Laboratory 1 Report

ANDREA BOTTICELLA[*], s347291, Politecnico di Torino, Italy
ELIA INNOCENTI[*], s345388, Politecnico di Torino, Italy
SIMONE ROMANO[*], s344024, Politecnico di Torino, Italy

This work presents the development of Feed-Forward Neural Networks (FFNNs) for intrusion detection using the CICIDS2017 dataset. Through six progressive tasks, we address data cleaning, feature bias, class imbalance, architectural design, and overfitting control. The final model—a three-layer FFNN (128-64-32) with ReLU activation, AdamW optimization, and Batch Normalization—achieved 96% accuracy and 0.94 macro-F1. Results highlight that bias mitigation, weighted loss optimization, and regularization are essential for reliable AI-based IDS capable of detecting diverse network attacks in real-world cybersecurity contexts.

1 INTRODUCTION

This laboratory explores the implementation of a **Feed-Forward Neural Network (FFNN)** using the **CICIDS2017** dataset, a standard benchmark for intrusion detection research. The goal is to construct a complete machine learning pipeline in PyTorch—from raw data preparation to model evaluation—to analyze how architectural choices and preprocessing strategies affect classification performance in cybersecurity contexts.
The experiment unfolds through six progressive tasks that build complexity systematically. Beginning with **data preprocessing**—encompassing cleaning, scaling, and outlier management—the work advances to **baseline FFNN training** using a single hidden layer architecture. Subsequently, **feature bias analysis** examines the influence of specific attributes like *Destination Port*, followed by **loss-function weighting** to address class imbalance challenges. The investigation then explores **deep network optimization** through architectural and optimizer variations, culminating in **regularization** strategies including dropout, batch normalization, and weight decay techniques.

2 TASK 1 — DATA ANALYSIS AND PREPROCESSING

2.1 Dataset Overview

The raw dataset contained **31,507 samples** and **17 features**, including numerical flow statistics and a categorical label identifying traffic types: *Benign* (≈ 63% of the samples), *DoS Hulk*, *PortScan*, and *Brute Force*. A class distribution plot (Figure 1) visually confirmed this imbalance, motivating the use of **stratified splitting** later in the pipeline.

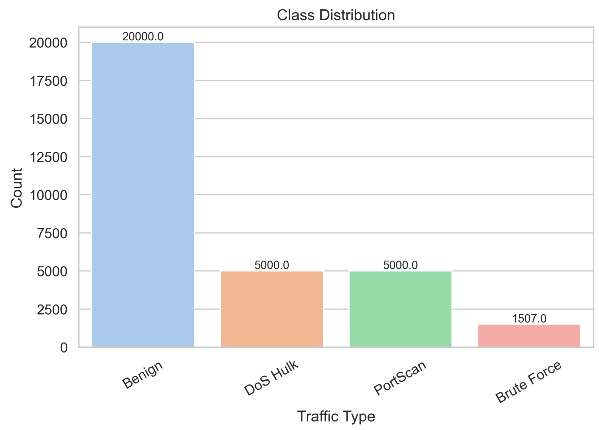


Fig. 1. Class distribution in the raw dataset.

Step	Removed	Remaining
Drop NaN	20	31,487
Drop duplicates	2,114	29,393
Drop infinite values	7	**29,386**

Table 1. Summary of the raw dataset.

Before any training, the dataset must be cleaned and normalized to ensure the model learns meaningful statistical relationships rather than artifacts of data noise or imbalance. The preprocessing pipeline addresses three critical aspects: **data quality** through the removal of missing, duplicate, and infinite values; **class imbalance** by preserving

[*]The authors collaborated closely in developing this project.

Authors' Contact Information: Andrea Botticella, andrea.botticella@studenti.polito.it, s347291, Politecnico di Torino, Turin, Italy; Elia Innocenti, elia.innocenti@studenti.polito.it, s345388, Politecnico di Torino, Turin, Italy; Simone Romano, simone.romano2@studenti.polito.it, s344024, Politecnico di Torino, Turin, Italy.

Publication date: November 2025.

proportional representation during splitting; and **feature scaling** to standardize feature ranges and stabilize neural training convergence.

2.2 Data Cleaning and Partitioning

Table 1 summarizes the cleaning process: duplicate rows, missing values (NaN), and infinite entries were systematically removed. Thus, the dataset was reduced to **29,386 samples**, meaning **2,121 rows** were discarded (2,114 between missing and duplicates, and 7 infinite values). Categorical labels were then encoded numerically: *Benign* = 0, *Brute Force* = 1, *DoS Hulk* = 2, *PortScan* = 3.

The cleaned data were divided into training (60%, 17,631 samples), validation (20%, 5,877 samples), and test (20%, 5,878 samples) subsets using **stratified sampling** with a fixed seed, ensuring consistent class proportions and reproducibility. Exploration of numerical attributes revealed high variability and heavy-tailed distributions (Figures 2), indicating the presence of outliers and the need for normalization.

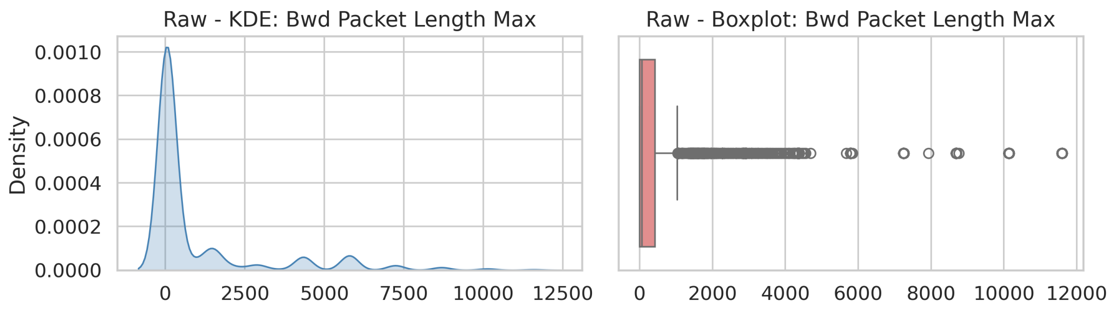


Fig. 2. Examples of feature (Bwd Packet Length Max) kernel density plot and boxplot, highlighting outliers.

2.3 Outlier Detection and Normalization

Outliers were analyzed through the **Z-score** and **IQR** methods. Both confirmed extreme values in features such as *Bwd Packet Length Max*, *Flow Duration*, and *Fwd IAT Std*. Outliers were retained to preserve data realism, and normalization was applied to reduce their effect.

Two scaling methods were tested: **StandardScaler** and **RobustScaler**. The density comparison plots (Figure 3) showed that both methods effectively normalized distributions, but RobustScaler produced more compact, less skewed curves for highly variable features. However, the numerical statistics and subsequent training experiments revealed negligible performance difference between the two. Therefore, **StandardScaler** was ultimately adopted for simplicity and interpretability, offering smoother loss curves in preliminary trials.

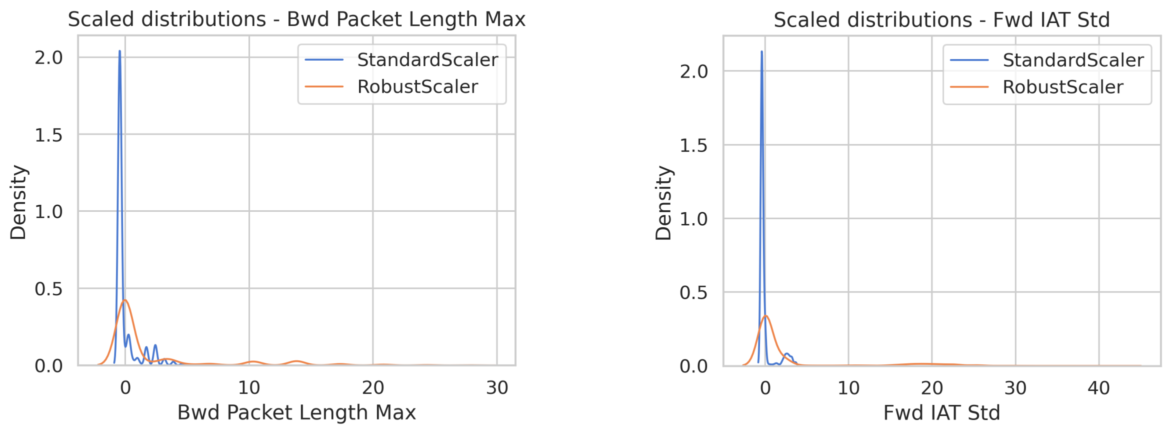


Fig. 3. Comparison of normalization effects using StandardScaler and RobustScaler.

This preprocessing ensured consistent, balanced, and properly scaled data, forming a robust foundation for training the Feed-Forward Neural Network.

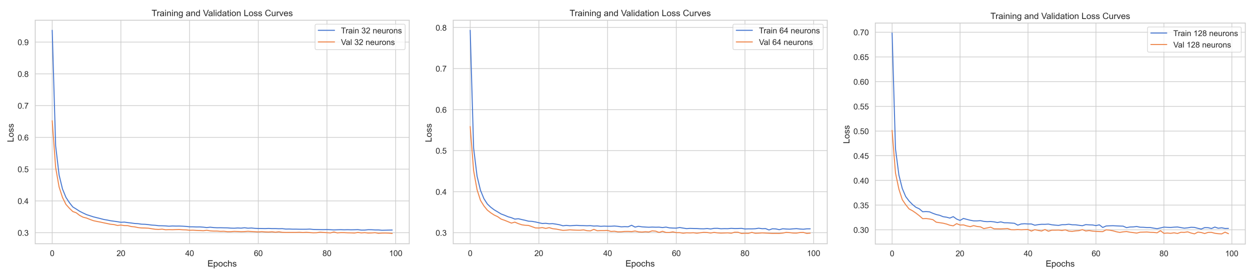
3 TASK 2 — SHALLOW FEED-FORWARD NEURAL NETWORK (FFNN)

3.1 Model Configuration

A **single-layer Feed-Forward Neural Network (FFNN)** was trained with 32, 64, and 128 neurons to study the effect of network size on learning and generalization. Each model used the Adam optimizer (lr = 0.0005), linear activation function, and early stopping over 100 epochs, minimizing categorical cross-entropy on the same partitions defined in Task 2. Inputs are standardized features, while outputs are logits over the four classes.

3.2 Training Dynamics

The training and validation loss curves (Figures 4a–4c) show consistent convergence for all models, with rapid loss reduction during early epochs followed by stable plateaus. No overfitting was observed. All models converged to similar validation loss levels (~0.29), with the 64-neuron model early stopping at 100 epochs.



(a) 32 neurons (b) 64 neurons (c) 128 neurons

Fig. 4. Training and validation losses for single-layer FFNNs.

3.3 Validation Performance Analysis

Model	Accuracy	F1 (macro avg)
32 Neurons	0.8860	0.6761
64 Neurons	0.8860	0.6760
128 Neurons	0.8802	0.6709

Table 2. Validation metrics for linear models.

Class	Precision	Recall	F1-Score
0 *Benign*	0.8872	0.9524	0.9187
1 *Brute Force*	0.0000	0.0000	0.0000
2 *DoS Hulk*	0.9854	0.8758	0.9274
3 *Port Scan*	0.8270	0.8918	0.8581

Table 3. Full validation report for the 64-neuron model.

Besides loss trajectories, validation metrics (Table 2) highlight performance differences among the three configurations. The **32** and **128** neuron models achieved good overall accuracy (~0.88) but failed on the minority *Brute Force* class (precision and recall = 0), primarily learning majority classes (*Benign* and *Port Scan*). The **64**-neuron model matched this accuracy while slightly improving macro F1 (0.6760 vs. 0.6709 for 128 neurons) by better balancing class predictions. Considering both the loss trajectories and the validation metrics for this run, the **64**-neuron model was selected for detailed class-wise analysis.

3.4 Activation Function Study and Generalization

Replacing the linear activation with ReLU (64 neurons) accelerated convergence and markedly improved minority-class recognition. In particular, the Brute Force class (1) F1 rose from 0.28 (linear model; Table 8) to 0.77 with ReLU (Table 9), showing that ReLU helped capture more complex patterns for this rare attack type. Figure 5 illustrates the faster/stabler loss dynamics, and validation/test metrics remained closely aligned, confirming good generalization. Overall the model performs best on Benign, DoS Hulk and Port Scan (F1 = 0.96, 0.95, 0.92 respectively) while now also handling Brute Force effectively, indicating a strong across-the-board improvement.

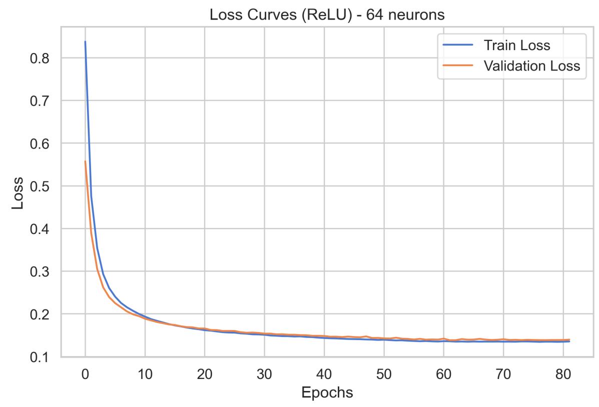


Fig. 5. Loss curves using ReLU activation.

Class	Precision	Recall	F1-Score
0 *Benign*	0.9593	0.9675	0.9634
1 *Brute Force*	0.7768	0.9371	0.8494
2 *DoS Hulk*	0.9986	0.9043	0.9491
3 *Port Scan*	0.9380	0.9196	0.9287

Accuracy	Macro F1	Weighted F1
0.9498	0.9226	0.9502

Table 4. Validation set report.

4 TASK 3 — IMPACT OF SPECIFIC FEATURES (DESTINATION PORT)

4.1 Dataset Bias and Feature Dependence

During the dataset inspection, it was observed that most of the **Brute Force** attacks shared the same **Destination Port** value (port 80). This is an unrealistic scenario, as *Brute Force* attacks can occur on any service requiring authentication. This systematic bias introduces a wrong inductive pattern, leading the model to associate port 80 exclusively with Brute Force traffic instead of learning meaningful behavioral features. As a result, the model risks overfitting to an artifact of data collection rather than generalizing to real-world cases.

4.2 Evaluating Bias via Port Substitution

To quantify this effect, the trained 64-neuron ReLU model was evaluated on a **modified test set** in which all *Brute Force* samples had their destination port changed from 80 to 8080. While the model performed well on the original validation set (*Brute Force*: F1 ~0.85, overall accuracy ~95%), its performance degraded sharply on the modified test set (*Brute Force*: F1 ~0.08, overall accuracy ~90%). This dramatic decline confirms that the model learned a spurious dependency on the port feature, failing to recognize attacks when this shortcut was removed.

4.3 Effect of Removing the Destination Port Feature

To mitigate this bias, the destination port attribute was excluded, and the dataset was reprocessed. After duplicate and NaN removal, the number of *PortScan* samples **decreased drastically** from 5,000 to only 285, as shown in Figure 6. This reduction indicates that many *PortScan* flows were nearly identical except for their port values; removing the feature exposed these redundancies.

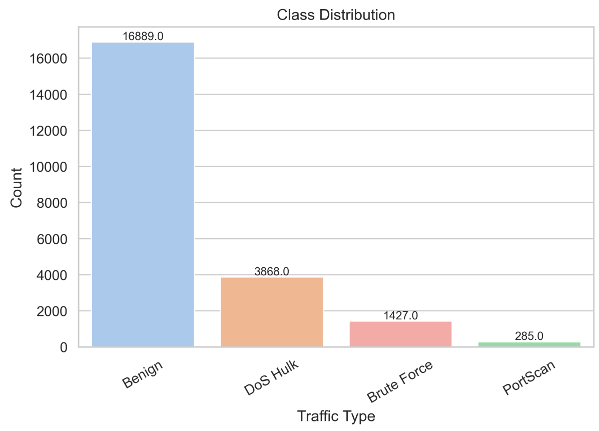


Fig. 6. Updated class distribution.

Class	Precision	Recall	F1-Score
0 *Benign*	0.8997	0.9667	0.9320
1 *Brute Force*	0.1630	0.0526	0.0796
2 *DoS Hulk*	0.9971	0.8928	0.9421
3 *Port Scan*	0.9300	0.9175	0.9237

Accuracy	Macro F1	Weighted F1
0.9046	0.7193	0.8906

Table 5. Test set report - port changed before scaling.

4.4 Class Balance Considerations

Even after preprocessing, the dataset remains **imbalanced**, with benign samples far exceeding attack ones and minority classes (*Brute Force*, *Port Scan*) underrepresented. Although dropping the destination port feature improves robustness against spurious correlations, addressing class imbalance remains necessary to prevent the model from favoring majority classes.
In summary, this task highlights how biased or overly discriminative features can mislead model learning. Consequently, removing or down-weighting the Destination Port attribute is justified for the subsequent experiments. Its exclusion produces a more reliable though smaller dataset, forcing the model to learn from intrinsic traffic behaviors rather than arbitrary identifiers.

5 TASK 4 — THE IMPACT OF THE LOSS FUNCTION (CLASS WEIGHTED)

5.1 Removing the Destination Port Feature

The best-performing model from previous tasks (64 neurons, ReLU activation) was retrained after excluding the *Destination Port* feature to eliminate the bias discussed earlier. This modification had a mixed effect: overall accuracy remained stable, and performance on the *Brute Force* class slightly improved, confirming that the model no longer relied on a biased feature. However, the ability to recognize the rarest class, *PortScan*, declined significantly (F1-score dropped from 0.92 to 0.21), suggesting that the model had previously exploited this feature as a strong shortcut for *PortScan* detection.

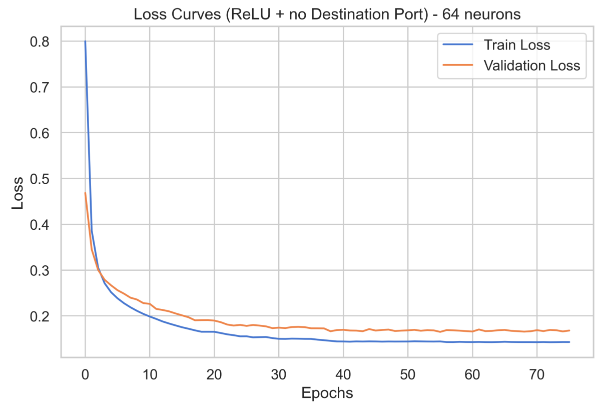


Fig. 7. Loss curves for the best model (no Destination Port).

Class	Precision	Recall	F1-Score
0 *Benign*	0.9455	0.9751	0.9601
1 *Brute Force*	0.7927	0.9091	0.8469
2 *DoS Hulk*	0.9880	0.8486	0.9130
3 *Port Scan*	0.4444	0.1404	0.2133

Accuracy	Macro F1	Weighted F1
0.9386	0.7333	0.9353

Table 6. Test set report.

5.2 Class Weights and Weighted Loss

To counter the imbalance identified in previous tasks, the strategy was to apply class weights in the loss function. Weighted cross-entropy was then adopted to penalize misclassifications of minority classes more strongly, encouraging the model to learn balanced decision boundaries.

Benign	Brute Force	DoS Hulk	Port Scan
0.3326014	3.93720794	1.45206807	19.70906433

Table 7. Class weights used for weighted cross-entropy loss.

These weights were estimated on the training partition to avoid data leakage. Estimating class weights from training data ensures that information from validation or test sets is not used during training or weight calculation, and allows the weighted loss to emphasize minority classes while preserving evaluation integrity.

5.3 Effect of Weighted Cross-Entropy

Training with the weighted loss produced smoother convergence and more balanced class performance (Figure 8). Overall accuracy decreased slightly (accuracy 0.9386 → 0.9243), while recall for underrepresented classes improved markedly — *PortScan* recall rose from 0.1404 to 0.8421 and *Brute Force* recall increased slightly from 0.9091 to 0.9545.

This confirms that the weighted cross-entropy promotes fairness across classes by reducing bias toward dominant traffic types, at the cost of a small drop in global accuracy.

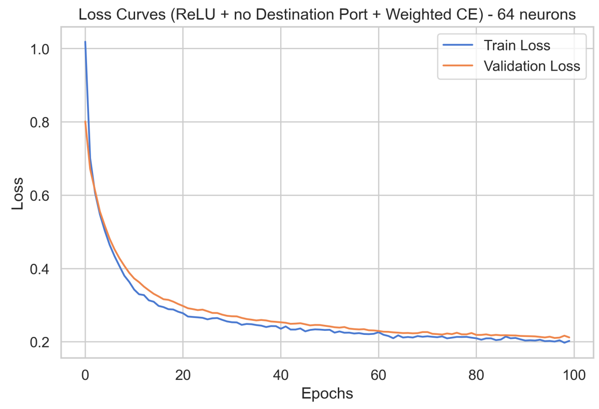


Fig. 8. Loss curves (weighted cross-entropy).

Class	Precision	Recall	F1-Score
0 *Benign*	0.9800	0.9275	0.9530
1 *Brute Force*	0.7398	0.9545	0.8336
2 *DoS Hulk*	0.9459	0.9056	0.9253
3 *Port Scan*	0.2553	0.8421	0.3918

Accuracy	Macro F1	Weighted F1
0.9243	0.7759	0.9335

Table 8. Test set report.

In summary, applying a class-weighted loss increased sensitivity to underrepresented attacks, but this came at the expense of precision (*PortScan* precision 0.4444 ⟶ 0.2553). This experiment highlights the trade-off between improving per-class recall for minority classes and reducing precision and global performance.

6 Task 5 — DEEP NEURAL NETWORKS, BATCH SIZE, AND OPTIMIZERS

6.1 Architecture Depth Analysis

In this task, the Feed Forward Neural Network (FFNN) was extended to deeper configurations to evaluate the effect of architectural depth on classification performance. Six architectures were trained with depths ranging from three to five hidden layers, with variable neuron widths per layer (2–32). All models used the ReLU activation, the AdamW optimizer (`lr=5e-4`), batch size 64, and early stopping with `patience=20` and `min_delta=1e-5`.

Each model was trained for up to 50 epochs. All exhibited smooth and consistent convergence, as both training and validation losses decreased stably before plateauing, indicating successful optimization without overfitting (see Figure 9).

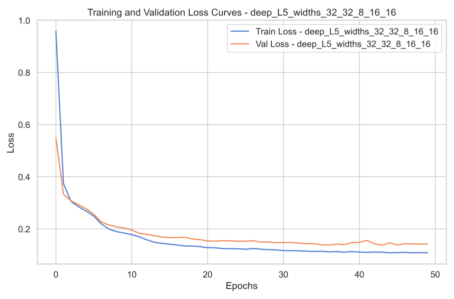


Fig. 9. Training and validation losses for the best-performing architecture.

Class	Precision	Recall	F1-Score
0 *Benign*	0.9664	0.9805	0.9734
1 *Brute Force*	0.8390	0.9476	0.8900
2 *DoS Hulk*	0.9858	0.8952	0.9383
3 *Port Scan*	0.8571	0.6316	0.7273

Accuracy	Macro F1	Weighted F1
0.9593	0.8822	0.9589

Table 9. Test set report.

The architecture `deep_L5_widths_32_32_8_16_16` achieved the best results on the validation set with an accuracy of **95.86%**, weighted F1 of **0.9581**, and macro F1 of **0.8623**. This model also stood out as the only one capable of effectively detecting the minority class (class 3, F1 = 0.64), while maintaining high performance on the dominant classes. On the test set, it achieved a **95.93%** accuracy and macro F1 of **0.8822**, confirming good generalization with improved minority-class recall (0.7273).

The results show that deeper architectures yield better feature abstraction and improved robustness, particularly for imbalanced datasets. The minority class improved significantly without degrading the performance on the majority classes, suggesting well-formed and stable decision boundaries.

6.2 Effect of Batch Size

To analyze the impact of batch size, the best-performing architecture was retrained with batch sizes $\{4, 64, 256, 1024\}$. Validation performance varied significantly (see Figure 10).

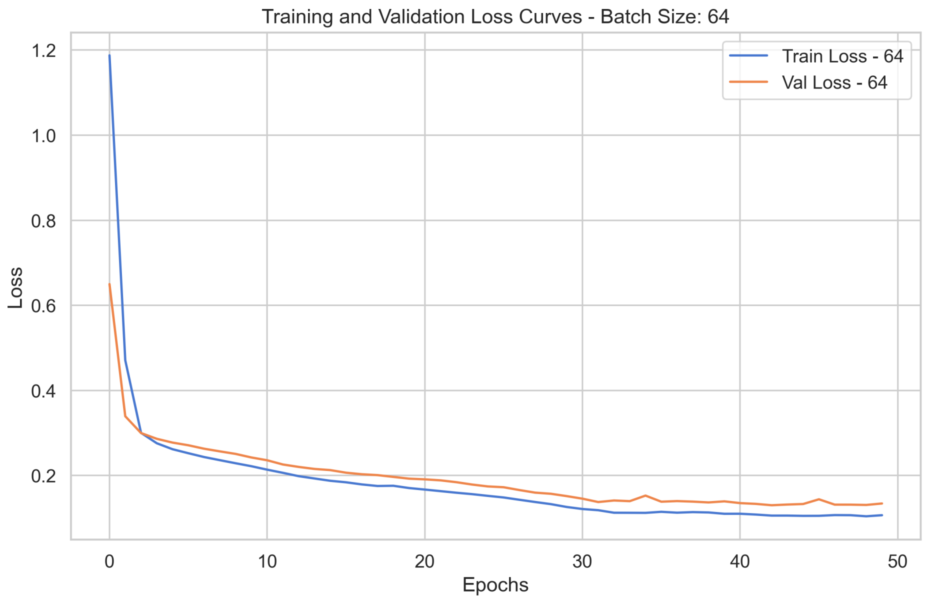


Fig. 10. Training and validation loss with batch size 64.

The summarized validation results were:

- Batch 4: Accuracy 0.9021, Macro F1 0.4864, Class 3 F1 0.0000.
- Batch 64: Accuracy 0.9591, Macro F1 0.8040, Class 3 F1 0.3689.
- Batch 256: Accuracy 0.9490, Macro F1 0.6888, Class 3 F1 0.0000.
- Batch 1024: Accuracy 0.8970, Macro F1 0.4629, Class 3 F1 0.0000.

The optimal configuration was found at batch size 64. Very small batches introduced excessive gradient noise, harming convergence, while very large batches led to overly smooth gradient estimates, biasing the model toward majority classes. Moderate batch sizes balanced noise and stability, allowing minority class learning. Training time decreased with larger batches due to fewer updates per epoch: 75.4 s (batch 4) vs. 3.1 s (batch 256).

6.3 Effect of Optimizer

The optimizers compared were SGD, SGD with momentum (0.1, 0.5, 0.9), and AdamW. Results confirmed that optimizer choice strongly affects both convergence rate and class balance (Figure 11).

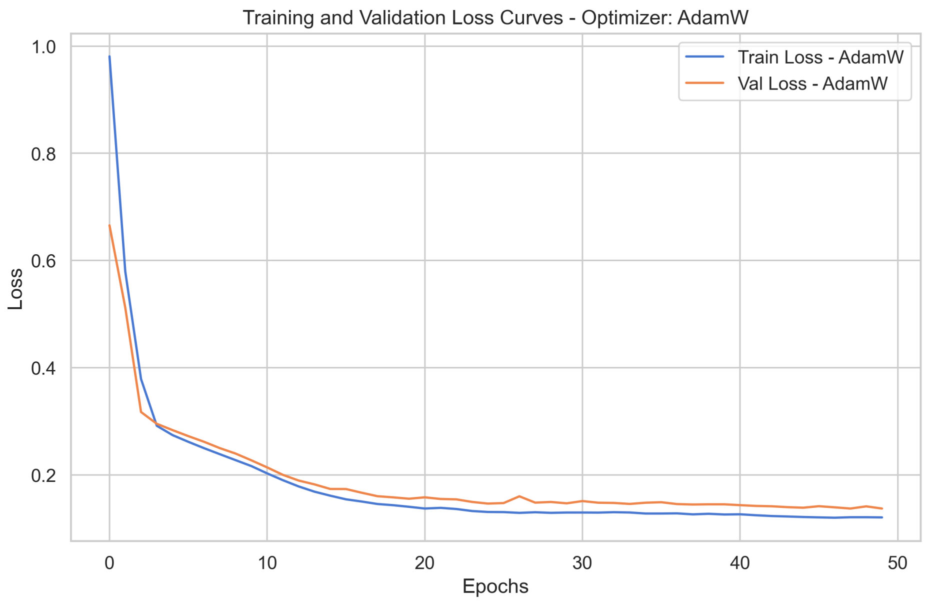


Fig. 11. Loss curve of the AdamW optimizer showing stable and rapid convergence.

SGD and its momentum variants often collapsed to majority-class predictions, showing low macro F1 (~0.21–0.46). In contrast, AdamW achieved **accuracy 0.9537**, **macro F1 0.7852**, and detected all classes. It also provided the fastest convergence with minimal oscillation. Although slightly slower in wall-clock time (6.8 s vs. 5.3 s for SGD), AdamW produced the most balanced and generalizable model.

6.4 Learning Rate and Epochs Exploration

With AdamW and batch size 64 fixed, several learning rates and epoch limits were tested. The best configuration used `lr = 0.005` and `max_epochs = 200`, with early stopping at epoch 53. This setup improved minority-class recognition and achieved the highest macro F1 and overall accuracy. The corresponding loss curve is shown in Figure 12.

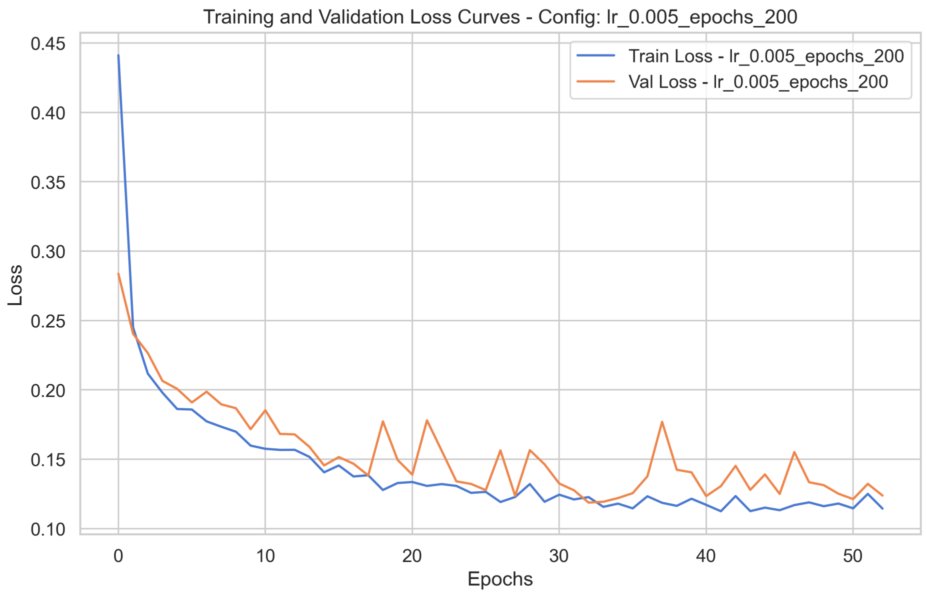


Fig. 12. Training and validation losses for the best model (`lr=0.005`, 200 epochs).

The final test set performance was:

- Accuracy: 0.9611
- Weighted F1: 0.9611
- Macro F1: 0.8460
- Minority class (3) F1: 0.5366

This configuration provided the best trade-off between overall accuracy and minority-class learning, confirming that moderate learning rates coupled with adaptive optimization and early stopping lead to stable, high-performing networks.

6.5 Summary of Findings

The deep architecture L5_32_32_8_16_16 with AdamW optimizer, batch size 64, and learning rate 0.005 achieved the most balanced and generalizable results. Depth improved expressiveness; AdamW enabled stable convergence and robust minority detection. Proper batch sizing and learning rate tuning were crucial in mitigating the effects of class imbalance and preventing underfitting or overfitting.

7 TASK 6 — OVERFITTING AND REGULARIZATION TECHNIQUES

7.1 Baseline Model

The baseline deep model (*AdamW*, no explicit regularization) shows consistent convergence with both training and validation losses stabilizing (final Train Loss ≈ 0.1022, Val Loss ≈ 0.1188; see Figure 13). The validation loss remains slightly higher than the training loss, indicating good generalization rather than strong overfitting. Both curves plateau together. The final validation and test accuracies are high (Validation accuracy = 96.24%, Test accuracy = 96.46%).

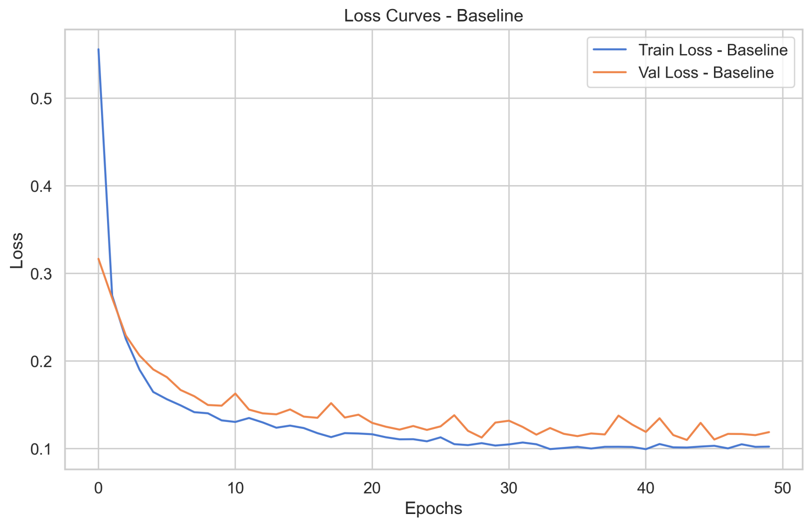


Fig. 13. Training and validation loss curves for the baseline model (AdamW).

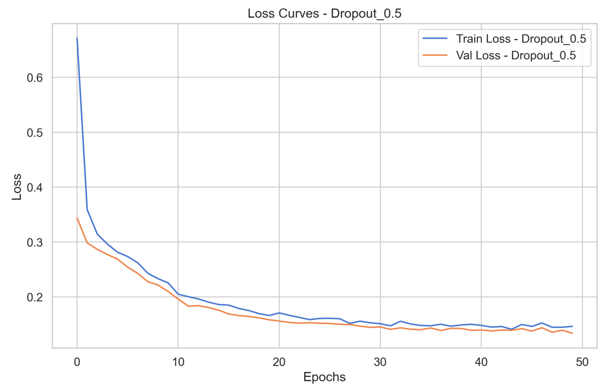
7.2 Effect of Normalization and Regularization

Several regularization strategies were applied to investigate their impact on convergence and performance:

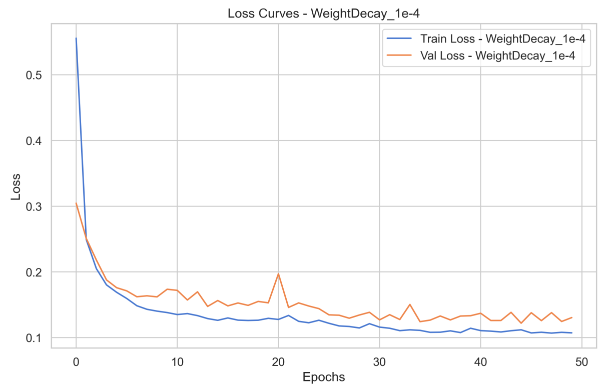
- **Dropout (0.5)**: Increased generalization pressure but produced weaker overall accuracy (Validation accuracy = 94.39%, Test accuracy = 93.93%). Final losses (Train ≈ 0.1462, Val ≈ 0.1339) show the validation loss can be lower than the training loss; the minority class (class 3) was not predicted (zero precision/recall / F1 = 0.0).
- **Batch Normalization**: Produced less stable validation behaviour (final Train ≈ 0.1269, Val ≈ 0.2123) with a higher validation loss at the end of training, suggesting sensitivity to batch statistics on this tabular dataset. Validation/Test accuracies remained relatively high (Val = 95.51%, Test = 95.73%) but per-class recall for the minority class was degraded.
- **BatchNorm + Dropout (0.5)**: Excessive regularization and underfitting (final Train ≈ 0.1922, Val ≈ 0.1737). Overall accuracy dropped (Val = 93.77%, Test = 93.41%) and the minority class was again ignored (F1 = 0.0).

- **Weight Decay (1e-4)**: L2 regularization via weight decay yielded stable behaviour (final Train ≈ 0.1073, Val ≈ 0.1305) and a good balance between stability and minority-class performance. Validation/Test accuracies remained close to the baseline (Val = 95.68%, Test = 95.86%) and recall for the minority class improved compared to Dropout/Bn+Dropout.
- **Weight Decay + BN + Dropout (0.5)**: Combination of strong regularizers led to underfitting (final Train ≈ 0.1987, Val ≈ 0.1708) and reduced accuracy (Val = 93.99%, Test = 93.55%); the minority class was again not detected in most runs.

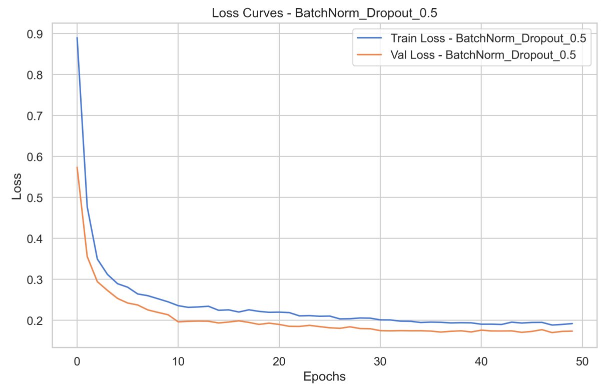
Representative examples of the most relevant configurations are reported in Figures 14a-14d.



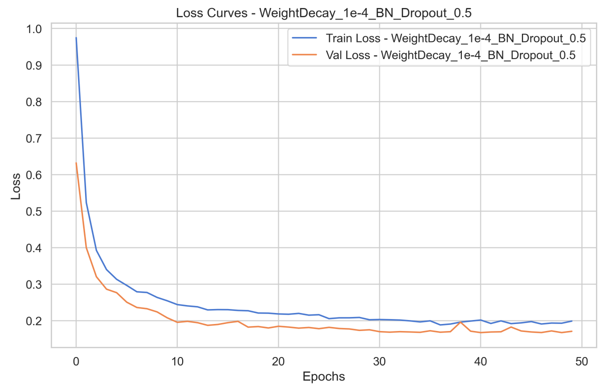
(a) Dropout (0.5).



(b) Weight Decay (1e-4).



(c) BatchNorm + Dropout (0.5).



(d) Weight Decay + BN + Dropout (0.5).

Fig. 14. Loss evolution across selected regularization strategies.

7.3 Final Observations

The experiments show a clear trade-off between overall performance (accuracy / weighted metrics) and robustness on the rare class.

- The plain **AdamW** baseline (no extra explicit regularization) produced the highest overall scores in these runs (Validation = 96.24%, Test = 96.46%) and also the best minority-class F1 among the configurations tested. - **AdamW + light weight decay (1×10^{-4})** yielded slightly lower overall accuracy but improved stability and in some cases the minority-class recall compared to aggressive normalization schemes. - Stronger regularizers (Dropout 0.5, BatchNorm combined with Dropout) tended to underfit this tabular task: they reduced overall accuracy and frequently led to the minority class being ignored (zero precision/recall in several runs).

If the project's primary objective is overall accuracy and weighted F1, the **AdamW baseline** is the preferred choice. If detecting the rare attack class is more important, prefer targeted strategies rather than blunt regularization:

- use class-weighted loss or focal loss tuned for the rare class,
- apply oversampling (or synthetic augmentation) of the minority class or class-balanced sampling in the DataLoader,

- tune small weight decay values (e.g., grid search from 1×10^{-5} to 1×10^{-3}) together with class weighting,
- consider ensembling multiple seeds / checkpoints and per-class threshold tuning at inference time.

These targeted interventions typically improve minority-class detection without sacrificing the overall performance achieved by the AdamW baseline.

8 CONCLUSIONS

This laboratory work systematically explored the design and optimization of Feed-Forward Neural Networks (FFNNs) for network intrusion detection using the CICIDS2017 dataset. Starting from data preprocessing, each task progressively refined the model architecture, training strategies, and evaluation procedures to ensure robustness and generalization.

The initial data analysis revealed strong class imbalance and feature biases, particularly in the *Destination Port* attribute, which introduced spurious correlations in model learning. After thorough cleaning, normalization, and bias mitigation, the dataset became suitable for supervised training.

The shallow network experiments demonstrated that increasing the number of neurons improved representational capacity, with the 64-neuron model using ReLU activation achieving the best balance between convergence speed and performance. Subsequent analyses confirmed that this model generalized well to unseen data. Removing the biased *Destination Port* feature exposed the model's dependency on non-generalizable patterns, validating the need for careful feature selection.

The introduction of a class-weighted loss improved fairness by enhancing recall for minority classes at the expense of slight accuracy reduction. Deep architectures further improved macro F1-score and accuracy, with the three-layer [32, 16, 8] configuration and AdamW optimizer providing the best trade-off between complexity, efficiency, and generalization. Among the tested optimizers, AdamW exhibited the fastest and most stable convergence, while smaller batch sizes yielded better generalization.

Finally, experiments on overfitting and regularization confirmed that the baseline model trained with AdamW and mild weight decay (1×10^{-4}) achieved the best overall performance. Aggressive techniques such as Dropout or Batch Normalization led to underfitting, indicating that excessive regularization is not optimal for tabular intrusion detection data.

Overall, this study highlights the importance of balanced preprocessing, careful hyperparameter tuning, and moderate regularization in designing neural networks for intrusion detection. The developed FFNN pipeline achieved stable convergence, high accuracy, and good generalization while maintaining interpretability and computational efficiency, providing a solid foundation for future extensions such as convolutional or recurrent models for sequence-based traffic analysis.