International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Residual LSTM Autoencoder: Enhancing Network Intrusion Detection with Forward Feed Mechanism

Mohammad Huzaif Anwar [1], *Nancy Biyaut* [2], *Rohit Sharma*[3]

[1,2,3] **U.G. Student Department of Information Science and Engineering**

[1,2,3]**Nitte Meenakshi College of Engineering, Yelahanka, Bengaluru, Karnataka, India 560064**

ABSTRACT—

In the realm of network security, the timely and accurate detection of network intrusions is of paramount importance. Traditional approaches to intrusion detection often rely on rule-based or signature-based methods, which struggle to adapt to the evolving landscape of cyber threats. In response to this challenge, this study introduces a novel approach known as the Residual LSTM Autoencoder with a Forward Feed Mechanism for network intrusion detection. The study “Residual LSTM Autoencoder: Enhancing Network Intrusion Detection with Forward Feed Mechanism” introduces a cutting-edge technique for detecting network intrusions. The proposed model leverages the power of Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), to capture the temporal dependencies and sequential patterns present in network traffic data. The incorporation of a residual mechanism not only enhances the model's capability to capture subtle anomalies but also effectively addresses the vanishing gradient problem. Our approach leverages the temporal dependencies within network traffic data, capitalizing on the sequential nature of LSTM, and mitigates vanishing gradient issues through the introduction of residual connections. Furthermore, the incorporation of a forward feed mechanism helps the model adapt to emerging threats in real-time. By integrating a forward feed mechanism, our proposed model enhances feature extraction and reconstruction capabilities, thus achieving higher detection accuracy. The experimental results highlight the remarkable performance of the Residual LSTM Autoencoder (RLAFF) model. It achieves an impressive average detection rate of 97.8% with a maximum detection accuracy of 98.8%, outperforming traditional techniques such as Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF) and XG-Boost models.

Keywords: Residual LSTM Autoencoder, Intrusion Detection, Forward Feed Mechanism, Network Security, Gradient problem, Real-time detection, Feature extraction, Feature reconstruction.

INTRODUCTION

In today's digital environment, cybersecurity has become a concern for organizations and individuals. The evolving and growing nature of cyber threats requires new mechanisms to protect critical networks and sensitive data. Traditional impact detection methods often rely on rule-based or signature-based techniques, which prove inadequate in new and unseen environments. This skepticism has led to the development of an intuitive understanding that uses the capabilities of artificial intelligence and deep learning. Network Intrusion Detection (NIDS) is an important security measure to protect networks against malicious attacks. However, traditional NIDS approaches often struggle to keep up with the ever-changing nature of cyber threats. Deep learning has emerged as a promising NIDS approach that provides the ability to learn complex patterns from network traffic data.

This paper presents a new Residual Connections LSTM-autoencoders with Feed Forward Neural Networks (RLAFF) architecture for NIDS. RLAFF provides the advantages of residual and short-term temporal (LSTM) networks to learn physical and temporal properties from network traffic data. The feeding process in the RLAFF architecture enables obtaining information about remote devices, which is important for attack detection. The research paper presents a method for network intrusion detection called "residual LSTM autoencoder" with integrated "feedforward mechanism". This innovation leverages the power of LSTM networks, a combination of neural networks (RNNs) aimed at processing computer events and data networks. It also adopted the work of autoencoders and artificial neural networks, aiming for good results and reducing the size to improve the reconstruction process.

The main idea is around LSTM networks, which are good at capturing network traffic data including time dependencies and sequence patterns. This allows the model to not only identify intrusion, but also detect previously unknown threats, making a significant difference to the normal intrusion process. The integration of the rest of the process further improves the quality of the model, making it possible to capture subtle changes while overcoming the vanishing problem. Additionally, the feed-forward system is integrated, helping to instantly adapt to emerging threats, making the system flexible and efficient. The RLAFF architecture is based on the following observations: Network data traffic is consistent in both space and time. Spatial correlation refers to the relationship between different characteristics of network packets, such as IP address, IP address, and port number. The term correlation refers to the time difference between different packets.

LSTM networks are well-suited for learning temporal patterns in network data, such as network traffic data. Residual networks have been shown to be effective at learning complex patterns in data and may also help solve the problem that deep neural networks lack. RLAFF architecture combines the above observations to create a powerful NIDS system. RLAFF is trained to enhance network access data. During training, RLAFF learns to identify and reconstruct physical and physiological differences between devices. When new network traffic data is provided to RLAFF, it can detect unusual patterns by comparing the reconstructed data to the input data.

The RLAFF architecture also uses a learning-communication machine to learn the distance between devices. Feedforward mechanisms are trained to predict the next order of data traffic in the network based on the current order. This allows RLAFF to detect unusual patterns in the current flow of network packets that may not be immediately noticeable. The RLAFF architecture has been evaluated on two NIDS datasets, NSL-KDD and UNSW-NB15. RLAFF outperforms other deep learning methods such as NIDS by performing state-of-the-art analysis of both datasets.

The application of LSTM autoencoders in network intrusion detection now has great potential to improve the overall security of computer networks. Its ability to detect the surrounding environment, detect the relationship between body and body, and detect new threats make it an important anti-cyber-attack tool. In this article, we now delve into the architecture of LSTM autoencoder and now try to demonstrate its good work in improving network intrusion detection. This new approach is capable of analyzing the network intrusion detection environment and providing strong protection against network threats.

RESEARCH MOTIVATION

- To improve the accuracy of traditional Network Intrusion Detection Systems (NIDS) systems in detecting sophisticated attacks and develop a more efficient NIDS system that can scale to large networks.
- To overcome the limitations of traditional Network Intrusion Detection Systems (NIDS) methods, including their challenges in learning intricate data patterns and their struggles in adapting to evolving cyber threats, it is crucial to develop solutions that can dynamically respond to the ever-changing landscape of attack strategies and techniques as they emerge.
- To effectively combat the ever-evolving landscape of cyber threats, characterized by the constant refinement of attack techniques by malicious actors to exploit network vulnerabilities, traditional security measures such as firewalls and antivirus software fall short in providing adequate protection.

OBJECTIVE

- **Solving the Missing Gradient Problem**: Implement residual connections to address the missing gradient problem in neural networks, thereby improving the flow of information and enhancing training quality and overall performance.
- **Intermittent Monitoring**: Integrate intermittent monitoring to capture network data and anomalies effectively, enabling the system to identify complex intrusion patterns and emerging threats.
- **Real-time Anomaly Detection**: Enhance the NIDS's capability for real-time anomaly detection, enabling swift responses to emerging threats and minimizing potential damage from cyberattacks.
- **Reduce False Positives and False Negatives**: Optimize the NIDS to minimize false positives and false negatives, ultimately boosting its overall accuracy and reducing the risk of unnecessary alerts or missed threats.
- **Improved Accuracy**: The ultimate objective is to elevate the accuracy of NIDS for network intrusion detection. This ensures its effectiveness in identifying and responding to both known and emerging threats while mitigating any adverse consequences.

LITERATURE REVIEW

[1] The ICESC 2020 study combined Principal Component Analysis (PCA) with Random Forest for intrusion detection. It improved accuracy, reduced dimensionality, and enhanced performance. Despite challenges like dimensionality reduction and computational complexity, the approach outperformed traditional algorithms, achieving a 96.78% accuracy rate and a 0.21% error rate in just 3.24 minutes. This research strengthens cybersecurity defenses against evolving threats.

[2] At the RTEICT-2022 conference, a novel Intrusion Detection System (IDS) was introduced, combining Artificial Neural Networks (ANN) with cuckoo search optimization. ANN handles classification, while cuckoo search optimizes ANN training through weight updates, resulting in superior performance. Evaluated on the NSL-KDD dataset, it achieved 99.35% accuracy, with low MAE, MSE, and RMSE, outperforming established methods such as FCANN, NNID, SRF, and IDS-ABC. ANN is inspired by the human brain's neural networks, comprising interconnected neurons for processing and transmitting signals.

[3] Presented at ICESIC 2022, this paper introduces a framework combining ensemble methods and SMOTE for improving Intrusion Detection Systems (IDS). It evaluates several state-of-the-art methods, including Random Forest, XG-Boost, Light-GBM, and Cat-Boost, both before and after applying SMOTE. The study highlights the significance of machine learning in intrusion detection and concludes that over-sampling and ensemble methods enhance cybersecurity effectiveness. The proposed framework, especially when applied to XG-Boost, achieves impressive results, with a Macro Average

Precision of 93.2% and a Macro Average F1 of 95.5%. It also attains the best Macro Average AUC of 99.4% and the best Macro Average Recall of 98.9%, underlining its strong performance in cyber threat detection.

[4] Presented at ICESIC 2022, this paper introduces a framework combining ensemble methods and SMOTE for improving Intrusion Detection Systems (IDS). It evaluates several state-of-the-art methods, including Random Forest, Xgboost, LightGBM, and CatBoost, both before and after applying SMOTE. The study highlights the significance of machine learning in intrusion detection and concludes that over-sampling and ensemble methods enhance cybersecurity effectiveness. The proposed framework, especially when applied to Xgboost, achieves impressive results, with a Macro Average Precision of 93.2% and a Macro Average F1 of 95.5%. It also attains the best Macro Average AUC of 99.4% and the best Macro Average Recall of 98.9%, underlining its strong performance in cyber threat detection.

[5] This research, presented at RTEICT-2019, proposes a novel method for detecting credit card transaction fraud in social media. It combines Benford's Law, which analyzes the distribution of first digits in datasets, with deep learning autoencoders. The autoencoder algorithm achieved a high ROC-AUC of 0.97%, indicating its effectiveness in fraud detection when combined with Benford's Law. It learned patterns of non-fraudulent transactions during training and successfully identified fraudulent transactions in testing.

[6] Presented at ICESIC 2022, this research introduces an Anomaly-Based Network Intrusion Detection System using LSTM and GRU, two recurrent neural networks suitable for anomaly detection. LSTM learns normal behavior patterns and detects deviations during testing. GRU, with a simpler architecture, is faster. Results showed Probe attacks with the highest accuracy at 95%, followed by Probe (92%), R2L (87%), and normal attacks (77%). Precision assessments indicated Probe attacks had the highest precision (0.96), demonstrating effective intrusion detection capabilities.

[7] This article featured in the IEEE Internet of Things Journal, introduces an advanced Network Intrusion Detection System (NIDS) that combines deep learning models like DNN, CNN, and LSTM. The study fine-tunes these models, favoring two-layer structures for optimal stability and performance. It also includes classic machine learning models like SVM and DT for comparison. A significant aspect of their work is the integration of Generative Adversarial Networks (GANs) to address data imbalance issues, leading to enhanced classification accuracy. The results are impressive, with the proposed models achieving high accuracies, up to 93.2% on the NSL-KDD dataset and 87% on UNSW-NB15. This research demonstrates the system's effectiveness in intrusion detection, especially in classifying minor categories, marking a significant advancement in NIDS.

[8] This research published in IEEE Access, introduces a hybrid Intrusion Detection System (IDS) that combines Convolutional Neural Networks (CNN) for spatial feature extraction and Long Short-Term Memory Networks (LSTM) for temporal feature extraction. They enhance the model with batch normalization and dropout layers, and it's evaluated using three datasets: CIC-IDS 2017, UNSW-NB15, and WSN-DS, for binary and multiclass classifications. The model demonstrates high detection rates, accuracy, and low false alarm rates, highlighting its effectiveness in intrusion detection and enhancing network security.

[9] The paper addresses the challenge of intrusion detection in large-scale networks due to increased advanced threats. It introduces the Long Short Term Memory Gate Recurrent Neural Network (LSTMgateRNN) for improved intrusion detection. This model combines LSTM features and gate functions to process attack data efficiently. In simulations using various datasets, LSTMgateRNN achieves an impressive 99% attack detection rate and outperforms existing models by 6-12%. This research contributes to enhancing network security and intrusion detection capabilities.

[10] The research published in IEEE-Transaction on networking addresses the vulnerability of Neural Networks (NNs) in Network Intrusion Detection Systems (NIDS) to adversarial attacks. They introduce the TIKI-TAKA framework, assessing NIDS robustness and proposing defense mechanisms. Adversarial attacks can evade NIDS with up to 35.7% success by altering time-based features. The proposed defenses, including model voting ensemble adversarial training, and query detection, restore intrusion detection rates to nearly 100%. These methods are effective in countering various malicious traffic, even potentially catastrophic attacks like botnets. This research emphasizes the importance of robust and reliable deep anomaly detectors in network security.

PROPOSED METHODOLOGY

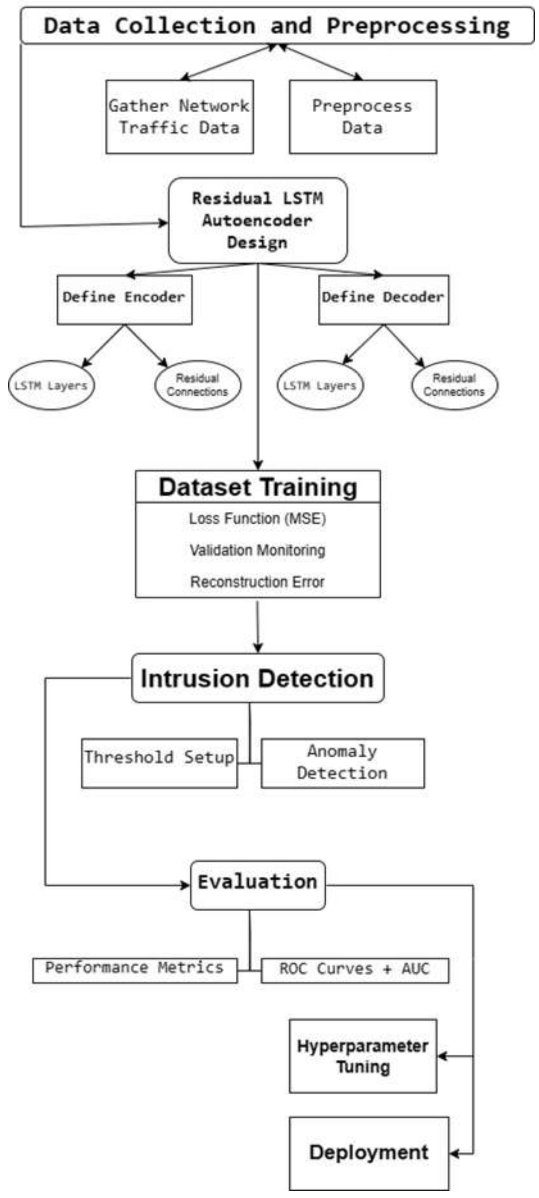


Fig. 1-Flow diagram of methodology

Step 1: Data Collection and Preprocessing

Data collection is a fundamental step in enhancing network intrusion detection, and it involves gathering network traffic data from various sources. This data serves as the foundation for subsequent analysis and intrusion detection.

There are two primary methods for collecting network traffic data:

A. Passive Collection: In passive collection, network traffic is observed and monitored without interfering with its flow. This can be achieved using network sniffing tools or by tapping into network cables. It is less disruptive to the network but may not capture all the traffic of interest.

B. Active Collection: Active collection involves generating network traffic for analysis by sending probes or packets into the network. Tools like traffic generators or network monitoring software can be employed for this purpose. While it allows for capturing specific types of traffic, it may introduce artifacts into the data.

The choice between passive and active collection depends on the specific analysis requirements. Passive collection is less disruptive but may have limitations in capturing all relevant traffic, whereas active collection can be tailored but introduces potential biases. Once the network traffic data is collected, it undergoes preprocessing to make it suitable for analysis:

- **Filtering**: Noise or irrelevant traffic is removed from the dataset to focus on relevant network activities.
- **De-identification**: Personally identifiable information (PII), such as IP addresses or MAC addresses, is removed to protect privacy and comply with data protection regulations.
- **Feature Extraction**: Relevant features, such as packet size, flow duration, or protocol type, are extracted from the data. These features are essential for the subsequent analysis.
- **Normalization**: Data is scaled to a common range, often [0, 1], to ensure that all features have similar scales, which is crucial for machine learning models.
- **Segmentation**: Data can be divided into smaller segments based on time intervals or types of traffic, which can facilitate more focused analysis.

These preprocessing steps ensure that the data is clean, accurate, and structured for effective analysis. It is important to tailor the preprocessing to the specific requirements of the intrusion detection system.

Step 2: Residual LSTM Autoencoder Design

Residual LSTM Autoencoder Design The coronary heart of the intrusion detection methodology lies within the Residual LSTM Autoencoder, an advanced neural network structure capable of shooting complicated temporal patterns inside network site visitors information. This design permits the device to differentiate between typical network conduct and ability intrusions correctly. It combines the strengths of Long Short-Term Memory (LSTM) layers and residual connections to effectively capture temporal dependencies and mitigate issues like the vanishing gradient hassle. This structure incorporates two important additives: the encoder and the decoder.

2.1: Autoencoders

Autoencoders are a type of unsupervised neural network architecture designed for data compression and reconstruction. They include an encoder and a decoder. The encoder takes in the enter data and compresses it into a latent representation. This latent representation captures important functions of the input records in a compressed shape. The decoder then takes this latent illustration and reconstructs the authentic statistics as intently as viable. By education the autoencoder to minimize the distinction among the enter and the reconstructed output, it learns to seize the maximum salient information inside the records and discard irrelevant info. This compressed illustration can be useful for various tasks along with dimensionality discount, wherein the purpose is to lessen the variety of capabilities while preserving vital data. Autoencoders also can be hired for anomaly detection, in which deviations from the learned ordinary patterns can be recognized.

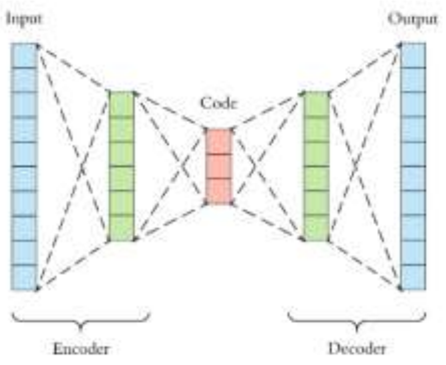


Fig. 2-Autoencoders

Autoencoder (AE) Architecture:

An autoencoder is composed of three layers: Input Layer, Hidden Layer, Output Layer.

Neuron Counts: The input and output layers have the same number of neurons, representing the dimensions of the input data. However, the hidden layer has fewer neurons, allowing it to capture a compressed representation of the input.

Operations in the Autoencoder:

- **Encoding Operation (Forward Pass):** During the encoding operation, the input data, represented as a vector [x1, x2, x3, ..., xm], is processed. The input is transformed into a hidden layer representation (y) using the following formula:**y = f1(wx + b)** y represents the hidden layer representation. f1 is the activation function of the encoder.b is the bias vector. w is the weight matrix.
- **Decoding Operation (Reconstruction - Reverse Pass):** In the decoding operation, the hidden representation (y) is transformed back into a new reconstructed vector (x¯) using the following equation:**x¯ = f2(w′y + b′)** x¯ is the reconstructed output. f2 is the activation function of the decoder. b′ and w′ are the bias vector and weight matrix for the output layer.

- **Training and Parameter Updates:** The neural network's parameters (including w, w′, b, and b′) are continuously adjusted during training to minimize the reconstruction error.
- **Reconstruction Error (Loss Function - L):** The loss or reconstruction error (L) is calculated using a non-linear function (typically the mean squared error) to measure the difference between the input data and the reconstructed output. The loss function is defined as:**L(x, x¯) = (1/m) * Σ (xj - x¯j)^2** L represents the reconstruction error. x is the input data.x¯ is the reconstructed output. m is the number of data points.

This loss function can be minimized using gradient descent. The gradient of the loss function with respect to the weight matrix for the encoder can be written as: **dL/dW = ∇L(x, x') * ∂x'/dW** where ∇L(x, x') is the gradient of the loss function with respect to the reconstructed input data, and ∂x'/dW is the partial derivative of the reconstructed input data with respect to the weight matrix for the encoder.

The gradient of the loss function with respect to the bias vector for the encoder can be written as: **dL/db = ∇L(x, x') * ∂x'/db** where ∇L(x, x') is the gradient of the loss function with respect to the reconstructed input data, and ∂x'/db is the partial derivative of the reconstructed input data with respect to the bias vector for the encoder.

The gradient of the loss function with respect to the weight matrix for the decoder can be written as: **dL/dW' = ∇L(x, x') * ∂x'/dW'** where **∇L(x, x')** is the gradient of the loss function with respect to the reconstructed input data, and ∂x'/dW' is the partial derivative of the reconstructed input data with respect to the weight matrix for the decoder.

The gradient of the loss function with respect to the bias vector for the decoder can be written as: **dL/db' = ∇L(x, x') * ∂x'/db'** where ∇L(x, x') is the gradient of the loss function with respect to the reconstructed input data, and ∂x'/db' is the partial

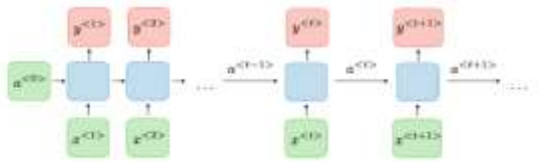
Training Objective:

The primary objective during training is to minimize the reconstruction error, which involves updating the neural network's parameters. This process enables the autoencoder to learn a compressed representation of the input data in the hidden layer, which can be used for tasks such as data compression, dimensionality reduction, and feature learning.

2.3: Recurrent neural network

Recurrent Neural Networks (RNNs) are specialized neural networks designed for sequential or time series data. They are widely used in applications such as language translation, natural language processing (NLP), speech recognition, and image captioning. Unlike traditional deep neural networks, RNNs have a memory that allows them to consider prior inputs when processing the current input and generating output. This sequential dependence is crucial for tasks where the order of elements matters.

Standard RNN (Recurrent Neural Network): In a standard RNN, the network processes sequential data by maintaining a hidden state that is updated at each time step. The hidden state at a particular time step is influenced by the input at that time step and the hidden state from the previous time step. The standard RNN is represented as a loop, indicating the recurrence of information at each time step. The Standard RNN allow previous outputs to be used as inputs while having hidden states. Fig 3 represents the architecture of traditional RNN



For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

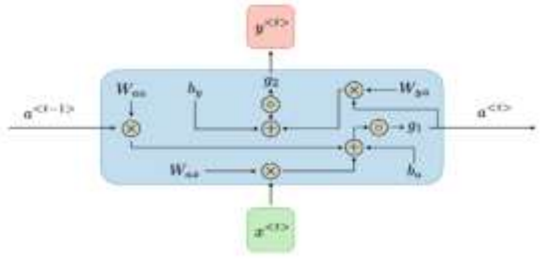


Fig. 3- Traditional RNN

Unfolded RNN (Recurrent Neural Network): The unfolded RNN is represented as a stack of layers, with each layer representing a single time step and has visualization technique that represents the recurrence over multiple time steps as distinct layers in a deep neural network. Each time step is treated as a separate layer, and the connections between layers represent the recurrence of information from one time step to the next. It provides a clearer view of how information flows through the network over time. The recurrent connections are shown explicitly as connections between the layers. This representation can be helpful for understanding how RNNs work, but it is less compact than the standard RNN representation.

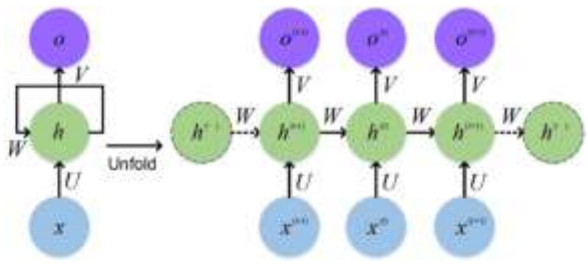


Fig. 4-Unfolded RNN

BASIC RNN STRUCTURE

1. Three Sets of Weights:

- Input-to-hidden weights (Wx)
- Hidden-to-hidden weights (Wh)
- Hidden-to-output weights (Wy)

2. Hidden State Calculation:

The hidden state (h_t) at time step t is calculated using the formula:

h_t = f(W_x * x_t + W_h * h_{t-1})

where h_t is the hidden state at time step t, x_t is the input at time step t, and h_{t-1} is the hidden state at the previous time step (t-1). W_x and W_h are weight matrices that are learned during training.

3. Forward Pass: During the forward pass, the RNN processes the input sequence step by step. At each time step t, it calculates the hidden state *h*_t using the above given formula. The hidden state from the previous time step (h_(t-1)) serves as the initial hidden state for the current time step.

4. Output Calculation: The RNN can produce an output at each time step or only at the final time step, depending on the task. The output at time step t is computed as**: y_t = g(Wy * h_t) where y_t** is Output at time step t and g is an activation function or identity function.

5. Backpropagation Through Time (BPTT): To train an RNN, you use a technique called Backpropagation Through Time (BPTT). BPTT is a form of backpropagation that extends back in time through the sequence. The loss is calculated for each time step. Gradients are computed with respect to the loss and the network's parameters (Wx, Wh, Wy) at each time step. These gradients are then accumulated and used to update the network's parameters through gradient descent.

2.3: Long Short Term Memory(LSTM)

Long Short-Term Memory (LSTM) networks are a specialized sort of recurrent neural network (RNN) designed to conquer the vanishing gradient problem encountered in traditional RNNs. This trouble arises whilst gradients come to be extremely small throughout education, hindering the community's capability to analyze long-time period dependencies in sequential records. LSTMs have a greater tricky architecture with memory cells and gating mechanisms, letting them keep and update information effectively over extended sequences. This functionality makes them well-appropriate for responsibilities like speech recognition, language modeling, and time collection prediction. LSTMs have established fulfillment in numerous applications, which includes natural language processing, speech reputation, and gadget translation, showcasing their effectiveness in managing sequential statistics with lengthy-term dependencies. The LSTM network architecture consists of three parts, as shown in the image below, and each part performs an individual function

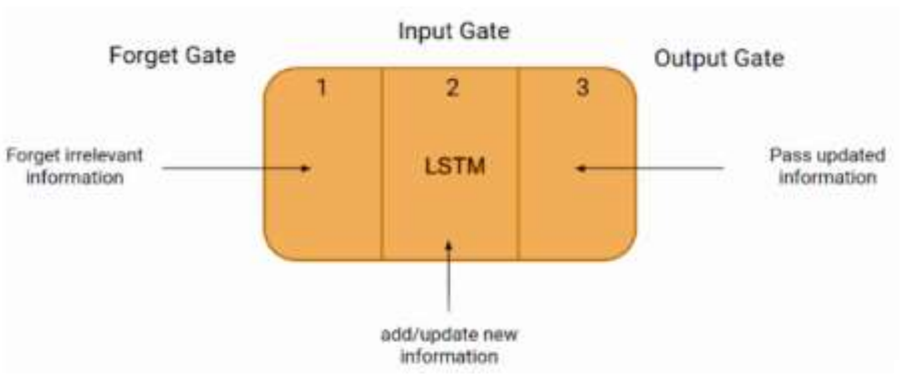


Fig. 5- LSTM Architecture

LSTM Architecture & Working

The LSTM architecture consists of several key components:

- Cell State (Ct): The cell state is a long-term memory storage unit that runs along the entire sequence. LSTMs have a separate cell state that runs through the entire sequence It allows LSTMs to selectively retain or discard information.
- Three Gates: LSTMs use three gates to control the flow of information: Forget Gate (f_t), Input Gate (i_t), and Output Gate (o_t).
- Forget Gate (f_t): Decides what information from the cell state should be forgotten. It is influenced by the previous hidden state h $_{(t-1)}$ and the current input x_t. Computed using the sigmoid activation function.

$F_t = \sigma (W_f * [h_{(t-1),} x_t] + b_f)$

- Xt: input to the current timestamp.
- b_f: weight associated with the input
- Ht-1: The hidden state of the previous timestamp
- Wf: It is the weight matrix associated with the hidden state

- Input Gate(i_t): Determines what new information should be stored in the cell state. It is influenced by the previous hidden state h $_{(t-1)}$ and the current input ***x$_t$***.Computed using the sigmoid activation function.

$i_{t=} \sigma (W_i * [h_{(t-1),} x_t] + b_i)$

- Cell State Update: Updates the cell state with new information. The cell state is updated by combining the information selected to be forgotten ($f_t * C_{(t-1)}$) and the new information to be stored ($i_t * C_t$).

The cell state is updated using two equations:

➢ $C_t = \tanh (W_C \cdot [h_{(t-1),}\ x_t] + b_C)$

This equation calculates the candidate cell state by taking the weighted sum of the previous hidden state ($h_{(t-1))}$ and the current input (x_t), and applying the hyperbolic tangent activation function (tanh). $C_t = f_t \cdot$

➢ $C_{(t-1)} + i_t * C_t$

This equation updates the cell state by combining the previous cell state ($C_{(t-1))}$ with the candidate cell state (C_t), weighted by the forget gate (f_t) and input gate (i_t) respectively.

- Output Gate (o_t): Decides the next hidden state (h_t) based on the updated cell state. It influences how much of the cell state information should be output. Computed using the sigmoid activation function.

$o_{t=} \sigma (W_o * [h_{(t-1),} x_t] + b_o)$

- Hidden State Update: Updates the hidden state using the cell state. The hidden state h_t is updated based on the output gate's decision and the updated cell state.

$h_t = ot * \tan(C_t)$

2.4: Forward Feed Neural Network:

A Feed-Forward Neural Network (FFNN) begins as a single-layer perceptron, the only form of a neural community. In this version, inputs go through multiplication via weights, and the ensuing weighted inputs are summed to form a complete. If this sum surpasses a predefined threshold (generally set at 0), the output is 1; in any other case, it's -1. FFNNs, a category of artificial neural networks, encompass more than one layers of interconnected neurons. Neurons within a layer connect to those within the next layer, permitting statistics to float exclusively in a single direction—from the enter layer to the output layer.

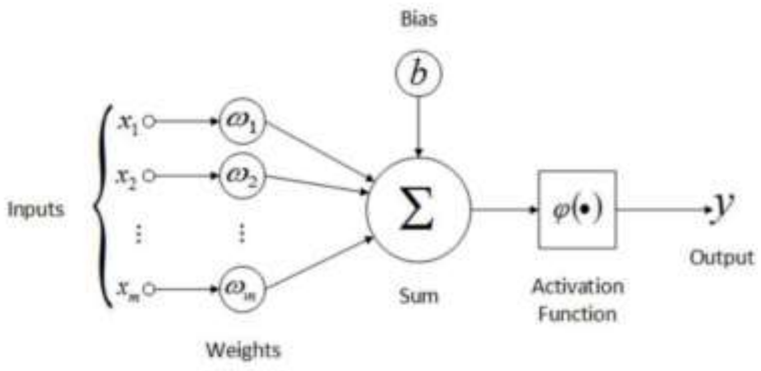


Fig. 6- Forward Feed Neural Network Architecture.

Forward Feed Neural Network Architecture(FFNN).

FFNNs can have any number of layers, but a typical FFNN has three layers: an input layer, a hidden layer, and an output layer. The input layer receives the input data, the hidden layer processes the input data, and the output layer produces the output data.

FFNN Forward Propagation:FFNNs work by performing forward propagation. Forward propagation is the process of passing the input data through the network and computing the output data. The following equations show the forward propagation process for a simple FFNN with two hidden layers:

h_1 = ϕ(W_1x + b_1)

h_2 = ϕ(W_2h_1 + b_2)

y = W_3h_2 + b_3

where:

- h 1 is the hidden state of the first hidden layer.
- h 2 is the hidden state of the second hidden layer.
- y is the output of the network.
- ϕ is an activation function, such as the sigmoid function or the ReLU function.
- W 1 , W 2 , and W 3 are the weight matrices between the layers.
- b 1 , b 2 , and b 3 are the bias vectors between the layers.

FFNN Training: FFNNs are trained using a supervised learning algorithm. Supervised learning algorithms learn from labeled data, which means that the input data and the desired output data are known. The following equations show the training process for a simple FFNN with two hidden layers:

Loss = ∑(y_i - ŷ_i)^2

∂Loss/∂W_3 = ∑(y_i - ŷ_i)h_2^T

∂Loss/∂b_3 = ∑(y_i - ŷ_i)

∂Loss/∂W_2 = ∑(y_i - ŷ_i)h_1^Tϕ(W_3h_2 + b_3)

∂Loss/∂b_2 = ∑(y_i - ŷ_i)ϕ(W_3h_2 + b_3)

∂Loss/∂W_1 = ∑(y_i - ŷ_i)x^Tϕ(W_2h_1 + b_2)

∂Loss/∂b_1 = ∑(y_i - ŷ_i)ϕ(W_2h_1 + b_2)

where:

- y i is the desired output for the $i$th input data point.
- y^i is the actual output for the $i$th input data point.
- Loss is the loss function.

The weight matrices and bias vectors are updated using gradient descent. Gradient descent is an iterative algorithm that updates the weights and biases in the direction that minimizes the loss function.

Step 4 : Training Datasets & Intrusion Detection

The combined approach of using Residual Connections LSTM-autoencoders with Feed Forward Neural Networks (RLAFF) can be applied as a methodology for network intrusion detection systems (NIDS). Here's a summary of how it can be used:

- ➢ Data Collection: The dataset ensured that it is comprehensive, encompassing a wide range of network traffic scenarios. Include both normal instances and diverse forms of anomalous behavior, covering known intrusion patterns.
- ➢ Pre-training: The importance of training the autoencoder is emphasize exclusively on normal data. The autoencoder learns to reconstruct normal patterns without exposure to anomalous instances. The autoencoder learns to reconstruct normal patterns accurately
- ➢ Fine-tuning: The pre-trained autoencoder is combined with an RLAFF. The encoder part of the autoencoder is used as a feature extractor and is connected to the RLAFF layers. Modify the RFFNN to have additional layers for classification or anomaly scoring. Emphasize that this integration allows the RFFNN to leverage the learned features from the autoencoder for enhanced performance.
- ➢ Architecture Modification: The adjustments made to the RLAFF architecture is elaborated to accommodate the integrated features. The addition of extra layers within the RLAFF for specific tasks such as classification or anomaly scoring is mentioned. In detecting intrusions the importance of fine-tuning is important.
- ➢ Supervised training: Train the combined model (autoencoder + RLAFF) using labeled data, where known intrusions are indicated. This can involve using a separate dataset with labeled intrusions or artificially introducing intrusions into the training data. The combined model undergoes supervised training using labeled data indicating instances of known intrusions. This step refines the model's ability to recognize and classify diverse intrusion patterns.
- ➢ Anomaly detection: During NIDS operation, incoming network traffic is processed through the combined model. An anomaly score, calculated by comparing reconstructed output with the original input, serves as an indicator of potential intrusions.. If the error or score exceeds a predetermined threshold, it indicates the presence of a potential intrusion.

In summary, the method combines the accuracy of autoencoders in regular pattern capture with the flexibility of residual feed forward neural networks (RLAFFs) to handle complex anomalies Model effectiveness depends on different training datasets, anomaly thresholds optimized, and on continuous updates to adapt to evolving network threats. NIDS) ensured that it.

Step 5: Intrusion Detection Performance Evaluation

5.1: ROC Curve

An ROC (Receiver Operating Characteristic) curve is a graphical representation that illustrates the performance of a binary classification model at various classification thresholds. The curve is created by plotting the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various threshold settings.

True Positive Rate (TPR) is the proportion of actual positive instances correctly predicted by the model i.e the fraction of true positives that are correctly identified. The TPR is calculated as follows:

TPR = TP / (TP + FN)

where: TP is the number of true positives, FN is the number of false negatives.

False Positive Rate (FPR) is the proportion of actual negative instances incorrectly predicted as positive by the model i.e FPR is the fraction of negative instances that are incorrectly identified as positive. The FPR is calculated as follows:

FPR = FP / (FP + TN) where: FP is the number of false positives, TN is the number of true negatives.

5.2: AUC Curve

Area under the curve (AUC) is a metric used to describe the overall performance of a binary classification model based on its receiver operating characteristic (ROC) curve The ROC curve presents the False Positive Rate (1 - specificity) against the True Positive Rate (sensitivity) on decision requirements.The AUC provides a scalar value that summarizes the ability of the model to distinguish between good and bad classes across all possible thresholds. Here is the breakdown of the AUC:

AUC Specification:

- A model with an AUC of 1.0 shows perfect discrimination, that is, perfect sensitivity (100%) and specificity (0% false positive rate) on all thresholds
- A model with an AUC of 0.5 shows performance equivalent to random chance. In this case, the ROC curve is a diagonal line from bottom-left to top-right.

- Models with AUCs ranging from 0.5 to 1.0 exhibit some discriminative power, with higher AUCs corresponding to better overall performance.
- AUC as a measure of discrimination:

In general, higher AUC indicates better discrimination. For example, an AUC of 0.8 indicates that a randomly selected positive sample of the sample has an 80% probability of being better fitted than a randomly selected negative sample

Relationship with ROC Curve. The AUC is closely related to the shape and location of the ROC curve. The model with the higher AUC tends to be on the ROC curve near the upper left corner. When comparing models, it is generally considered that the one with the highest AUC has the best overall performance for discrimination.

5.3: Performance Metrics

Performance metrics are quantifiable measures of the success or effectiveness of a system or process. They are used to track progress, identify areas for improvement, and make informed decisions. Performance metrics, whether applied extensively to system evaluation or particularly within the geographical regions of device mastering and data science, serve as quantifiable measures to gauge the fulfillment and effectiveness of a machine, method, or version.

Extending this idea to the domain of intrusion detection structures (IDS), overall performance metrics come to be instrumental in comparing the device's functionality to discover and classify intrusions. Just as in system getting to know, those metrics in IDS are crucial for monitoring the device's progress, pinpointing areas that require enhancement, and facilitating informed selection-making inside the ongoing quest for effective intrusion detection and prevention.

Some common performance metrics used in the evaluation of classification models are.

Accuracy: It represents the proportion of correctly classified models among all predictions made by the model. It is a key metric that provides an overall assessment of the correctness of the model.

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

Precision (Positive Predictive Value): Precision measures the accuracy of positive predictions made by the model. It answers the question: Of all instances predicted as positive, how many were correctly predicted? It's particularly relevant when the cost of false positives is high.

$$Precision = \frac{TP}{TP + FP}$$

Recall (Sensitivity or True Positive Rate): Recall, also known as sensitivity or true positive rate, gauges the model's ability to capture all relevant instances of the positive class. It answers the question: Of all actual positive instances, how many did the model correctly predict?

$$Recall = \frac{TP}{TP + FN}$$

F1 Score: The F1 Score combines precision and recall into a single metric. It balances the trade-off between precision and recall, providing a comprehensive measure of a model's performance. It is particularly useful when there is an uneven class distribution.

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

In summary, while accuracy provides a holistic view of a model's correctness, precision and recall focus on specific aspects related to positive predictions. The F1 Score, by combining these metrics, offers a balanced evaluation, especially in situations where there is an imbalance between the positive and negative classes. Each metric serves a unique purpose in assessing different facets of a classification model's performance.

Step 6: Hyperparameter Tuning

Hyperparameter tuning is an integral part of training any machine learning model, and is especially important for intrusion detection systems (IDS). Hyperparameters are parameters that define the learning process of a machine learning model. Hyperparameters are external design parameters that are not learned from the data but are set prior to the training process. Tuning of these hyperparameters requires finding the best combination that results in the best model performance. There are numbers of hyperparameters that can be tuned in IDS. Some of the most important hyperparameters are:

- Learning Rate: The learning rate controls how quickly the model updates its parameters. A high number of classes can lead to faster convergence, but it can also lead to overconvergence. Lower learning rates can lead to slower convergence, but can also help prevent overconvergence.
- Power of regularization: Regularization is a technique that can be used to prevent overfitting. This works by adding a penalty to the loss function to discourage the model from finding complex patterns that are not supported by the data. The force constant controls the severity of the penalty.
- Epochs: An epoch is a single pass through the entire training data set. The number of epochs determines the duration of the model training. Too few training sessions can prevent the model from correctly recognizing the data, while too many training sessions can lead to overfitting.

Our approach in Hyperparameter Tuning are.

- There are several ways to tune hyperparameters in IDS one way is grid search. A grid search is a systematic approach to explore various combinations of hyperparameter values. To perform a grid search, the user specifies the values set for each hyperparameter he wants to tune. The model is then trained and evaluated for each combination of hyperparameter values. The combination of hyperparameter values that produces the best results is then selected as the best hyperparameter.
- Another approach to hyperparameter tuning is to use random search. Random search is similar to network search, but instead of trying all possible combinations of hyperparameter values, the random search algorithm randomly samples hyperparameter values from specified ranges perform hyperparameter tuning, especially when The number of hyper parameters because the tuning is high.

EXPERIMENT RESULTS

A. Dataset

The NSL-KDD database is extensively used in research to test and demonstrate various identification methods. Considered one of the most popular databases in the industry, it solves the problems of multiple trains and test data samples. The database is an updated version of the KDD'99 database, which offers several advantages.Compared to KDD'99, NSL-KDD does not include duplicate records of trains and test parts. This exclusion is necessary to prevent the classifier from iterating records again. The total number of records in the NSL-KDD contains the same problem as the number of records in the main dataset.

The number of records in the train and test sections is realistic and sufficient to evaluate, so that the entire data set is used rather than a random subset sample This order of choice ensures that from comes out consistently across multiple analyzes based on this data set.In the NSL-KDD database, each feature vector has 43 components, with 41 features per class. Element 42 represents strategic attack, while element 43 indicates complexity. The data set includes attack types such as DOS, Probe, R26, and U2R etc.

In summary, the NSL-KDD is a valuable tool for effect detection research due to its adequate sample size in terms of true representation components, avoidance of duplicate records, and train testing.

Figure 7 shows the distribution of attacks and frequent scenarios in the data set. The process of detecting an attack or the absence of an attack involves two steps: first, it determines whether an attack occurred for each data sample, as described in this review

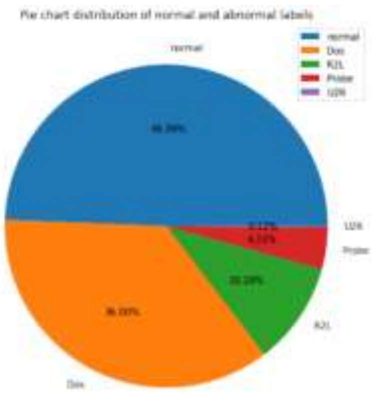


Fig 7. Distribution of Attacks

Figure 8 illustrates the distribution of attack instances along with their respective names and total counts. This representation provides a comprehensive view of the dataset by detailing the number of occurrences for each specific attack type.

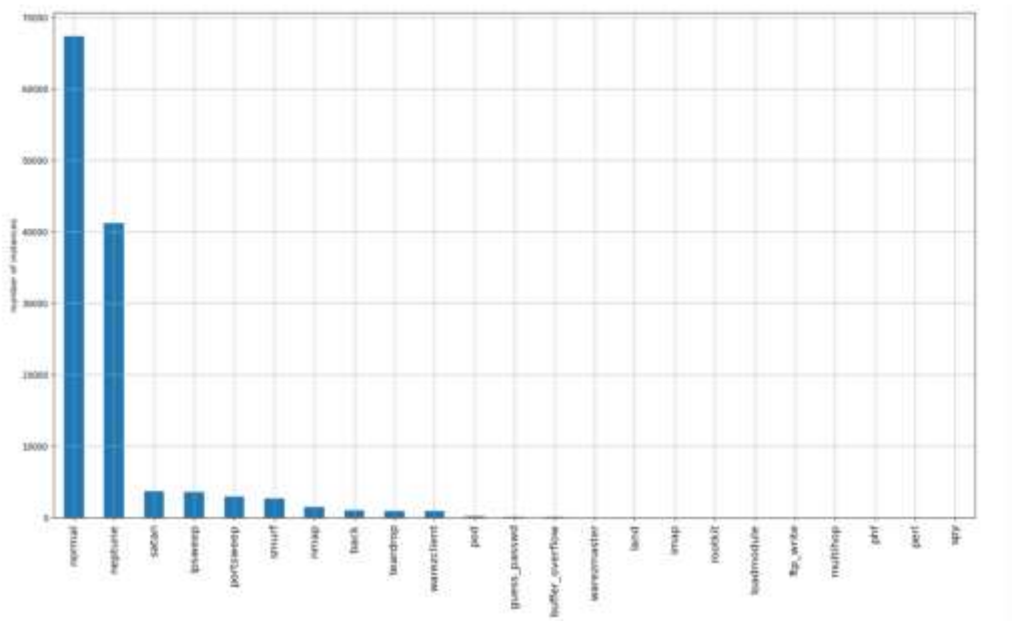


Fig. 8- Distribution of attacks instances.

B. EXPERIMENTS

All models were trained using the Python-based deep learning toolkit TensorFlow TensorFlow makes efficient mathematical operations on GPUs easier, increasing computational performance. These simulations were trained on a system with a Core i7 CPU, an NVIDIA RTX3060 graphics card and 128 GB of RAM. The use of a GPU reduced the training time by a ten times. However, speed was not adequately considered in the project.Efforts were made to minimize validation loss for both AutoEncoder and LSTM models..

C. RESULTS

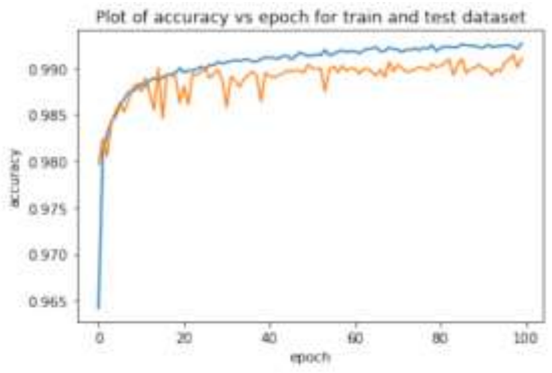
In the pursuit of improving community intrusion detection, our proposed Residual Connections LSTM-autoencoders with Feed Forward Neural Networks (RLAFF) model demonstrated robust performance across various stages of training and testing.. We meticulously allocated 50% of the dataset for training, ensuring the version's exposure to a numerous set of situations. This training subset was further validated with the residual 20%, and the model's efficacy was rigorously evaluated on the remaining 30% of the dataset, serving as the comprehensive test bed for assessing its real-world applicability.

It took 60 epochs of training our RLAFF model to adapt and prevent overfitting. The important step was taken that the training set at the beginning of each period was randomized, reducing the risk of the sample memorizing specific patterns and ensuring additional input data effectively Notably, our model was developed entirely from scratch

In our experiment, we performed two specific steps to optimize the performance and versatility of the model. The first experiment focused on binary classification, where RLAFF was skillfully trained to distinguish between normal network activity and potential intrusions Notably, it demonstrated its expertise in distributed attacks into malicious or benign categories, providing an initial understanding of the ability to flag and isolate threats.

The second experiment examined the depth of multidimensional networks. Our RLAFF model acknowledged the challenge of classification, and specifically distinguished between types of intrusions, such as denial of service (DoS), detection, user-rooted (U2R),Probe, Brute Force attacks, remote-local (R2L), and general network behavior. This extended classification enables a more nuanced and granular understanding of network operations, and contributes to a more sophisticated and effective intrusion detection system.

Figures 9 and 10 provide crucial insights into our Residual LSTM Autoencoder model's binary classification training. Fig 10 depicts the loss vs epoch for both training and test datasets, showcasing convergence and generalization. In Fig 9, the targeted loss vs epoch plots specifically for binary classification highlight the model's ability to distinguish between normal and intrusive network activities.



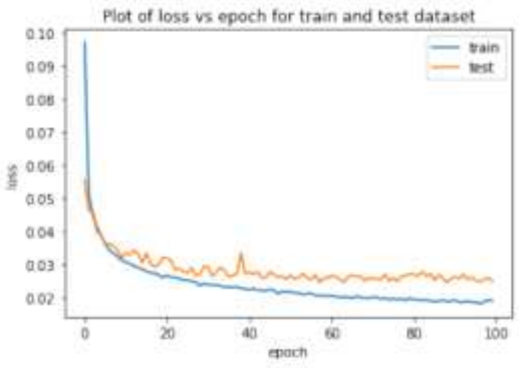
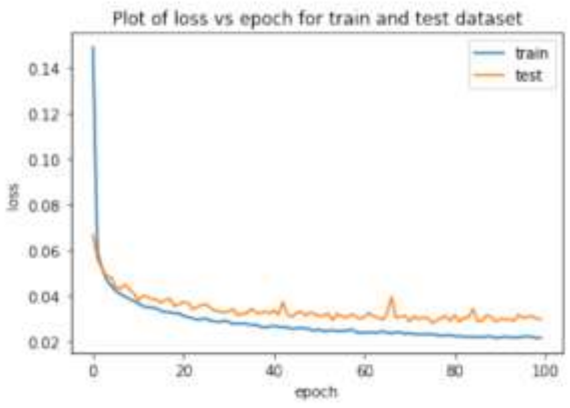


Fig. 9- accuracy vs epoch plots for binary classification.

Fig. 10- loss vs epoch plots for binary classification.

Figures 11 and 12 provide a brief overview of the training of our Residual LSTM Autoencoder model for several classifiers. Figure 11 illustrates the model's loss vs epoch plots, and highlights its efficiency in dealing with various barriers. In Fig. 12, accuracy vs epoch plot illustrates how impressive the model is at accurately classifying samples across multiple intrusions. These diagrams encompass the learning dynamics of the model and highlight its effectiveness in several classifications for network access detection.



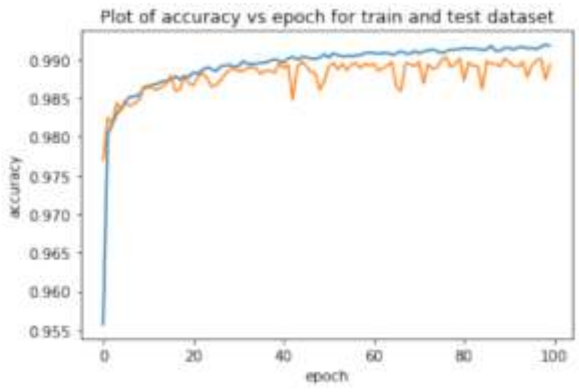


Fig. 11- loss vs epoch plots for multi-classification.

Fig. 12- accuracy vs epoch plots for binary classification.

In Figure 13, our model's ROC curve demonstrates exceptional performance with an AUC (Area Under the Curve) result of 0.9948067. This high AUC underscores the model's outstanding ability to maintain a very high true positive rate while minimizing false positives, affirming its accuracy in distinguishing between normal and intrusive network activities.

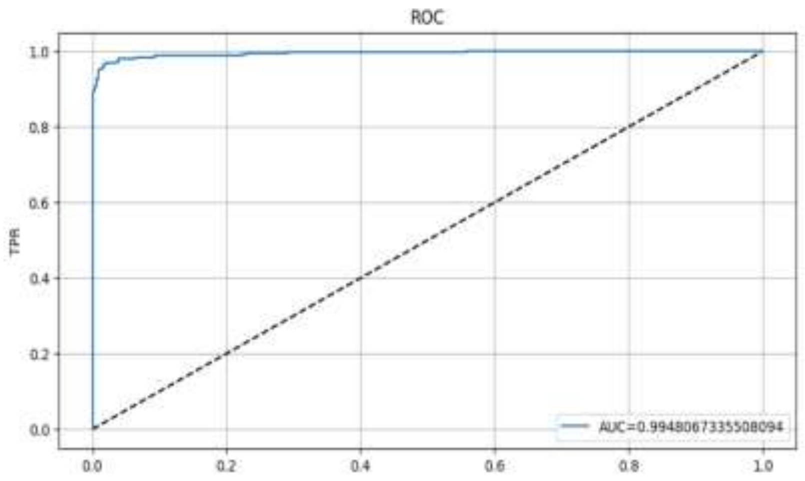


Fig. 13-ROC curves of RLAFF model

In Figure 14, the F1 score vs. threshold plot depicts the dynamic relationship between model performance and decision thresholds. This graph serves as a crucial tool for identifying an optimal balance between precision and recall based on the specific needs of the network intrusion detection application.

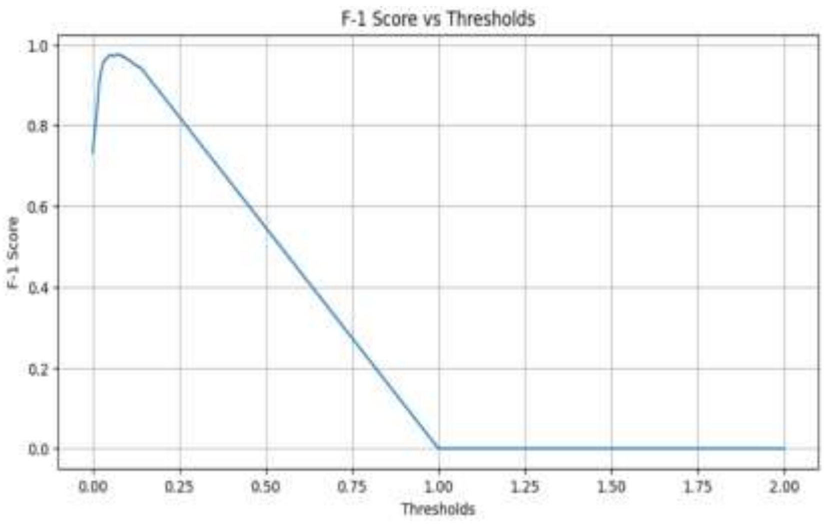
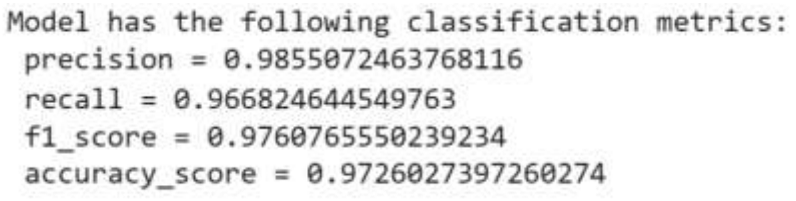


Fig. 14- F1 score vs. Threshold of RLAFF model

Our rigorous performance comparison against similar models revealed a substantial advancement in our RLAFF model.Further, through dedicated optimization efforts, we elevated the overall performance metrics, attaining a precision of 98.55%, recall of 96.68%, an F1 score of 97.61%, and an accuracy score of 97.26%. These enhancements underscore the effectiveness of our model in network intrusion detection, outperforming its counterparts and fortifying its reliability in real-world scenarios.



```
Model has the following classification metrics:
 precision = 0.9855072463768116
 recall = 0.966824644549763
 f1_score = 0.9760765550239234
 accuracy_score = 0.9726027397260274
```

CONCLUSION

In summary, the Residual LSTM Autoencoder with a Forward Feed Mechanism (RLAFF) not only addresses the shortcomings of traditional intrusion detection methods but also sets a new standard for accuracy and adaptability. By leveraging the temporal dependencies within network traffic data and mitigating challenges like the vanishing gradient problem through residual connections, RLAFF excels in capturing subtle anomalies. The model's performance, with an average detection rate of 97.8% and a maximum accuracy of 98.8%, outshines conventional approaches such as Support Vector Machine, Decision Tree, Random Forest, and XG-Boost models. Its ability to distinguish between various intrusion types, including denial of service, user-rooted, probe, brute force, and remote-local attacks, adds a layer of sophistication to intrusion detection. Furthermore, the ROC curve with an AUC result of 0.9948067 and the F1 score vs. threshold plot underscore the model's robustness in achieving a high true positive rate while minimizing false positives. RLAFF's precision of 98.55%, recall of 96.68%, F1 score of 97.61%, and accuracy score of 97.26% demonstrate its reliability in accurately classifying network activities. The model's success in real-world scenarios positions it as a crucial tool for safeguarding networks and contributing to the ongoing advancement of intrusion detection technology.

REFERENCE

1. S. Waskle, L. Parashar and U. Singh, "Intrusion Detection System Using PCA with Random Forest Approach," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2020, pp. 803-808, doi: 10.1109/ICESC48915.2020.9155656.

2. Imran, M., Khan, S., Hlavacs, H. *et al.* Intrusion detection in networks using cuckoo search optimization. *Soft Comput* **26**, 10651–10663 (2022). https://doi.org/10.1007/s00500-022-06798-2

3. F. Li, W. Ma, H. Li and J. Li, "Improving Intrusion Detection System Using Ensemble Methods and Over-Sampling Technique," 2022 4th International Academic Exchange Conference on Science and Technology Innovation (IAECST), Guangzhou, China, 2022, pp. 1200-1205, doi: 10.1109/IAECST57965.2022.10062178. M. Muniandy. O. Gabriel, and T. Ern, “Implementation of pharmaceutical drug traceability using blockchain technology,” Int. J., vol. 2019, p. 35, Jun. 2019

4. S. Suhana, S. Karthic and N. Yuvaraj, "Ensemble based Dimensionality Reduction for Intrusion Detection using Random Forest in Wireless Networks," 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2023, pp. 704-708, doi: 10.1109/ICSSIT55814.2023.10060929.
5. K. L. Kurien and A. A. Chikkamannur, "Benford's Law and Deep Learning Autoencoders: An approach for Fraud Detection of Credit card Transactions in Social Media," 2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT), Bangalore, India, 2019, pp. 1030-1035, doi: 10.1109/RTEICT46194.2019.9016804.
6. R. Koniki, M. D. Ampapurapu and P. K. Kollu, "An Anomaly Based Network Intrusion Detection System Using LSTM and GRU," 2022 International Conference on Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2022, pp. 79-84, doi: 10.1109/ICESIC53714.2022.9783500.
7. C. Park, J. Lee, Y. Kim, J. -G. Park, H. Kim and D. Hong, "An Enhanced AI-Based Network Intrusion Detection System Using Generative Adversarial Networks," in IEEE Internet of Things Journal, vol. 10, no. 3, pp. 2330-2345, 1 Feb.1, 2023, doi: 10.1109/JIOT.2022.3211346.
8. A. Halbouni, T. S. Gunawan, M. H. Habaebi, M. Halbouni, M. Kartiwi and R. Ahmad, "CNN-LSTM: Hybrid Deep Neural Network for Network Intrusion Detection System," in IEEE Access, vol. 10, pp. 99837-99849, 2022, doi: 10.1109/ACCESS.2022.3206425.
9. Kshirsagar, P. R., Yadav, R. K., Patil, N. N., & Makarand L, M. (2022). Intrusion Detection System Attack Detection and Classification Model with Feed-Forward LSTM Gate in Conventional Dataset.
10. C. Zhang, X. Costa-Pérez and P. Patras, "Adversarial Attacks Against Deep Learning-Based Network Intrusion Detection Systems and Defense Mechanisms," in IEEE/ACM Transactions on Networking, vol. 30, no. 3, pp. 1294-1311, June 2022, doi: 10.1109/TNET.2021.3137084.
11. M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, “Autoencoder-based feature learning for cyber security applications,” in 2017 International joint conference on neural networks (IJCNN). IEEE, 2017, pp. 3854–3861.
12. S. Minaee and Z. Liu, “Automatic question-answering using a deep similarity neural network,” in 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP). IEEE, 2017, pp. 923–927.
13. M. S. Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut, “Ddosnet: A deep-learning model for detecting network attacks,” in 2020 IEEE 21st International Symposium on” A World of Wireless, Mobile and Multimedia Networks”(WoWMoM). IEEE, 2020, pp. 391–396.
14. S. Hosseini and B. M. H. Zade, “New hybrid method for attack detection using combination of evolutionary algorithms, svm, and ann,” Computer Networks, vol. 173, p. 107168, 2020.
15. Ullah S, Ahmad J, Khan MA, Alkhammash EH, Hadjouni M, Ghadi YY, Saeed F, Pitropakis N. A New Intrusion Detection System for the Internet of Things via Deep Convolutional Neural Network and Feature Engineering. *Sensors*. 2022; 22(10):3607. https://doi.org/10.3390/s22103607.