Optimizer Algorithms Analysis for Intrusion Detection System on Deep Neural Network

H. A. Danang Rimbawa, Agung Nugroho, Muhammad Abditya Arghanie
Cyber Defense Engineering Study Program-Faculty of Science and Defense Technology,
The Republic of Indonesia Defense University, Bogor, Indonesia

***Abstract*—Intrusion Detection Systems (IDS) play a critical role in identifying potential threats and intrusions in real-time within information technology infrastructures. The development of IDS using Deep Neural Networks (DNN) with the UNSW-NB15 dataset has shown significant potential in improving attack classification accuracy. However, the performance of the DNN-based IDS models is highly dependent on the choice of optimization algorithm. This study compares the performance of several commonly used optimizers in DNN training, including SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Adafactor, and Nadam. The quantitative analysis demonstrates that Adam achieves the highest accuracy among all optimizers tested, while Adadelta performs the worst. RMSprop shows instability in both validation accuracy and loss convergence, indicating challenges in adapting the learning rate for consistent learning. The ANOVA analysis yields an F-statistic of 34.687, which is greater than the F-critical value of 2.140 at a significance level of $\alpha = 0.05$. This result confirms a statistically significant difference in performance among the tested optimization algorithms. These findings provide valuable insights for selecting the most appropriate optimizer to enhance the performance of DNN-based intrusion detection systems. Furthermore, this research contributes to the existing literature by offering a comprehensive comparative evaluation of optimizers, supporting future studies in improving IDS optimization strategies.**

***Keywords*—Deep Neural Networks (DNN); Intrusion Detection System (IDS); optimization algorithms; UNSW-NB15 dataset**

I. Introduction

In today's rapidly evolving digital era, information technology infrastructure has become increasingly vulnerable to sophisticated and complex cyber-attacks. Cyber threats take many forms, including malware, phishing, and targeted intrusions that compromise sensitive data and critical systems. Intrusion Detection Systems (IDS) play a vital role in safeguarding networks by continuously monitoring traffic, identifying suspicious behavior, and providing timely alerts. Effective IDS implementation significantly enhances an organization's ability to detect, prevent, and respond to cyber-attacks swiftly and accurately.

Advancements in IDS technologies have enabled the development of more intelligent and adaptive solutions. Modern IDS increasingly integrates machine learning [2] and deep learning to detect attack patterns that traditional rule-based or signature-based systems may overlook. As organizations become more dependent on digital infrastructures and online services, artificial intelligence (AI) has emerged as an essential component in strengthening cybersecurity defenses. AI-driven IDS solutions enable rapid and precise detection, allowing organizations to proactively mitigate threats before substantial damage occurs.

Previous studies have explored the use of artificial intelligence, particularly Deep Neural Networks (DNN), within the IDS frameworks. Aleesa et al. demonstrated promising results using DNNs optimized with Adam on the UNSW-NB15 dataset, which contains a diverse set of modern cyber-attack scenarios and is widely used for benchmarking IDS performance. However, the effectiveness of DNN-based IDS heavily depends on the choice of optimization algorithm employed during training, which directly influences convergence stability, classification accuracy, and model robustness.

Although several optimization algorithms have been evaluated, existing studies often focus on limited contexts, such as specific datasets, single optimizers, or models other than IDS. For example, Dogo et al. evaluated various optimization methods within CNN architectures; however, their work does not provide a comprehensive comparison tailored specifically for IDS using deep learning, nor does it include statistical validation to confirm whether performance differences among optimizers are significant. This highlights a clear gap in the literature, in which no study has systematically examined multiple optimizers on IDS with DNN while applying rigorous statistical methods.

To address this gap, this research conducts a systematic comparative analysis of eight optimization algorithms, namely Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSprop), Adaptive Moment Estimation (Adam), Adadelta, Adagrad, Adamax, Adafactor, and Nesterov-accelerated Adam (Nadam), applied to a DNN-based IDS trained on the UNSW-NB15 dataset. The novelty of this work lies in its quantitative approach, including the use of ANOVA to statistically validate whether there are meaningful differences in optimizer performance. By integrating empirical testing and formal statistical analysis, this study provides a deeper understanding of how optimizer selection affects IDS accuracy, stability, and convergence behavior.

The contributions of this study are threefold. First, it provides the most comprehensive statistical comparison to date of commonly used optimization algorithms within a DNN-based IDS framework. Second, the study identifies the most suitable optimization algorithm for the UNSW-NB15 dataset, offering practical guidance for cybersecurity researchers and practitioners seeking to enhance IDS performance. Third, the

findings demonstrate the substantial impact of optimizer selection on IDS design and deployment, contributing valuable insights for strengthening proactive cybersecurity strategies in response to increasingly sophisticated cyber threats.

II. Related Works

In this section, the researchers discuss the application of Deep Neural Networks (DNN) in Intrusion Detection Systems (IDS) by analyzing several optimization algorithms, including SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Adafactor, and Nadam. This study focuses on evaluating and comparing the performance of these optimization algorithms to enhance the effectiveness of DNN-based IDS. Previous research on DNN for IDS has also examined various architectures and learning strategies aimed at improving detection accuracy and computational efficiency. Collectively, these studies contribute to the development of more robust and adaptive IDS models capable of addressing evolving cyber threats.

To provide a structured overview of prior research, Table I summarizes relevant studies, including the models used, optimizers applied, datasets utilized, and their relevance to the present work. The inclusion of this table ensures that related works are not only listed but also explicitly integrated into the discussion, in accordance with the reviewer's comments.

TABLE I. Summary of Related Works

Author	Model	Optimizer	Dataset	Relevance
[1]	CNN	N/A	UNSW-NB15	The paper discusses the use of CNN in IDS using the UNSW-NB15 dataset, so that it can be used as a reference for using the UNSW-NB15 dataset in research.
[3]	DNN	Adam	UNSW-NB15	The paper discusses the use of DNN with the Adam optimizer on IDS using the UNSW-NB15 dataset, which makes it relevant to this study.
[5]	CNN	$_{V}$SGD, SGD_{M}, SGD_{NM}, ADAGrad, RMSProp, ADAM, ADADelta, ADAMax, NADAM	Cats and Dogs, Natural Images, and Fashion MNIST	The paper discusses the comparison of optimization algorithms for image classification using CNN, so that it can be used as a reference for comparing optimization algorithms on IDS in this research.
[6]	CNN	$_{V}$SGD, SGD_{M}, SGD_{NM}, ADAGrad, RMSProp, ADAM, ADADelta, ADAMax, NADAM	MNIST, CIFAR-10. LFW, Kaggle Flowers	The paper discusses the comparison of optimization algorithms for image classification using CNN, so that it can be used as a reference for comparing optimization algorithms on IDS in this research.
[7]	VAE	NADAM	NSL-KDD, UNSW-NB15	The paper discusses the use of VAE with the NADAM optimizer on IDS using the UNSW-NB15 dataset, so it can be used as a reference in this research.
[8]	DNN	SGD	NSL-KDD	The paper discusses the use of DNN with the SGD optimizer on IDS using the NSL-KDD dataset, so that it can be used as a reference in research.

As shown in Table I, [1] explores the use of Convolutional Neural Networks (CNN) for IDS using the UNSW-NB15 dataset. This study highlights the suitability of UNSW-NB15 for intrusion detection research due to its comprehensive coverage of modern cyberattack types. The relevance of [1] lies in validating the dataset selection adopted in the present research.

The study [3], also summarized in Table I, examines the use of DNN with the Adam optimizer on the UNSW-NB15 dataset. Insights from [3] support the need to evaluate multiple optimizers, as Adam has shown promising performance but has not been compared extensively against other optimization algorithms in a standardized IDS environment.

The study [5] provides a comparative analysis of optimization algorithms for CNN-based image classification tasks across multiple datasets. Although this domain differs from IDS, the comparative methodology is relevant and demonstrates the importance of understanding optimizer behavior across various learning contexts. This serves as a conceptual foundation for evaluating optimization algorithms within IDS, which is the core objective of the present study.

Similarly, [6] compares multiple optimization algorithms using CNN on several datasets, including MNIST, CIFAR-10. LFW and Kaggle Flowers. As indicated in Table I, this study offers methodological guidance for multi-optimizer evaluation and reinforces the need for robust comparative experimentation.

The study [7] discusses the use of Variational Autoencoders (VAE) with the Nadam optimizer for IDS using NSL-KDD and UNSW-NB15. This finding demonstrates that optimization algorithms significantly influence IDS performance across different machine learning models, further justifying the present study's focus on optimizer comparison.

The study [8] evaluates DNN with the SGD optimizer using the NSL-KDD dataset. As summarized in Table I, this work highlights how optimizer selection affects DNN performance in IDS applications.

Collectively, the studies outlined in Table I reinforce the importance of evaluating optimization algorithms and provide the theoretical and methodological foundation for this research. They emphasize the necessity of systematically comparing multiple optimizers on the same IDS task using a consistent DNN architecture and dataset. Furthermore, integrating these studies into the discussion allows the current research to clearly position its contributions within the broader body of literature and identify remaining gaps, particularly the lack of comprehensive optimizer comparisons for IDS using the UNSW-NB15 dataset.

III. Materials and Methods

In this study, the researchers examine the use of Deep Neural Networks (DNN) for Intrusion Detection Systems (IDS) based on the UNSW-NB15 dataset by analyzing the performance of several optimization algorithms, including SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Adafactor, and Nadam. The primary focus of this work is to evaluate how these optimizers influence model convergence, accuracy, and stability when applied to DNN-based IDS.

Previous research on DNNs for IDS has typically focused on exploring different network architectures or improving feature extraction techniques. However, studies that provide a systematic comparison of optimization algorithms within the same experimental setting are still limited. This research addresses that gap by offering a structured and comprehensive evaluation of multiple optimizers under identical conditions, enabling a clearer understanding of their relative effectiveness for intrusion detection tasks. Through this comparative analysis, the study contributes to the development of more efficient IDS models capable of responding to evolving cyber threats.

A. UNSW-NB15 Dataset

The UNSW-NB15 dataset was developed and published by the University of New South Wales in 2015. Since its introduction, this dataset has become one of the most important data sources in network security research. UNSW-NB15 contains various categories of attacks that represent different types of threats that may occur in a real network environment. In addition, the dataset records the number of unique IP addresses involved during the simulation and data collection process, providing a comprehensive view of diverse network traffic characteristics. This dataset has been widely used to develop and evaluate intrusion detection techniques, as well as to assess the performance of various security algorithms [4].

TABLE II. UNSW-NB15 Dataset

Category	Number of Attacks
Normal	93000
Analysis	2677
Backdoor	2329
DoS	16353
Exploit	44525
Fuzzers	24246
Generic	58871
Reconnaissance	13987
Shellcode	1511
Worms	174
Total	257673

As shown in Table II, the UNSW-NB15 dataset consists of nine major attack categories, including Normal, Analysis, Backdoor, DoS, Exploit, Fuzzers, Generic, Reconnaissance, Shellcode, and Worms, with a total of 257,673 attack instances. This distribution illustrates the broad coverage of cyberattack types contained in the dataset, making it suitable for performance evaluation of machine learning-based IDS models.

The dataset provides a rich set of features extracted from packet headers, protocol information, connection duration, transferred bytes, and many other attributes. These features enable detailed analysis of network behavior and support the development of more effective intrusion detection models. With this level of feature diversity, researchers can perform thorough data preprocessing, identify critical traffic patterns, and develop advanced intrusion detection techniques using machine learning approaches such as Deep Neural Networks (DNN). The features included in the UNSW-NB15 dataset serve as an essential foundation for building, training, and validating IDS models in contemporary cybersecurity research.

B. Deep Neural Network

Deep Neural Networks (DNN) incorporate one or more hidden layers between the input and output layers of an artificial neural network. These hidden layers enable the model to learn complex patterns and relationships within the dataset. During the learning process, the DNN iteratively adjusts its parameters to minimize the difference between predicted outputs and actual targets. This optimization process involves propagating input data through the network, generating predictions, comparing them with the true labels, and calculating the resulting error. Using appropriate optimization algorithms, the network updates its parameters to reduce this error and improve overall performance [6].

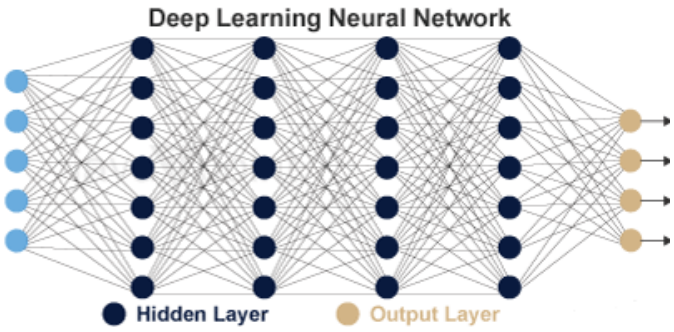

Fig. 1. DNN architecture.

Fig. 1 illustrates the general architecture of a DNN. Each layer in a DNN performs specific transformations on the input data, gradually extracting higher-level features as the information passes through the network. These layers consist of interconnected neurons, where each neuron computes a weighted sum of its inputs and applies an activation function to introduce non-linearity into the model. This nonlinear transformation allows DNNs to learn complex and abstract representations from data, making them highly effective for tasks such as image recognition, video analysis, and intrusion detection [7].

The design of the hidden layers plays a crucial role in determining the performance of a DNN. Factors such as the number of hidden layers, the number of neurons in each layer, and the activation functions used have a direct impact on the model's ability to learn and generalize. In the context of Intrusion Detection Systems (IDS), it is essential to consider

how modifications to the hidden layer structure may influence the model's responsiveness to changing network environments and evolving attack patterns [8].

C. Optimization Algorithms

The selection of an optimization algorithm for a neural network is one of the most important steps in the process of developing an effective and efficient model. With the wide variety of algorithms available, the right choice can make a big difference in model performance and accuracy. One of the key factors in choosing an algorithm is understanding the characteristics of the data and the task at hand. Each algorithm has its own advantages and disadvantages and is suitable for certain types of problems.

The use of optimization algorithms in Deep Neural Networks (DNN) can affect performance, convergence speed, and even the possibility of the resulting model. Several factors that need to be considered in selecting an algorithm include the type of problem being faced, data size, network architecture, and available computing resources. In this study, we compare the performance of different optimization algorithms on the same task using the same model and dataset.

D. Stochastic Gradient Descent (SGD)

The Stochastic Gradient Descent (SGD) optimization algorithm is commonly used in training Deep Neural Networks (DNN). This algorithm works by iterating through the training dataset and updating the model parameters based on the gradient of the loss function estimated from a number of samples (usually a batch) at each iteration [9]. In DNN, SGD updates the weights and biases of the neural network to minimize the value of the loss function produced by the network on the training data. The SGD algorithm can be a simple algorithm and can be successfully used in machine learning models [10].

E. Adagrad

The AdaGrad optimization algorithm individually adjusts the learning rate of model parameters. A rapid decrease in learning rate occurs for parameters with the largest loss partial derivatives, while parameters with small partial derivatives experience a relatively small decrease in learning rate, which is implemented by utilizing all historical squared gradient values. In general, for some learning models, these algorithms work well for simple quadratic problems but often stop too early when training neural networks [11].

Adaptive Gradient (AdaGrad) uses a temporal history of gradient updates to improve convergence speed and reduce reliance on manually tuning learning rates, making it a popular choice for Deep Neural Network (DNN) optimization. The algorithm adaptively updates the learning rate to enable accelerated convergence without the effort of adjusting the learning rate. Therefore, this method is a popular, ready-to-use choice for DNN optimization. The Stochastic Gradient Descent (SGD) optimization algorithm is commonly used in training [12].

F. AdaDelta

Adaptive Delta (Adadelta) uses model parameters with individually determined learning rates. The learning rate in Adadelta decreases dynamically over time. This concept differs from many other optimization algorithms that use a fixed or linearly decreasing learning rate. In Adadelta, this decrease in learning speed occurs gradually until it finally reaches a point where the learning process cannot continue any further. This reduction in learning rate can be an important factor in maintaining model stability and convergence, especially in the case of large and complex data sets. By adaptively adjusting the learning rate, Adadelta helps minimize the risk of getting stuck in undesirable local minimum and increases the chances of finding better solutions globally [13].

G. RMSProp

Root Mean Square Propagation (RMSProp) with momentum updates parameters by applying momentum to the rescaled gradient. RMSProp sets the learning rate by reducing it by the root mean square value of the gradients accumulated from previous updates. In RMSProp, the learning rate is reduced to a small value, allowing updates to be automatically adjusted to approach the local minimum. RMSProp performs better in a nonconvex setting by converting the gradient accumulation into an exponentially weighted moving average. RMSProp uses an exponentially decreasing moving average to ignore the history of past extreme points so that it can quickly converge after encountering a convex bowl [14].

H. Adam

Adaptive Moment Estimation (Adam) [11], [15] is one of the most widely used optimization algorithms in deep learning by calculating individual adaptive learning rates for different parameters of the first and second gradient moment estimates. Adam combines the advantages of AdaGrad, which works well with sparse gradients, and RMSProp, which works well in online and non-stationary settings. Additionally, Adam includes a bias correction in the first and second moment estimates to account for their initialization.

Adam requires little adjustment to the learning rate and is easy to implement, and does not change as the gradient diagonal scale changes. Adam is also computationally efficient and requires little memory. Additionally, Adam is suitable for non-stationary purposes and problems with very noisy and sparse gradients. Adam combines the advantages of RMSProp and momentum. Adam [14] is often used as a replacement for traditional stochastic gradient reduction methods that dynamically update the learning rate for each parameter and are considered computationally efficient with low memory requirements. Mean Square Propagation (RMSProp) with momentum updates parameters by applying momentum to the rescaled gradient.

I. AdaMax

AdaMax [15] is a development of the Adam optimization algorithm, which takes the basis of the infinity norm. The AdaMax algorithm starts by calculating the gradient against the stochastic objective, then estimating the first moment bias, and calculating the infinite norm is then weighed exponentially. Biased first moment estimation is a method to estimate the average gradient at each time step, while the exponentially weighted infinity norm is used to measure the magnitude of the largest gradient at each time step. Parameter adjustments in

Adamax are carried out by considering the biased first moment estimates as well as the exponentially weighted infinite norm. AdaMax can help improve the efficiency and performance of optimization algorithms, especially in machine learning contexts involving complex models with many parameters. In [16], AdaMax's performance is superior in numerical function optimization in model learning for image classification.

J. Adafactor

Adafactor [17] simply maintains the average number of moves per row and per column and estimates the second moment per parameter based on this number. With this approach, Adafactor efficiently updates model parameters by considering information from simplified exponential moving averages, i.e., only per-row and per-column calculations. This allows Adafactor to have fast and efficient performance in optimizing models, especially on large-scale problems or in deep learning contexts where model complexity and data volume are very high. The simplicity of the Adafactor approach allows it to be implemented easily and can be applied in a variety of learning models.

K. Nadam

Nesterov-accelerated adaptive momentum estimation (Nadam) [18] simplifies the Nesterov acceleration to approximate the first moment of the Adam gradient. But the acceleration does not use any extrapolated point gradients. The Nadam approach indeed simplifies the acceleration process in the Adam method by focusing on estimating the first moment of the gradient. The acceleration [19] implemented in Nadam does not consider the gradients of the extrapolated points, which may reduce the effectiveness of the algorithm in adapting the learning steps. Additionally, theoretical deficiencies in guaranteeing convergence can lead to uncertainty in the performance and stability of algorithms across different modeling conditions and datasets.

L. Methodology

A clear and well-documented research methodology is essential for ensuring the reliability, validity, and reproducibility of a study. By presenting a systematic approach, this research provides strong theoretical and practical contributions that can support further academic development and practical implementation in the field of intrusion detection systems. A transparent methodology also increases confidence in the findings and strengthens the conclusions drawn from the analysis.

The overall method used in this study is illustrated in Fig. 2, which outlines the sequential stages of the research process.

This study begins by using the UNSW-NB15 dataset as the primary input for the intrusion detection system. A data preprocessing phase is then conducted to ensure data quality and integrity. This includes cleaning, normalizing, and preparing the dataset for subsequent modeling stages. After preprocessing, a Deep Neural Network (DNN) is employed to analyze the processed data. DNNs are widely recognized for their capability to learn complex and nonlinear patterns, making them highly effective for tasks related to network security and intrusion detection.

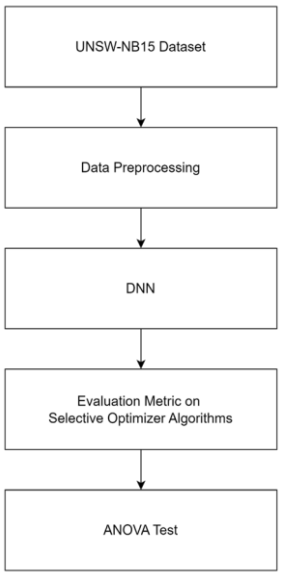

Fig. 2. Research methodology.

Following the implementation of the DNN model, several optimization algorithms are evaluated to assess their impact on training performance and detection accuracy. Evaluation metrics are used to measure each optimizer's effectiveness in improving model learning outcomes. Finally, ANOVA analysis is performed to determine whether significant differences exist among the optimization algorithms tested. This statistical approach [31] helps identify which optimizers perform better and provides insights into their suitability for intrusion detection applications.

1) Data preprocessing: Data preprocessing was carried out to ensure the quality and integrity of the dataset used in this analysis. This process involves removing empty or incomplete entries because missing values may interfere with the analytical process and reduce the accuracy of the results. Irrelevant or unnecessary data was also eliminated to simplify the dataset and maintain analytical focus. Additionally, duplicate entries were identified and removed to prevent distortion and ensure that every instance in the dataset is unique and relevant to the study.

Next, one-hot encoding was applied to categorical features to convert them into binary representations suitable for the model. This step is essential to ensure that the model can correctly interpret and process categorical variables.

Fig. 3. Step-by-step data preprocessing.

Fig. 3 illustrates the sequential steps of the data preprocessing stage. Data normalization was then applied to adjust the scale of numerical features, so they fall within a uniform range. This helps to improve model convergence and prevents certain features from dominating the learning process due to differences in scale. After normalization, the dataset was

divided into three subsets: training data, testing data, and validation data. This partitioning is crucial for evaluating model performance on unseen data and ensuring that the results are validated objectively.

By conducting this comprehensive preprocessing procedure, this study ensures that the data used in the analysis is clean, relevant, and properly prepared to support effective model development and performance evaluation.

2) DNN: Deep Neural Networks (DNN) are utilized in this research to analyze the UNSW-NB15 dataset. A DNN is a neural network architecture capable of learning increasingly complex feature representations through multiple layers of interconnected neurons. These layers enable the model to capture hierarchical patterns in the data, allowing it to understand both low-level and high-level characteristics that are essential for intrusion detection.

A DNN consists of sequential processing layers, starting from the input layer and continuing through multiple hidden layers before reaching the output layer. Each neuron in a hidden layer receives inputs from neurons in the previous layer, applies a weight to each input, and computes a weighted sum followed by an activation function. Through this multilayer transformation, the network gradually converts raw input data into more abstract representations. As information flows through deeper layers, the DNN becomes capable of recognizing complex attack patterns, subtle anomalies, and relationships that are not easily detected through traditional analytical methods.

By using DNN, this study can identify intricate attack behaviors and subtypes that may not be captured by conventional machine learning models. DNN architecture also supports adaptability by enabling the model to adjust to evolving intrusion patterns over time. This contributes to improving detection accuracy and reducing false positives, thereby enhancing the overall performance of the IDS.

The DNN architecture used in this research is presented in Table III, which outlines the model's hyperparameters, including the number of hidden layers, number of neurons, activation functions, number of epochs, and batch size. These settings form the basis for training the model and evaluating the performance of the optimization algorithms examined in this study.

TABLE III. DNN ARCHITECTURE

Hyperparameter	Value/ Type
Hidden Layers	3
Neurons	100
Hidden Layer Activation	Relu
Output Layer Activation	Softmax
Epochs	10
Batch Size	100

3) Evaluation metric on selective optimizer algorithms: The metric evaluation stage in Selective Optimizer Algorithms assesses the performance of several optimization algorithms in the Intrusion Detection System (IDS). The evaluation metrics used in this research include accuracy and loss. This metric helps evaluate how well the optimization algorithm can learn complex patterns in IDS data and make accurate predictions. So you can identify the most suitable and effective optimization algorithm for use on IDS. Choosing the right optimization algorithm can improve the performance and reliability of the IDS in detecting network security threats in a timely and accurate manner. Therefore, this evaluation plays an important role in developing a more effective and responsive network security system.

4) ANOVA test: Analysis of Variance (ANOVA) [30] is a statistical technique used to compare data between three or more different groups. ANOVA considers between-group variation and within-group variation. Between-group variation measures how large the differences in the means are between different groups, while within-group variation measures how large the variation in the values within each group. If the variation between groups is much greater than the variation within groups, then there is an indication that there are significant differences between the groups. ANOVA can be used to analyze data performance from various optimizer algorithms. For example, if there are eight optimizer algorithms being evaluated, ANOVA can be used to determine whether there are significant differences in overall performance between the algorithms. This research will analyze significant differences in performance between the optimized optimizer algorithms tested.

IV. RESULTS AND DISCUSSION

A. SGD Result

Stochastic Gradient Descent (SGD) is one of the most widely used optimization techniques for training machine learning models, particularly in artificial neural networks and Deep Neural Networks (DNN) [20]. SGD optimizes the objective function by following the gradient direction with respect to the model parameters. Its primary goal is to minimize the prediction error by iteratively updating the parameters based on the gradient of the loss function computed from small batches of training data.

In this research, the SGD optimizer is applied to train the DNN model using the UNSW-NB15 dataset. Fig. 4 presents the training and validation accuracy obtained during the model training process, while Fig. 5 shows the corresponding training and validation loss curves.

The results indicate that both training and validation accuracy generally increase with each epoch, demonstrating that the model progressively improves its ability to correctly classify intrusion data. Meanwhile, the loss curves in Fig. 5 show a consistent decrease across epochs for both training and validation datasets. A decreasing loss value reflects better model adaptation to the data because the loss function measures the discrepancy between predicted outputs and actual labels.

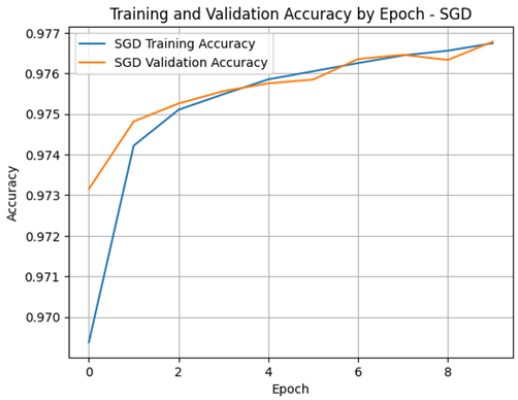

Fig. 4. Training and validation accuracy for SGD.

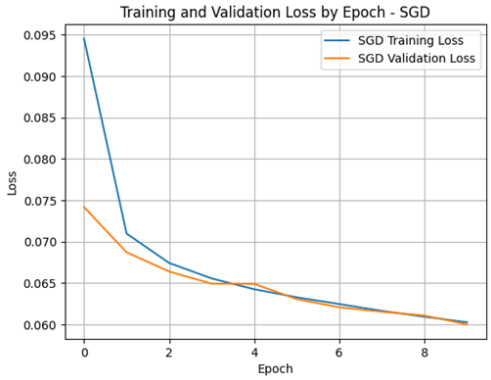

Fig. 5. Training and validation loss for SGD.

This gradual decline in loss indicates effective convergence, meaning the model is successfully refining its parameters to reduce prediction errors. The behavior of the loss curves also suggests that SGD is performing efficiently in updating parameters based on the gradients calculated from mini-batches of data. This characteristic enables faster updates and supports the model in learning more generalizable patterns from the dataset.

B. AdaGrad Result

The AdaGrad algorithm adaptively updates the learning rate to accelerate convergence without requiring manual tuning efforts [21]. This algorithm automatically adjusts the learning rate during the training process, enabling each parameter to be updated based on its historical gradient information. As the model approaches an optimal solution, AdaGrad dynamically modifies the learning rate to ensure efficient parameter optimization.

Fig. 6 presents the training and validation accuracy for AdaGrad, showing how the model's performance improves over successive epochs. AdaGrad enables faster learning in the early stages by assigning larger updates to infrequent features, resulting in a noticeable improvement in accuracy.

Fig. 7 illustrates the training and validation loss for AdaGrad, which consistently decreases as the number of epochs increases. A decreasing loss value indicates that the model is learning effectively and reducing prediction errors over time.

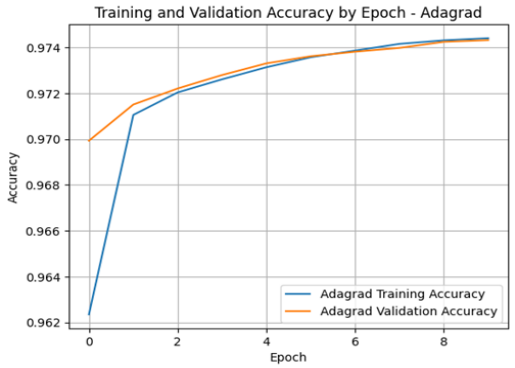

Fig. 6. Training and validation accuracy for AdaGrad.

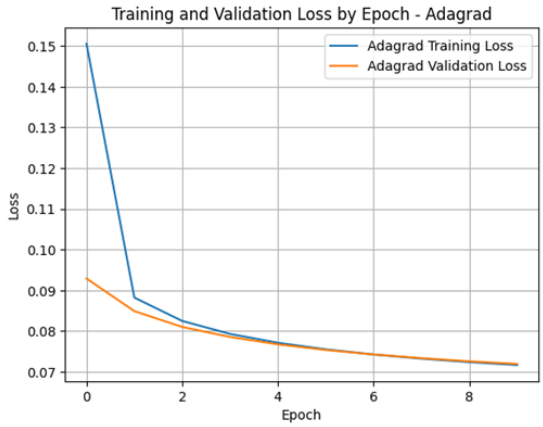

Fig. 7. Training and validation loss for AdaGrad.

The graphs of training and validation accuracy and loss provide a clear representation of how the model evolves across iterations. The increase in accuracy and the decrease in loss demonstrate that the model progressively improves and moves closer to an optimal solution. This behavior highlights AdaGrad's ability to adjust the learning rate dynamically for each parameter, thereby enhancing convergence and improving performance during the training process.

C. AdaDelta Result

Adadelta makes smaller adjustments to frequently updated parameters and larger adjustments to parameters that are updated less frequently [22]. During the model training process, Adadelta adaptively regulates the magnitude of parameter updates based on how often each parameter changes. When a parameter is updated frequently, Adadelta applies smaller adjustments to maintain model stability, as minor refinements are typically sufficient to improve performance.

Fig. 8 shows the training and validation accuracy obtained using the Adadelta optimizer. As illustrated in Fig. 8, the accuracy for both training and validation increases consistently across epochs, indicating stable and sustainable convergence. This demonstrates that the Adadelta algorithm effectively guides the model toward improved accuracy by adaptively optimizing the learning rate based on parameter-specific characteristics.

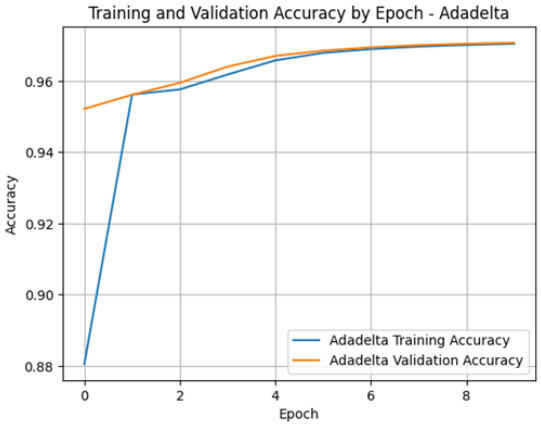

Fig. 8. Training and validation accuracy for AdaDelta.

Fig. 9 presents the training and validation loss for the Adadelta optimizer. The consistent decline in both training and validation loss indicates that the model continues to learn effectively over time. The stable reduction in loss confirms that Adadelta successfully adjusts the learning rate and parameter updates, enabling efficient convergence. This pattern reflects the model's increasing ability to learn from data and move closer to an optimal solution.

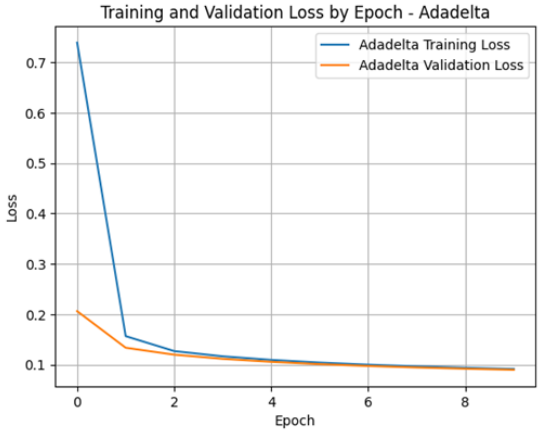

Fig. 9. Training and validation loss for AdaDelta.

D. RMSProp Result

RMSProp is commonly used in training artificial neural network models to accelerate convergence and improve overall training performance by adaptively regulating the learning rate. In RMSProp, the learning rate is adjusted individually for each model parameter based on the exponential moving average of previous squared gradients [23]. This mechanism addresses the limitations of a fixed learning rate by automatically adapting the step size throughout the training process.

Fig. 10 illustrates the training and validation accuracy produced by RMSProp. The graph shows that both training and validation accuracy fluctuate across epochs. Such instability may occur when the learning rate is excessively high, causing divergence, or too low, resulting in very slow convergence. Additionally, complex model architecture or many parameters can contribute to inconsistent learning behavior, as the model may struggle to capture stable patterns within the dataset.

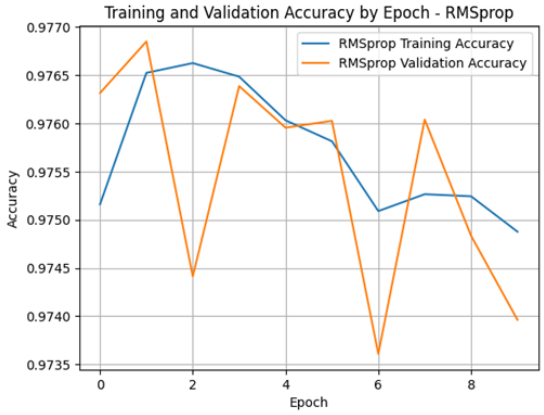

Fig. 10. Training and validation accuracy for RMSProp.

Fig. 11 presents the training and validation loss curves for RMSProp. The results indicate that the training and validation loss increase steadily across epochs, while the accuracy values fluctuate. This pattern suggests that the model is experiencing overfitting. Overfitting occurs when the model becomes too closely aligned with the training data and loses its ability to generalize to unseen data, such as validation samples. One of the common causes of overfitting is excessive model complexity, particularly in deep neural network architectures with large capacity [24].

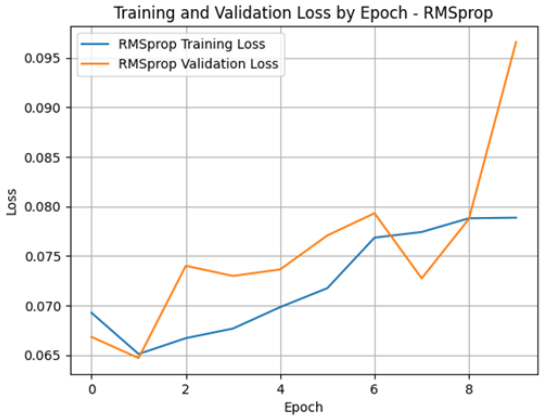

Fig. 11. Training and validation loss for RMSProp.

E. Adam Result

Adam (Adaptive Moment Estimation) is an optimization algorithm designed to combine the advantages of momentum methods and adaptive learning rate techniques [25]. Adam utilizes the accumulated gradient values from previous iterations, weighted by a decay factor, which represents the principle of momentum. Momentum provides acceleration during optimization by considering both the direction and magnitude of parameter changes from earlier iterations. By incorporating historical gradient information, the model can move more efficiently toward the optimal solution.

Additionally, Adam uses the squared gradient from previous updates to estimate variance or fluctuations in the gradient. This mechanism enables Adam to adjust the learning rate adaptively for each parameter. As a result, parameters with high variance receive smaller updates, while those with low

variance receive larger updates. This balance helps stabilize the learning process and prevents drastic fluctuations in parameter adjustments.

Adam uses the first-order derivative of the cost function to update parameters, providing low computational complexity and making it highly efficient. By combining momentum-based gradient estimation with root mean square (RMS) gradient estimation, Adam generates parameter updates with an automatically adjusted learning rate [25]. This adaptive adjustment ensures that parameter updates remain controlled and appropriate throughout the training process.

Fig. 12 shows the training and validation accuracy produced for Adam. The results indicate stable accuracy improvement across epochs during both training and validation. This demonstrates Adam's ability to adaptively modify the learning rate based on gradient information and previous parameter updates, ensuring smooth optimization progress.

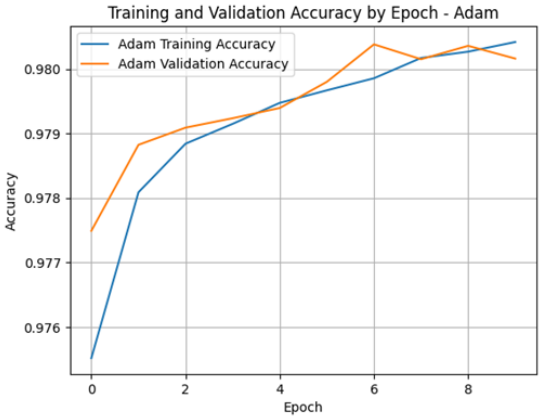

Fig. 12. Training and validation accuracy for Adam.

Fig. 13 presents the training and validation loss for Adam. Both curves show a consistent reduction in loss, indicating effective learning and minimal fluctuations during the optimization process. Adam gradually reduces the learning rate as the model approaches the global minimum, preventing oscillations near the minimum and enabling more precise convergence. This behavior contributes to the stable accuracy gains and steady declines in loss across epochs.

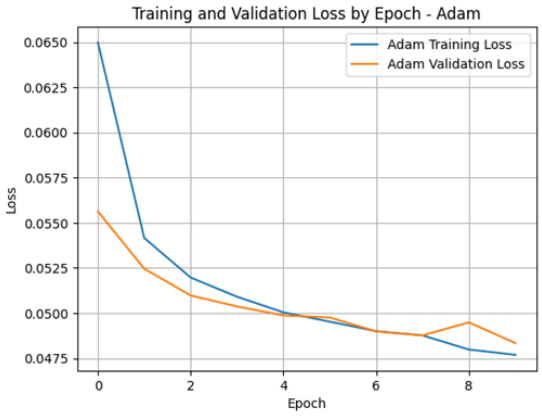

Fig. 13. Training and validation loss for Adam.

By leveraging both momentum and RMS gradient estimates, Adam produces more consistent and reliable parameter updates compared to many traditional optimizers. This results in faster and more stable optimization, making Adam particularly effective for training deep neural networks. The performance observed in this study confirms Adam's capability to handle complex optimization tasks efficiently, as reflected in the stable improvement of accuracy and consistent reduction of loss throughout the training and validation stages.

F. Adamax Result

The performance of the Adamax optimizer is considered strong in numerical function optimization compared to several other optimization algorithms [16]. Adamax is a variant of the Adam (Adaptive Moment Estimation) optimization method, designed to provide more stable parameter updates, especially when the magnitude of the gradient becomes very large. This characteristic enables Adamax to maintain steady convergence during the training of deep learning models.

Fig. 14 shows the training and validation accuracy for the Adamax optimizer. Adamax demonstrates good performance compared to Adam, particularly in situations where the gradient moments are large, and Adam begins to show deviations from optimal updates. The Adamax method utilizes two core components, namely the gradient moment and the root mean square (rms) value of the gradient, to determine parameter adjustments. By incorporating both components, Adamax can generate more adaptive and stable parameter updates, which contribute to faster convergence in complex numerical optimization problems.

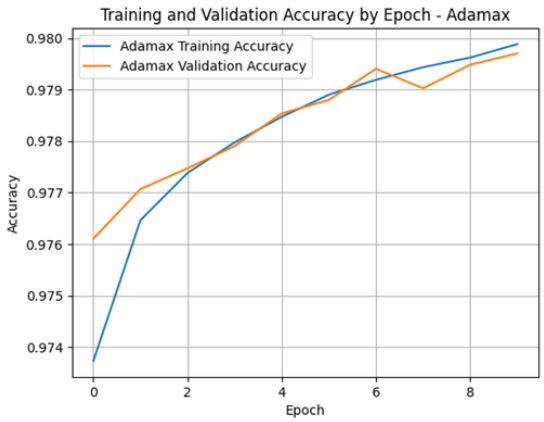

Fig. 14. Training and validation accuracy for Adamax.

However, in this study, the results indicate that Adam performs better than Adamax on the UNSW-NB15 dataset using DNN. This performance difference may occur because each dataset possesses unique distributions and structural patterns that influence model learning behavior. Additionally, the architecture of the DNN model employed in the experiment can also affect the relative performance of Adam and Adamax.

Fig. 15 presents the training and validation loss for the Adamax optimizer.

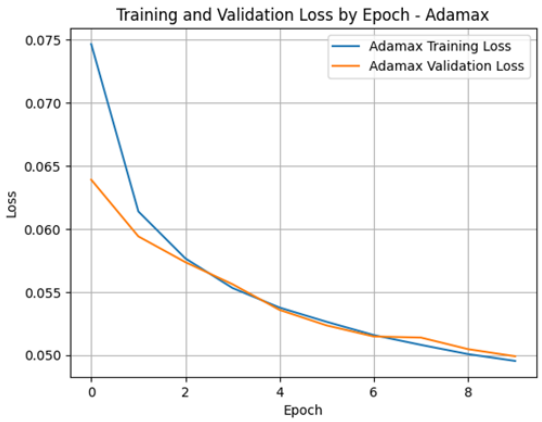

Fig. 15. Training and validation loss for Adamax.

G. AdaFactor Result

The Adafactor optimization algorithm was developed to reduce the memory requirements of the AdaGrad method, particularly for training large language models [26]. Adafactor can operate efficiently by significantly minimizing additional memory usage without sacrificing performance. This is achieved by storing second-moment estimates using a compressed representation that exploits the structural properties of the parameter tensors. As a result, Adafactor enables the training of large-scale neural networks while alleviating memory constraints, which often become a limiting factor in complex model development. By maintaining strong optimization performance while reducing memory overhead, Adafactor offers a more resource-efficient solution in artificial intelligence research and large-scale model training.

Fig. 16 presents the training and validation accuracy for Adafactor. The graph shows that accuracy increases steadily across epochs for both training and validation datasets. This consistent improvement indicates that Adafactor can adjust learning rates adaptively and perform stable parameter updates.

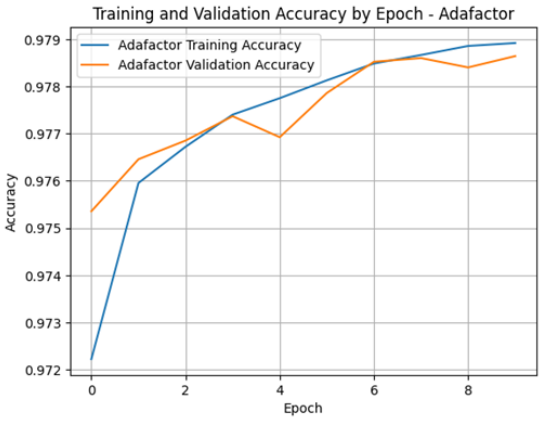

Fig. 16. Training and validation accuracy for AdaFactor.

Fig. 17 illustrates the training and validation loss for Adafactor. The figure shows a continuous and consistent decline in the loss function as training progresses. This stable reduction in loss demonstrates that the algorithm effectively manages the learning rate and updates model parameters efficiently throughout the training process. The combination of increasing accuracy and decreasing loss reflects the algorithm's ability to support model convergence and maintain stability during training.

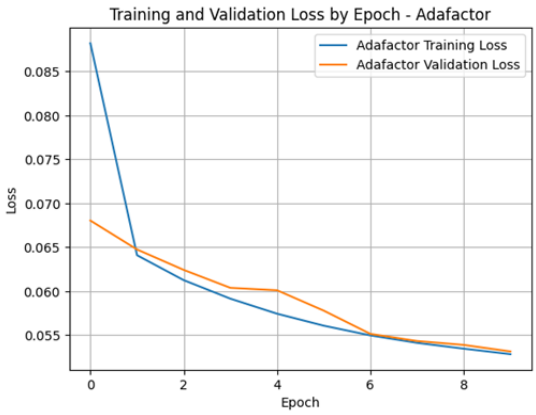

Fig. 17. Training and validation loss for AdaFactor.

H. Nadam Result

Nadam was developed to address inconsistencies that may arise between momentum calculations and parameter updates in optimization algorithms [27]. In the Adam optimization algorithm, momentum is used to incorporate gradient information from previous iterations when determining parameter updates. However, inconsistencies may occur between the calculated momentum and the actual parameter adjustments performed during optimization. Nadam resolves this issue by integrating Nesterov momentum into Adam, resulting in more consistent and targeted parameter updates. This ensures that momentum contributes more reliably to the learning process, making the optimization pathway more stable and predictable while improving convergence.

Fig. 18 illustrates the training and validation accuracy obtained using Nadam. The accuracy curves show a generally stable upward trend during the training and validation phases, indicating that Nadam effectively improves the model's ability to learn patterns from the UNSW-NB15 dataset. The adaptive mechanism in Nadam allows the algorithm to fine-tune parameter updates more carefully, contributing to improved accuracy across epochs.

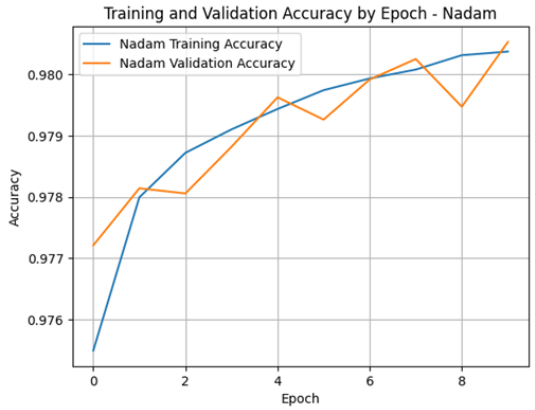

Fig. 18. Training and validation accuracy for Nadam.

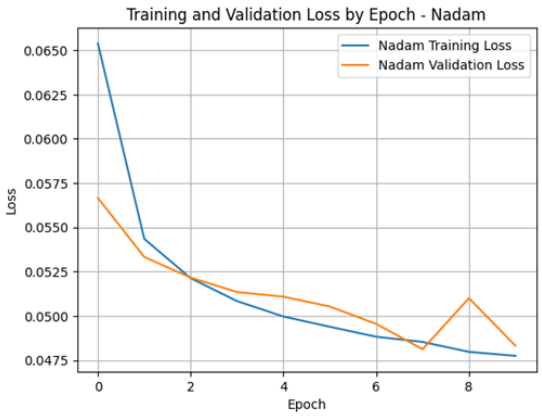

Fig. 19. Training and validation loss for Nadam.

Fig. 19 presents the training and validation loss for Nadam. The results show a steady reduction in both training and validation loss values across epochs. This consistent decline suggests that the model continues to refine its predictions and minimize error during the learning process. The stability of the loss curves indicates that Nadam efficiently adjusts the learning rate and updates model parameters with a controlled trajectory, reducing the likelihood of sudden fluctuations.

An important advantage of Nadam is its ability to produce consistent parameter updates while reducing the risk of overshooting, a condition in which large parameter changes cause the optimization process to move past the optimal point. By ensuring that updates are more controlled and directed, Nadam minimizes this risk. This contributes to a more reliable training process, leading to improved model stability and convergence. As a result, Nadam provides a robust alternative to Adam, particularly in cases where more precise parameter adjustments are needed.

I. Comparison of Optimizer Algorithms

In this research, a comparison was conducted among several commonly used optimization algorithms, including SGD (Stochastic Gradient Descent), RMSprop, Adam, Adadelta, Adagrad, Adamax, Adafactor, and Nadam. The objective of this study is to evaluate the relative performance of each algorithm in the context of model training, with particular focus on accuracy, stability, and convergence behavior.

The results indicate that Adam provides the best overall performance. Its adaptive learning rate mechanism, which adjusts based on gradient information, enables faster convergence and consistently higher accuracy across epochs. Nadam is identified as the second-best performer due to its ability to address some limitations found in Adam, especially regarding momentum estimation and parameter update consistency.

Fig. 20 presents the comparison of all optimization algorithms. Adadelta shows noticeably lower performance compared to the other algorithms. This performance degradation is partly attributed to its more complex hyperparameters, such as decay rate and epsilon, which require precise tuning. Incorrect hyperparameter settings may hamper its convergence speed and overall effectiveness, especially on complex datasets or architectures. Because Adadelta updates the learning rate based on accumulated past gradients, the learning process may become too slow or insufficiently responsive.

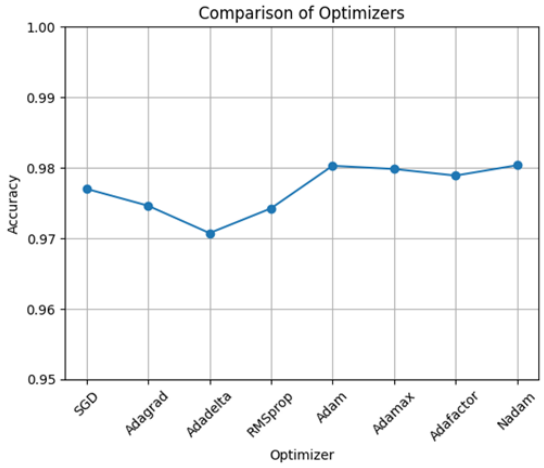

Fig. 20. Comparison of optimizer algorithms.

Fig. 21 illustrates the comparison of training accuracy. Across all algorithms, training accuracy improves significantly at each epoch. This consistent improvement demonstrates that the model continuously learns and generalizes the patterns present in the training data. The steady increase in accuracy suggests that none of the models exhibit symptoms of severe overfitting, as the learning process continues to progress in a stable manner without abrupt fluctuations. This also indicates that the adjustments made by each optimizer during training enable the network to better predict the correct labels from the training set.

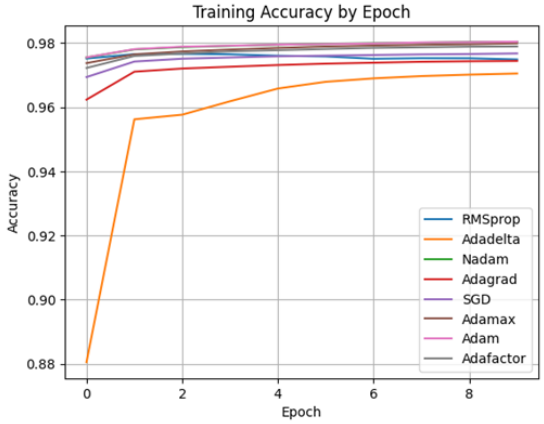

Fig. 21. Comparison of training accuracy.

Fig. 22 shows the comparison of validation accuracy. Validation accuracy also improves across epochs for all algorithms except RMSprop. The inconsistent behavior of RMSprop indicates difficulty in adjusting the learning rate effectively when processing validation gradients. This may be due to its sensitivity to specific hyperparameter values, such as learning rate and decay rate. Improper settings can lead to unstable or slow convergence on the validation dataset.

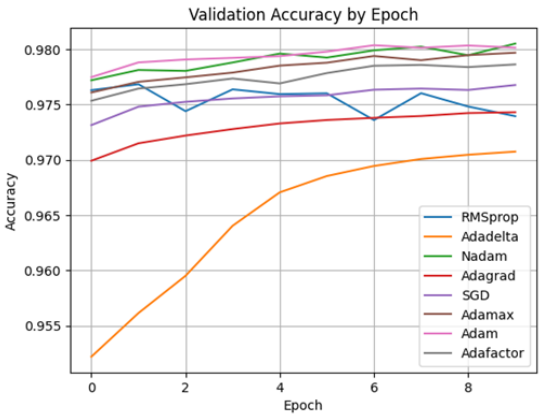

Fig. 22. Comparison of validation accuracy.

Fig. 23 depicts the comparison of training loss. All algorithms experience a steady decline in training loss at each epoch, demonstrating continual improvement in reducing prediction error during the learning process. A consistent decrease in training loss is a strong indication that the model is progressing well and learning meaningful representations from the data.

Fig. 24 presents the comparison of validation loss across all optimizers. The validation loss decreases steadily across all algorithms except RMSprop. This decline indicates improved generalization capability to unseen data. RMSprop's irregular behavior reinforces the earlier observation that this optimizer may struggle with gradient-based adjustments on validation data, especially under suboptimal hyperparameter configurations.

The performance accuracy values for each optimizer used in this research are summarized in Table IV.

J. ANOVA Test

One method to compare classifiers is by using the Analysis of Variance (ANOVA) test to evaluate the accuracy produced by each classifier. ANOVA determines whether there are statistically significant differences between the accuracies generated by the classifiers tested [28]. In this study, ANOVA is used to evaluate whether there are significant performance differences among the optimization algorithms applied to a DNN-based classifier. ANOVA assesses the equality of multiple means by comparing the variance between groups with the variance within groups [29].

Statistical hypothesis testing was conducted to determine whether significant differences exist among the optimization algorithms. ANOVA evaluates how the variation between groups compares with the variation within groups. A statistically significant result, indicated by a p-value lower than the predetermined significance level, supports the decision to reject the null hypothesis. The hypotheses used in this research are presented in Table V.

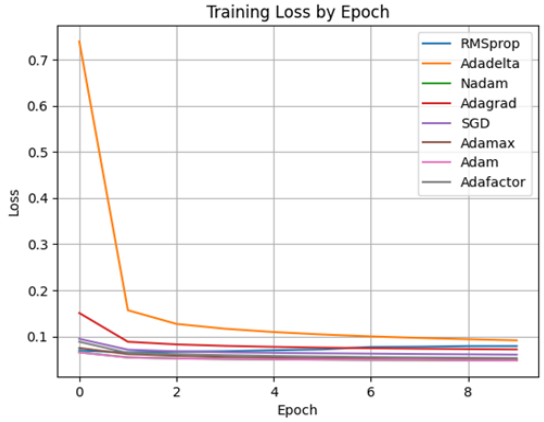

Fig. 23. Comparison of training loss.

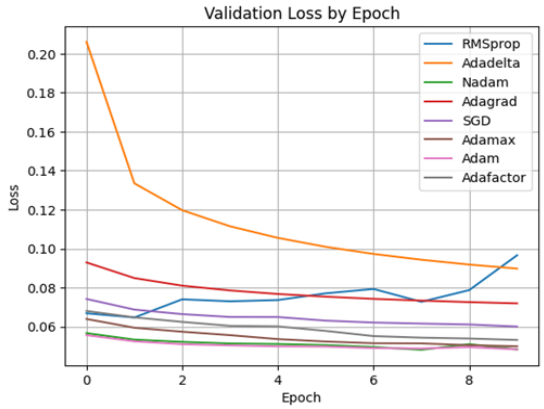

Fig. 24. Comparison of validation loss.

TABLE IV. VALIDATION ACCURACY OF OPTIMIZER ALGORITHMS

Epoch	SGD	RMSProp	Adam	AdaDelta	AdaGrad	Adamax	AdaFactor	Nadam
1	0.976316	0.952194	0.97721	0.969927	0.973152	0.976105	0.977492	0.975356
2	0.97685	0.956142	0.97814	0.971505	0.974818	0.977068	0.978824	0.976458
3	0.974413	0.959513	0.978056	0.972206	0.975259	0.977473	0.97909	0.976854
4	0.976387	0.964044	0.978818	0.972792	0.975564	0.977904	0.979236	0.977369
5	0.975956	0.967068	0.979625	0.973301	0.975755	0.978536	0.979395	0.976925
6	0.976027	0.968543	0.979262	0.973612	0.975849	0.978798	0.9798	0.977862
7	0.97369	0.969454	0.979917	0.97381	0.976354	0.979405	0.98038	0.978523
8	0.97604	0.970086	0.980251	0.973975	0.976458	0.979025	0.98015	0.978601
9	0.974834	0.970468	0.979473	0.974241	0.976332	0.979482	0.980358	0.978403
10	0.973962	0.970753	0.980529	0.974316	0.976779	0.979703	0.98016	0.978643

TABLE V. RESEARCH HYPOTHESIS

Hypothesis	Description
H0	There is no significant difference in performance between the tested optimizer algorithms in detecting intrusions on systems using DNN.
H1	There is a significant difference in performance between the tested optimizer algorithms in detecting intrusions on systems using DNN.

Table V summarizes the null hypothesis (H0) and the alternative hypothesis (H1). The null hypothesis states that there is no significant difference in the performance of the tested optimizers in detecting intrusions using DNN. The alternative hypothesis states that a significant difference exists. These hypotheses guide the statistical analysis to identify whether the collected data support the null or the alternative hypothesis. If the results show significant differences between groups, the null hypothesis is rejected.

After performing ANOVA on the optimizer performance data, based on the Validation Accuracy of Optimizer Algorithms, the results were obtained in the form of F-statistic and p-values. If the p-value is smaller than the significance level, typically $\alpha = 0.05$, the null hypothesis is rejected. This means that at least two optimization algorithms differ significantly in performance. This pattern indicates that choosing the appropriate optimizer can have a meaningful impact on the performance of an intrusion detection system. If the p-value is greater than α, the null hypothesis cannot be rejected, meaning the evidence is insufficient to confirm a significant difference. This lack of significance may be due to small sample sizes or low variability in the data.

A high F-statistic value in an ANOVA test indicates that the variation between group means is greater than the variation within the groups. A higher F-value reflects real performance differences among the optimization algorithms tested. This is important when choosing an optimal algorithm for intrusion detection, as performance differences may influence the effectiveness of the system. High F-values also highlight the need for further analysis, such as pairwise comparisons, to better understand which algorithms differ and to what extent. This information can support decision-making in future research and system implementation.

The results of the ANOVA test conducted on the UNSW-NB15 dataset using the DNN model are presented in Table VI. This test evaluates whether significant performance differences exist among the optimization algorithms.

Based on Table VI, the ANOVA analysis produced an F value of 34.68717049, while the F-critical value calculated at a significance level of $\alpha = 0.05$ was 2.139655512. Since the F value exceeds the F-critical value, the null hypothesis is rejected. This confirms that significant performance differences exist among the tested optimization algorithms. These results support the alternative hypothesis, which states that the optimization algorithms differ significantly in detecting intrusions using DNN. The implication is that selecting the appropriate optimization algorithm can substantially enhance the performance of intrusion detection systems.

TABLE VI. RESULT OF ANOVA TEST

ANOVA: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance
RMSprop	10	9,754394	0.9754394	1,2869E-06
Adadelta	10	9,648265	0.9648265	4,4301E-05
Nadam	10	9,791281	0.9791281	1,12562E-06
Adagrad	10	9,729685	0.9729685	1,9651E-06
SGD	10	9,75632	0.975632	1,11424E-06
Adamax	10	9,783499	0.9783499	1,3964E-06
Adam	10	9,794885	0.9794885	7,99467E-07
Adafactor	10	9,774994	0.9774994	1,21769E-06

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.001614882	7	0.000230697	34,68717049	1,20649E-20	2,139655512
Within Groups	0.000478858	72	6,6508E-06			
Total	0.00209374	79				

V. CONCLUSIONS AND LIMITATIONS

Based on the experimental findings, the Adam optimizer demonstrated the best overall performance on the UNSW-NB15 dataset when applied to a Deep Neural Network-based Intrusion Detection System. Adam consistently produced stable improvements in accuracy and steady reductions in loss across epochs, indicating its effectiveness in accelerating convergence and maintaining model stability. Nadam ranked as the second-best performer, while Adadelta yielded the lowest accuracy among the evaluated algorithms. RMSprop exhibited noticeable fluctuations in validation accuracy and validation

loss, suggesting that it may require more careful hyperparameter tuning or may be less suitable for certain IDS configurations.

The results from the ANOVA analysis provide strong statistical validation of these observations. The obtained F-statistic value of 34.687, which exceeds the critical value of 2.139 at $\alpha = 0.05$, confirms that significant performance differences exist across the eight optimization algorithms. These findings highlight the importance of optimizer selection as a critical determinant of the overall performance and reliability of DNN-based intrusion detection systems.

While Adam and Nadam emerged as promising candidates for IDS implementation, several limitations must be considered. This study relied on a single dataset and did not incorporate cross-validation or extensive hyperparameter optimization, which may influence generalizability. Additionally, the evaluation focused primarily on accuracy and loss, without including broader IDS performance metrics such as precision, recall, F1-score, false positive rate, or AUC. Incorporating these metrics would provide a more complete assessment of the practical effectiveness of each optimization method in real-world intrusion detection scenarios.

To advance future research, several directions are proposed. Expanding the experimental framework to include multiple datasets with different traffic and attack characteristics would strengthen generalizability. More comprehensive hyperparameter optimization and the inclusion of additional performance metrics would support more rigorous evaluation and facilitate practical deployment. Further exploration of optimizer behavior across different deep learning architectures, operational environments, and adversarial conditions may also offer deeper insight into the stability, robustness, and adaptability of each algorithm. Overall, this study contributes to the growing body of knowledge on optimization strategies for IDS and underscores the critical role of optimizer selection in improving cybersecurity resilience.

REFERENCES

[1] J. Sinha and M. Manollas, "Efficient Deep CNN-BiLSTM Model for Network Intrusion Detection," in Proceedings of the 2020 3rd International Conference on Artificial Intelligence and Pattern Recognition, New York, NY, USA: ACM, Jun. 2020. pp. 223–231. doi: 10.1145/3430199.3430224.

[2] M. Rodríguez, Á. Alesanco, L. Mehavilla, and J. García, "Evaluation of Machine Learning Techniques for Traffic Flow-Based Intrusion Detection," Sensors, vol. 22, no. 23, p. 9326, Nov. 2022, doi: 10.3390/s22239326.

[3] A. Aleesa, M. Thanoun, A. Mohammed, and N. Sahar, "DEEP-INTRUSION DETECTION SYSTEM WITH ENHANCED UNSW-NB15 DATASET BASED ON DEEP LEARNING TECHNIQUES," Journal of Engineering Science and Technology, vol. 16, pp. 711–727, Mar. 2021.

[4] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in 2015 Military Communications and Information Systems Conference (MilCIS), IEEE, Nov. 2015, pp. 1–6. doi: 10.1109/MilCIS.2015.7348942.

[5] E. M. Dogo, O. J. Afolabi, and B. Twala, "On the Relative Impact of Optimizers on Convolutional Neural Networks with Varying Depth and Width for Image Classification," Applied Sciences, vol. 12, no. 23, p. 11976, Nov. 2022, doi: 10.3390/app122311976.

[6] D. Soydaner, "A Comparison of Optimization Algorithms for Deep Learning," Intern J Pattern Recognit Artif Intell, vol. 34, no. 13, p. 2052013, Dec. 2020. doi: 10.1142/S0218001420520138.

[7] M. A. Albahar and M. Binsawad, "Deep Autoencoders and Feedforward Networks Based on a New Regularization for Anomaly Detection," Security and Communication Networks, vol. 2020. pp. 1–9, Jul. 2020. doi: 10.1155/2020/7086367.

[8] S. P. Thirimanne, L. Jayawardana, L. Yasakethu, P. Liyanaarachchi, and C. Hewage, "Deep Neural Network Based Real-Time Intrusion Detection System," SN Comput Sci, vol. 3, no. 2, p. 145, Mar. 2022, doi: 10.1007/s42979-022-01031-1.

[9] A. M. Aleesa, B. B. Zaidan, A. A. Zaidan, and N. M. Sahar, "Review of intrusion detection systems based on deep learning techniques: coherent taxonomy, challenges, motivations, recommendations, substantial analysis and future directions," Neural Comput Appl, vol. 32, no. 14, pp. 9827–9858, Jul. 2020. doi: 10.1007/s00521-019-04557-3.

[10] M. Merenda, C. Porcaro, and D. Iero, "Edge Machine Learning for AI-Enabled IoT Devices: A Review," Sensors, vol. 20. no. 9, p. 2533, Apr. 2020. doi: 10.3390/s20092533.

[11] H. Han, H. Kim, and Y. Kim, "Correlation between Deep Neural Network Hidden Layer and Intrusion Detection Performance in IoT Intrusion Detection System," Symmetry (Basel), vol. 14, no. 10. p. 2077, Oct. 2022, doi: 10.3390/sym14102077.

[12] H. Robbins and S. Monro, "A Stochastic Approximation Method," The Annals of Mathematical Statistics, vol. 22, no. 3, pp. 400–407, Sep. 1951, doi: 10.1214/aoms/1177729586.

[13] Y. Tian, Y. Zhang, and H. Zhang, "Recent Advances in Stochastic Gradient Descent in Deep Learning," Mathematics, vol. 11, no. 3, p. 682, Jan. 2023, doi: 10.3390/math11030682.

[14] S. Chaudhury and T. Yamasaki, "Robustness of Adaptive Neural Network Optimization Under Training Noise," IEEE Access, vol. 9, pp. 37039–37053, 2021, doi: 10.1109/ACCESS.2021.3062990.

[15] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A Survey of Optimization Methods From a Machine Learning Perspective," IEEE Trans Cybern, vol. 50. no. 8, pp. 3668–3681, Aug. 2020. doi: 10.1109/TCYB.2019.2950779.

[16] M. Reyad, A. M. Sarhan, and M. Arafa, "A modified Adam algorithm for deep neural network optimization," Neural Comput Appl, vol. 35, no. 23, pp. 17095–17112, Aug. 2023, doi: 10.1007/s00521-023-08568-z.

[17] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Dec. 2014.

[18] E. Hassan, M. Y. Shams, N. A. Hikal, and S. Elmougy, "The effect of choosing optimizer algorithms to improve computer vision tasks: a comparative study," Multimed Tools Appl, vol. 82, no. 11, pp. 16591–16633, May 2023, doi: 10.1007/s11042-022-13820-0.

[19] N. Shazeer and M. Stern, "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost." 2018.

[20] X. Xie, P. Zhou, H. Li, Z. Lin, and S. Yan, "Adan: Adaptive Nesterov Momentum Algorithm for Faster Optimizing Deep Models." 2023.

[21] T. Dozat, "Incorporating Nesterov Momentum into Adam," 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:70293087

[22] F. Lopez, E. Chow, S. Tomov, and J. Dongarra, "Asynchronous SGD for DNN training on Shared-memory Parallel Architectures," in 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, May 2020. pp. 1–4. doi: 10.1109/IPDPSW50202.2020.00168.

[23] Y. Wang, J. Liu, J. Misic, V. B. Misic, S. Lv, and X. Chang, "Assessing Optimizer Impact on DNN Model Sensitivity to Adversarial Examples," IEEE Access, vol. 7, pp. 152766–152776, 2019, doi: 10.1109/ACCESS.2019.2948658.

[24] R. Elshamy, O. Abu-Elnasr, M. Elhoseny, and S. Elmougy, "Improving the efficiency of RMSProp optimizer by utilizing Nestrove in deep learning," Sci Rep, vol. 13, no. 1, p. 8814, May 2023, doi: 10.1038/s41598-023-35663-x.

[25] Z.-Y. Zhang et al., "Towards Understanding the Overfitting Phenomenon of Deep Click-Through Rate Models," in Proceedings of the 31st ACM International Conference on Information & Knowledge

Management, New York, NY, USA: ACM, Oct. 2022, pp. 2671–2680. doi: 10.1145/3511808.3557479.

[26] D. Yi, J. Ahn, and S. Ji, "An Effective Optimization Method for Machine Learning Based on ADAM," Applied Sciences, vol. 10. no. 3, p. 1073, Feb. 2020. doi: 10.3390/app10031073.

[27] E. Hassan, M. Y. Shams, N. A. Hikal, and S. Elmougy, "The effect of choosing optimizer algorithms to improve computer vision tasks: a comparative study," Multimed Tools Appl, vol. 82, no. 11, pp. 16591–16633, May 2023, doi: 10.1007/s11042-022-13820-0.

[28] X. He, F. Xue, X. Ren, and Y. You, "Large-Scale Deep Learning Optimizations: A Comprehensive Survey," Nov. 2021.

[29] S. Ruder, "An overview of gradient descent optimization algorithms," Sep. 2016.

[30] Nurrahma and R. Yusuf, "Comparing Different Supervised Machine Learning Accuracy on Analyzing COVID-19 Data using ANOVA Test," in 2020 6th International Conference on Interactive Digital Media (ICIDM), IEEE, Dec. 2020. pp. 1–6. doi: 10.1109/ICIDM51048.2020.9339676.

[31] R. A. Fisher, "Statistical Methods for Research Workers," 1992, pp. 66–70. doi: 10.1007/978-1-4612-4380-9_6.