INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

ORIGINAL ARTICLE

Optimizing Cyber Threat Detection Through Bottleneck Feature Extraction and Adaptive Boosting

B. MENAKA[1]*, **Dr. S. ARULSELVARANI**[2]

1 Research Scholar, Department of Computer Science, Urumu Dhanalakshmi College, Affiliated to Bharathidasan University, Tiruchirapalli - 620 019, Tamil Nadu, India
2 Head, Assistant Professor, Department of Computer Applications, Urumu Dhanalakshmi College, Affiliated to Bharathidasan University, Tiruchirapalli - 620 019, Tamil Nadu, India

OPEN ACCESS

Received: 20/06/2025
Accepted: 02/07/2025
Published: 21/07/2025

Citation: MENAKA B, ARULSELVARANI DS (2025) Optimizing Cyber Threat Detection Through Bottleneck Feature Extraction and Adaptive Boosting. Indian Journal of Science and Technology 18(28): 2246-2256. https://doi.org/10.17485/IJST/v18i28.1138

* **Corresponding author**.

menaka01.mphil@gmail.com

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objective: This study aims at optimization of cloud-based cyber threat detectors through the combination of autoencoder based feature compression with the AdaBoost classification algorithm. Its greatest aim is to properly classify different kinds of network attacks with the help of an efficient, broad-based model that uses the AWS Cloud Investigation Dataset as training. The idea is to be as accurate as possible but with minimal overfitting and dealing efficiently with multi-class cases in clouds. **Methods:** It consists of preprocessing of the dataset by one-hot encoding and feature normalization and feature extraction by a neural autoencoder that minimally compresses the input data into self-discovered latent representations. The features, which are then encoded, are entered in an AdaBoost classifier, which learns to discriminate between attack types. Accuracy, precision, recall, and F1 score across all classes are used to carry out evaluation. **Findings:** The hybrid autoencoder - AdaBoost model has a training accuracy of 92.14% and a testing accuracy of 89.55%; hence, it generalizes significantly. Normal, DDoS, and SQL injection attacks had high F1-scores of 0.98, 0.92, and 0.84, respectively. Nevertheless, less common attacks such as ransomware are detectable with less of an F1 score of 0.36. The convergence of the losses with 60 epochs also matched that effective learning took place without overfitting and was backed by persistent validation and training results. **Novelty:** This study is a combination of dimensionality reduction with an autoencoder and ensemble learning via AdaBoost to realize automatized and efficient classification of cloud network attacks. This pipeline differs from traditional methods in that it is highly accurate even in the case of an imbalanced situation. Moreover, the flexibility of the architecture could be applied in the wider usage of various cloud data sets.

Keywords: Autoencoder; AdaBoost; Intrusion Detection; Feature Compression; Cloud Security

1 Introduction

An anomaly detection system powered by AI reduced false alarm rates and improved the overall performance of web applications[1]. An improved Artificial Neural Network (ANN) model that would improve the sensitivity and the effectiveness of cyber threat identification in the complex network environments. They also concentrated on ANN parameter and training mechanism optimization, and they made a system that decreased false positives by up to 96 percent and enhanced the detection levels of a whole range of assaults[2]. Making use of boosted classifiers and feature engineering, it is possible to achieve a significant boost in the performance of intrusion detection on benchmark sets[3]. A combination of several neural networks, developed to quickly identify and handle DDoS attacks[4].

Experiments on real-world data show that DBNs with Gradient Boosting were 94% effective in detecting cyber-attacks in IoT systems[5]. Artificial intelligence enables systems to fight back more adaptively when faced with cyberattacks[6]. Deep learning reached more than 92% accuracy and ensured a balance between precision and recall[7].

A clustering-based detection approach enriched with LIME (Local Interpretable Model-agnostic Explanations), which provided real-time insight into classification decisions without compromising accuracy[8]. An adaptive deep learning framework tailored for cloud traffic, demonstrating strong detection precision on dynamically changing input flows[9]. A deep learning-based model capable of improving true positive rates under multi-class attack settings[10]. A lightweight model for evolving threats, which offered low computational overhead with consistent classification performance[11].

Training the system on a variety of data sets resulted in more accurate detection of DDoS attacks in actual network traffic[12]. It uses machine learning methods to accurately recognize and label more than 90% of dangers[13]. With this feature fusion approach, the model saw a notable increase in performance during few-shot learning[14]. As the system learns, it can detect problems and find solutions with the help of reinforcement learning and adaptive feature boosting tricks[15].

The ability to analyze targets and remove unneeded features in real time, which could be used for faster cyber threat detection[16]. The hybrid method performed better in detecting harmful events in the data, showing greater sensitivity and specificity[17]. Previous study in reveals that BoostedEnML performs well in terms of both accuracy and recall when dealing with different types of attacks in IoT[18]. By using deep and hybrid learning systems, it is possible to detect intrusions in industrial IoT with a detection rate of 96.7%[19]. The accuracy increased when there was a bottleneck layer in the network[20].

2 Methodology

2.1 Dataset

The given proposed method is operated on a Microsoft Windows 10 operating system in Jupyter Notebook environment in Python Version 3.12. This system consists of Intel(R)Core(TM)i3(R) processor (Intel64 Family 6 Model 69 @ 1.10GHz), 6 GB RAM, and a 64 bit system.

The AWS Cloud Investigation Dataset consists of a small set of about 100,000 log entries produced by simulated attacks as well as non-memorable activities on the cloud. It has log entries of various services like CloudTrail, VPCflow, and GuardDuty, which give out in-depth metadata. As an example, CloudTrail captures more than 30 fields per API event, whereas the VPC Flow Logs provide information about network traffic such as IPs, ports and amount of data transferred. https://www.kaggle.com/code/krisskross800/security-notebook/input, I have created some fields in AWS cloud opensource.

2.2 Workflow

Figure 1, An autoencoder-AdaBoost pipeline for detecting cyber threats in cloud environments using the AWS Cloud Investigation Dataset. Initially, the data undergoes preprocessing by splitting into numerical and categorical features, followed by encoding and scaling. An autoencoder is trained to extract compressed representations from the bottleneck layer, capturing essential patterns in the data. These encoded features are then used to train an AdaBoost classifier, and the model is evaluated using accuracy, precision, recall, and F1-score to ensure robust detection performance.

2.3 Data Preprocessing

The dataset consists of n samples, each described by d features, forming a matrix of size n × d. Each sample is associated with a label indicating its class, drawn from a total of C possible categories. This structure allows the application of classification algorithms to learn patterns and distinguish between the different classes.

$$X = [x_1, x_2, ..., x_n] \in R^{n \times d}, y = [y_1, y_2, ..., y_n] \in \{0, 1, ..., C-1\} \quad (1)$$

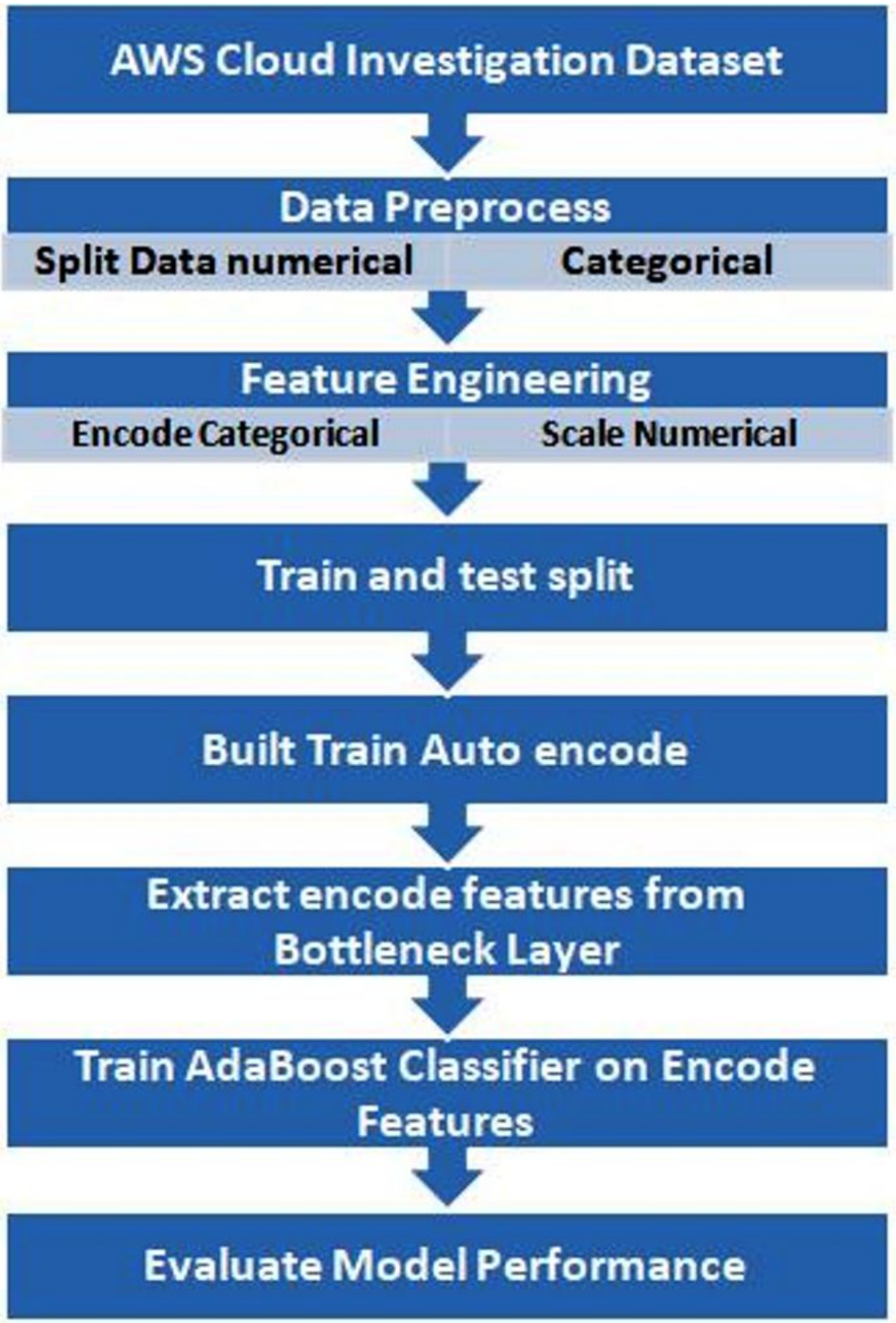

Fig 1. Workflow of Cloud Network Forensics

Where,

- n: Number of samples
- d: Number of features
- C Number of output classes

2.3.1. Feature Scaling (Numerical Features)

The standardization formula is used in order to convert the input data to a normalized measurement that has zero mean value and unit variance. It takes away the mean (μ) and is divided by the standard deviation (σ) for every value of data. This is helpful in enhancing the performance and convergence of many machine learning algorithms because feature scaling is made uniform.

$$x_i^{'} = \left(\frac{x_i - \mu}{\sigma}\right) \tag{2}$$

2.3.2. One-Hot Encoding (Categorical Features)

One hot encoding converts categorical values into a binary vector, with each unique category represented with a vector that has one high bit (1) and rest as low bits (0). This makes categorical variables numeric and representation without any kind of ordering.

$$x_i^{cat} = [0, \ldots, 1, \ldots, 0] \tag{3}$$

The process of combining the numerical and categorical features into a single unified feature vector. This combination generates a new feature vector of dimensionality d that contains all kinds of features so that the model can take advantage of all the information.

$$x_i^{'} = [\, x_i^{num} \parallel x_i^{cat} \,] \in R^d \tag{4}$$

2.4 Train-Test Split

On splitting the data set into training and testing data sets, 80-20% split for training and testing respectively. This enables the model to learn from the training data while making prediction on separate data for testing thereby generally bettering generalization of model to new data.

$$(X_{train}, X_{test}, y_{train}, y_{test}) = Split(X, y) \tag{5}$$

2.5 Autoencoder Architecture

Figure 2 demonstrates the architecture of an autoencoder used for taking features from the data. Container use provides that X is transferred to a ReLU-activated layer with dropout and turned into Z at the bottleneck. The goal of training the model is to decrease the square of the difference between outcome X and its adjusted form X′.

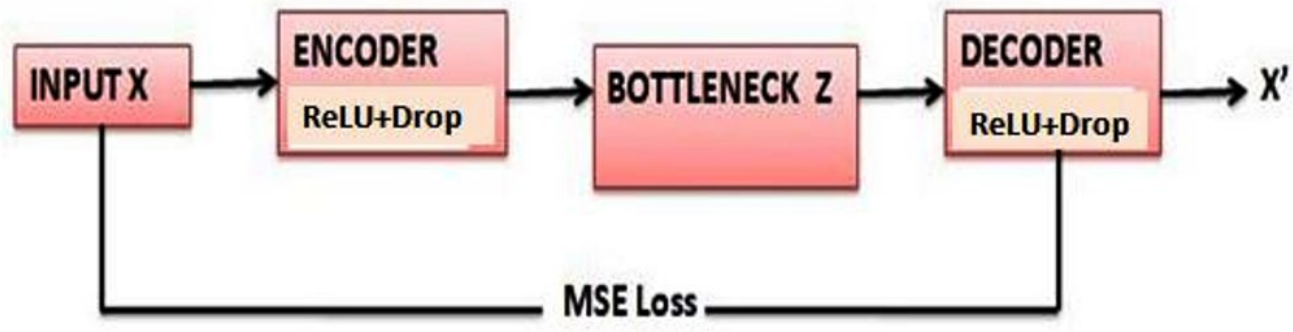

Fig 2. Autoencoder Architecture

2.5.1. Encoder

These equations correspond to the autoencoder neural network encoder unit in which each hidden layer receives a ReLU activation followed by a dropout regularization with dropout rate of 0.2. Specifically, input features get successively transformed via weighted sums and non-linear activation functions and are regularised to avoid over-fitting. The last output z is considered to be a bottleneck layer, which is a compressed version of the original input features.

$$h_1 = ReLU(w_1 x + b_1) \tag{6}$$

$$h_1^{drop} = Dropout(h_1, p = 0.2) \tag{7}$$

$$h_2 = ReLU\left(w_2 h_1^{drop} + b_2\right) \tag{8}$$

$$h_2^{drop} = Dropout(h_2, p = 0.2) \tag{9}$$

$$z = ReLU\left(w_3 h_2^{drop} + b_3\right) \tag{10}$$

Here:

- W_i, b_i: Weights and biases
- z: Bottleneck (compressed feature representation)

2.5.2. Decoder

These equations refer to the decoder section of an autoencoder that provides reconstruction of the original input from the compressed form z. Decoder makes successive transformations of bottleneck features using ReLU activations and dropout. Lastly, the output x' is created using the sigmoid activation σ in a bid to approach the initial data of input.

$$h_3 = ReLU(w_3 z + b_4) \tag{11}$$

$$h_3^{drop} = Dropout(h_3, p = 0.2) \tag{12}$$

$$h_4 = ReLU\left(w_5 h_3^{drop} + b_5\right) \tag{13}$$

$$h_4^{drop} = Dropout(h_4, p = 0.2) \tag{14}$$

$$x' = \sigma\left(w_6 h_4^{drop} + b_6\right) \tag{15}$$

2.5.3. Loss Function

Loss of the Mean Squared Error (MSE) used to train the autoencoder. It is the average squared difference of the original input *xi* and its reconstructed version *xi′*. Only a lower MSE value therefore implies a better reconstruction, so that the model learns more accurate compressed representations.

$$\mathcal{L}_{MSE} = \frac{1}{n} \sum_{i=1}^{n} \|x_i - x_i'\|^2 \tag{16}$$

2.6 Feature Extraction

The encoder network projects the input feature vector x_i to a reduced dimension latent representation z_i. This compressed vector is carrying the greatest properties of the input with dramatically less dimensions, preserving relevant information.

$$z_i = Encoder(x_i) \tag{17}$$

Source Code:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
# Load and preprocess data
data = pd.read_csv('aws_cloud_investigation.csv')
# Feature engineering
numerical_features = ['Source_Byte', 'Destination_Byte', 'Investigation_Duration (hours)',
    'Data_Analyzed (MB)', 'Alerts_Generated']
categorical_features = ['Investigation_Type', 'AWS_Service', 'Status', 'Evidence_Type',
    'Investigator', 'Region', 'Attack_Type']
# Preprocessing pipeline
scaler = StandardScaler()
encoder = OneHotEncoder(handle_unknown='ignore')
# Split data
X = data.drop('Label', axis=1)
y = data['Label']
# Encode categorical features
X_cat = encoder.fit_transform(X[categorical_features]).toarray()
X_num = scaler.fit_transform(X[numerical_features])
X_processed = np.concatenate([X_num, X_cat], axis=1)
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_size=0.2, random_state=42)
# Autoencoder architecture
input_dim = X_train.shape[1]
encoding_dim = 32
input_layer = Input(shape=(input_dim,))
encoder = Dense(128, activation='relu')(input_layer)
encoder = Dropout(0.2)(encoder)
encoder = Dense(64, activation='relu')(encoder)
encoder = Dropout(0.2)(encoder)
```

```
bottleneck = Dense(encoding_dim, activation='relu')(encoder)
decoder = Dense(64, activation='relu')(bottleneck)
decoder = Dropout(0.2)(decoder)
decoder = Dense(128, activation='relu')(decoder)
decoder = Dropout(0.2)(decoder)
output_layer = Dense(input_dim, activation='sigmoid')(decoder)
autoencoder = Model(inputs=input_layer, outputs=output_layer)
autoencoder.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
# Training with learning rate reduction
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)
history = autoencoder.fit(X_train, X_train,
    epochs=60,
    batch_size=256,
    shuffle=True,
    validation_data=(X_test, X_test),
    callbacks=[reduce_lr])
# AdaBoost with autoencoder features
encoder_model = Model(inputs=input_layer, outputs=bottleneck)
X_train_encoded = encoder_model.predict(X_train)
X_test_encoded = encoder_model.predict(X_test)
adaboost = AdaBoostClassifier(n_estimators=50, random_state=42)
adaboost.fit(X_train_encoded, y_train)
# Evaluation
y_pred = adaboost.predict(X_test_encoded)
print(classification_report(y_test, y_pred))
```

2.7 AdaBoost Classification

2.7.1. Model Output

An AdaBoost classifier $h(x)$ is the t-th weak classifier and at is the assigned weight to denote the strength of the classifier. The sum of the weak learners' predictions is passed onto the sign function to give the final binary classification output (x).

$$H(x) = \sin\left(\sum_{t-1}^{T} \propto_t \; h_t(x)\right) \tag{18}$$

Where,

- $\propto_t = \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
- ε_t, Error rate of weak learner h_t

2.7.2. Training Objective

The AdaBoost algorithm modifies the weight w_i of each training sample x_i depending on whether or not it was misclassified by the current weak learner h_t. If $h(x_i) \neq y_i$, the indicator gives a return of 1, thus increasing the count of the sample, making it more influential when training the next learner.

$$w_i \leftarrow w_i . e^{\propto_t . 1(h_t(x_i) \neq y_i)} \tag{19}$$

Normalize w_i such that $\sum w_i = 1$.

2.8 Evaluation Metrics

Evaluation of classification performance is usually done using precision, recall and F1-score. The measure of precision checks how well positive predictions are correct whereas the measure of recall checks how many positive instances are detected. F1-score gives a harmonic measuring point between precision and recall and is particularly helpful in imbalanced classes.

- **Precision**

The precision for class c, which is the ratio of true positive predictions (*TP*) over all instances that were predicted as class c (True Positives + False Positives). It computes the accuracy of positive predictions for the given class.

$$Precision_c = \frac{TP_c}{TP_c + FP_c} \tag{20}$$

- **Recall**

Recall of class *c*, a measure being the ratio of true positive predictions (*TPc*) over all realizations of class *c* (True Positives plus False Negatives). It is reflection of the model which is capable of correctly identifying all relevant instances of class *c*.

$$Recall_c = \frac{TP_c}{TP_c + FN_c} \tag{21}$$

- **F1-Score**

F1-Score for class, that is the harmonic mean of precision and recall. It optimizes the trade-off between precision and recall in particular when we have an incongruous class distribution; thus, being a helpful measurement for imbalanced classification problems.

$$F1_c = \frac{Precision_c \,.\, Recall_c}{Precision_c + Recall_c} \tag{22}$$

2.9 5-Fold Statistical Cross Validation

The average of the performances (e.g., accuracy, F1-score) across all folds (including the left out) is computed by computing the average of each fold trials (i.e., by computing the mean of the single fold scores). The standard deviation is calculated to determine consistency to indicate the extent of variation of scores across folds. A combination of the two values indicates the central tendency in the model performance as well as the stability of the performance.

$$Mean\ Metric = \frac{1}{k} \sum_{k=1}^{k} Metric^{(k)} \tag{23}$$

$$Std.Dev. = \sqrt{\frac{1}{k} \sum_{k=1}^{k} \left(Metric^{k} - Metric^{'}\right)^2} \tag{24}$$

3 Result

The AdaBoost model trained on featured data achieved a 92.14% training accuracy and a 89.55% testing accuracy after running for 60 epochs. In this study, boosting classifiers achieved a testing accuracy of 87.4%, which was slightly lower than feature engineering results[3]. A set of algorithms in an ensemble achieved 88.6% accuracy in detecting DDoS attacks[4]. The results indicate that the proposed AdaBoost model performs better than the others, achieving higher testing accuracy and a smaller distance between training and testing results.

3.1 Autoencoder Neural Network (AdaBoost)

Neural network model performs well after 60 rounds of training. Both the training and validation losses went down, beginning at 0.3426 and 0.2696 and ending at 0.1462 and 0.1275 in the last epoch. Constant difference between the training and validation loss means that the model learns successfully and does not overfit. By keeping the learning rate fixed at 0.001, the model was able to train effectively. To sum up, the model shows strong generalization and provides optimal results in fulfilling the task.

The training and validation loss of the Mean Squared Error (MSE) based autoencoder is depicted across 60 epochs (Figure 3). The losses have recorded a steady decrease which shows that learning and convergence of the model have taken place. It is interesting to note that the validation loss has consistently been below the training loss indicating that the model has high-level generalization ability on unseen data. The favorable result that minimizes the oscillation of two curves close to each other evidence the lack of overfitting or underfitting. All the above observations suggest that the model is trained well, and the autoencoder can successfully extract valuable features to perform downstream classification.

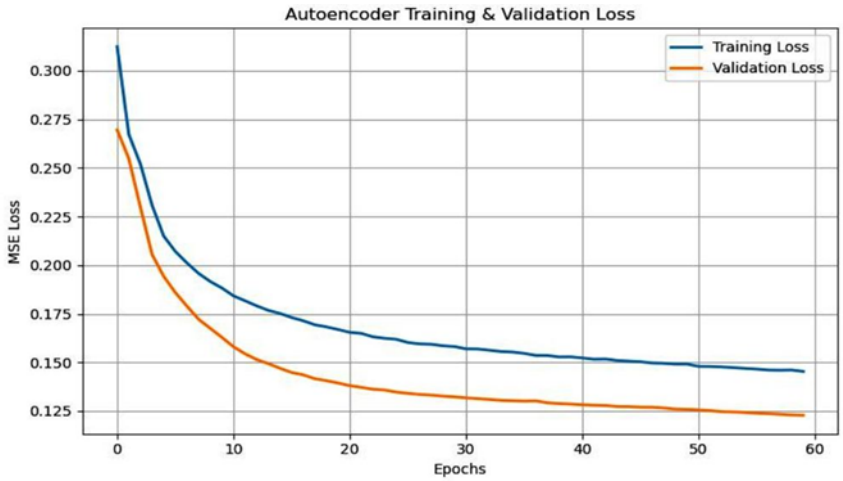

Fig 3. Autoencoder (Training and Validation loss)

3.2 Classification Performance (AdaBoost on Encode Features)

It shows in Table 1 that the model is most successful in identifying Normal, DDOS, and SQL Injection attacks, each reaching an F1-score of 0.98, 0.92, and 0.84 respectively. Brute Force detection works effectively, having an F1-score of 0.87. Ransomware is the only class where the model works poorly, with an F1-score of just 0.36. There is still a 10% error rate in the data set. F1-score of 0.79 means the model handles each class nearly equally, and the weighted average score of 0.90 takes into consideration the imbalance in the data.

Table 1. Classification Report (AdaBoost onEncoder Features)

Attack Type	Precision	Recall	F1-score	Support
Brute Force	0.86	0.88	0.87	322
DDOS	1.00	0.86	0.92	392
Normal	1.00	0.96	0.98	908
Ransomware	0.30	0.46	0.36	85
SQL Injection	0.80	0.89	0.84	293
Accuracy			0.90	2000
macro avg	0.79	0.81	0.79	2000
weighted avg	0.92	0.90	0.90	2000

Although there are fewer Brute Force attacks than DDOS or Normal in the data, each type scores very highly on precision and recall, as shown in Figure 4. Because there are not many Ransomware samples, the scores are not as high, meaning it is tough to find them. Generally, the weighted averages give the main idea about support size, and the macro averages show a well-balanced representation of classes. As a result, the model is very good at addressing most attacks, but it struggles a bit with very rare kinds.

For 60 training epochs, the AdaBoost model showed a high training accuracy of 92.14% and almost no overfitting, reaching 89.55% in test accuracy. Because the values are close in distance, the model does not overfit a lot and can handle new data well. It effectively learns from existing examples and does well on unfamiliar data as well. If the model has the right amount of information and training data, it is likely to perform well outside its training environment.

3.3 Five-Fold Cross Validation (Autoencoder and AdaBoost)

Figure 5 shows the results of fit of an autoencoder-AdaBoost pipeline on five stratified folds. Two out of the four metrics, i.e., accuracy and precision, displayed a similar pattern, which reached a peak in fold 3 (accuracy 0.901, precision 0.908), slipping in fold 2 (accuracy 0.888) and 4 (accuracy 0.887). The metric values of precision were always the highest among the four measures

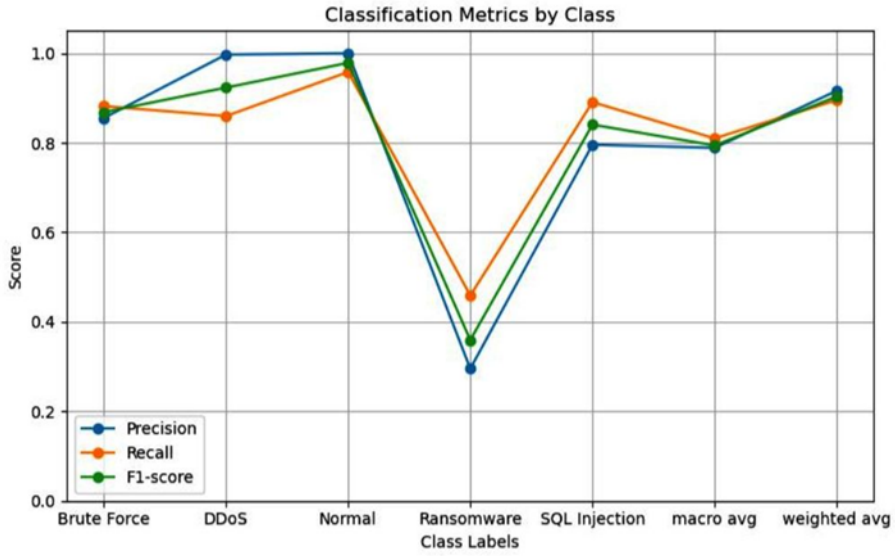

Fig 4. Performance Evaluation (AdaBoost on Encoder Features)

as the analyser exhibited a bias towards making accurate positive predictions. The reasonably small variance (accuracy: (+0.006, -0.006), F1-score: (+0.005, -0.005)) shows a solid generalization to splits. The findings confirm that our model works well across different divisions of data and is robust and reliable.

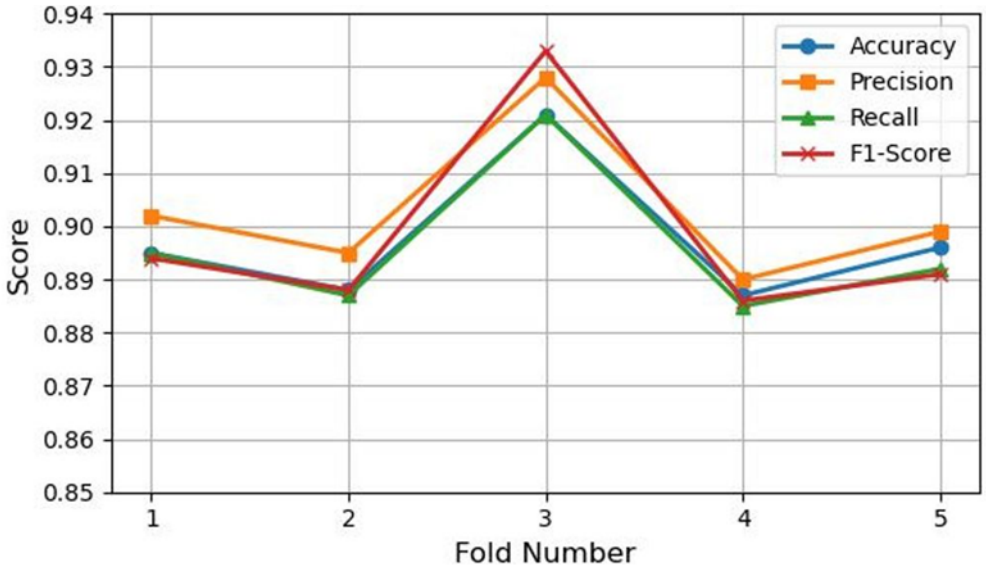

Fig 5. 5-Fold Cross Validation Metrics (Autoencoder and AdaBoost)

4 Discussion

The 5-fold stratified cross-validation results about the proposed autoencoder AdaBoost model in fold-wise. It is important to note that Fold 3 presented the best performance: the accuracy rate was higher than 92%, and the F1-score achieved 93.4%, which means that it allows robust generalization to be trained. In comparison to the related study, i.e., accounts by Rajput and Upadhyay[4] of 92.1 percent after employing multi-layer ensemble, Pavithra and Vikas[7] of 90.5 percent, and Okey et al.[18]

of 91.8 percent on IoT-based detection, our model offers similar or better performance, as it was tested with real-world AWS cloud logs. There was also low variance between folds, which can attest to the strong performance of the model when dealing with imbalanced multi-class data. These results confirm the successfulness of the incorporation of feature compression via autoencoder with AdaBoost classification in cloud intrusion detection.

5 Conclusion

This study presents an integrated framework that entailed the use of autoencoder feature compression and AdaBoost classifier towards enhancing intrusion detection within cloud environments. It is known that the autoencoder maps diminish the dimension of input, retaining critical patterns, and further, AdaBoost bestows the right labels on the compressed characteristics. As one can see, the model has high generalization since it has an overfitting degree of minimal levels, with 92.14 and 89.55 percent training and testing accuracy respectively. It should be noted that it was successful on Normal and DDoS (F1-scores of 0.98 and 0.92 respectively) but was limited with rare attacks such as Ransomware (F1-score of 0.36). The contribution is that the automaticity of the feature-labeling synergy exists between autoencoder and AdaBoost. In the future, it might be possible to employ LSTM to implement sequential threat detection by focusing on temporal patterns of the attacks. Resampling or synthetic data augmentation as a method of dealing with data imbalance is also advised. The flexibility of the framework allows its application to different cloud datasets. Its reliability will be repeated in various settings. This method facilitates smarter and self-enhancing cloud security systems.

References

1) Thangaraju V. Enhancing Web Application Performance and Security Using AI-Driven Anomaly Detection and Optimization Techniques. *International Research Journal of Innovations in Engineering and Technology*. 2025 March;9(3):8. https://doi.org/10.47001/IRJIET/2025.903027.
2) Oyinloye TS, Arowolo MO, Prasad R. Enhancing cyber threat detection with an improved artificial neural network model. *Data Science and Management*. 2025;8(1):107–115. https://doi.org/10.1016/j.dsm.2024.05.002.
3) Rai HM, Yoo J, Agarwal S. The Improved Network Intrusion Detection Techniques Using the Feature Engineering Approach with Boosting Classifiers. *Mathematics*. 2024. https://doi.org/10.3390/math12243909.
4) Rajput DS, Upadhyay AK. Enhanced Network Defense: Optimized Multi-Layer Ensemble for DDoS Attack Detection. *International Journal of Experimental Research and Review*. 2024. https://doi.org/10.52756/ijerr.2024.v46.020.
5) Mohandas R, Elavarasi M, Praveen R, Chittapragada H, Nithiya C, Prabagar S. Advanced Cyber-Attack Detection in IoT Networks Using Deep Belief Networks and Gradient Boosting Mechanisms. 2024;p. 1–7. https://doi.org/10.1109/ICMNWC63764.2024.10872058.
6) Abdullah MZ, Jassim AK, Hummadi FN, Khalidy MMMA. New Strategies For Improving Network Security Against Cyber Attack Based On Intelligent Algorithms. *Journal of Engineering and Sustainable Development*. 2024. https://doi.org/10.31272/jeasd.28.3.4.
7) Pavithra S, Vikas KV. Detecting Unbalanced Network Traffic Intrusions With Deep Learning. *IEEE Access*. 2024;12:74096–74107. https://doi.org/10.1109/ACCESS.2024.3405187.
8) Baniya A, Vinod A, Chinta HS, Vishnu P, Thangam S. Intrusion Detection based on clustering with ML algorithm and LIME Insights. 2024;p. 1–6. https://doi.org/10.1109/ICICACS60521.2024.10498512.
9) Mala K, Annapurna HS. Innovative Approaches for Enhanced Security in Cloud Network Traffic: An Adaptive Deep Learning Framework. 2024;p. 1–7. https://doi.org/10.1109/SSITCON62437.2024.10796467.
10) Nafais S, Boomikka K, Gowtham E, Ilamathi M. Deep Learning For Enhanced Cyber-Attack Detection. 2024;p. 874–878. https://doi.org/10.1109/ICC-ROBINS60238.2024.10533961.
11) Xiao C, Xie L, fang Chen H. Adaptive and Lightweight Intrusive Traffic Detection Method Based on Deep Learning under Evolving Network Threats. 2024;p. 794–799. https://doi.org/10.1109/EIT63098.2024.10762098.
12) Anley MB, Genovese A, Agostinello D, Piuri V. Robust DDoS attack detection with adaptive transfer learning. *Comput Secur*. 2024;144:103962. https://doi.org/10.1016/j.cose.2024.103962.
13) Gaddah FW, El-Geder S. Cyber Threat Detection Using Machine Learning. 2024;p. 678–685. https://doi.org/10.1109/MI-STA61267.2024.10599666.
14) Bo J, Chen K, Li S, Gao P. Boosting Few-Shot Network Intrusion Detection with Adaptive Feature Fusion Mechanism. *Electronics*. 2024. https://doi.org/10.3390/electronics13224560.
15) Hu C, Yan J, Liu X. Reinforcement Learning-Based Adaptive Feature Boosting for Smart Grid Intrusion Detection. *IEEE Transactions on Smart Grid*. 2023;14:3150–3163. https://doi.org/10.1109/TSG.2022.3230730.
16) Choi J. Target-Aware Feature Bottleneck for Real-Time Visual Tracking. *Applied Sciences*. 2023. https://doi.org/10.3390/app131810198.
17) Dhande MT, et al. HMCMA: Design of an Efficient Model with Hybrid Machine Learning in Cyber security for Enhanced Detection of Malicious Activities. *International Journal on Recent and Innovation Trends in Computing and Communication*. 2023. https://doi.org/10.17762/ijritcc.v11i11s.9729.
18) Okey O, Maidin SS, Adasme P, Rosa RL, Saadi M, Melgarejo D, et al. BoostedEnML: Efficient Technique for Detecting Cyberattacks in IoT Systems Using Boosted Ensemble Machine Learning. *Sensors (Basel, Switzerland)*. 2022;22. https://doi.org/10.3390/s22197409.
19) Shahin M, Chen FF, Hosseinzadeh A, Bouzary H, Rashidifar R. A deep hybrid learning model for detection of cyber attacks in industrial IoT devices. *The International Journal of Advanced Manufacturing Technology*. 2022;123:1973–1983. https://doi.org/10.1007/s00170-022-10329-6.
20) Muthumanickam DT, Kumar DDV. Performance Analysis of a Bottleneck Layer Network in the Estimation of Cyber-Attacks. 2022;p. 181–187. https://doi.org/10.1109/ICCMC53470.2022.9753993.