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 report presents a systematic study of Feed-Forward Neural Networks (FFNNs) for multi-class network-flow classification using a curated subset of CICIDS2017. We build an end-to-end pipeline in PyTorch that encompasses data cleaning, stratified splitting, outlier inspection, and feature normalization; we then investigate how architectural depth, activation functions, batch size, optimizers, loss re-weighting, and regularization affect learning dynamics and generalization.

The work is organized in six tasks. First, we establish a robust preprocessing protocol and compare Standard vs. Robust scaling, ultimately adopting standardization fitted on the training split. Second, we train shallow one-hidden-layer models with different widths and show that switching from a linear activation to ReLU substantially improves minority-class recognition while preserving overall accuracy and stable convergence. Third, we expose a harmful inductive bias: the feature *Destination Port* is spuriously correlated with the *Brute Force* class. A simple port substitution at inference time causes a marked performance drop, and removing the feature reshapes class supports due to duplicate collapse. Fourth, we mitigate class imbalance with a class-weighted cross-entropy that improves macro-level metrics and recall for rare classes without materially harming overall accuracy. Fifth, we extend to deeper FFNNs and analyze the effect of batch size and optimizers, observing that smaller batches tend to generalize better at higher computational cost and that AdamW provides the most reliable optimization among tested choices. Finally, we assess overfitting and regularization and find that a small weight decay (AdamW) yields the best trade-off on this tabular task, whereas aggressive dropout or batch normalization can underfit or destabilize validation loss.

Overall, our results underline three practical lessons for intrusion-detection FFNNs on tabular features: (i) treat feature-induced biases explicitly; (ii) prefer training-set-fitted standardization and class-weighted losses when imbalance matters; and (iii) adopt modest capacity with careful regularization (light weight decay), ReLU activations, and a robust optimizer such as AdamW.

Contents

1 INTRODUCTION

This laboratory investigates Feed-Forward Neural Networks (FFNNs) for intrusion detection on tabular, flow-level features derived from CICIDS2017. Our objective is twofold: first, to engineer a transparent and reproducible supervised-learning pipeline in PyTorch, and second, to quantify how modeling choices influence both optimization behavior and generalization in presence of class imbalance and feature-induced biases.

We proceed in six steps. We start by establishing a robust preprocessing protocol that removes missing values and duplicates, handles non-finite entries, and applies standardization fitted on the training split; we also inspect distributions and outliers to justify the scaling choice. We then study shallow architectures (one hidden layer) and show how activation functions (linear vs. ReLU) alter minority-class recognition without compromising stability. Next, we expose a spurious correlation between the feature *Destination Port* and the *Brute Force* label, demonstrating

*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: October 2025.

that a small perturbation at inference time (changing port 80 to 8080) degrades performance markedly; removing the feature further reveals how duplicate collapse reshapes class supports.

Building on these insights, we adopt class-weighted cross-entropy to improve macro-level metrics and minority recall, extend the architecture to deeper FFNNs to explore capacity, and analyze the impact of batch size and optimizer. Finally, we quantify overfitting and evaluate regularization strategies. Across the study, AdamW emerges as the most reliable optimizer, smaller batches tend to generalize better at higher computation cost, and light weight decay provides the best regularization balance for this tabular task.

Beyond raw metrics, the broader lesson for AI and Cybersecurity is methodological: reliable evaluation requires attention to data provenance, feature biases, and class supports; principled preprocessing and targeted regularization often matter as much as architectural complexity. Throughout the report we report not only accuracy but also per-class precision/recall and macro/weighted F1-scores to capture minority-class behavior, and we use early stopping with validation-based model selection to prevent overfitting.

2 DATA ANALYSIS AND PREPROCESSING

2.1 Dataset and labels

We work with a tabular subset of CICIDS2017 comprising flow-level statistics (e.g., Flow Duration, Flow IAT Mean, Bwd/Fwd packet-length summaries, Flow Bytes/s, flags-related counts) and a categorical label in {*Benign*, *DoS Hulk*, *PortScan*, *Brute Force*}. Each row is a flow; no raw packet sequences are used. The goal is multi-class classification on these features.

The selected features include (non-exhaustively): *Flow Duration*, *Flow IAT Mean*, *Fwd PSH Flags*, *Bwd Packet Length Mean/Max*, *Flow Bytes/s*, *Down/Up Ratio*, *SYN Flag Count*, *Fwd Packet Length Mean/Max*, *Fwd IAT Std*, *Packet Length Mean*, *Subflow Fwd Packets*, *Flow Packets/s*, *Total Fwd Packets*, and *Destination Port*. The label is a four-class categorical variable.

2.2 Cleaning protocol and effects

The preprocessing adheres to a conservative protocol: non-finite values are replaced and dropped, missing values are removed, and duplicates are eliminated. Concretely, starting from a raw table of ~ 31 507 rows, we remove missing, duplicate, and infinite entries and obtain ~ 29 386 clean samples (as printed in the notebook). In total, the cleaning step removes ~ 2,121 rows (~ 2,114 NaN/duplicates + ~ 7 infinite). This step is essential to prevent degenerate gradients, spurious leakage through repeated rows, and unstable scalers.

The label column is encoded with a `LabelEncoder`; the mapping is kept for interpretability when printing classification reports.

2.3 Stratified data split

We split the data into training, validation, and test partitions with a 60/20/20 ratio using stratification by label to preserve class proportions. Random seeds are fixed for reproducibility. All transformers (e.g., scalers) are *fit* on the training split only and *applied* to validation and test to avoid leakage. The notebook reports initial split sizes around 17.6k/5.9k/5.9k for train/val/test, with the majority class dominating.

2.4 Outliers and normalization

We inspect distributions on the training split using Z-score and IQR-based analyses and visualize representative features via KDE and boxplots. Several features exhibit heavy tails and extreme values (e.g., *Bwd Packet Length Max*, *Flow Duration*, *Bwd Packet Length Mean*, *Fwd IAT Std*). We therefore compare two scalers on the training split: StandardScaler (zero mean, unit variance) and RobustScaler (median and IQR). While RobustScaler is less sensitive to outliers and produced tighter distributions for heavy-tailed features, we ultimately adopt *standardization* because it yielded smoother learning curves and indistinguishable or slightly better downstream accuracy in preliminary runs, while keeping interpretation and hyperparameter tuning simple.

All scaling hyperparameters are estimated on training data and then applied unchanged to validation and test sets.

2.5 Evaluation metrics and selection protocol

Given class imbalance, we monitor overall accuracy together with per-class precision, recall, and F1-scores, and we report macro- and weighted-F1 aggregates. Models are selected by the epoch that minimizes validation loss (early stopping with small `min_delta` and `patience`); the best weights are restored before final evaluation on the test set.

3 SHALLOW NEURAL NETWORK (1 LAYER)

3.1 Architectures and training setup

We train three single-hidden-layer FFNNs that differ only by hidden width: $\{32, 64, 128\}$ neurons. Inputs are standardized features; outputs are logits over the four classes. The loss is cross-entropy, the optimizer is AdamW with learning rate 5×10^{-4}, the batch size is 64, and early stopping monitors validation loss. We first use a linear activation in the hidden layer to establish a conservative baseline, then replace it with ReLU on the best width selected by validation.

Implementation details follow the notebook: the training loop averages batch losses per epoch for both training and validation partitions; early stopping stores the best-performing state dict whenever the validation loss improves by more than `min_delta` and aborts when no such improvement occurs for `patience` epochs. We used `min_delta` $= 10^{-5}$ and `patience` = 20 across widths.

3.2 Convergence and model selection

All three models exhibit smooth training and validation loss decay and reach a stable plateau within the epoch budget. We select the model snapshot that achieves the minimum validation loss, restoring those weights before evaluation. Using this criterion favors architectures that generalize better rather than those that merely minimize training loss.

3.3 Validation performance and class-wise behavior

On the validation split, larger widths generally yield stronger macro-level metrics, although differences are modest. Without nonlinearity (linear activation), minority-class recognition is fragile: the most challenging class often shows very low recall. In our runs, 128 neurons achieved the highest macro F1 (~ 0.76) and accuracy (~ 0.897) on validation, while the 64-neuron model achieved the lowest validation loss over epochs and exhibited the most stable convergence. Switching the best-width network to ReLU substantially improved the *Brute Force* class, reaching an F1 around 0.85 on validation, while preserving strong performance on majority classes. This indicates that even shallow nonlinear capacity is beneficial for this tabular task.

3.4 Test-set generalization

Comparing validation and test reports for the selected model (64 neurons with ReLU) shows closely aligned metrics, supporting the conclusion that the training protocol and early stopping prevented overfitting and that the model generalizes well. In the remainder of the study we therefore use the best shallow configuration with ReLU as a competitive baseline when comparing deeper models and data/feature manipulations.

4 THE IMPACT OF SPECIFIC FEATURES (DESTINATION PORT)

4.1 Why this feature matters

In the provided dataset, *Destination Port* is strongly correlated with certain labels. In particular, all flows labeled as *Brute Force* originate from port 80. If a model exploits this shortcut, it will appear accurate under the original distribution but fail under even small deviations at inference time. We therefore design two experiments to diagnose and mitigate this bias.

4.2 Perturbation at inference time

We modify the *test* split only by replacing `Destination Port` = 80 with 8080 for rows labeled *Brute Force*, leaving all other samples unchanged. Evaluating the best shallow model on this perturbed test set reveals a pronounced drop in precision (~ 0.17), recall (~ 0.05), and F1 (~ 0.08) for *Brute Force*, with overall accuracy falling from ~ 95% to ~ 90.4%. This confirms that the original model captured a spurious correlation rather than a causal signal.

4.3 Removing the feature and reprocessing

To remove the shortcut entirely, we drop the `Destination Port` column from the original table and repeat the full preprocessing (cleaning, stratified splitting, scaling fitted on training). An immediate side effect is a sharp change in class supports: without the port column, many rows that previously differed only by destination port become exact duplicates and are eliminated by the cleaning step. The net effect is that some attack classes — most notably *PortScan* — lose a large fraction of instances. For example, *PortScan* counts drop from ~ 5,000 in the raw dataset to ~ 285 after duplicate removal on the no-port table; *Brute Force* support in validation/test also becomes very small (~ 57 samples per split in our runs).

This phenomenon illustrates that "debiasing by feature removal" can reshape the dataset in subtle ways. It is still the right choice to prevent shortcut learning here, but it must be accompanied by evaluation strategies that remain meaningful when some classes become rare.

4.4 Implications

After removing the feature, training on the new splits yields models that no longer rely on ports, but class imbalance is now more severe. Consequently, downstream experiments (next sections) complement this change with class-weighted losses and careful reporting of macro metrics to avoid over-optimistic conclusions. Where class supports become extremely small, we emphasize macro-F1 and per-class recall, and we discuss the limitations of single-split estimates.

5 THE IMPACT OF THE LOSS FUNCTION

5.1 Motivation and setup

After removing `Destination Port` (Task 4), class imbalance becomes more pronounced. To prevent the classifier from ignoring rare classes, we switch from an unweighted cross-entropy to a class-weighted one. We estimate weights with `compute_class_weight(class_weight = 'balanced')` using *only* the training labels, convert them to a tensor, and pass them to `nn.CrossEntropyLoss`.

5.2 Effects on training dynamics and metrics

Class weighting slightly alters optimization but consistently improves macro-averaged F1 by increasing recall on under-represented classes. Overall accuracy remains similar or decreases marginally because errors on the majority class are penalized more heavily, but the resulting classifier is better aligned with the security objective of detecting rare attacks. In our runs, the weighted loss materially improved the recognition of the rarest classes without destabilizing convergence. Qualitatively, we observe a redistribution of errors from rare to frequent classes, which is an acceptable trade-off for security-sensitive detection.

5.3 Guidance

In intrusion-detection settings with skewed class distributions, class-weighted losses are a simple and effective default. They should be paired with stratified splits, per-class reporting, and—when supports are very small—resampling or augmentation to produce reliable estimates.

6 DEEP NEURAL NETWORK

6.1 Design of deep architectures

We extend the shallow baseline to 3–5 hidden layers with modest widths (2–32 neurons per layer). The aim is to quantify whether added depth improves representation on tabular features and how it interacts with other training choices. We instantiate six architectures (two per depth), for example: L3: [16, 8, 4] and [32, 16, 8]; L4: [32, 16, 8, 4] and [16, 16, 8, 8]; L5: [32, 32, 16, 8, 4] and [16, 8, 8, 4, 2]. The most reliable configuration in our experiments is a 3-layer network with widths `[32, 16, 8]` (tag `deep_L3_widths_32_16_8`), trained with ReLU and AdamW.

6.2 Loss curves and convergence (qualitative summary)

Deeper models converge smoothly with early stopping. Three-layer networks consistently achieve lower validation losses than the single-layer baseline, while 4–5 layers offer no systematic improvement for this feature set and can overfit unless regularized. Saved loss curves (not reproduced here) document these trends.

6.3 Batch size experiments

Using the best architecture, we vary batch size in {4, 64, 256, 1024}. Smaller batches produce noisier but more generalizable updates and achieve the best validation macro metrics, at the cost of markedly longer training times. Conversely, very large batches train quickly but underperform on validation. We retain batch size 64 as a good compromise between stability, runtime, and generalization, also because it produced smoother loss curves during early stopping.

6.4 Optimizer comparison

We compare SGD, SGD with momentum (0.1/0.5/0.9), and AdamW under matched learning rates and early stopping. AdamW converges fastest and reaches lower validation loss. Momentum helps SGD, narrowing the gap, but AdamW remains the most reliable choice without additional tuning effort.

6.5 Per-class test reports and a troubling observation

On the no-port splits, some trained models failed to predict the rarest class at all in certain runs, despite high overall accuracy. This behavior stems from extremely low support for that class in validation/test after duplicate removal and from residual imbalance. It underscores the importance of macro-level metrics and, when feasible, resampling or augmentation for robust estimates. For the selected deep model (`deep_L3_widths_32_16_8`), the reported test metrics include accuracy ~ 0.9530, weighted F1 ~ 0.9523, and macro F1 ~ 0.8282, with a lower F1 (~ 0.50) for *PortScan* compared to other classes.

6.6 Implication

Dataset transformations can mitigate shortcut learning yet inadvertently reduce minority supports. For credible evaluation, pair feature curation with class-aware losses and sampling strategies; otherwise, improvements in overall accuracy may hide systematic failures on rare but security-critical classes.

7 OVERFITTING AND REGULARIZATION

7.1 Experiment goal

We quantify overfitting risks in a higher-capacity FFNN and evaluate three regularizers: dropout, batch normalization, and L2 weight decay (via AdamW's weight_decay). The baseline uses six hidden layers with widths `[256, 128, 64, 32, 16]` and ReLU.

7.2 Baseline training dynamics (no regularization)

Training and validation losses decrease smoothly and plateau at close values with early stopping, suggesting limited overfitting on this dataset. Nevertheless, the gap between training and validation remains nonzero, and minority-class metrics are sensitive to capacity, motivating targeted regularization.

7.3 Dropout experiments

With dropout $p = 0.5$, training loss increases as expected and validation curves become noisier. On our tabular features, this level of dropout tended to underfit and, in some runs, degraded the recognition of the rarest class. Smaller dropout rates may be beneficial, but aggressive dropout was not optimal here.

7.4 Batch Normalization (BatchNorm)

Batch normalization between dense layers accelerated training but sometimes introduced validation instability and did not consistently improve macro metrics. In this setting, the benefit of input standardization and AdamW overshadowed BatchNorm's gains.

7.5 Weight decay (L2) with AdamW

A small weight decay (e.g., 10^{-4}) offered the most reliable improvement: it reduced variance in validation loss and modestly improved macro metrics without harming majority-class accuracy. Given the tabular nature of the features, light L2 regularization is a safe default.

7.6 Combined configuration and final model selection

When combined, heavy regularization (BatchNorm + Dropout 0.5 + weight decay) was excessive, yielding underfitting and lower macro F1. Our final recommendation for this task is a modest-capacity deep FFNN trained with AdamW and light weight decay; add mild dropout only if overfitting symptoms (rising validation loss with falling training loss) become apparent.

7.7 Practical recommendations from experiments

(1) Prefer light L2 regularization (AdamW weight decay) for tabular-flow FFNNs; it improves stability with minimal tuning.

(2) Use mild dropout only as needed; aggressive dropout can underfit and harm minority-class recall.
(3) Batch normalization is not universally beneficial on standardized tabular inputs; evaluate its impact empirically.
(4) Always monitor per-class metrics and macro F1; overall accuracy can mask failures on rare, security-critical classes.

8 CONCLUSIONS

This report walked through a complete FFNN pipeline and used empirical evidence from the notebook to draw conclusions regarding architecture choices, preprocessing, and regularization.

Key takeaways:

- **Preprocessing matters:** scaling, padding/truncation, and handling duplicates/NaNs materially affect training dynamics and the effective class balance.
- **Feature biases can be dangerous:** a single feature (Destination Port) can introduce a brittle shortcut for a classifier. Removing the feature reduced this bias but also altered class supports via duplicate collapse, illustrating the complexity of data curation.
- **Regularization should be measured, not maximal:** for these standardized tabular features, *light weight decay* with AdamW yielded the best trade-off. Aggressive dropout or batch normalization did not consistently help and sometimes degraded minority-class metrics.
- **Class imbalance requires active handling:** using class-weighted loss helped recover recall on rare classes; nevertheless, extremely small supports (e.g., 57 samples) lead to unreliable per-class estimates and require either resampling/augmentation or different evaluation strategies.
- **Operational implication for cybersecurity:** models that rely on spurious correlations are fragile in deployment. Robust detection requires attention to dataset provenance, feature engineering, and evaluation protocols that prioritize macro metrics.

8.1 Future work

Possible extensions:

- Explore sequential or attention-based models when raw sequences become available, to complement tabular features.
- Explore data augmentation or generative techniques to increase minority-class support.
- Perform stratified cross-validation and confidence estimation for rare-class performance.

A APPENDIX

Publication date: October 2025.