Politecnico di Torino

AI and Cybersecurity

# Project 1: Introduction to Deep Learning

# Using a Feed Forward Neural Network (FFNN)

Angelo Barbera - s326432, Giorgia Moscato - s324647, Alessandro Genova - s330893

# Table Of Contents

# 1 Task 1: Data Preprocessing

Preparing the dataset through preprocessing is a foundamental step to make the dataset ready for the Deep Learning task. To do this we need to:

- **Remove missing values (NaN) and duplicate entries:** we removed duplicates with `drop_duplicates` because they can bias the learning process and can reduce generalization. We remove missing values with `dropna` for simplicity (an alternative approach would have been to estimate them).
- **Ensure data consistency and correct formatting:** we removed infinite and negative values to ensure that the dataset contains only meaningful data. Moreover the labels are converted in integer numbers assigning an index to each label (label encoding) using `label_encoder.fit_transform(data['Label'])` because the neural network need to receive in input numeric data.
- **Split the dataset to extract a training, validation and test sets:** we split the dataset to use 60% of the data for training, 20% for validation, and 20% for testing using the function `train_test_split`.
- **Standardization:** we computed each feature's mean, standard deviation, minimum, and maximum values to identify potential outliers. Since some features had a wide range of values, instead of using min-max scaling that is sensitive to outliers, we chose standardization that ensures that every feature has a variance of 1 and mean equal to 0. We used `scaler.fit_transform` for the training set and used the same scaler for testing and validation set.

**Q: How do you preprocess the test partition data? Is the preprocessing the same as for the training partition?**
The preprocessing steps for the testing set are the same of the training one, the only difference is on the standardization phase. The mean and the standard deviation are calculated on the training set because calculating them on the entire dataset can give to the model information about unseen data, leading to incorrect performance estimates. These values are also used to standardize the data belonging to the validation and training set because in real applications only old data is available to standardize the data in input to the model.

After the preprocessing we obtain the following results:

```
# Remove missing values (NaN), duplicate entries, negative values
Removed 2111 duplicates
Removed 3 missing values
Removed 9 infinite and negative values

# Number of data for each class after preprocessing
Label
Benign         19240
PortScan        4849
DoS Hulk        3868
Brute Force     1427
```

# 2 Task 2: Shallow Neural Network

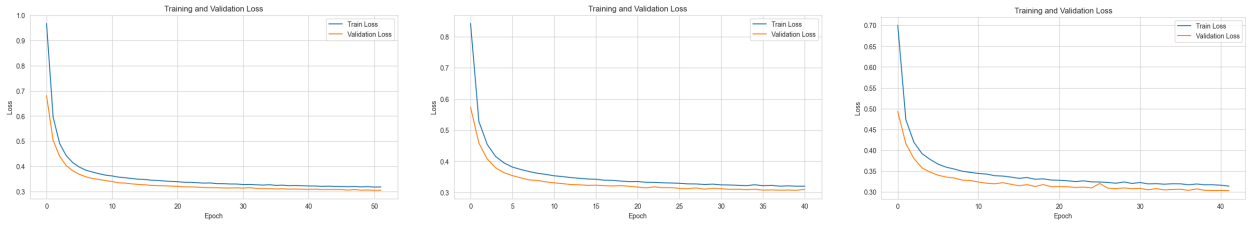
## 2.1 Training process

The training is performed giving in input to the model for each epoch the mini-batches of the training set with size equal to 64, with respect of using the entire dataset, the gradient descent converges faster, the model generalizes better, the training process is faster and it allows to work with large datasets that does not fit in memory. For each mini-batch is cleared the previous gradient (`zero_grad()`), it is computed the model output (`model(batch_X)`) and the loss (`loss=criterion(outputs, batch_y)`, then it is performed

the backpropagation (`loss.backward()`) and the model parameters are updated (`optimizer.step()`). For each epoch the training loss is computed through the mean of the loss computed for each mini-batch. During the training is also computed the validation loss using the mini-batches of the validation set. Finally we implemented the early stopping to avoid overfitting. The early stopping is implemented evaluating the validation loss each epoch, saves the model whenever the validation loss improves by more than `min_delta`, and stops training if the validation loss fails to improve for `patience` consecutive epochs.

## 2.2 Model performance

**Q: How does the loss curve evolve during training on the training and validation set?**
Figure 1 show three plots with respectively 32, 64 and 128 hidden neurons. In all the cases both training and validation losses decrease rapidly over the first 10 epochs, then each curve reaches a plateau and early stopping is activated.




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

Figure 1: Loss curves

**Q: How do you select the best model across epochs? Which model do you use for validation and test?**
During the i-th epoch, we use different mini-batches for training and validation. For training, we update the model's weights after each mini-batch via backpropagation. For validation, the model is evaluated after the epoch's training mini-batches are processed, on the validation mini-batches. It is selected the model with the lowest validation loss. For testing, we use the final model obtained after training is completed.

**Q: What is the overall classification performance in the validation and test datasets and considering the different classes?**
The overall classification performance is measured through the accuracy, we obtain for the validation accuracy: **88.96%** (32 neurons), **88.80%** (64 neurons) and **89.08%** (128 neurons) and for the test accuracy: **88.55%** (32 neurons), **88.53%** (64 neurons), **88.72%** (128 neurons). The performance per class is measured through precision, recall and f1-score. Considering the different classes, the overall classification performance in the validation and test datasets are the following ones:

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 89.20 | 95.50 | 92.24 | 88.89 | 95.40 | 92.03 |
| Brute Force | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Dos Hulk | 98.16 | 88.75 | 93.22 | 98.69 | 86.02 | 91.92 |
| Port Scan | 82.24 | 88.74 | 85.37 | 81.70 | 89.88 | 85.60 |

Table 1: Validation and Test Scores with 32 neurons

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 89.17 | 95.39 | 92.18 | 88.88 | 95.46 | 92.05 |
| Brute Force | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Dos Hulk | 97.88 | 88.75 | 93.09 | 98.69 | 86.02 | 91.92 |
| Port Scan | 81.68 | 88.21 | 84.82 | 81.65 | 89.57 | 85.43 |

Table 2: Validation and Test Scores with 64 neurons

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 89.22 | 95.68 | 92.33 | 88.88 | 95.72 | 92.17 |
| Brute Force | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Dos Hulk | 98.44 | 89.00 | 93.49 | 98.69 | 86.02 | 91.92 |
| Port Scan | 82.53 | 88.53 | 85.42 | 82.62 | 89.67 | 86.00 |

Table 3: Validation and Test Scores with 128 neurons

**Q: Why is the performance of the model so poor?**
If we look at the per class metrics we can say that the model does not perform well because is too simple, indeed we have only single linear layer and we do not consider the class imbalance. The model is not able to classify the Brute Force samples.

## 2.3 Using ReLU as activation function

After finding that the 128-neuron model performed best, we replaced its activation function with ReLU. Fig. 3 and Fig. 2 show the confusion matrices of the linear model and the non linear one with 128 neurons. We can see that the performance improves overall, especially for the class 1 (Brute Force) which is the less represented class. We obtained for the validation accuracy **95.64%** and for test accuracy **95.63%**. The performance per class is shown in table 4.

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 96.44 | 97.36 | 96.90 | 96.11 | 97.51 | 96.80 |
| Brute Force | 80.73 | 94.29 | 86.99 | 83.84 | 94.83 | 89.00 |
| Dos Hulk | 99.45 | 92.20 | 95.69 | 99.72 | 90.85 | 95.08 |
| Port Scan | 94.58 | 91.89 | 93.22 | 94.59 | 92.20 | 93.38 |

Table 4: Validation and Test Scores with 128 neurons and ReLU

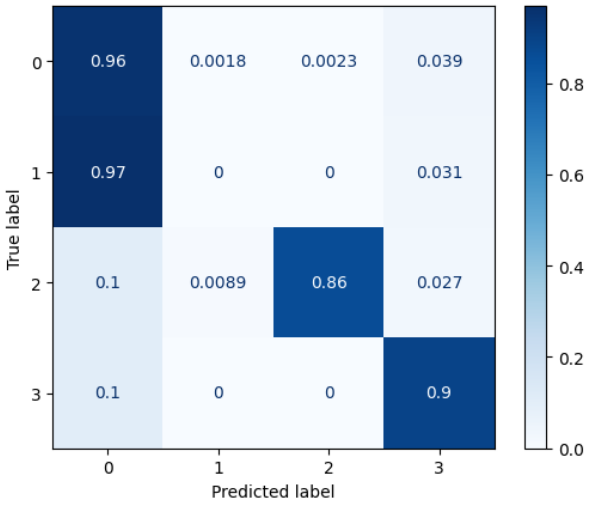



Figure 2: Linear model

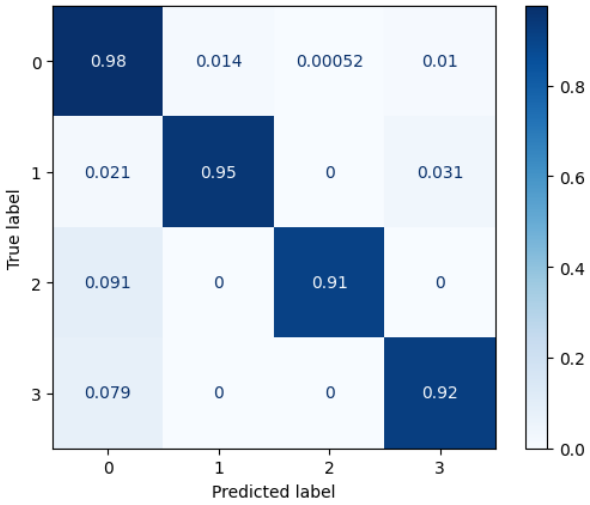



Figure 3: Non linear model

# 3 Task 3: The impact of specific features

**Q: All Brute Force attacks in your dataset originate from port 80. Is this a reasonable assumption?**
No. While all brute-force attacks within our dataset originate from port 80, this is a characteristic of the dataset itself, not a universal truth about brute-force attacks. By training the model on this data, it will learn a strong bias and incorrectly associate brute-force attacks exclusively with port 80. Because of this inductive bias, the model will perform poorly on new data where those same attacks might use different ports, which is a major problem for real-world application.

## 3.1 Replace port 80 with port 8080

We replaced port 80 with port 8080 for the Brute Force attacks in the test set. Then we tested the previously trained model, the accuracy is **91.27%** and the performance per class is the following one:

| Class | Precision | Recall | F1 |
|---|---|---|---|
| Benign | 90.19 | 97.50 | 93.71 |
| Brute Force | 26.38 | 6.55 | 10.49 |
| Dos Hulk | 99.72 | 90.85 | 95.07 |
| Port Scan | 94.59 | 92.20 | 93.38 |

Table 5: Performance metrics for different classes.

**Q: Does the performance change? How does it change? Why?**
The performance is significantly worse than before because the model was trained to classify Brute Force attacks based on data where these attacks occurred on port 80, so it learned to associate Brute Force attacks with that port. When the test data presented brute-force attacks on port 8080, the model failed to recognize them.

## 3.2 Remove the feature port from the original dataset

**Q: How many PortScan do you now have after preprocessing How many did you have before?**
After removing the 'port' feature and repeating all preprocessing steps, the number of PortScan instances was reduced from **4849** to **285**.

**Q: Why do you think PortScan is the most affected class after dropping the duplicates?**
PortScan is the most affected class since its entries previously differed only by the port feature. Without the port column, almost all PortScan records became duplicates on the remaining features so were eliminated when duplicates were dropped.

**Q: Are the classes now balanced?**
Now the dataset is unbalanced due to the under-representation of the PortScan class.

## 3.3 Training process

We repeated the training process with the best architecture found in the previous step.

**Q: How does the performance change? Can you still classify the rarest class?**
The validation accuracy is **94.93%** and the performance per class is the following one:

| Class | Precision | Recall | F1 |
|---|---|---|---|
| Benign | 95.44 | 98.10 | 96.75 |
| Brute Force | 81.27 | 92.75 | 86.63 |
| Dos Hulk | 99.86 | 88.12 | 93.62 |
| Port Scan | 28.57 | 7.41 | 11.76 |

Table 6: Performance metrics for different classes.

The performance on the Port Scan class is very poor, as the model fails to correctly classify samples belonging to this class.

## 3.4 Using weighted loss

We applied weighted loss and repeated the training process with the new loss.

**Q: How does the performance change per class and overall? In particular, how does the accuracy change? How does the f1 score change?**
The validation accuracy in the previous model was **94.93%** while now it's **91.30%** so the overall performance seems to not improve and if we look at the performance per class, we have:

| Class | Precision | Recall | F1 |
|---|---|---|---|
| Benign | 98.80 | 90.54 | 94.49 |
| Brute Force | 74.85 | 93.84 | 83.28 |
| Dos Hulk | 91.12 | 93.43 | 92.26 |
| Port Scan | 20.73 | 94.44 | 34.00 |

Table 7: Performance metrics for different classes

But if we compare the two confusion matrices we can notice that there are fewer false negatives after applying the weighted loss. Even though the number of false positives increases, it's better to classify something that was not an attack as an attack, rather than failing to classify an actual attack as such.

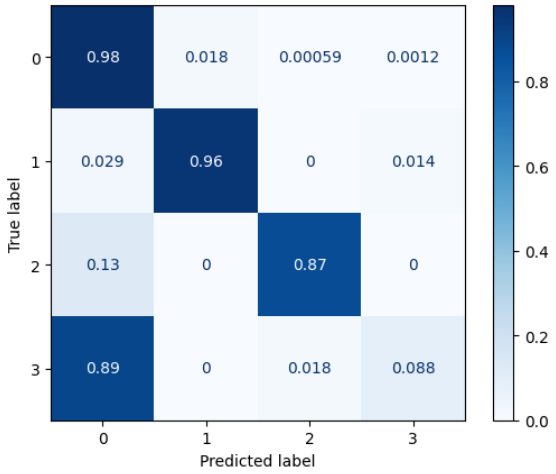



Figure 4: Before weighted loss

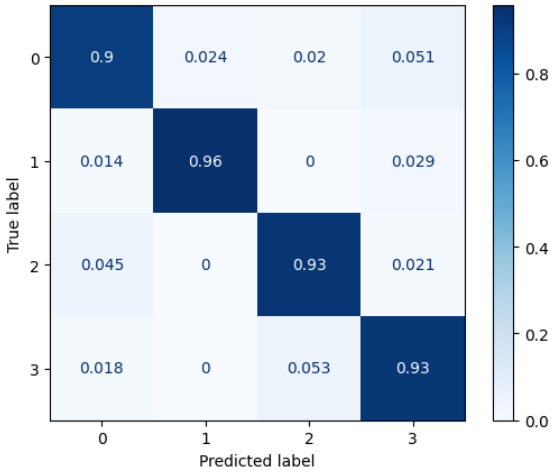



Figure 5: After weighted loss

# 4 Task 4: Deep Neural Network

## 4.1 Design the first Deep Learning

We define three different architectures:
The first architecture consist of three layer: the first layer maps the input to 16 neurons, followed by a second layer with 8 neurons, and finally an output layer.
The second architecture has four layers: the input is first mapped to 32 neurons, then to 16 neurons, followed by 8 neurons, and finally to the output layer.
The third architecture consists of five layers. The first layer maps the input to 32 neurons, then successively into 16, 8, and 4 neurons, before mapping to the output.
The losses of the three architectures are very similar, during the first 15 epochs, all models' losses decrease at the same rate. The most noticeable difference is in how many epochs it takes for the loss to stabilize, the first and third architectures require 50 epochs, whereas the second stabilizes after 30 epochs.

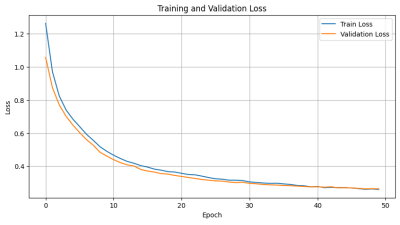



Figure 6: Architecture 1

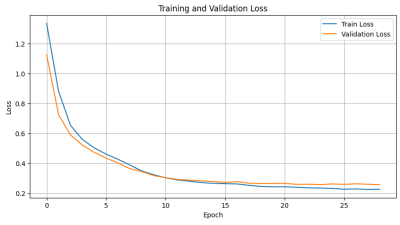



Figure 7: Architecture 2

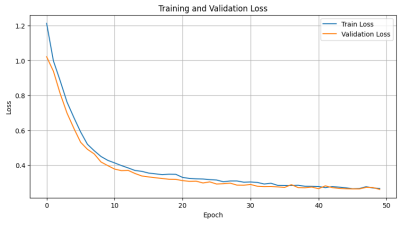



Figure 8: Architecture 3

We obtained for the validation accuracy: **91.94%** (Architecture 1), **90.65%** (Architecture 2), **92.65%** (Architecture 3) and for test accuracy: **92.16%** (Architecture 1), **90.96%** (Architecture 2), **93.01%** (Architecture 3). The overall classification performance in the validation and test sets is the following:

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 97.02 | 92.70 | 94.81 | 96.69 | 93.10 | 94.87 |
| Brute Force | 71.03 | 94.20 | 80.99 | 73.33 | 95.65 | 83.01 |
| Dos Hulk | 96.40 | 88.11 | 92.07 | 97.05 | 87.15 | 91.83 |
| Port Scan | 26.37 | 88.88 | 40.67 | 28.49 | 89.47 | 43.22 |

Table 8: Architecture 1

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 98.63 | 89.83 | 94.02 | 98.59 | 89.87 | 94.03 |
| Brute Force | 71.94 | 93.84 | 81.44 | 73.13 | 95.65 | 82.88 |
| Dos Hulk | 87.20 | 93.04 | 90.03 | 87.36 | 94.08 | 90.60 |
| Port Scan | 22.47 | 90.74 | 36.02 | 24.40 | 89.47 | 38.34 |

Table 9: Architecture 2

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 97.90 | 92.88 | 95.32 | 97.93 | 93.13 | 95.47 |
| Brute Force | 76.56 | 88.76 | 82.21 | 76.92 | 90.57 | 83.19 |
| Dos Hulk | 89.46 | 93.42 | 91.40 | 90.83 | 93.57 | 92.18 |
| Port Scan | 31.75 | 87.03 | 46.53 | 34.22 | 89.47 | 49.51 |

Table 10: Architecture 3

All three architectures demonstrate good performance on benign, brute force and dos traffic, but Architecture 3 achieves the highest F1 scores across every class. Its most significant gains appear in Port Scan detection, which is the most challenging class. Hence we consider the architecture 3 the best one.

## 4.2 The impact of batch size

**Q: Does the performance change? And why? Report both the validation and test results.** The batch size influences the performance of the model because it corresponds to the number of samples processed in each epoch. Small batches (from 1 to 64) generates noisy estimates and the training is slow but the updates are fast, it provides good generalization for small dataset and does not require a lot of memory. Large batches (from 64) generates smooth gradient estimate, the training is fast but it may have poor generalization and requires more memory. We obtained for the validation accuracy: **94.97%** (Size 1), **89.00%** (Size 32), **90.34%** (Size 64), **89.47%** (Size 128), **86.04%** (Size 512) and for test accuracy: **94.72%** (Size 1), **89.47%** (Size 32), **90.65%** (Size 64), **89.80%** (Size 128), **86.47%** (Size 512). The overall classification performance in the validation and test sets is the following:

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 95.25 | 98.19 | 96.70 | 95.18 | 97.95 | 96.55 |
| Brute Force | 81.33 | 93.12 | 86.82 | 80.73 | 95.65 | 87.56 |
| Dos Hulk | 99.71 | 88.12 | 93.56 | 99.14 | 87.41 | 92.90 |
| Port Scan | 100.00 | 3.70 | 7.14 | 50.00 | 1.75 | 3.39 |

Table 11: Batch size 1

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 99.23 | 88.23 | 93.41 | 99.01 | 88.92 | 93.69 |
| Brute Force | 64.84 | 94.20 | 76.81 | 66.17 | 95.65 | 78.22 |
| Dos Hulk | 82.39 | 89.89 | 85.97 | 85.28 | 89.04 | 87.12 |
| Port Scan | 22.94 | 98.15 | 37.19 | 23.14 | 98.25 | 37.46 |

Table 12: Batch size 32

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 98.64 | 90.54 | 94.42 | 98.55 | 91.06 | 94.66 |
| Brute Force | 74.00 | 93.84 | 82.75 | 76.08 | 95.65 | 84.75 |
| Dos Hulk | 99.14 | 87.86 | 93.16 | 99.14 | 87.03 | 92.69 |
| Port Scan | 14.99 | 96.30 | 25.94 | 15.63 | 92.98 | 26.77 |

Table 13: Batch size 64

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 98.90 | 87.76 | 92.99 | 98.70 | 88.06 | 93.08 |
| Brute Force | 78.72 | 93.84 | 85.62 | 79.04 | 95.65 | 86.56 |
| Dos Hulk | 83.87 | 95.32 | 89.23 | 84.56 | 95.21 | 89.57 |
| Port Scan | 17.58 | 88.89 | 29.36 | 19.47 | 89.47 | 31.97 |

Table 14: Batch size 128

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 99.00 | 85.09 | 91.52 | 98.87 | 85.65 | 91.79 |
| Brute Force | 68.01 | 91.67 | 78.09 | 68.17 | 93.12 | 78.71 |
| Dos Hulk | 98.01 | 87.36 | 92.38 | 98.15 | 86.90 | 92.18 |
| Port Scan | 10.23 | 98.15 | 18.53 | 11.07 | 96.49 | 19.86 |

Table 15: Batch size 512

Analysing the result the best batch size for this type of classification task is 32 because it provides the higher recall scores meaning that there are few false negatives. However the impact of the batch size is not so meaningful since the scores are similar, with the exception of batch size 1. Increasing the batch size too much can harm generalization, as reflected in the Port Scan f1 in the batch size 512 case.

**Q: How long does it take to train the models depending on the batch size? And why?**
The time needed to train the model is the following: **768 sec.** (Size 1), **44 sec.** (Size 32), **38 sec.** (Size 64), **24 sec.** (Size 128), **14 sec.** (Size 512). Increasing the batch size reduces the training time because fewer forward passes are required, resulting in fewer gradient updates.

## 4.3 The impact of the Activation Function

**Q: Does the performance change? Why does it change? Report both the validation and the test results.**
The performance changed because we used different activation functions. We obtained for the validation

accuracy: **81.13%** (Linear), **93.65%** (Sigmoid), **91,81%** (ReLU) and for test accuracy: **81.88%** (Linear), **93.54%** (Sigmoid), **92,05%** (ReLU). The overall classification performance in the validation and test sets is the following:

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 99.40 | 78.83 | 87.93 | 99.34 | 79.86 | 88.54 |
| Brute Force | 36.55 | 89.13 | 51.84 | 37.69 | 90.94 | 53.29 |
| Dos Hulk | 94.01 | 87.23 | 90.49 | 93.35 | 86.65 | 89.88 |
| Port Scan | 12.38 | 94.44 | 21.89 | 13.54 | 91.23 | 23.58 |

Table 16: Linear

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 94.83 | 96.86 | 95.83 | 94.71 | 96.82 | 95.76 |
| Brute Force | 72.70 | 88.77 | 79.93 | 72.67 | 90.58 | 80.65 |
| Dos Hulk | 97.89 | 88.12 | 92.75 | 98.02 | 87.41 | 92.41 |
| Port Scan | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 17: Sigmoid

| Class | Validation | | | Test | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| Benign | 98.81 | 90.78 | 94.62 | 98.65 | 91.03 | 94.69 |
| Brute Force | 72.96 | 93.84 | 82.09 | 74.37 | 95.65 | 83.68 |
| Dos Hulk | 82.91 | 95.70 | 88.85 | 83.48 | 95.47 | 89.07 |
| Port Scan | 37.80 | 88.89 | 53.04 | 40.32 | 87.72 | 55.25 |

Table 18: ReLU

**Q: Explain why and how the different activation functions affect performance in this architecture.**
Using a linear activation function, the layers of the neural network collapse into a single layer with a single linear transform. This lead to high recall but low precision. The network learns decision boundaries that can separate only the easiest patterns. Using a sigmoid introduces a non linearity in the network but this can cause the vanishing gradient problem since this function is bounded between 0 and 1. Even if the performance on the first 4 classes is not so poor, the network is not able to classify samples belonging to the port scan class. Using ReLU the network shows the best performance. It introduces non linearity in the model, solves the vanishing gradient problem because it is not bounded as the sigmoid and allows the network to learn good decision boundaries.

## 4.4 The impact of the Optimizer

**Q: Is there a difference in the trend of the loss functions?**
Using SGD the loss decrease is gradual, there is a noticeable gap between training and validation loss and the training loss is not so smooth.
Using SGD with momentum 0.1 there is a faster initial descent than SGD, but it is smoother; with momentum 0.5 there is a phases of rapid decline of the loss around epochs 10-25 then it decreased slowly; With momentum 0.9 there are two phases of rapid decline around epochs 0-8 and 15-25 then the loss stabilizes and reach the lowest value of 0.2.
Using AdamW the loss decreases rapidly for the first 15 epochs then it stabilizes. The performance of the model is very poor with SGD and SGD with momentum 0.1 and 0.5, this is also clear from the value of the validation loss. The best performance is achieved with SGD with momentum 0.9 and AdamW.

**Q: How long does it take to train the models with the different optimizers? And why?**
The time needed to train the models is about the same for all the optimizers (between 50 and 60 seconds) with the difference that SGD required 40 epochs to stabilize, SGD with momentum 50 epochs, AdamW 37 epochs. AdamW converges faster because it uses momentum for the gradient estimate and the squared gradient so it maintains the memory of the past gradient (like SGD with momentum) and rescale it using the square root of the squared gradient. It uses an exponent to decay the correction, the change is significant in the early steps but negligible later.

**Q: Evaluate the effects of the different learning rates and epochs for the best optimizer and architecture found above**
We evaluated the best optimizer (AdamW) with a low learning rate (0.00001) and a high learning rate (0.05). In the first case after 50 epochs the loss started to stabilize around the value 1, and the model achieved low performance. In the second case the model started to overfit and the early stopping triggered at epoch 9 leading to very poor performance.

# 5 Task 5: Overfitting and Regularization

**Q: What do the losses look like? Is the model overfitting?**
Yes, the model is overfitting and this is noticeable by the losses: the validation after epoch 5 increases while the training loss decreases. This means that the model is adapting too much to the training samples. The loss is shown in Fig. 9.

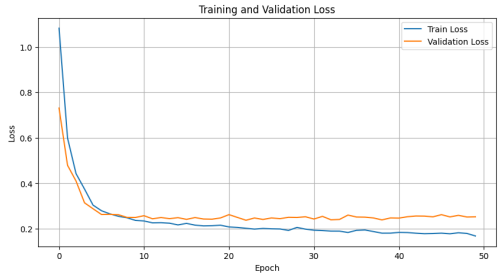



Figure 9: Before regularization

**Q: What impact do the different normalization techniques have on validation and testing performance?**
Introducing in the model dropout layers reduces overfitting preventing the neurons to become dependent by each other and allowing to learn more specific features of the data. Batch normalization reduces overfitting making the model less sensible to weight initialization. AdamW's weight regularization reduce overfitting encouragin the model to generalize better discouraging weights specific to training data. From the plots in Fig. 10, 11 and 12 it is possible to observe that using dropout reduces significantly overfitting and combining it with batch normalization makes the validation loss smoother. Moreover AdamW's weight decay provides the same improvement as dropout + batch normalization.

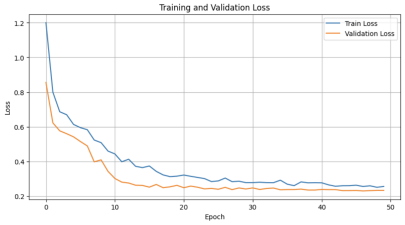



Figure 10: Dropout


Figure 11: Dropout + batch norm


Figure 12: Weight decay (0.0001)