



Anomaly Detection System for Network Traffic Using Backpropagation Neural Network Optimized with Adam Algorithm

Ibrahim Mathew Michael, Obunadike G. N. and Nazifi Shuaibu

Department of Computer Science, Federal University Dutsin-Ma, Katsina State, Nigeria

*Corresponding Author's e-mail: mathewmichaelu@gmail.com

ABSTRACT

Network anomaly detection requires classifiers that distinguish benign traffic from malicious traffic while controlling false alarms. This study evaluates a feed-forward Backpropagation Neural Network (BPNN) trained with the Adam optimizer for binary classification using the NSL-KDD benchmark. The implemented network uses eight retained traffic features, Dense layers of 8 and 15 tanh neurons, and a two-node Softmax output. The supplied implementation uses an 80:20 train-test split with random_state = 42; 20% of the training portion is used as validation. Training uses 100 epochs, batch size 32, learning rate 0.001, and sparse categorical cross-entropy. From the reported confusion matrix (TN=1534, FP=17, FN=23, TP=431), the derived accuracy is 97.99%, precision 96.21%, recall 94.93%, F1-score 95.57%, specificity 98.90%, false-positive rate 1.10%, and false-negative rate 5.07%. An ROC-AUC of 0.992 is retained as reported, but the raw ROC coordinates were not supplied. The results demonstrate strong benchmark performance but do not establish optimizer superiority, zero-day detection, contemporary-network generalization, or production readiness. Controlled optimizer ablation, repeated runs, statistical testing, attack-category analysis, and validation on newer datasets are therefore identified as necessary follow-up work.

Keywords: Anomaly detection, Backpropagation Neural Network, Adam optimizer, NSL-KDD, Intrusion detection, False-positive rate

1.0 INTRODUCTION

Network intrusion detection systems monitor network activity and alert when traffic is judged malicious or sufficiently abnormal. Anomaly-based detection is important because it learns patterns of traffic behaviour rather than relying only on predefined signatures. A systematic review of anomaly-based network intrusion detection research found that preprocessing, feature transformation, model choice, evaluation metrics, and dataset quality materially affect reported performance (Yang et al., 2022).

NSL-KDD remains a widely used benchmark because it provides labelled normal and attack traffic and was developed to reduce redundancy problems in KDD'99. It contains 41 attributes and attack categories including DoS, Probe, R2L and U2R. Its continued use is useful for controlled benchmarking, but recent research also emphasizes that it is an old, simulated benchmark and does not fully represent current operational networks (Tavallae et al., 2009; Yang et al., 2022; Ayantayo et al., 2023).

This study focuses on a compact feed-forward BPNN trained with Adam. Adam uses adaptive estimates of gradient moments and is widely used for neural-network optimization (Kingma & Ba, 2015). The use of Adam, however, does not by itself prove superiority over SGD, Momentum, RMSProp or other optimizers. Such a conclusion requires a controlled comparison using the same architecture, preprocessing, data partition and evaluation protocol.

The objective is to implement and evaluate the supplied BPNN-Adam model for binary normal-versus-anomalous classification on NSL-KDD. The contributions are: (i) explicit documentation of the implemented architecture and training protocol; (ii) transparent reporting of the confusion matrix and derived error rates; and (iii) a clear separation between benchmark evidence and claims that were not experimentally tested, including zero-day capability, real-time readiness, scalability and optimizer superiority.

1.1 Related Work

Yang et al. (2022) reviewed 119 highly cited anomaly-based intrusion-detection studies and emphasized the importance of preprocessing, feature selection, evaluation methodology and dataset choice. Their review also identified NSL-KDD and KDD'99 as heavily used datasets while noting the need for more representative data.

Ayantayo et al. (2023) evaluated deep-learning feature-fusion architectures using NSL-KDD and UNSW-NB15 and discussed class imbalance and generalization. Other recent studies have investigated recurrent, ensemble, hybrid and optimization-based intrusion-detection methods, illustrating that reported performance is strongly dependent on architecture and experimental protocol.

Recent NSL-KDD research has also examined feature selection and model comparison. For example, a 2023 IEEE study investigated mutual-information and correlation-based feature selection across several classifiers. These studies reinforce the need to report exactly which features and transformations are used. The present work therefore identifies eight retained features but does not claim a particular selection algorithm because its code was not supplied.

The gap addressed by the revised study is methodological transparency. The earlier manuscript simultaneously described 41 input nodes, eight selected features and 10-fold cross-validation. The supplied implementation instead supports eight retained variables, an 80:20 split, a 20% validation split from the training data, and Adam-based training. The revised manuscript reports the latter protocol.

2. MATERIALS AND METHODS

2.1 Research Design

The study uses supervised binary classification. The workflow comprises NSL-KDD preparation, preprocessing, retention of eight features, train-test splitting, validation within the training portion, BPNN construction, Adam optimization, training and held-out evaluation.

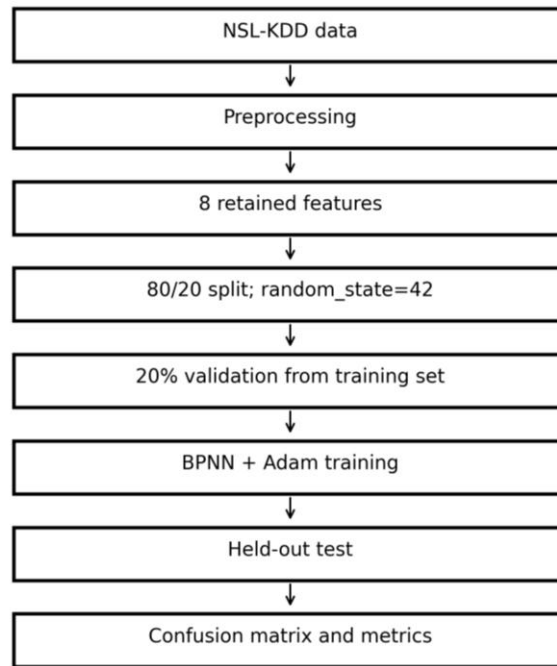


Figure 1: Revised experimental workflow

2.2 Dataset

The study uses NSL-KDD. The dataset has 41 original attributes and labelled normal and attack traffic. The four attack groups are DoS, Probe, R2L and U2R (Tavallaee et al., 2009). The source chapter reports 34,144 training and 8,571 testing instances. However, the supplied Chapter 4 confusion matrix contains 2,005 observations. These figures cannot both represent the same final test evaluation. The revised manuscript therefore uses the internally consistent confusion matrix for metric derivation and flags the sample-count discrepancy for reconciliation from the original execution log before submission.

2.3 Feature Representation and Preprocessing

Eight retained variables are documented: Duration, Protocol_type, Service, Src_bytes, Dst_bytes, Flag, Src_port and Dst_port. The source methodology states that categorical variables were encoded and numeric variables normalized/standardized. The exact preprocessing script and feature-selection code were not supplied, so the revised manuscript does not attribute the selection to a specific algorithm and does not invent an exact scaler.

Table 1: Eight retained features documented in the supplied methodology

Feature	Role
Duration	Connection duration
Protocol_type	Protocol category
Service	Network service
Src_bytes	Source byte count
Dst_bytes	Destination byte count
Flag	Connection status
Src_port	Source port
Dst_port	Destination port

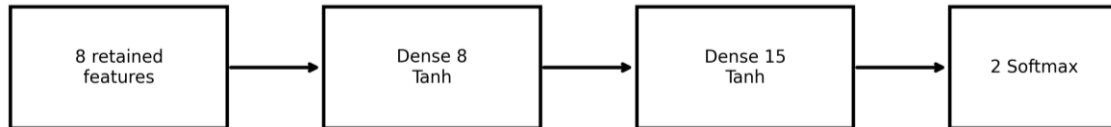
2.4 Data Splitting and Validation

The supplied code uses `train_test_split(..., test_size=0.2, random_state=42)`. During `model.fit()`, `validation_split=0.2` is applied to the training data. Thus the implementation is a train/validation/test procedure, not 10-fold cross-validation. The earlier 10-fold claim has

been removed. The split code does not specify stratify=labels; therefore, deliberate preservation of class proportions cannot be claimed from the code.

2.5 BPNN Architecture

The supplied Keras implementation is Sequential with input_shape equal to the number of retained features, followed by Dense(8,tanh), Dense(15,tanh), and Dense(2,softmax). With eight retained features, the computational architecture is therefore 8 input variables → 8-neuron dense layer → 15-neuron dense layer → 2-node output.



Adam lr=0.001 | 100 epochs | batch=32 | sparse categorical cross-entropy

Figure 2: Implemented BPNN architecture reconstructed from the supplied Keras code

2.6 Adam Optimization

Adam is the optimizer specified in the supplied implementation, with learning rate 0.001 and default settings otherwise. Adam was introduced as a first-order stochastic optimization method using adaptive moment estimates (Kingma & Ba, 2015). Because no matched SGD, Momentum or RMSProp experiments were supplied, the study does not claim that Adam is experimentally superior.

2.7 Training Configuration

Table 2: Model and training configuration

Parameter	Setting
Input variables	8 retained features
Dense layer 1	8 neurons, tanh
Dense layer 2	15 neurons, tanh
Output	2 neurons, Softmax
Optimizer	Adam
Learning rate	0.001
Loss	Sparse categorical cross-entropy
Epochs	100
Batch size	32
Test split	20%
Validation split	20% of training
Random state	42

2.8 Evaluation

The evaluation uses accuracy, precision, recall, F1-score and ROC-AUC. Specificity, false-positive rate and false-negative rate are additionally derived from the reported confusion matrix. The supplied code includes probability-based ROC construction using `y_pred_probs[:,1]`, which is the appropriate basis for a conventional ROC curve. Because raw ROC points were not supplied, the reported AUC=0.992 is retained but not used to fabricate a curve.

2.9 Reproducibility

The source methodology identifies Python 3.x, TensorFlow/Keras, NumPy, Pandas, Matplotlib and Seaborn. It also lists an Intel i5-class processor, at least 8 GB RAM and an optional NVIDIA GTX1050-class GPU. Exact package versions, operating-system build, GPU software

versions, actual hardware, and complete preprocessing code were not supplied and should be archived before final submission.

3. RESULTS AND DISCUSSION

3.1 Confusion Matrix

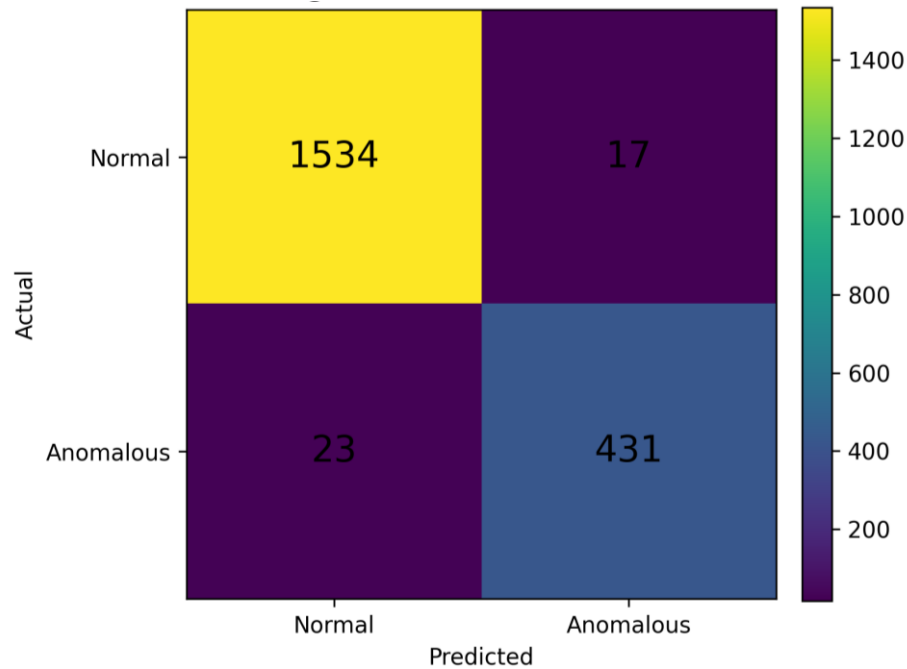


Figure 3: Confusion matrix reconstructed from the reported counts

Table 3: Confusion matrix and totals

	Predicted Normal	Predicted Anomalous	Total
Actual Normal	1534 (TN)	17 (FP)	1551
Actual Anomalous	23 (FN)	431 (TP)	454
Total	1557	448	2005

The reported matrix contains 17 false positives and 23 false negatives. These counts provide the basis for the internally consistent metric calculations below.

3.2 Performance Metrics

Table 4: Performance measures

Metric	Derived value
Accuracy	98.00%
Precision	96.21%
Recall	94.93%
F1-score	95.57%
Specificity	98.90%
False-positive rate	1.10%
False-negative rate	5.07%
Reported ROC-AUC	0.992

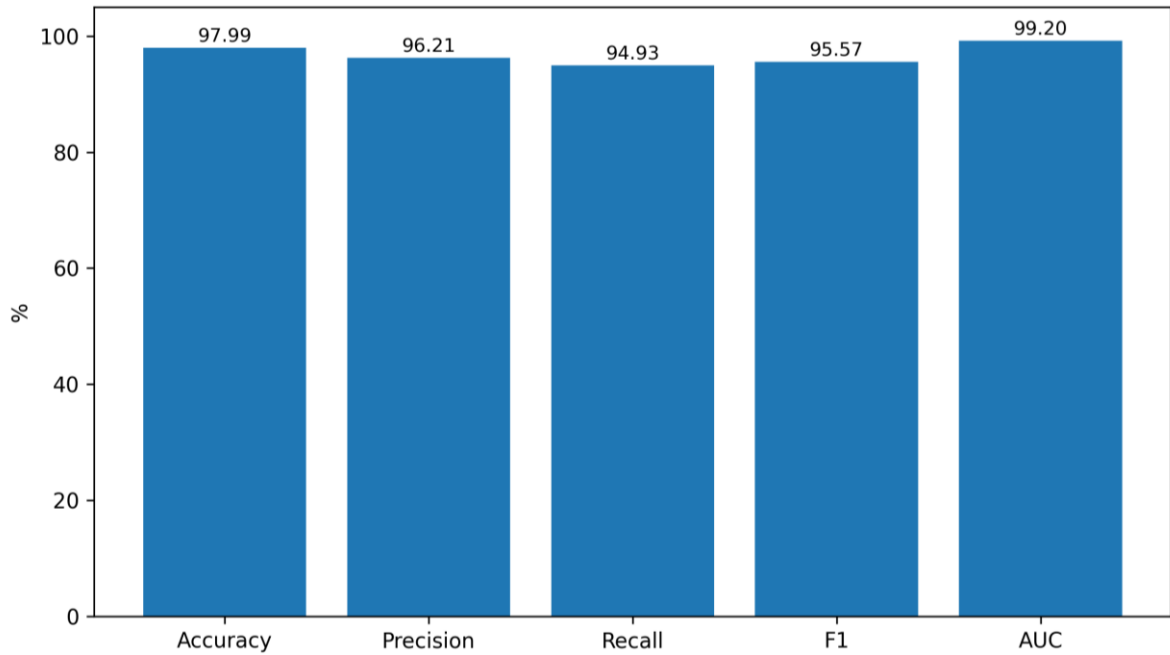


Figure 4: Performance metrics derived from the reported confusion matrix plus the reported AUC.

The confusion matrix yields 97.99% accuracy, 96.21% precision, 94.93% recall and 95.57% F1-score.

3.3 Error Analysis

The false-positive rate is approximately 1.10% because 17 of 1,551 actual normal records were incorrectly labelled anomalous. The false-negative rate is approximately 5.07% because 23 of 454 actual anomalous records were classified as normal.

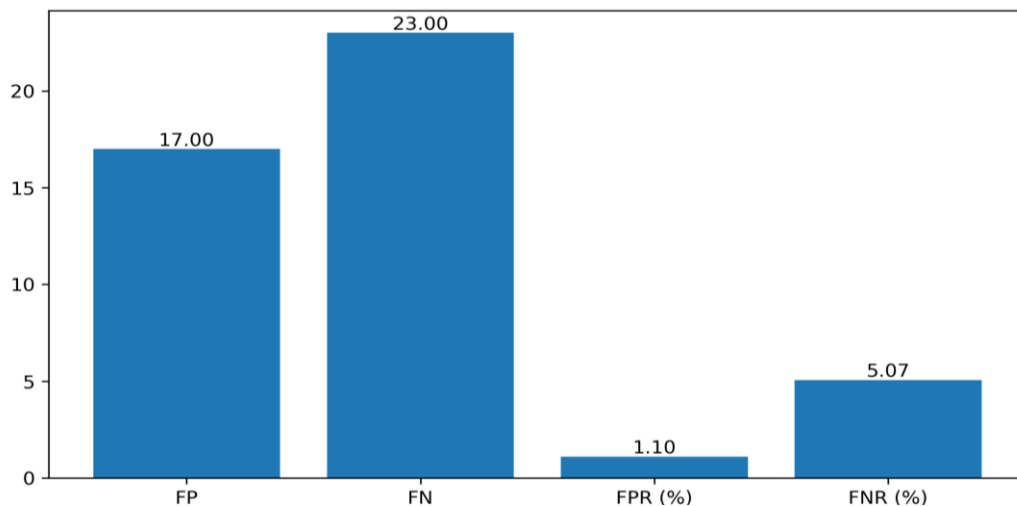


Figure 5: Error profile derived from the reported confusion matrix

3.4 ROC-AUC

The source manuscript reports ROC-AUC=0.992. The probability-based ROC construction in the supplied code is appropriate, but the raw FPR/TPR arrays were not provided. A faithful ROC figure therefore cannot be reconstructed without inventing observations. The original exported ROC figure should be inserted from the actual run before final submission.

3.5 Discussion

The results indicate strong binary-classification performance on the reported NSL-KDD evaluation. Precision of 96.21% indicates that most records classified as anomalous were anomalous in the reported test matrix, while recall of 94.93% indicates that most anomalous records in that matrix were detected. Specificity of 98.90% indicates that normal traffic was usually classified correctly.

The interpretation must remain bounded by the experiment. Only one train-test split and one validation split are documented. There are no repeated independent runs, confidence intervals, statistical tests, or controlled optimizer comparisons. Consequently, the observed performance cannot be presented as statistically superior to other training algorithms.

The earlier manuscript attributed gains to Adam, cross-validation, enhanced preprocessing and scalability. Those causal statements have been removed because the supplied evidence does not isolate their effects. In particular, no 10-fold cross-validation implementation or optimizer ablation was supplied.

NSL-KDD is useful for benchmark comparison but is not a substitute for contemporary operational traffic. Recent literature continues to identify dataset age, class imbalance and differences between benchmark traffic and current networks as important limitations (Yang et al., 2022; Ayantayo et al., 2023). A high benchmark score should therefore not be interpreted as proof of production readiness.

The next experimental step should be a controlled ablation in which the same architecture, features, split and preprocessing are trained with Adam, SGD, Momentum and RMSProp. Multiple random seeds should be used, with mean and standard deviation or confidence intervals reported. Evaluation should also include attack-category results and at least one newer intrusion-detection dataset.

3.6 Comparison with Prior Approaches

The earlier manuscript reported a numerical baseline attributed to a Sabhnani and Serpen-inspired BPNN. The supplied materials do not contain the implementation, execution output, preprocessing details or test predictions needed to verify those values. The baseline is therefore not presented as a controlled experimental comparator.

Published studies are likewise treated as contextual rather than ranked against the present model because they use different datasets, feature sets, partitions, class definitions and architectures. Recent work using NSL-KDD and UNSW-NB15 demonstrates the diversity of current experimental designs (Ayantayo et al., 2023).

Table 5: Contextual comparison without unsupported performance ranking

Source/context	Dataset	Use in revised paper
Yang et al. (2022)	Multiple IDS datasets	Methodological and dataset context
Ayantayo et al. (2023)	NSL-KDD + UNSW-NB15	Recent deep-learning context
Present study	NSL-KDD	Primary BPNN-Adam experiment

3.7 Limitations

NSL-KDD is an established but dated benchmark and does not establish performance on contemporary enterprise, cloud, IoT or encrypted traffic.

Chapter 3 reports 34,144 training and 8,571 test instances, while the Chapter 4 confusion matrix contains 2,005 observations; this must be reconciled.

The exact feature-selection and preprocessing code was not supplied.

No controlled optimizer ablation, repeated-run uncertainty analysis or statistical test was supplied.

No DoS, Probe, R2L and U2R class-specific results were supplied.

No inference-time, throughput or memory experiment was supplied.

Raw ROC coordinates and epoch-level training history were not supplied, so those figures must be exported from the original run rather than reconstructed.

3.8 Contribution to Knowledge

The study contributes an explicitly documented compact BPNN formulation for binary network-anomaly classification using eight retained NSL-KDD traffic features and Adam optimization. Its methodological contribution is the clearer specification of architecture, training parameters, split protocol and error measures, together with a disciplined separation between benchmark evidence and untested claims such as optimizer superiority, zero-day detection, scalability and deployment readiness.

4.0 CONCLUSION

A compact BPNN trained with Adam was evaluated for binary anomaly detection on NSL-KDD. The reported confusion matrix yields 97.99% accuracy, 96.21% precision, 94.93% recall, 95.57% F1-score, 98.90% specificity, 1.10% false-positive rate and 5.07% false-negative rate. The reported ROC-AUC of 0.992 is retained subject to verification from the original probability outputs.

The evidence supports strong benchmark performance for the implemented run, but it does not establish statistical superiority over alternative optimizers, modern-network generalization, zero-day capability or production readiness. Before final submission, the sample-count discrepancy should be reconciled, preprocessing and feature-selection code archived, original ROC/training plots exported, and the model evaluated using repeated runs, optimizer ablation, attack-category analysis and newer datasets.

REFERENCES

- Ayantayo, A., Kaur, A., Kour, A., Schmoor, X., Shah, F., Vickers, I., Kearney, P., & Abdelsamea, M. M. (2023). Network intrusion detection using feature fusion with deep learning. *Journal of Big Data*, 10, 167. <https://doi.org/10.1186/s40537-023-00834-0>
- Awad, A. A., Ali, A. F., & Gaber, T. (2023). An improved long short term memory network for intrusion detection. *PLOS ONE*, 18(8), e0284795. <https://doi.org/10.1371/journal.pone.0284795>
- Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.
- Tavallaei, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 1–6.
- Yang, Z., Liu, X., Li, T., Wu, D., Wang, J., Zhao, Y., & Han, H. (2022). A systematic literature review of methods and datasets for anomaly-based network intrusion detection. *Computers & Security*, 116, 102675. <https://doi.org/10.1016/j.cose.2022.102675>
- Investigating novel machine learning based intrusion detection models for NSL-KDD data sets. (2023). 2023 International Conference on IT and Industrial Technologies. <https://doi.org/10.1109/ICIT59216.2023.10335831>

APPENDIX A: CORE IMPLEMENTATION EXCERPT

```
X_train, X_test, y_train, y_test = train_test_split(
    features_scaled[selected_features], labels,
    test_size=0.2, random_state=42)

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(8, input_shape=(X_train.shape[1],), activation='tanh'),
    tf.keras.layers.Dense(15, activation='tanh'),
```

```
tf.keras.layers.Dense(2, activation='softmax')
])

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

history = model.fit(
    X_train, y_train,
    epochs=100, batch_size=32, validation_split=0.2)

y_pred_probs = model.predict(X_test)
y_pred_classes = y_pred_probs.argmax(axis=1)
fpr, tpr, _ = roc_curve(y_test, y_pred_probs[:, 1])
roc_auc_value = auc(fpr, tpr)
```