Machine learning for intrusion detection using CIC-IDS2017

Ruturaj Jayant Kotwal
MSc Artificial Intelligence and Robotics
Hof University of Applied Sciences
Hof, Bavaria, Germany

Advait Santosh Lanjekar
MSc Artificial Intelligence and Robotics
Hof University of Applied Sciences
Hof, Bavaria, Germany

***Abstract* — As cyber threats grow increasingly sophisticated and frequent, there is a pressing need for advanced Intrusion Detection Systems (IDSs) capable of accurately detecting malicious network activity. In this study, we use deep learning techniques to improve IDS performance, overcoming the challenges faced by traditional signature-based methods in identifying emerging and zero-day attacks. Leveraging the CIC-IDS2017 dataset, we employ Multilayer Perceptrons (MLPs) in Keras to classify benign & malicious network traffic under both binary and multiclass configurations. By integrating efficient feature selection using the XGBoost algorithm, we achieve state-of-the-art accuracy (~99%), all while optimizing for minimal memory usage. This work highlights the potential of MLPs and related techniques as powerful tools in the ongoing effort to strengthen cybersecurity defenses.**

***Keywords* — Intrusion Detection System (IDS), CIC-IDS2017, Cybersecurity, Machine Learning, Deep Learning, Multilayer Perceptrons (MLPs), Feature Selection, XGBoost, Binary Classification, Multiclass Classification, Zero-Day Attacks.**

I. INTRODUCTION

1. Problem definition

The rapid growth of the internet, particularly with the rise of IoT and Software-Defined Networks (SDN) [1], has resulted in a considerable increase in cyber-security threats. IoT devices, despite enhancing productivity, are extremely vulnerable to assaults due to inherent security flaws. Traditional signature-based IDSs struggle to detect novel or modified attacks, while anomaly-based IDSs often suffer from high false alarm rates [2].

2. Study goal

Machine Learning (ML) and its subset, Deep Learning (DL), provide effective techniques for detecting sophisticated network assaults such as zero-day threats and Distributed Denial-of-Service (DDoS). These techniques enable Intrusion Detection Systems (IDSs) to recognize patterns in network traffic that indicate malicious behavior [2].

This study focuses on using MLPs for both binary and multiclass classification of network traffic, leveraging the CIC-IDS2017 dataset [3] as a benchmark. To boost detection accuracy and reduce computational complexity, the XGBoost algorithm [4] is used for feature selection, ensuring the models concentrate on the most relevant features. Additionally, the SMOTE algorithm is employed to address class imbalance, further enhancing the model's ability to effectively detect malicious activities.

3. Contributions of the paper

This study makes the following contributions to the field of intrusion detection:

3.1. High Detection Accuracy: Achieved high accuracy in both binary and multiclass classification of network traffic using Multilayer Perceptrons.

3.2. Efficient Feature Selection: Applied XGBoost to extract the most critical features, enhancing model accuracy while reducing computational overhead.

3.3. Improved Class Imbalance Handling: Addressed class imbalance using SMOTE, ensuring better detection of minority attack classes.

3.4. Advanced Evaluation Metrics: Used advanced metrics like False Alarm Ratio (FAR) and Attack Miss Ratio (AMR) to provide a detailed evaluation of model performance.

II. STATE OF THE ART

Previous approaches to IDSs using ML have aimed to address challenges such as data imbalance, feature representation, and the high dimensionality of network traffic data. A novel ML-based IDS model introduced in recent research [5] tackles these issues through a combination of techniques, including Random Oversampling (RO) for balancing datasets, feature embedding based on clustering methods like K-means and Gaussian Mixture (GM), and Principal Component Analysis (PCA) for dimensionality reduction. The model was evaluated using three benchmark datasets: UNSW-NB15 [6], CIC-IDS2017, and CIC-IDS2018 [7, 8], achieving exceptional accuracy rates. On the CIC-IDS2017 dataset, Decision Trees (DT), Random Forest (RF), and Extra Trees (ET) achieved up to 99.99% accuracy, demonstrating the model's ability to outperform existing state-of-the-art methods and handle both binary and multilabel classification tasks [5].

The study [5] highlights the importance of feature selection and dimensionality reduction in improving IDS performance. By employing techniques like PCA and embedding nuanced patterns through clustering, the proposed model reduces computational complexity while maintaining high accuracy. Additionally, the integration of multiple machine learning algorithms, such as DT, RF, ET, and XGBoost, allowed the model to effectively leverage the strengths of each technique, improving detection rates and minimizing false positives. While the research makes substantial advancements, such as improving accuracy by up to 2.99% on CIC-IDS2017, it acknowledges the absence of deep learning methods,

leaving an opportunity for future work to incorporate models like Recurrent Neural Networks (RNNs) and Deep Neural Networks (DNNs) to further enhance detection capabilities in complex and dynamic network environments [5].

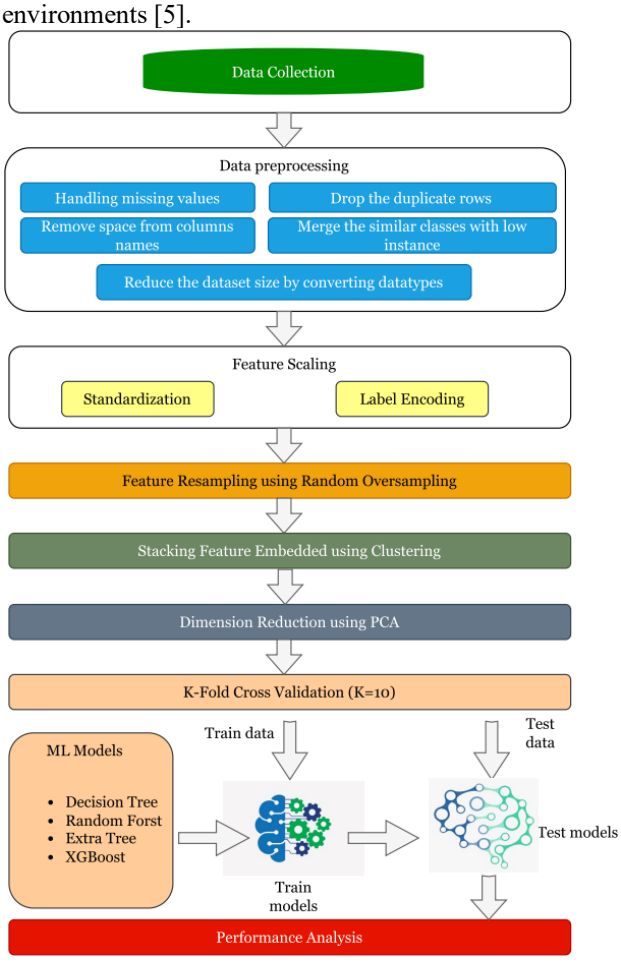

Figure 1. Proposed framework by [5] for stacking feature embedded with PCA for intrusion detection.

III. APPROACH

1. Research objective

This study aims to improve the performance of IDSs by leveraging MLPs for binary and multiclass classification of network traffic using the CIC-IDS2017 dataset. The study focuses on achieving high accuracy while optimizing computational efficiency.

2. Key hypotheses

2.1. Feature Selection with XGBoost improves model accuracy and reduces computational complexity by focusing on the most relevant features.

2.2. SMOTE effectively addresses class imbalance, enhancing the detection of minority attack classes.

2.3. Multilayer Perceptrons (MLPs) can achieve state-of-the-art accuracy (~99%) in both binary and multiclass classification tasks.

3. Dataset overview

The CIC-IDS2017 dataset was created to overcome the drawbacks of previous intrusion detection datasets, such as DARPA98, KDD99, and ADFA13, which have obsolete attack types, insufficient traffic variety, anonymized payloads, and incomplete feature sets. This dataset includes benign traffic and seven updated attack categories, such as brute force, DoS, DDoS, infiltration, and botnet attacks, reflecting real-world network conditions. It was generated using realistic traffic profiles and meets 11 essential criteria for modern IDS evaluation [3].

The dataset includes 80 network traffic features extracted with the CICFlowMeter [9] tool, ensuring complete coverage of packet-level details. These features were examined using machine learning methods to determine which subsets are most successful at detecting various attack types. Publicly available and widely used, CIC-IDS2017 has become a benchmark for intrusion detection research, offering diverse, high-quality data for developing and testing IDS models [3].

4. Dataset preparation

4.1. Importing the CIC-IDS2017 Dataset

The dataset is loaded from an Apache Parquet [10] file into a Pandas DataFrame. It has been meticulously cleaned and preprocessed by [11]. Furthermore, all metadata have been removed [12] to eliminate potential shortcut learning issues.

4.2. Data Preprocessing

To prepare the dataset for modeling, missing and duplicate entries were removed, and the target variable (Label) was configured for both binary and multiclass classification. In the binary setup, benign traffic was labeled as 0 and malicious as 1. For multiclass classification, attack types were encoded with unique integer labels.

A significant class imbalance was observed, with benign traffic comprising 84.92% of the binary dataset. A naive model predicting only the majority class would achieve a baseline accuracy of 84.92%, highlighting the need for advanced models to achieve meaningful results. Visualizations confirmed the imbalance, emphasizing the challenge for effective classification.

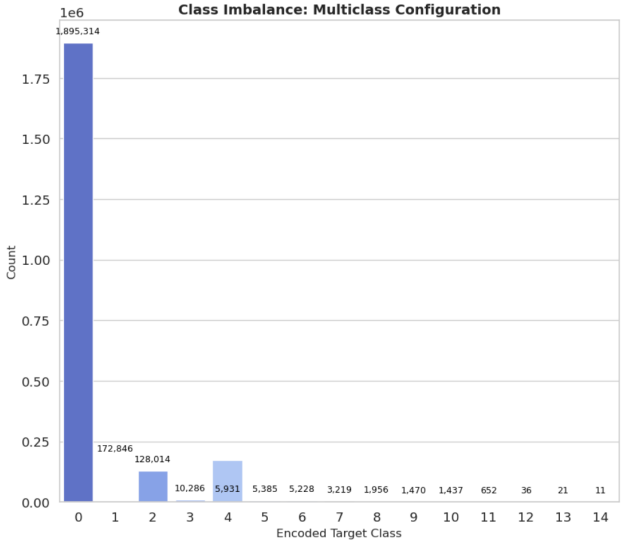

Figure 2. Multiclass target distribution.

4.3. Splitting the Dataset

To ensure robust model training and evaluation, the dataset was divided into three subsets: training (75%), validation (10%), and test (15%). Effective hyperparameter tuning, performance measurement, and assessment of the model's capacity to generalize unknown data are made possible by this division.

A custom function, *extractAllSets*, was employed for this process. This function leverages the *train_test_split* method [13] twice to achieve the desired

proportions. Importantly, stratified splitting was used to maintain the class distribution across all subsets, ensuring that the imbalance observed in the original dataset is reflected in each partition. This preserves consistency and avoids biased performance estimates.

Reproducibility was ensured by using a fixed random seed and enabling dataset shuffling for improved training dynamics.

4.4. Feature Selection with XGBoost

XGBoost [4], a powerful tree-based ensemble method, was used for selecting features. By determining feature significance scores, which measure each feature's contribution to the model's prediction performance, XGBoost automatically selects features during the training phase. When used to split nodes in decision trees, these scores are determined by how much a feature lowers Gini impurity [14]. Features that have the greatest impact on impurity reduction are deemed more significant.

The Gini impurity is a measure of how mixed the classes are in a dataset. For a binary classification problem with classes 0 and 1, the Gini impurity (G) for a node with N samples is calculated as:

$$G = 1 - \sum_{i=0}^{1} p_i^2$$

where p_i is the probability of class i in the node [14].

In the context of XGBoost, the feature that most effectively reduces this impurity is chosen at each decision tree node, and the importance score of that feature is derived from the total reduction in impurity across all trees in the ensemble [14]. Features with importance scores above 0.01 were selected. Feature selection was conducted separately for binary and multiclass classification tasks, with the most important features identified and retained for each configuration.

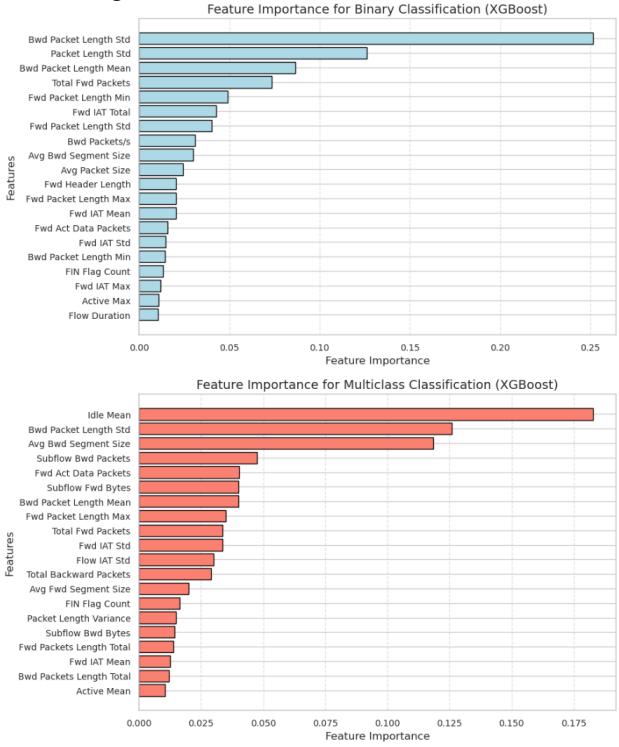

Figure 3. Feature Importance Comparison for Binary and Multiclass Classification Using XGBoost.

Out of 77 initial features, 20 were selected based on their importance scores above 1%, reducing the dataset's dimensionality and focusing on the most relevant features for modeling.

4.5. Feature Engineering

Feature engineering is a critical step in improving model performance by transforming raw data into useful features for prediction. In this study, feature engineering is applied exclusively to the binary classification case after performing feature selection.

4.5.1. New Features

4.5.1.1. Combined Feature Importance

This feature is generated by calculating a weighted sum across related features and allocating weights to features according to their significance. This helps capture the overall relevance of selected features.

4.5.1.2. Feature Ratios and Differences

New features are derived by calculating the difference between related features, such as the difference in packet lengths (*Bwd Packet Length Std* vs. *Packet Length Std*), or the difference in packet counts between forward and backward directions.

4.5.1.3. Interaction Features

This feature captures interactions between features, such as the product of *Bwd Packets/s* and *Flow Duration*. This interaction term helps capture the combined effect of these features.

4.5.2. Secondary Evaluation

After generating the new features, a secondary evaluation is performed to assess their effectiveness using the same XGBoost feature selection approach as before. The features are evaluated against a threshold of 1% importance to identify the most relevant ones. The feature engineering process successfully added 4 new features, with two being selected by XGBoost for further modeling. The feature set was refined, keeping 20 significant features for the binary configuration, while discarding features like *Fwd Act Data Packets* and *Fwd Packet Length Min*.

4.6. Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a statistical approach used to analyze datasets, summarize their key characteristics, and uncover patterns or anomalies, often through visualizations and statistical methods. In this study, EDA was performed automatically using the *ProfileReport* module from the *ydata_profiling* library [15]. This tool provides a comprehensive summary of the dataset, streamlining the exploration process [16].

While EDA is typically performed at the beginning of the data preparation pipeline (before feature selection and dataset splitting), for clarity and readability, this report examines only the final 20 features that emerged after feature selection and feature engineering. Analyzing all 77 initial features, or features excluded in earlier stages, is neither practical nor necessary for this report.

4.7. SMOTE

SMOTE (Synthetic Minority Oversampling Technique) [17] is applied in the binary configuration of the dataset to address class imbalance. In the CIC-IDS2017 dataset, the class distribution is highly skewed, with 84.92% of samples representing benign activity (Class 0) and 15.08% representing malicious activity (Class 1). This imbalance can lead to poor detection of rare attack types, as machine learning models tend to favor the majority class.

4.7.1. Key steps in SMOTE

- Identifying the minority class.
- Selecting a data point: A random sample from the minority class is chosen.
- Finding nearest neighbors: Using the k-Nearest Neighbors (k=5), SMOTE identifies other similar samples in the minority class.
- Generating synthetic data: A new synthetic point is created between the chosen data point and one of its nearest neighbors. This is done by adding a scaled difference vector (using a random number λ between 0 and 1) to the original data point:

$$s = x + \lambda \cdot (y - x)$$

Here, x is the selected point, y is a neighbor, and λ is randomly chosen.

4.7.2. Implementation

- Sampling strategy: A sampling ratio of 0.5 was used, meaning the minority class size becomes half of the majority class size.
- Data leakage prevention: SMOTE is applied only to the training set. The validation and test sets remain unchanged to reflect the real-world distribution, ensuring unbiased evaluation and reliable model performance.

4.7.3. Limitations

- It assumes that synthetic data points adequately represent the true complexity of the minority class.
- The generated data may not account for nuanced patterns in the minority class distribution.

5. Binary Classification

This section discusses using Multilayer Perceptrons (MLPs) for binary classification [18] to differentiate between benign and malicious network data.

5.1. Overview of Models

Five MLP models were developed with varying sizes. Despite differences in architecture, all models share a consistent data processing pipeline and identical hyperparameters (e.g., optimizer, learning rate scheduler). The models were trained and evaluated using multiple metrics. Developers can choose between models based on the specific requirements of accuracy versus memory constraints.

5.2. Weight initialization

Two initialization techniques were employed to stabilize training:

5.2.1. Glorot Initialization : Prevents vanishing/exploding gradients by sampling weights from a uniform distribution based on input and output units. Suitable for activation functions like tanh and sigmoid [19].

5.2.2. He Initialization: Tailored for ReLU activations, samples weights from a normal distribution with variance proportional to input units, addressing gradient vanishing issues [19].

5.3. Model architectures

The models range from minimal to maximal complexity:

- Minimal model (Model 1): Features 4 layers with 8 to 2 units and uses Glorot and He initializations.
- Maximal model (Model 5): Has 9 layers with 256 to 2 units, allowing for higher accuracy but requiring more memory.
- Intermediate models (Models 2–4) progressively scale in size.

Each model uses tanh, selu, and sigmoid activations with initialization optimized for their respective roles.

5.4. Training details

- Hyperparameters: Pre-tuned, including the Adam optimizer with a binary cross-entropy loss function for binary classification tasks.
- Early Stopping: Monitors validation accuracy, halting training after 8 epochs of no improvement and restoring the best weights.
- Learning Rate Scheduling: Adjusts the learning rate dynamically for better performance.
- Batch Size: Models performed best with a batch size of 1024.

5.5. Standard Evaluation

Model	Test Loss	Test Accuracy	Test Precision	Test Recall	Test F1 Score
model_1_b	0.1607	94.21%	76.96%	87.91%	82.07%
model_2_b	0.1448	94.57%	77.98%	89.17%	83.20%
model_3_b	0.1029	96.28%	86.34%	89.47%	87.88%
model_4_b	0.0953	96.92%	87.03%	93.47%	90.14%
model_5_b	0.0414	98.77%	94.93%	96.99%	95.95%

Table 1. The evaluation of models was performed using widely recognized metrics.

5.6. Advanced evaluation metrics

- False Alarm Ratio (FAR): Proportion of false positives among all predicted positives, emphasizing the model's reliability [20].
- Attack Miss Ratio (AMR): Proportion of missed attacks among all actual negatives, critical for evaluating detection efficiency [20].

Model	Test FAR	Test AMR
model_1_b	0.0467	0.1209
model_2_b	0.0447	0.1083
model_3_b	0.0251	0.1053
model_4_b	0.0247	0.0653
model_5_b	0.0092	0.0301

Table 2. Confusion matrix was used to derive FAR and AMR for each model.

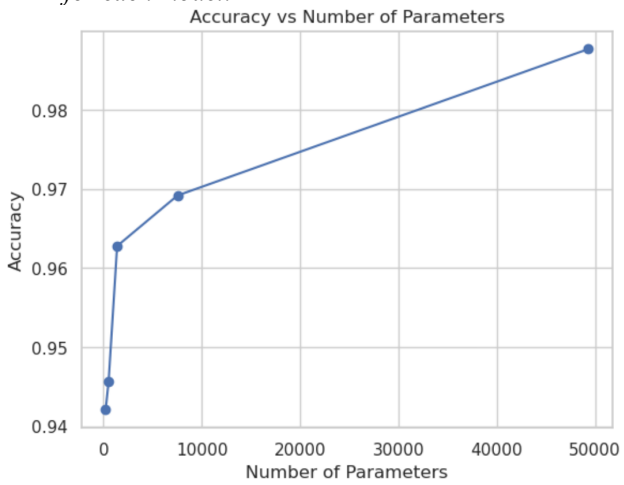

Figure 4. A trade-off between accuracy and model size (number of parameters) was analyzed. As the number of parameters increased, accuracy improved.

5.7. Latency

Latency, the time taken by the model to make predictions on test samples, was measured.

Model	Latency (ms)
model_1_b	0.039
model_2_b	0.040
model_3_b	0.040
model_4_b	0.041
model_5_b	0.043

Table 3. All models demonstrated similar latency, with differences being negligible in milliseconds. It may not influence decision-making.

6. Multiclass classification

In multiclass classification, the objective is to differentiate between one benign class and multiple attack types. This setup demands higher complexity in model design to ensure fine-grained detection, as misclassifications between attack types can lead to ineffective or inappropriate mitigation actions.

6.1. Model overview

Five neural network architectures were developed, ranging from minimal to maximal complexity:

6.1.1. Model_1_m (Minimal): A simple three-layer model with 863 parameters.

6.1.2. Model_2_m: Adds one hidden layer, increasing parameters to 1,135.

6.1.3. Model_3_m: Introduces a third hidden layer, totaling 1,407 parameters.

6.1.4. Model_4_m: A deeper architecture with four hidden layers and 1,679 parameters.

6.1.5. Model_5_m (Maximal): The most complex design with five hidden layers and 5,391 parameters.

All models use *tanh* and *selu* activations, softmax output, and progressive scaling to balance complexity and performance. The procedures of binary classification are repeated, adjusted for the multiclass configuration.

6.2. Evaluation metrics across models

Metric	Model_1_m	Model_2_m	Model_3_m	Model_4_m	Model_5_m
Test Loss	0.3042	0.3160	0.2895	0.2099	0.2106
Test Accuracy	90.47%	88.21%	88.35%	92.59%	93.00%
Precision	90.30%	88.05%	87.28%	93.31%	92.82%
Recall	90.47%	88.21%	88.35%	92.59%	93.00%
F1 Score	89.55%	87.44%	87.26%	92.56%	92.75%
FAR	0.0365	0.0532	0.0297	0.0458	0.0310
AMR	0.4034	0.4223	0.4399	0.1500	0.2249

6.3. Parameter comparison

Model	Total Parameters	Trainable Parameters	Non-trainable Parameters	Memory Usage	Latency (ms)
Model_1_m	863	863	0	3.37 KB	0.038
Model_2_m	1135	1135	0	4.43 KB	0.039
Model_3_m	1407	1407	0	5.50 KB	0.040
Model_4_m	1679	1679	0	6.56 KB	0.041
Model_5_m	5391	5391	0	21.06 KB	0.042

6.4. Key observations

- Model_5_m achieves the best accuracy (93.00%) but with 6.25 times more parameters than Model_1_m.
- Latency differences are minimal across all models, making memory usage the primary consideration for deployment.

6.5. Accuracy vs Number of parameters

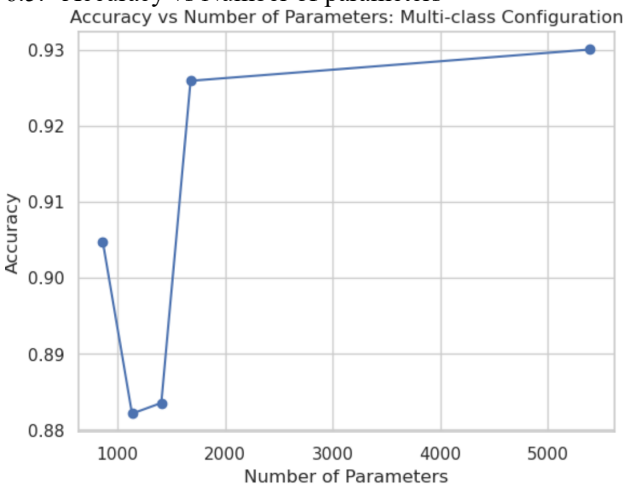

Figure 5. The trade-off between accuracy and model complexity is visualized in the graph.

6.6. Visualization of training and validation metrics

The visualization of training and validation loss and accuracy over epochs provides critical insights into the learning behavior of each model. By observing these plots, we can assess how well the models generalize to unseen data, identify signs of overfitting or underfitting, and evaluate the effectiveness of the chosen architectures. The comparison between training and validation curves highlights the performance consistency across all models during the learning process.

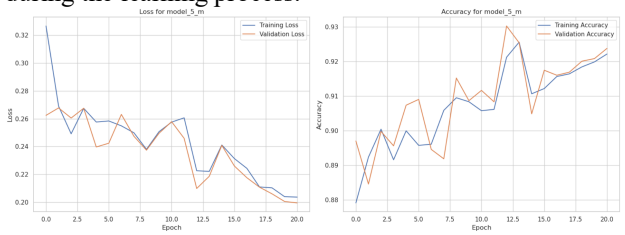

Figure 6. Training and Validation Loss & Accuracy over epochs for model_5_m. This visualization supports model selection by showcasing convergence rates and stability across epochs.

6.7. Comparison to binary classification

While binary classification achieves better accuracy levels, multiclass classification offers several added advantages:

- Granularity: Ability to identify the specific attack type.
- Actionable insights: Enables tailored responses to different attacks.
- Complexity: Greater challenge in decision boundaries, reflected in the higher resource usage.

6.8. Conclusion

Model_5_m demonstrates good performance in multiclass classification, with the highest accuracy (93.00%) and the lowest FAR (0.0310) and AMR (0.2249). Despite the trade-off in memory usage, its latency remains efficient (0.042 ms). This model is recommended for systems prioritizing precision and recall over resource constraints, while Model_1_m provides an excellent lightweight alternative.

IV. FUTURE OUTLOOK

Future improvements can be achieved by integrating advanced scaling techniques to normalize

input data more effectively, potentially improving model generalization. Outlier analysis could enhance data preprocessing, removing noise and anomalies that may affect model training.

Also, using Explainable AI (XAI) approaches may provide more in-depth insights into model decision-making, increasing cybersecurity experts' trust and interpretation. Together, these approaches have the potential to increase detection accuracy, reduce false alarms, and provide more actionable information about system vulnerabilities.

V. CONCLUSION

This study focused on designing and evaluating neural network architectures for intrusion detection using both binary and multiclass classification. The models ranged from minimal to maximal complexity, balancing trade-offs between accuracy, memory usage, and latency. Results showcased state-of-the-art performance, with the highest accuracy reaching 98.77% for binary classification tasks. The evaluation included advanced metrics such as False Alarm Ratio (FAR) and Attack Miss Ratio (AMR), emphasizing practical applicability in real-world cybersecurity scenarios. Overall, the models demonstrated robust performance and scalability while maintaining low latency and efficient resource utilization.

REFERENCES

[1] Sultana, N., Chilamkurti, N., Peng, W. et al. Survey on SDN based network intrusion detection system using machine learning approaches. Peer-to-Peer Netw. Appl. 12, 493–501 (2019). https://doi.org/10.1007/s12083-017-0630-0

[2] Mohanda Sarhan, Siamak Layeghy, Nour Moustafa, Marcus Gallagher, Mariuss Portmann on Feature extraction for machine learning – based intrusion detection in IOT networkshttps://www.sciencedirect.com/science/article/pii/S2352864822001754

[3] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”, 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018.

[4] Tianqi Chen, Carlos Guestrin on XGBoost: A Scalable Tree Boosting System https://arxiv.org/abs/1603.02754

[5] Talukder, M.A., Islam, M.M., Uddin, M.A. et al. Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction. J Big Data 11, 33 (2024). https://doi.org/10.1186/s40537-024-00886-w

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

[7] University of Brunswick on CSE-CIC-IDS2018 on AWShttps://www.unb.ca/cic/datasets/ids-2018.html

[8] A Realistic Cyber Defense Datasethttps://registry.opendata.aws/cse-cic-ids2018/

[9] Habibi Lashkari, Arash. (2018). CICFlowmeter-V4.0 (formerly known as ISCXFlowMeter) is a network traffic Bi-flow generator and analyser for anomaly detection. https://github.com/ISCX/CICFlowMeter. 10.13140/RG.2.2.13827.20003.

[10] https://parquet.apache.org/

[11] Laurens D ‘Hooge on Flow-based Intrusion Detection Datasethttps://www.kaggle.com/datasets/dhoogla/cicids2017

[12] D’hooge, L., Verkerken, M., Volckaert, B., Wauters, T., De Turck, F. (2022). Establishing the Contaminating Effect of Metadata Feature Inclusion in Machine-Learned Network Intrusion Detection Models. In: Cavallaro, L., Gruss, D., Pellegrino, G., Giacinto, G. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2022. Lecture Notes in Computer Science, vol 13358. Springer, Cham. https://doi.org/10.1007/978-3-031-09484-2_2

[13] Skitlearn.org on train test split https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[14] Sam Black on Calculating a features importance with Gini Importance https://sam-black.medium.com/calculating-a-features-importance-with-xgboost-and-gini-impurity-3beb4e003b80

[15] YData Profiling https://docs.profiling.ydata.ai/latest/

[16] Satyam Tripathi on Pandas Profiling (ydata-profiling) in Python: A guide for beginnershttps://www.datacamp.com/tutorial/pandas-profiling-ydata-profiling-in-python-guide

[17] N. V Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer on SMOTE: Synthetic Minority over-sampling technique https://www.jair.org/index.php/jair/article/view/10302

[18] Neri Van Otten on Multilayer perceptron explained and how to train and optimize MLPs https://spotintelligence.com/2024/02/20/multilayer-perceptron-mlp/

[19] Sanjay Dutta on understanding glorot and hHe initialization: A guide for students https://medium.com/@sanjay_dutta/understanding-glorot-and-he-initialization-a-guide-for-college-students-00f3dfae0393

[20] Short tutorial on multi-class Metrics https://www.kaggle.com/code/nolovelost/short-tutorial-on-multi-class-metrics#4.-Specialization:-Intrusion-Detection-System

[21] Laurens D’hooge, Miel Verkerken Bruno Volckaert, Tim Wauters and Filip De Turck https://link.springer.com/chapter/10.1007/978-3-031-09484-2_2

[22] Cody Marie Wild on One Feature Attribution Method to (Supposedly) Rule Them All:h Shapley Values ttps://towardsdatascience.com/one-feature-attribution-method-to-supposedly-rule-them-all-shapley-values-f3e04534983d

[23] Quantile functions https://bookdown.org/kevin_davisross/probsim-book/quantile-functions.html

[24] CIC-IDS-2017-MLPs-v.1.0 https://www.kaggle.com/code/nolovelost/cic-ids-2017-mlps-v-1-0