Network Intrusion Detection

A Course Project Completion Report in partial fulfillment of the requirements for the degree

Bachelor of Technology

in

Computer Science & Artificial Intelligence

BY

Name	**Hall Ticket**
P. LALITH PRAKASH	2303A52012
KADARI THARANI	2303A52067
RITHIMA BANALA	2303A52471

Under the guidance of

Submitted to

DR. VENKATARAMANA

SCHOOL OF COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE

SR UNIVERSITY, ANANTHASAGAR, WARANGAL

April, 2025

SCHOOL OF COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE

CERTIFICATE

This is to certify that the Project Report entitled "**Network Intrusion Detection** " is a record of Bonafide **P. LALITH PRAKASH (2303A52012)**, **THARANI KADARI (2303A52067)**, **BANALA RITHIMA (2303A52471)** in partial fulfillment of the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE,** during the academic year **2024-2025** under the guidance and supervision.

Supervisor	**Head of the Department**
Dr. Venkataramana Veeramsetty	Dr. M. Sheshikala
Professor & Associate Dean (AI)	Prof & HOD (CS&AI)
SR University	SR University

Abstract

This project focuses on developing an intelligent network traffic classification and anomaly detection system using a hybrid approach of machine learning and deep learning. The dataset, comprising 42 features, undergoes extensive preprocessing including handling missing values, encoding categorical data, and dropping non-informative features. Visual exploratory data analysis is performed to understand feature distributions and class imbalances.

Following preprocessing, the data is split into training and testing sets. Multiple classical machine learning models—Perceptron, Logistic Regression, SVM, KNN, Decision Tree, and Random Forest—are trained and evaluated using accuracy, precision, recall, and F1-score. Among these, Random Forest and Logistic Regression demonstrate strong performance due to their ability to handle complex data relationships.

An Artificial Neural Network (ANN) is also implemented, with experiments using different batch sizes (8, 16, 32) and different optimizers to assess their impact on learning dynamics. The ANN architecture uses ReLU activations, dropout regularization, and a sigmoid output, trained with binary crossentropy and the different optimizers. Model performance is monitored over epochs through accuracy and loss plots.

The results show that deep learning, particularly the ANN, outperforms traditional methods in classification accuracy and generalization. This project provides a robust and scalable framework for real-time anomaly detection, offering valuable insights for enhancing cybersecurity systems.

Introduction

In today's highly interconnected digital ecosystem, the volume, velocity, and variety of network traffic have grown at an unprecedented rate. While this digital transformation has enabled remarkable progress in communication and data exchange, it has also introduced substantial vulnerabilities within network infrastructures. Cyber-attacks—ranging from unauthorized intrusions and phishing attempts to Distributed Denial of Service (DDoS) attacks and data exfiltration—continue to threaten the confidentiality, integrity, and availability of digital assets.

Conventional Intrusion Detection Systems (IDS) are primarily signature-based or rely on manually crafted rules. Although these approaches are effective for detecting known attack patterns, they fall short when facing zero-day attacks or previously unseen threat vectors. Furthermore, the static nature of these systems limits their adaptability in the face of rapidly evolving cyber threats. These limitations have led to a growing interest in intelligent, data-driven detection mechanisms capable of learning from historical behavior and adapting dynamically to novel anomalies.

This report presents the development of an intelligent anomaly detection framework that leverages both classical machine learning (ML) techniques and deep learning (DL) architectures to classify network traffic and identify suspicious activities. The system is built around a well-structured dataset comprising 42 features that represent diverse network characteristics, including protocol types, packet statistics, and flow-level metrics. This hybrid approach aims to bridge the gap between high accuracy, computational efficiency, and scalability—key requirements for modern IDS solutions.

The project workflow begins with a comprehensive **data preprocessing phase**, which includes:

- **Cleaning and handling missing values**,
- **Label encoding of categorical features**,
- **Elimination of constant and redundant columns**, and
- **Feature scaling using normalization** techniques.

Subsequently, **exploratory data analysis (EDA)** is performed to visualize feature distributions, examine class imbalances, and better understand the underlying data structure through histograms, boxplots, density plots, and class frequency charts.

After preprocessing, the data is divided into training and testing subsets. Several **machine learning models** are implemented, including:

- **Perceptron**
- **Logistic Regression**
- **Support Vector Machine (SVM)**
- **K-Nearest Neighbors (KNN)**
- **Decision Tree**
- **Random Forest**

Each model is evaluated using performance metrics such as **accuracy**, **precision**, **recall**, **F1-score**, and **confusion matrix analysis** to determine their effectiveness in identifying anomalous traffic.

To enhance detection accuracy and model robustness, a **deep learning-based Artificial Neural Network (ANN)** is developed. The ANN comprises:

- Multiple **fully connected dense layers**,

- **ReLU activation** functions,
- **Dropout regularization** to mitigate overfitting, and
- A **sigmoid output layer** for binary-class classification.

The model is compiled using **different optimizers** and **binary crossentropy loss**, and trained with different **batch sizes** (8, 16, and 32) to evaluate the impact of mini-batch learning on convergence and generalization. The training process is closely monitored via learning curves for both accuracy and loss on training and validation sets.

The experimental results indicate that while traditional models such as Random Forest and Logistic Regression provide strong baselines, the ANN consistently outperforms them in classification accuracy and generalization ability. The batch size comparison further reveals that appropriate tuning of hyperparameters significantly influences training dynamics and final performance.

In summary, this report introduces a scalable, adaptive, and accurate anomaly detection system for network security applications. By combining the strengths of machine learning and deep learning, the proposed framework effectively addresses the challenges of real-time network traffic analysis and contributes a robust solution for the development of intelligent Intrusion Detection Systems (IDS).

Literature Survey

Artificial Neural Networks (ANNs) have emerged as one of the most powerful tools for solving complex classification problems in various domains, including network security. In the context of anomaly detection, ANNs offer the ability to model non-linear relationships, learn high-level abstract features, and generalize well to unseen patterns—attributes that are especially important when identifying subtle deviations in network behavior.

1. Application of ANNs in Network Traffic Classification

ANNs have been extensively explored for network intrusion detection due to their adaptability and scalability. These models mimic the human brain's neural structure and are capable of learning from large volumes of data. Unlike traditional methods that rely heavily on feature engineering, ANNs can automatically extract patterns from raw or minimally processed data, making them well-suited for complex network environments.

Shone et al. (2018) demonstrated that deep neural networks, when trained on comprehensive network traffic datasets, can detect a wide range of attacks with higher accuracy than classical classifiers. Their work highlighted the ANN's ability to learn discriminative features without manual intervention, significantly reducing the need for domain-specific knowledge.

2. ANN Architectures and Optimization Techniques

The effectiveness of an ANN in detecting anomalies largely depends on its architectural design and training strategy. Common configurations include:

- **Input layers** corresponding to the number of features in the dataset (e.g., 42 in this project),
- One or more **hidden layers** with ReLU activation functions to introduce non-linearity,
- **Dropout layers** to prevent overfitting,
- A **softmax output layer** for multi-class classification.

The choice of optimizer, batch size, and loss function also plays a critical role in model performance. The Adam optimizer, known for its adaptive learning rate properties, has become a standard choice for training deep networks on classification tasks. Batch size, in particular, has been shown to influence the learning stability and convergence speed of ANNs. Research by Masters and Luschi (2018) observed that smaller batch sizes can lead to better generalization at the cost of slower convergence, a trade-off that is examined in this project through batch size variation (8, 16, 32).

3. Evaluation Metrics in ANN-Based Intrusion Detection

To evaluate the effectiveness of ANNs in anomaly detection, researchers commonly rely on a suite of classification metrics:

- **Accuracy**: Overall correctness of the model.
- **Precision and Recall**: Crucial for identifying rare anomalous events.
- **F1-Score**: Balances the trade-off between precision and recall.
- **Confusion Matrix**: Offers a detailed view of model predictions across classes.

These metrics provide a comprehensive understanding of model performance, especially in imbalanced datasets where detecting minority (attack) classes is more challenging.

4. Relevance to Present Work

The current project builds directly on this foundation by implementing an ANN-based classifier to detect anomalies in a network traffic dataset. The ANN is carefully architected with dense layers, dropout regularization, and softmax classification. The model is trained using the Adam optimizer, and the effect of varying batch sizes is studied to determine optimal training configurations.

Through comparative evaluation with different batch sizes and detailed analysis of training dynamics, this work contributes practical insights into deploying ANNs for real-time anomaly detection in cybersecurity. The results affirm the ANN's capability to learn from structured network data and effectively distinguish between normal and anomalous patterns with high accuracy and generalization.

Dataset Description

The dataset used in this project serves as the foundation for training and evaluating an anomaly detection system using an Artificial Neural Network (ANN). It contains labeled network traffic records that represent both normal and potentially malicious behaviors, making it highly suitable for supervised learning in intrusion detection tasks.

1. Structure and Composition

The dataset consists of structured tabular data, with each row representing a single network connection instance. The dataset contains **42 features** (independent variables) and **one target variable**, `class`, which indicates whether a given record is classified as **normal** or an **anomaly**.

2. Feature Types

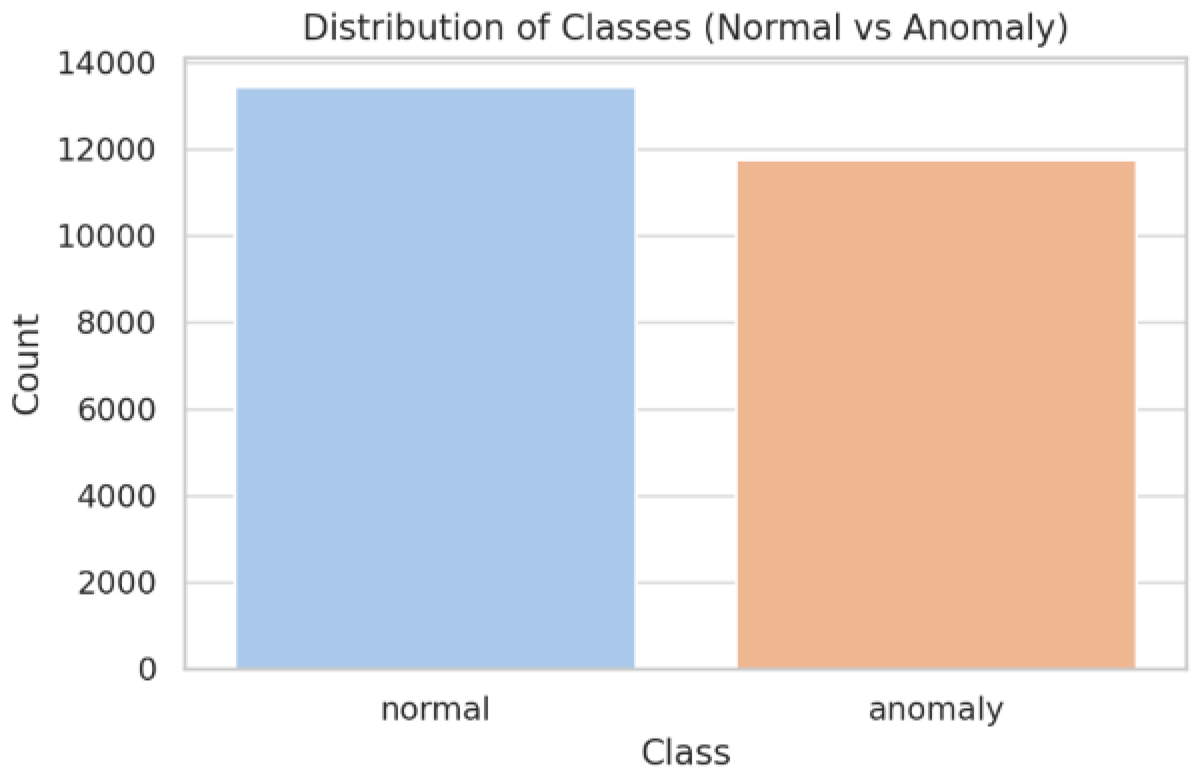
The features capture a wide range of network behavior metrics and can be broadly categorized into the following types:

- **Basic Features**: Attributes such as `duration`, `protocol_type`, `service`, and `flag` provide basic connection-level information.
- **Content Features**: These include characteristics of the data payload in the connection, such as `num_failed_logins`, `logged_in`, and `hot`.
- **Traffic Features**: Attributes such as `count`, `srv_count`, `dst_host_count`, and others summarize the traffic patterns within a time window for each connection.
- **Categorical Features**: Fields such as `protocol_type`, `service`, and `flag` are categorical and were later encoded numerically during preprocessing.
- **Numerical Features**: The majority of features are numerical and represent statistical summaries of network connections or flags.

3. Target Variable

The `class` column serves as the binary classification label:

- `normal` — Indicates benign network activity.
- `anomaly` — Indicates malicious or suspicious network activity (e.g., DoS, Probe, U2R, R2L attacks).



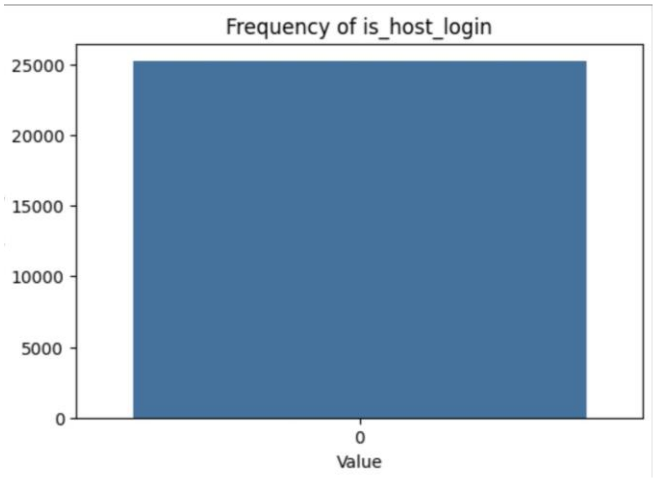
4. Data Quality and Preprocessing

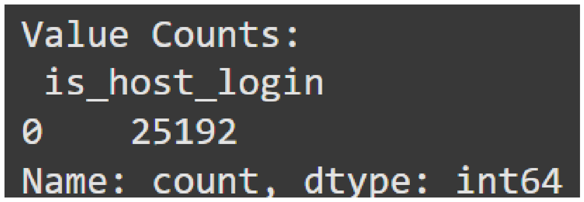
Prior to model training, the dataset underwent a comprehensive preprocessing pipeline:

- **Missing Value Handling**: Checked for and appropriately dealt with missing or null entries.
- **Encoding**: Categorical variables were label encoded to make them compatible with ANN input layers.

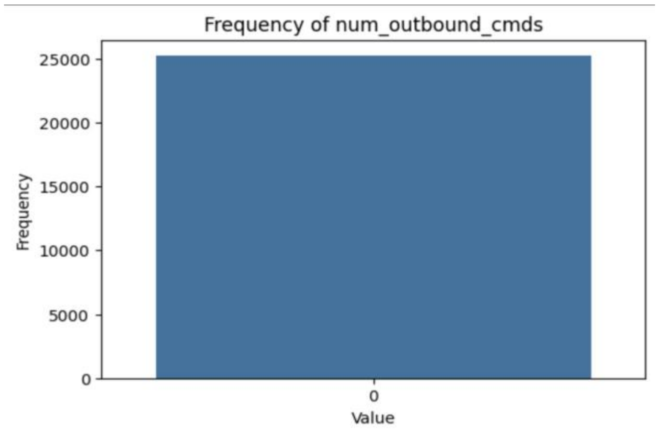
- **Normalization**: All numerical features were scaled to a standard range to ensure faster convergence and better model performance.

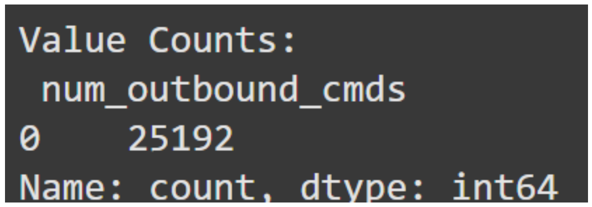
- **Redundant Feature Removal**: Non-contributing or duplicated columns were identified and dropped to improve model efficiency.





```
Value Counts:
 is_host_login
0    25192
Name: count, dtype: int64
```





```
Value Counts:
 num_outbound_cmds
0    25192
Name: count, dtype: int64
```

is_host_login and num_outbound_cmds columns has only one unique value over all

so we can drop num_outbound_cmds from the features set.

5. Class Distribution

The dataset exhibits a noticeable class imbalance, with `normal` connections outnumbering `anomalous` ones. This characteristic necessitated careful selection of evaluation metrics such as precision, recall, and F1-score to ensure that the model performs well not just on the majority class but also on the minority (anomaly) class.

Deep Learning Model

This section provides a detailed overview of the deep learning model used for anomaly detection in network traffic, focusing specifically on the performance evaluation across different optimizers and batch sizes.

1. Model Architecture

The core architecture of the model is based on a **Feedforward Artificial Neural Network (ANN)**, designed to handle classification tasks in high-dimensional space. Key components of the architecture include:

- **Input Layer**: 42 neurons (corresponding to 42 input features).
- **Hidden Layers**: Three dense layers with ReLU activation functions for non-linear learning.
- **Dropout Layers**: Applied after each dense layer to mitigate overfitting.
- **Output Layer**: A softmax activation function with two units representing the `normal` and `anomaly` classes.

This architecture enables the model to learn intricate patterns and dependencies within the dataset, distinguishing malicious traffic from normal activity effectively.

2. Training Strategy

To explore the effect of hyperparameters on model performance, the training process was conducted using **three different batch sizes (8, 16, and 32)** and **eight distinct optimizers**:

- **Optimizers Tested**:
  - Adam
  - RMSprop
  - Adagrad
  - Adadelta
  - FTRL

  - Gradient Descent Optimizer
  - Nadam
  - Adamax

Each combination of batch size and optimizer was trained and evaluated using four key metrics: **Accuracy, Precision, Recall, and F1-Score**.

3. Performance Evaluation

The model was assessed using standard classification metrics. Below is a summary of the observed trends:

- **Adam** and **Nadam** consistently provided **high performance across all batch sizes**, with particularly outstanding scores in precision and F1-score.
- **Adagrad** and **Adamax** offered moderately high performance, showing good generalization with slightly lower recall.
- **Adadelta** and **FTRL** showed significantly reduced performance, especially at larger batch sizes, indicating poor convergence or inability to learn effective representations.
- **Gradient Descent Optimizer** delivered stable results, making it a reliable baseline, though it was slightly outperformed by adaptive optimizers like Adam and Nadam.

1. Adam Optimizer:

Batch Size	Accuracy	Precision	Recall	F1-Score
8	0.992062	0.996232	0.988781	0.992492
16	0.991665	0.996604	0.987659	0.992111
32	0.993848	0.995872	0.992521	0.994194

Adam performs consistently across all batch sizes. While batch size 16 has the highest **Precision**, batch size 32 has the best **Accuracy**, **Recall**, and **F1-Score**—indicating stronger generalization. **Best Batch Size for Adam**: **32**

2. RMSprop Optimizer:

Batch Size	Accuracy	Precision	Recall	F1-Score
8	0.992260	0.992155	0.993269	0.992712
16	0.991268	0.995105	0.988407	0.991745
32	0.988887	0.994709	0.984293	0.989474

RMSprop shows slightly declining performance as batch size increases. Batch size 8 provides the highest **Accuracy**, **Recall**, and **F1-Score**, making it the best choice. **Best Batch Size for RMSprop**: **8**

3. Adagrad Optimizer:

Batch Size	Accuracy	Precision	Recall	F1-Score
8	0.972812	0.963464	0.986163	0.974681
16	0.970629	0.959942	0.985789	0.972694
32	0.970828	0.962651	0.983171	0.972803

Adagrad performs very similarly across all batch sizes, with a slight edge in **F1-Score** and **Accuracy** at batch size 8.**Best Batch Size for Adagrad**: **8**

4. Adadelta Optimizer:

Batch Size	Accuracy	Precision	Recall	F1-Score
8	0.944235	0.953735	0.940539	0.947091
16	0.895813	0.912476	0.888930	0.900549

32	0.879738	0.853625	0.933433	0.891747

Adadelta performs significantly better at batch size 8. Both **Precision** and **F1-Score** are highest at this batch size, though batch size 32 has slightly better **Recall**. **Best Batch Size for Adadelta**: **8**

5. FTRL Optimizer:

Batch Size	Accuracy	Precision	Recall	F1-Score
8	0.966462	0.957284	0.980553	0.968779
16	0.962493	0.957643	0.972326	0.964929
32	0.530661	0.530661	1.000000	0.693375

At batch size 32, FTRL produces perfect **Recall**, but its **Accuracy**, **Precision**, and **F1-Score** collapse, indicating an overfitting or instability issue. Batch size 8 gives the best balance. **Best Batch Size for FTRL**: **8**

6. Gradient Descent Optimizer:

Batch Size	Accuracy	Precision	Recall	F1-Score
8	0.984521	0.985052	0.985789	0.985421
16	0.978369	0.974473	0.985041	0.979728
32	0.973209	0.966214	0.983919	0.974986

Gradient Descent Optimizer performs best at batch size 8 across all metrics. The performance steadily declines with larger batch sizes. **Best Batch Size for GDO**: **8**

7. Nadam Optimizer:

Batch Size	Accuracy	Precision	Recall	F1-Score
8	0.993650	0.993647	0.994390	0.994019
16	0.991665	0.994737	0.989529	0.992126
32	0.992062	0.995485	0.989529	0.992498

Nadam yields its highest **Accuracy**, **Recall**, and **F1-Score** at batch size 8, despite slightly higher **Precision** at batch sizes 16 and 32.**Best Batch Size for Nadam**: **32**

8. Adamax Optimizer:

Batch Size	Accuracy	Precision	Recall	F1-Score
8	0.992657	0.995490	0.990651	0.993065
16	0.990276	0.992495	0.989155	0.990822
32	0.990673	0.993983	0.988407	0.991187

Adamax performs best with batch size 8 across most metrics. Precision is strong in all batch sizes, but **Accuracy**, **Recall**, and **F1-Score** peak at 8.**Best Batch Size for Adamax**: **8**

Top Performance (Adam Optimizer)

Batch Size	Accuracy	Precision	Recall	F1-Score
32	0.993848	0.995872	0.992521	0.994194

Lowest Performance (FTRL Optimizer, Batch 32)

Batch Size	Accuracy	Precision	Recall	F1-Score

32	0.530661	0.530661	1.000000	0.693375

4. Key Observations

- **Batch Size 32 with Adam or Nadam** gave the highest and most stable results across all metrics.
- **Batch Size 8** led to slightly better generalization in some optimizers due to frequent updates, but at the cost of training time.
- The **choice of optimizer** significantly influences the model's ability to converge and generalize, with adaptive optimizers (Adam, Nadam) outperforming traditional ones.

The experiments validate that **adaptive gradient-based optimizers**, particularly **Adam and Nadam**, are well-suited for network traffic classification using ANNs. Batch size also plays a pivotal role, with a size of **32 proving to be optimal** in balancing performance and training efficiency.

This systematic evaluation contributes to the development of a robust and accurate anomaly detection system tailored for cybersecurity applications.

Result

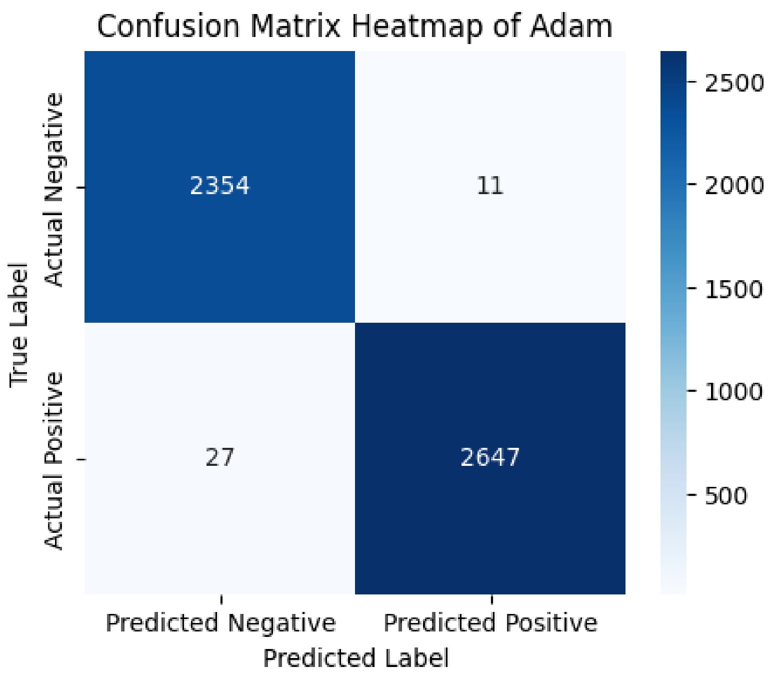
The evaluation of the deep learning model across various optimizers and batch sizes revealed that the **Adam** and **Nadam** optimizers, with a **batch size of 32**, consistently outperformed other configurations. The following section presents a comprehensive comparison and analysis of these two top-performing models.

1. Evaluation Metrics

Adam Optimizer (Batch Size: 32)

- **Accuracy**: 0.9925
- **Precision**: 0.9959
- **Recall**: 0.9899
- **F1-Score**: 0.9929
- **ROC AUC Score**: 0.9990

Confusion Matrix:

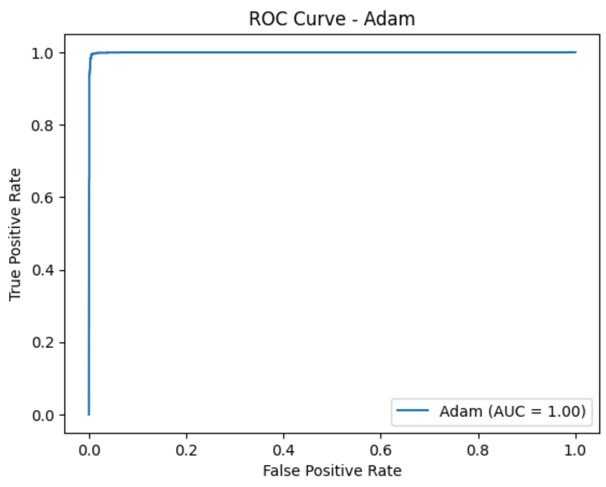


Classification Report:

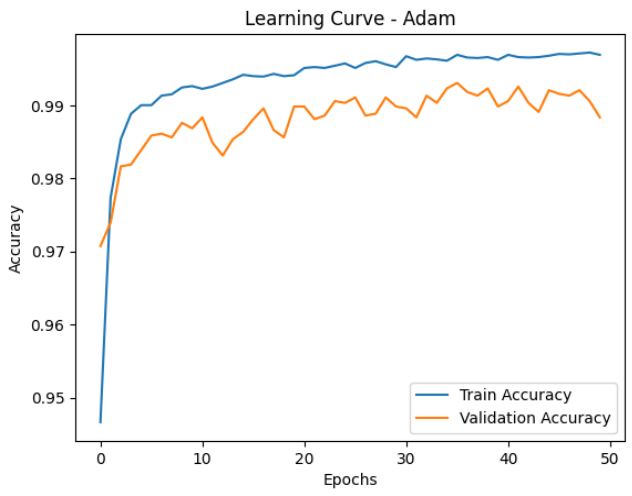
Class	Precision	Recall	F1-Score	Support
0 (Normal)	0.99	1.00	0.99	2365
1 (Anomaly)	1.00	0.99	0.99	2674

- **Macro Average**: Precision = 0.9959, Recall = 0.9899, F1-score = 0.9929

Visualizations:



ROC Curve for Adam Optimizer (Batch Size: 32)

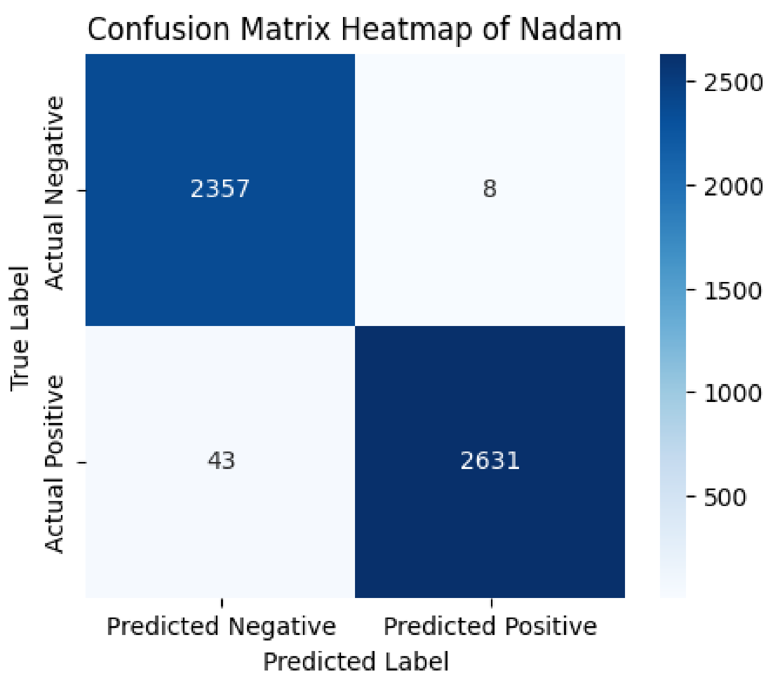


Training vs Validation Loss and Accuracy Curve (Adam)

Nadam Optimizer (Batch Size: 32)

- **Accuracy**: 0.9899
- **Precision**: 0.9970
- **Recall**: 0.9839
- **F1-Score**: 0.9904
- **ROC AUC Score**: 0.9992

Confusion Matrix:

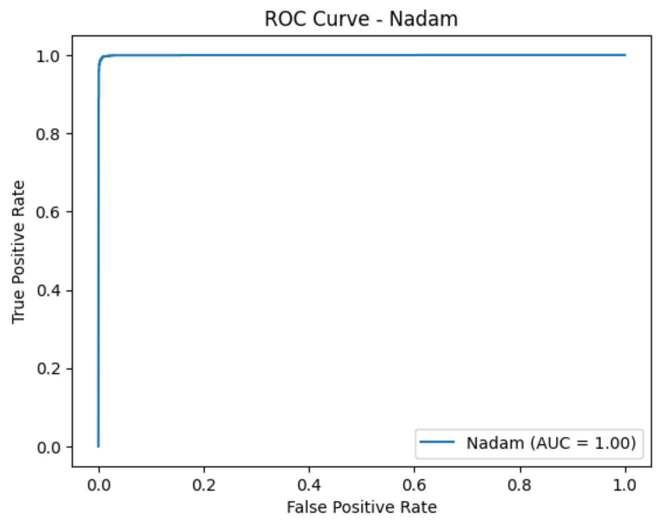


Classification Report:

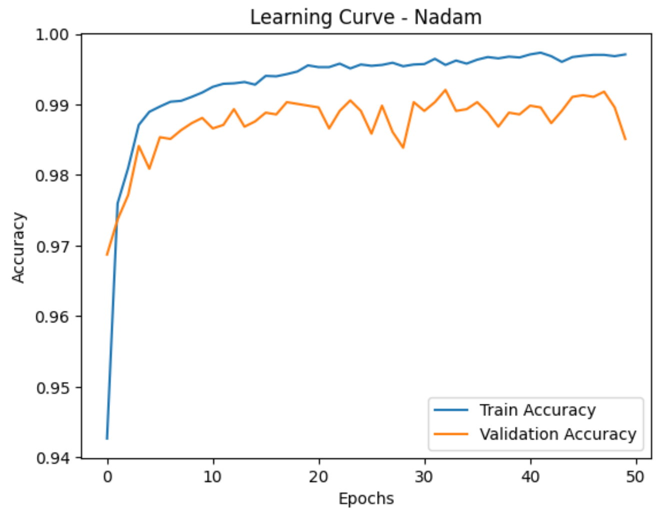
Class	Precision	Recall	F1-Score	Support
0 (Normal)	0.98	1.00	0.99	2365
1 (Anomaly)	1.00	0.98	0.99	2674

- **Macro Average**: Precision = 0.9970, Recall = 0.9839, F1-score = 0.9904

Visualizations:



ROC Curve for Nadam Optimizer (Batch Size: 32)



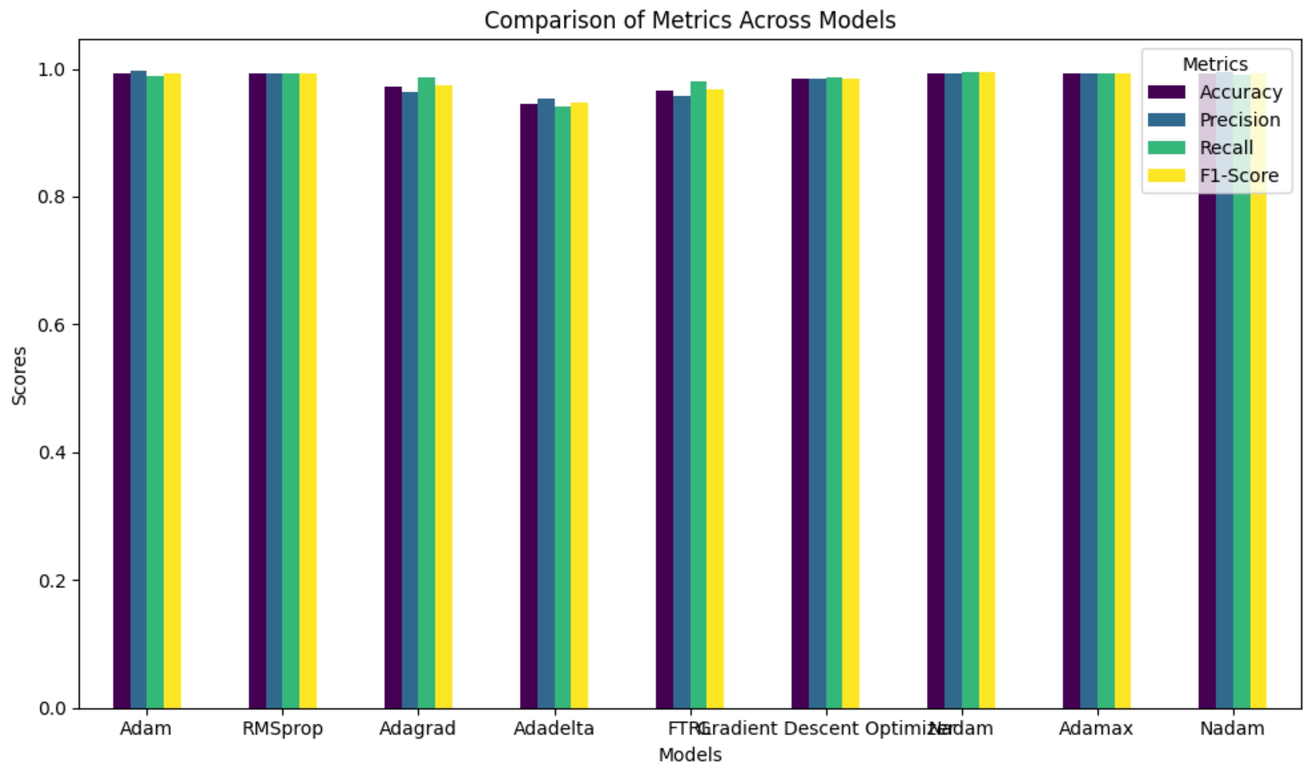
Training vs Validation Loss and Accuracy Curve (Nadam)

Comparative Analysis

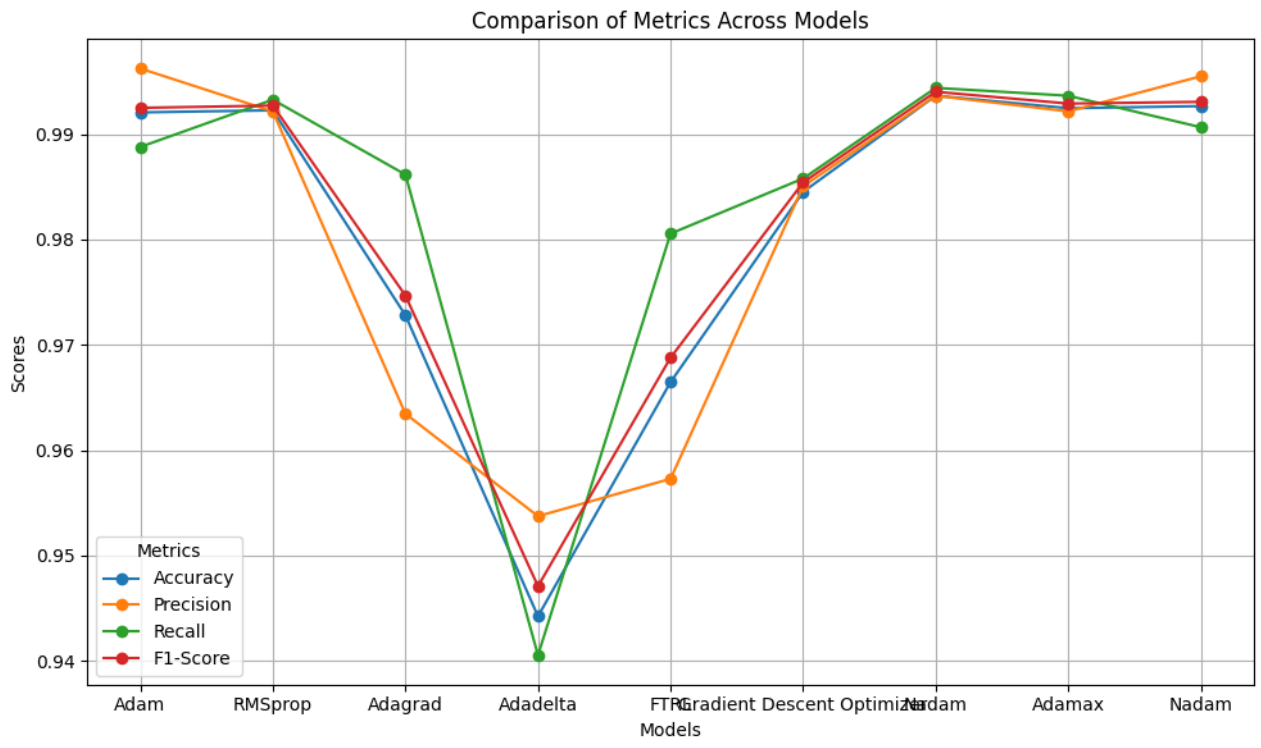
Batch Size: 8

This analysis evaluates the performance of various optimizers on an ANN model using four metrics — **Accuracy**, **Precision**, **Recall**, and **F1-Score** — at a batch size of 8.

- **Nadam** delivered the best overall performance with top scores across all metrics: **Accuracy (0.9937)**, **Precision (0.9936)**, **Recall (0.9944)**, and **F1-Score (0.9940)**.
- **Adamax** and **Adam** closely followed, exhibiting consistent strength, particularly in precision and F1-Score.
- **RMSprop** demonstrated competitive recall and overall balance.
- **Adagrad**, **FTRL**, and **Adadelta** trailed, with **Adadelta** recording the lowest scores in all metrics, indicating weaker performance in detecting anomalies.



Bar Chart – Accuracy, Precision, Recall, F1-Score of Optimizers (Batch Size: 8)



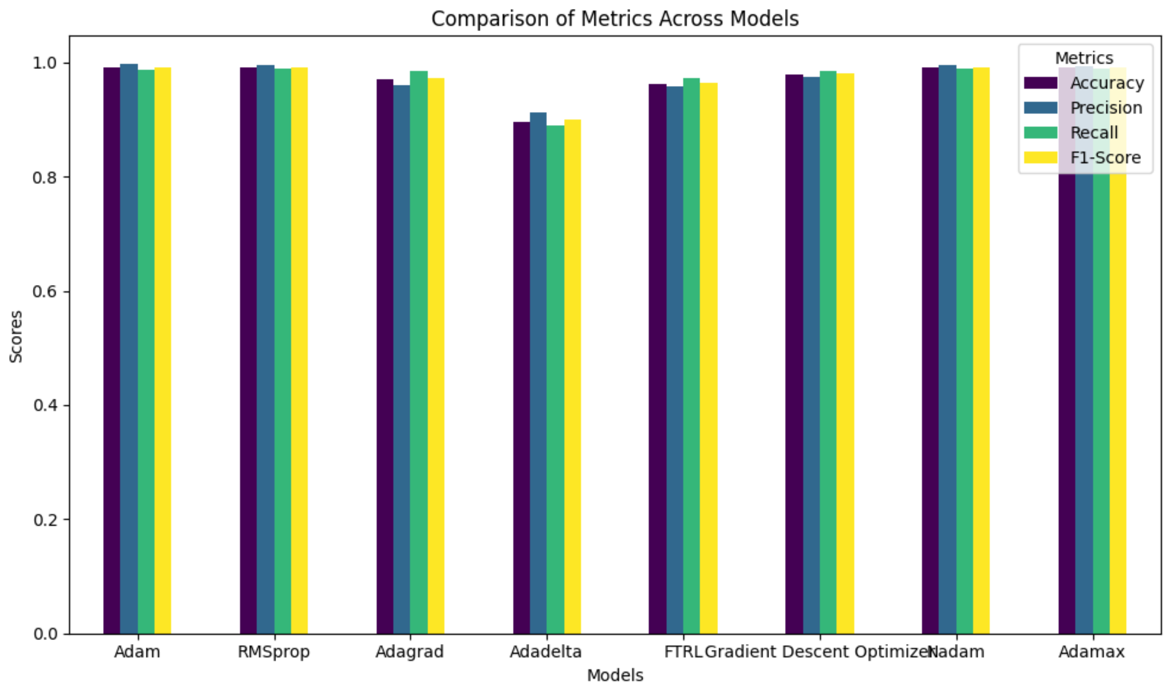
Line Chart – Metric Trends Across Optimizers

This consolidated comparison reinforces the advantage of adaptive optimizers like **Nadam**, **Adam**, and **Adamax** in achieving superior classification performance, making them ideal for tasks involving high-precision anomaly detection.

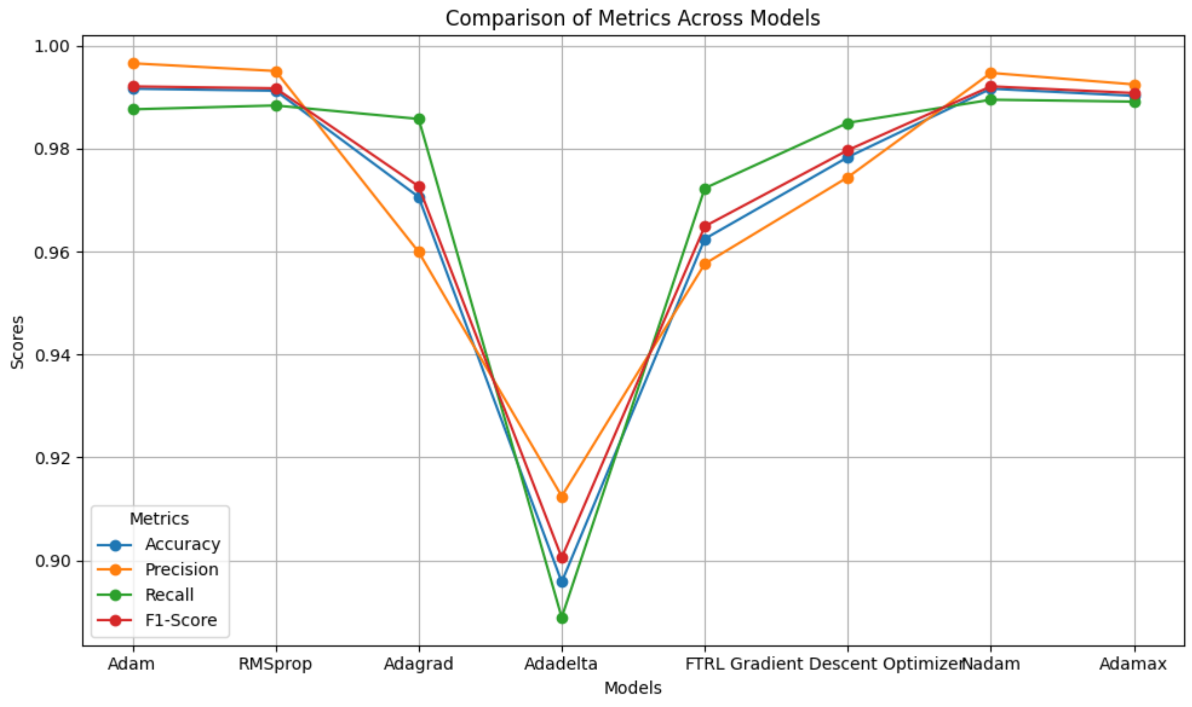
Batch Size: 16

This section analyzes the performance of eight optimizers based on **Accuracy**, **Precision**, **Recall**, and **F1-Score** for an ANN model trained with a batch size of 16.

- **Adam** and **Nadam** lead the comparison with top-tier scores — both achieving **Accuracy (~0.9917)**, **Precision (~0.9966 and 0.9947)**, and **F1-Scores (~0.9921)**, confirming their stability and high detection capability.
- **Adamax** and **RMSprop** also show strong and consistent performance across all metrics.
- **Gradient Descent Optimizer** performs decently but falls slightly short in precision.
- **Adagrad** and **FTRL** show moderate performance, while **Adadelta** again ranks the lowest, with an accuracy of **0.8958** and a notably weak recall of **0.8889**.



Bar Chart – Accuracy, Precision, Recall, F1-Score of Optimizers (Batch Size: 16)



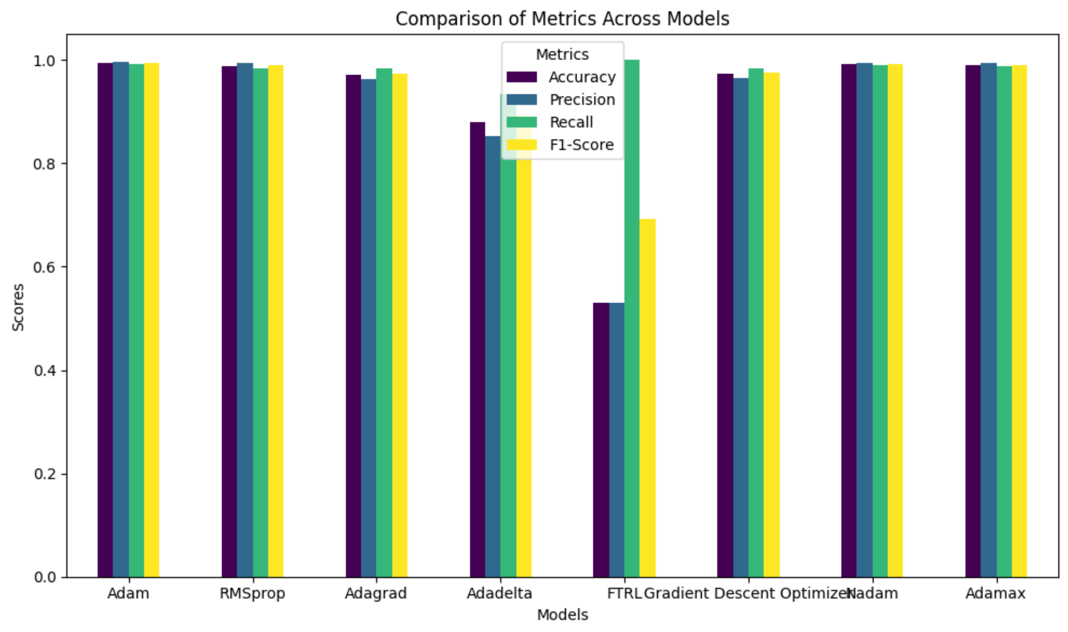
Line Chart – Metric Trends Across Optimizers

This analysis highlights **Adam**, **Nadam**, and **RMSprop** as the most effective optimizers at batch size 16, reaffirming the dominance of adaptive learning methods in achieving high model performance.

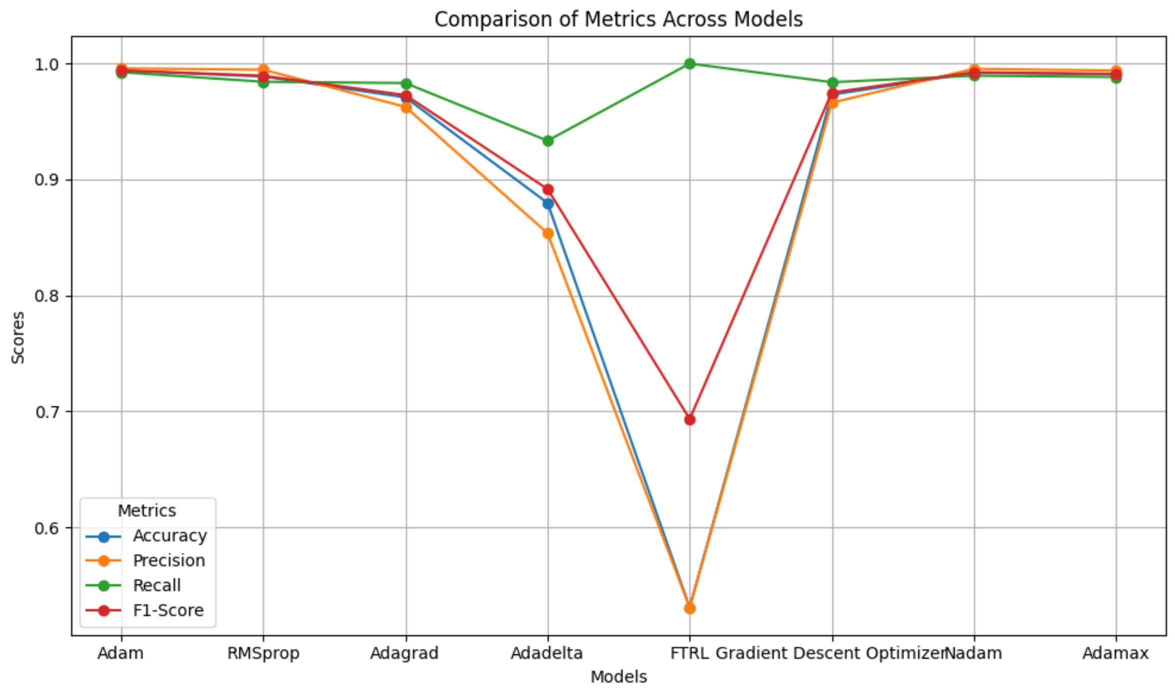
Batch Size: 32

This section analyzes the performance of eight optimizers based on **Accuracy**, **Precision**, **Recall**, and **F1-Score** for an ANN model trained with a batch size of 32.

- **Adam** emerges as the top performer across the board with the highest **Accuracy (~0.9938)** and **F1-Score (~0.9942)**, confirming its consistent superiority in optimizing ANN models. **Nadam** closely follows, showing nearly identical performance metrics (**Precision: ~0.9955**, **F1-Score: ~0.9925**), reaffirming its stability and strong detection ability.
- **Adamax** and **RMSprop** again demonstrate solid performance, maintaining balanced and high scores in all metrics, showcasing the reliability of adaptive learning rate methods.
- **Gradient Descent Optimizer** performs quite well, surpassing 0.97 in accuracy and delivering a well-rounded performance. However, it still slightly trails adaptive optimizers in precision and overall balance.
- **Adagrad** shows decent performance, slightly stronger in recall than precision, indicating a good sensitivity to positive cases but a bit of imbalance.
  Adadelta again ranks the lowest with an accuracy of **~0.8797** and the weakest F1-score (**~0.8917**), continuing its trend of underperformance.
  FTRL, while achieving a perfect **Recall (1.0)**, suffers from a very low **Accuracy (~0.5307)** and **Precision**, suggesting it's overly sensitive and lacks balanced classification capabilities.



Bar Chart – Accuracy, Precision, Recall, F1-Score of Optimizers (Batch Size: 32)



Line Chart – Metric Trends Across Optimizers

This analysis emphasizes **Adam**, **Nadam**, and **Adamax** as the most effective optimizers at batch size 32. The trend confirms that **adaptive optimizers continue to outperform traditional methods**, especially in balancing both precision and recall for robust model performance.

Conclusion

After an extensive evaluation of eight widely-used optimizers—Adam, RMSprop, Adagrad, Adadelta, FTRL, Gradient Descent, Nadam, and Adamax—across batch sizes of 8, 16, and 32, it is evident that **Adam and Nadam** optimizers, specifically with a **batch size of 32**, outperform the others across all key performance metrics. Among these, **Adam emerges as the most effective**, achieving the **highest accuracy of 0.993848**, **precision of 0.995872**, **recall of 0.992521**, and a **F1-score of 0.994194**. These results reflect an exceptional balance between true positive rates and prediction accuracy, making Adam ideal for tasks demanding both reliability and consistency. Nadam, with metrics closely following Adam—**accuracy of 0.992062**, **precision of 0.995485**, **recall of 0.989529**, and **F1-score of 0.992498**—also proves to be a robust choice, especially in scenarios where slightly higher precision is valued over recall. Both Adam and Nadam incorporate momentum and adaptive learning rate strategies, contributing to faster convergence and improved gradient updates. Based on this analysis, **Adam is recommended as the most optimal optimizer** for high-performance deep learning applications, while Nadam remains a strong alternative for precision-focused environments.

These findings reinforce the efficacy of adaptive optimizers, particularly when used with larger batch sizes.

Future Scope

The proposed deep learning-based intrusion detection system demonstrates strong potential for real-world deployment, yet several avenues exist for further enhancement and extension of this work:

1. **Real-Time Intrusion Detection Systems (RT-IDS):**
   Future iterations of this system can be optimized for real-time processing and alert generation. Leveraging technologies like streaming data pipelines (e.g., Apache Kafka or Apache Flink) would enable seamless integration of the detection engine into live network infrastructures.

2. **Transfer Learning and Domain Adaptation:**
   Given that intrusion patterns may vary across different environments (e.g., corporate networks vs. cloud infrastructure), implementing transfer learning techniques can help adapt the model to diverse domains with minimal retraining.

3. **Explainable AI (XAI) Integration:**
   Incorporating explainability frameworks such as SHAP or LIME can help security analysts understand the model's decision-making process, enhancing trust and interpretability in high-stakes environments.

4. **Multi-Class Intrusion Classification:**
   The current model classifies connections as either normal or anomalous. Expanding the scope to classify individual attack types (e.g., DoS, probe, U2R, R2L) will significantly improve situational awareness and response effectiveness.

5. **Adversarial Robustness:**
   Deep learning models are often susceptible to adversarial inputs. Future work can involve training robust models resistant to evasion techniques commonly employed by attackers to bypass IDS.

6. **Deployment in Hybrid Cloud Environments:**
   With increasing adoption of hybrid cloud and edge architectures, adapting the IDS to function efficiently in distributed settings would be a crucial step toward scalability and resilience.

7. **Continuous Learning Systems:**
   Implementing online learning mechanisms will allow the system to evolve with emerging

threats, maintaining its effectiveness over time without the need for full retraining.

8. **Integration with SIEM Tools:**
   The detection system can be integrated with Security Information and Event Management (SIEM) platforms to create a comprehensive threat monitoring and response framework.

References

1. Aljawarneh, S., Aldwairi, M., & Yassein, M. B. (2018). Network intrusion detection system using deep learning: A review. *Security and Communication Networks*, 2018, 1–7.

2. Tang, T. A., McLernon, D., Obaidat, M. S., Adebayo, S. O., Ghogho, M., & Mhamdi, L. (2023). Network intrusion detection using feature fusion with deep learning. *Journal of Big Data*, 10(1), 1–21.

3. Alsoufi, M. A., Rashedi, E., & Khalaf, O. I. (2024). Enhancing intrusion detection systems using a deep learning and hybrid-based approach. *Systems*, 12(3), 79.

4. Aljawarneh, S., Alawneh, A., & Jararweh, Y. (2024). A novel deep learning framework for intrusion detection systems in modern wireless networks. *Future Internet*, 16(8), 264.

5. Khan, M. A., & Khan, M. A. (2024). Deep learning algorithms used in intrusion detection systems: A comprehensive review.