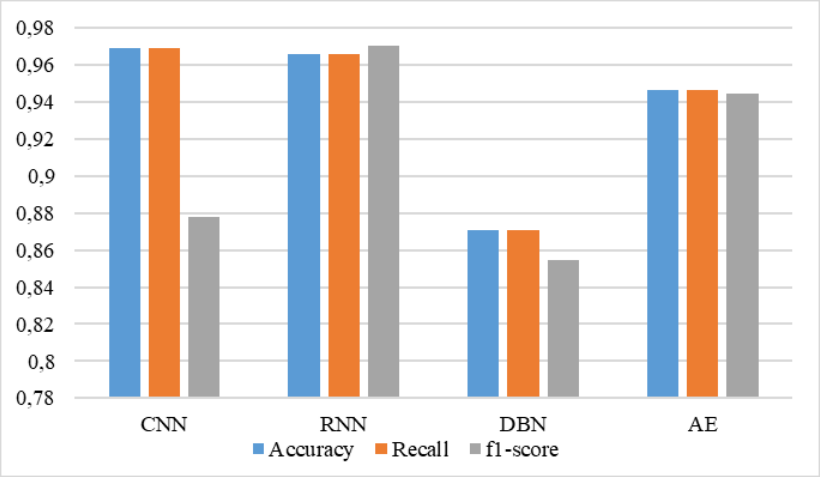

Fig. 6. Performance comparison chart

This chapter discusses the evaluation metrics used to assess the model's performance. Additionally, a confusion matrix plot is presented to visualize normal and categorical attack data. The confusion matrix provides a representation of data distribution. Performance analysis indicates that CNN achieves the highest accuracy and recall, while RNN yields the best F1-score.

6. CONCLUSIONS

This study investigated the use of Deep Belief Networks (DBNs) and Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) layers for network intrusion detection using the NSL-KDD dataset. RNNs outperformed DBNs in recall, accuracy, and F1-score, particularly in classifying attack types, due to their ability to process sequential network traffic data. DBNs, pre-trained with Restricted Boltzmann Machines (RBMs), underperformed due to challenges in hyperparameter tuning. The research addresses a critical gap in intrusion detection systems by demonstrating the advantages of deep learning models over traditional methods. It highlights RNNs with LSTM layers as a superior approach for adaptive and accurate intrusion detection. This study's novelty lies in its comparative analysis, revealing the limitations of DBNs and providing insights for optimization, thus advancing the field of network security both scientifically and practically.

REFERENCES

Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., & Ahmad, F. (2021). Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, *32*(1), e4150. https://doi.org/10.1002/ett.4150

Edeh, D. I. (2021). Network intrusion detection system using deep learning technique. Master of Science, Department of Computing, University of Turku.

Fu, K., Cheng, D., Tu, Y., & Zhang, L. (2016). Credit card fraud detection using convolutional neural networks. In A. Hirose, S. Ozawa, K. Doya, K. Ikeda, M. Lee, & D. Liu (Eds.), *Neural Information Processing* (Vol. 9949, pp. 483–490). Springer International Publishing. https://doi.org/10.1007/978-3-319-46675-0_53

Ghani, H., Virdee, B., & Salekzamankhani, S. (2023). A deep learning approach for network intrusion detection using a small features vector. *Journal of Cybersecurity and Privacy*, *3*(3), 451-463. https://doi.org/10.3390/jcp3030023

Kim, J., Kim, J., Thi Thu, H. L., & Kim, H. (2016). Long short term memory recurrent neural network cassifier for intrusion detection. *2016 International Conference on Platform Technology and Service (PlatCon)* (pp. 1–5). https://doi.org/10.1109/PlatCon.2016.7456805

Kültür, E. (2022). Network intrusion detection with a deep learning approach. Master's thesis, Middle East Technical University (Turkey).

Nasr, M., Bahramali, A., & Houmansadr, A. (2018). DeepCorr: Strong flow correlation attacks on tor using deep learning. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1962–1976). https://doi.org/10.1145/3243734.3243824

Sama, L. (2022). Network intrusion detection using deep learning. Doctoral dissertation, Victoria University.

Sharafaldin, I., Lashkari, A. H., Hakak, S., & Ghorbani, A. A. (2019). Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. *2019 international carnahan conference on security technology (ICCST)* (pp. 1-8). IEEE. https://doi.org/10.1109/CCST.2019.8888419

Shiri, F. M., Perumal, T., Mustapha, N., & Mohamed, R. (2024). A comprehensive overview and comparative analysis on deep learning models. *Journal on Artificial Intelligence*, *6*, 301-360. https://doi.org/10.32604/jai.2024.054314

Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*(1), 41–50. https://doi.org/10.1109/TETCI.2017.2772792

Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2016). Deep learning approach for network intrusion detection in software defined networking. *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 258–263. https://doi.org/10.1109/WINCOM.2016.7777224

Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, *5*, 21954-21961. https://doi.org/10.1109/ACCESS.2017.2762418

Zhang, Y., Chen, X., Jin, L., Wang, X., & Guo, D. (2019). Network intrusion detection: Based on deep hierarchical network and original flow data. *IEEE Access*, *7*, 37004–37016. https://doi.org/10.1109/ACCESS.2019.2905041

Zhang, Z., Zhou, X., Zhang, X., Wang, L., & Wang, P. (2018). A model based on convolutional neural network for online transaction fraud detection. *Security and Communication Networks*, *2018*, 1–9. https://doi.org/10.1155/2018/5680264