Lab-1
Group-5

Dennis D'Amico - s343841, Pierre Perrier - s350300,
Manuele Sforzini - s349555

0 Goal of the laboratory

This laboratory focuses on making the student experience first-hand how a simple neural network is built and how hyperparameters, optimizers and architectures impact on the result of a neural network.

In particular, this laboratory asks the students to create a neural network that is able to correctly classify the content of a dataset into Benign, PortScan, DoS Hulk and Brute Force, given a .csv file with labeled data streams.

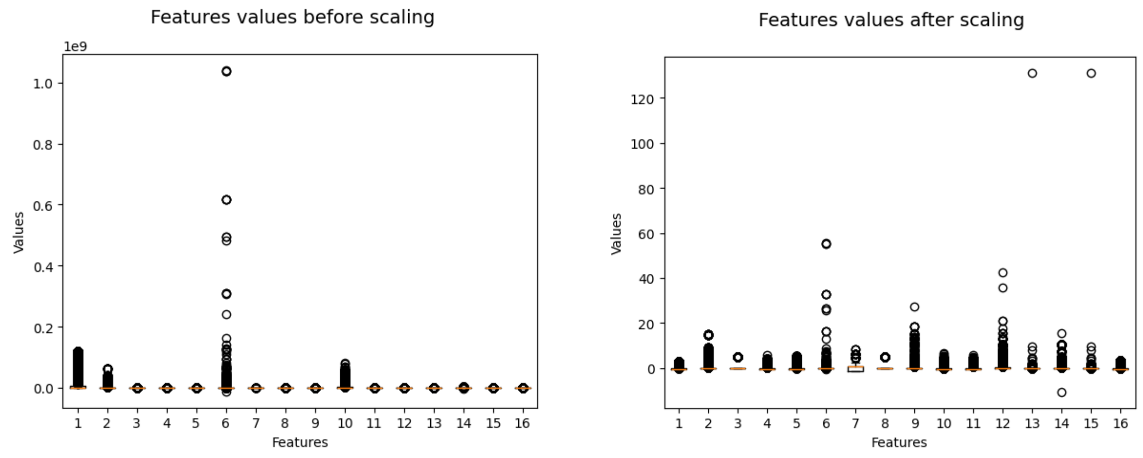
1 Task 1: Preprocessing

The first operation required by the laboratory is to preprocess the given dataset:

Q:How many samples did you have before and after removing missing and duplicates entries? We first set infinites as NaN with the $pd.set\_option('mode.use\_inf\_as\_na', True)$ command and then removed all null values with $df = df.dropna()$, passing from 31507 rows in the original dataset to 31480 after removing null values and to 29386 after removing duplicates. We decided to remove rows containing null values as they were not so many, so removing them does not prevent us to create an effective classifier.
We then set the seed to add randomness and split the dataset into training, validation and test sets, using the following commands:

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

Q:How did you normalize the data? Why did you choose it? We chose to use the StandardScaler to normalize the data, as it is designed to work with numerical features, like the ones in this dataset, and works well also in presence of outliers. It computes the mean and the standard deviation of each feature.



(a) Boxplot before standardization

(b) Boxplot after standardization

Figure 1: Comparison of data feature with and without scaling

As seen from Figure 1, before standardization there were some outliers in class 6, whose values were close to 1e9. Instead, after standardization, there are still some outliers, but they are closer in range to normal values. Probably, the neural network would not be able to process these high values, which would be likely classified as infinite. So, as stated before, it was decided to just remove them from the dataset, as the removal should not have a big impact in the performance.

2 Task 2: Shallow Neural Network

The second task asks to create 3 different models, differing from each other in the number of neurons per layer. In particular, the first model has 32 neurons, the second 64 and the third 128. In our solution, we created a variable *neurons_list* $= [32, 64, 128]$ that was cycled in a for loop and, for each iteration, the number of neurons was given as input to the model with the following command:

```python
neurons_list = [32, 64, 128]
for neurons in neurons_list:
    model = ShallowNN(16, neurons)
```

Where the input parameter equal to 16 corresponds to the number of features in the dataset.
Here is the implementation of the `ShallowNN` class:

```python
class ShallowNN(nn.Module):
    def __init__(self, input_size, hidden_units):
        super(ShallowNN, self).__init__()
        self.hidden_layer = nn.Linear(in_features=input_size,
            out_features=hidden_units)  # Number of neurons
        self.output_layer = nn.Linear(in_features=hidden_units, out_features=4)

    def forward(self, x):
        x = self.hidden_layer(x)
        return self.output_layer(x)
```

In this function, the number of neurons passed is used to correctly set the number of neurons in the hidden layers.

Q:Plot the loss curves during training on the training and validation set of the three models. What is their evolution? Do they converge? As seen in Figure 2, the losses in each model converge. While the first model converges and stops at epoch 84, the second model at epoch 77 and the last model at around epoch 64, all of them with best validation loss of about 0.29. **Q: How do you select the best model across epochs?** As hinted from the previous answer, the best model is selected with early stopping: as the validation loss does not improve for a different number of epochs, the training is stopped to prevent overfitting. In the end, the model is restored to this state. **Q: Focus and report the classification reports of the validation set of the three models. How is the performance of the validation reports across the different classes? Is the performance good or poor? Why?** The performance changes for the different classes. In particular, each model struggles with class 1 (*Brute Force*), which is never learned and is also the least represented class, as shown in Table 1. This is probably why class 1 is the hardest to be learned.

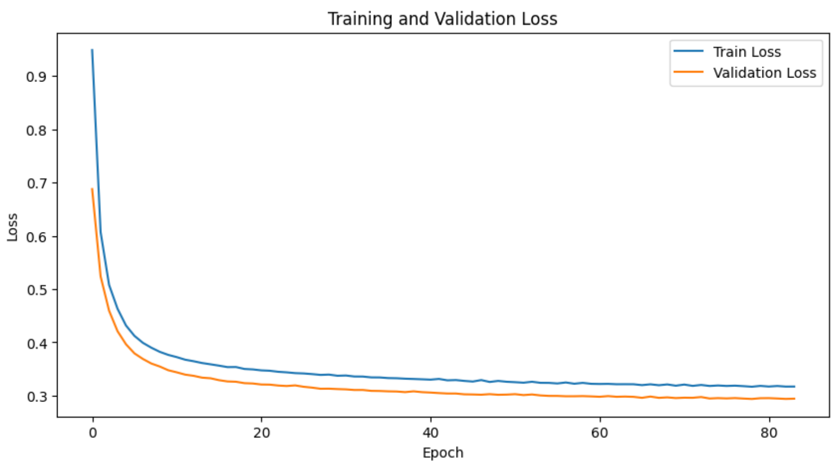
	Benign	**Brute Force**	**Dos Hulk**	**Port Scan**
Training set	11515	866	2292	2958
Validation set	3900	286	776	915

Table 1: Number of samples per class in Training and Validation set

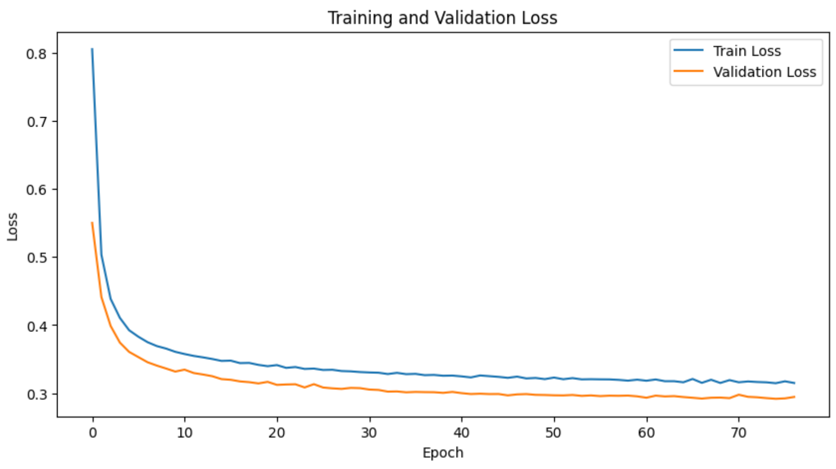
For the other classes, instead, the model provides a good performance, with high values in precision and recall. The reports are shown in Table 2a, 2b and 2c.

Q: Now, focus on the best model you chose. Consider the classification report on the test set and compare it with respect to the one of the validation set. Is the performance similar? I.e., does the model generalize? We consider the model with ***64 neurons*** as the best one, in fact it has very similar results in comparison to the one with ***128 neurons*** and, having less neurons, it is more efficient. The performance in validation and test is very similar, in particular we get

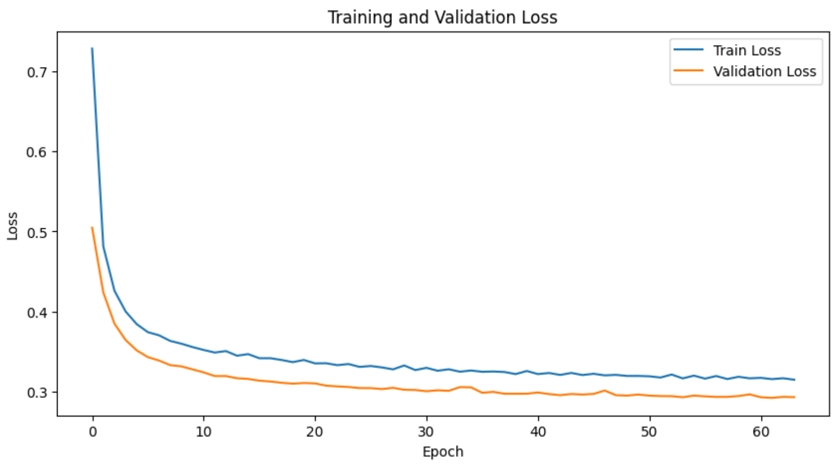
- Validation Accuracy: 88.4465



(a) Training with 32 neurons



(b) Training with 64 neurons



(c) Training with 128 neurons

Figure 2: Training curve of neural network with 32, 64, 128 neurons

- Test Accuracy: 88.6356

The precision and recall values for both sets are also closely aligned, indicating that the model generalizes well and is not overfitting. The 64-neuron model provides a good balance between performance and efficiency, making it the optimal choice.

Now the best model is updated, changing the activation function to *ReLu*.

Q:Focus and report the classification report of the validation set. Does the model

perform better in a specific class? Yes, now class 1 is finally learned by the model, as we can see in Table 2d.

(a) 32 neurons with Linear Activation

	precision	recall	f1-score
0	0.90	0.95	0.92
1	0.00	0.00	0.00
2	0.99	0.87	0.93
3	0.78	0.90	0.84
accuracy			0.88
macro avg	0.67	0.68	0.67
weighted avg	**0.85**	0.88	0.86

(b) 64 neurons with Linear Activation

	precision	recall	f1-score
0	0.89	0.95	0.92
1	0.00	0.00	0.00
2	0.99	0.87	0.93
3	0.78	0.89	0.83
accuracy			0.88
macro avg	0.67	0.68	0.67
weighted avg	**0.85**	0.88	0.86

(c) 128 neurons with Linear Activation

	precision	recall	f1-score
0	0.89	0.96	0.92
1	0.00	0.00	0.00
2	0.99	0.87	0.92
3	0.83	0.91	0.87
accuracy			0.89
macro avg	0.68	0.68	0.68
weighted avg	**0.85**	0.89	0.87

(d) 64 neurons with ReLu

	precision	recall	f1-score
0	0.96	0.98	0.97
1	0.82	0.95	0.88
2	1.00	0.90	0.95
3	0.96	0.92	0.94
accuracy			0.96
macro avg	0.94	0.94	0.94
weighted avg	**0.96**	0.96	0.96

Table 2: Validation Classification report for different architectures

Would it be correct to compare the results on the test set? The test set is always used to determine whether a model is good or not, it is never used to compare results and adjust hyperparameters.

3 Task 3: The impact of Specific Features

The third task was about analyzing the impact that some specific features in the dataset may have on the classification results.
In particular, we know that the dataset has some biases, as for example the fact that all the rows labeled as *Brute Force* have *Destination Port* = 80.

Q: Is this a reasonable assumption? This is clearly not a reasonable assumption, as it may force the Neural Network to learn that Brute Force attacks can happen only against port 80; so, if a Brute Force attack targets another port, it would not be recognized by the network.

Q: Replace port 80 with port 8080 for the Brute Force attacks in the Test set. Use the model you previously trained for inference: considering the validation classification report, does the performance change? How does it change? Why? If we change port 80 with port 8080 for Brute Force attacks in the Test set, using the model trained before, what we get is that the performance of the model changes, in particular regarding class 1, the one related to Brute Force attacks. We can see a huge drop in all the metric values for class 1:

Metric	Precision	Recall	f1-score
Validation	0.82	0.95	0.88
Test	0.16	0.05	0.07

Table 3: Metrics in the Validation report and Test report, the latter after the update in the dataset

The other classes are not affected that much, we have only small differences with respect to the validation report. We also have a difference in the Test Accuracy:

- Test Accuracy with the original dataset: 95.7128%
- Test Accuracy with the updated dataset: 91.4086%

This change in the metric values happens due to the assumption that all the brute force attacks in the training set happened on port 80.
Now the feature *Destination Port* is removed from the original dataset and all the preprocessing steps are repeated.

Q: How many PortScan do you now have after preprocessing (e.g., removing duplicates)? How many did you have before? We can see that in this case the modification in the dataset had an important impact on the rows labeled as *PortScan*: in fact, there were originally 4999 rows with this label, while now, after having removed duplicates, there are only 285 of them.

Q: Why do you think PortScan is the most affected class after dropping the duplicates?*PortScan* is certainly the most affected class by the duplicates drop because a Port-Scan attack is strictly connected to the destination port feature, as a scanner interacts with different destination ports to find, for example, some services that he can interact with.

Q: Are the classes now balanced? Now classes are not balanced: the number of samples for each class are reported in Table 4

Class	Benign	Brute Force	DoS Hulk	PortScan
Count	16889	1427	3868	285

Table 4: Number of samples per class

4 Task 4: The impact of the Loss Function

Now, using the best model found in a previous task, the training process is repeated with the updated version of the dataset.

Q: How does the performance change? Can you still classify the rarest class? The least represented class, *PortScan*, now is more difficult to be classified, in the test we can see this fact given the bad values for the metrics: Precision, Recall and f1-score, respectively 0.50, 0.27 and 0.35

Instead, the other classes are not affected that much by the update in the dataset.
Now, to improve performance, a weighted loss function is introduced via the parameter *weight* in the *CrossEntropyLoss* function.

Q: Which partition do you use to estimate the class weights? Only the training partition was used to estimate the weights, as the test partition should remain completely unseen until tests are performed.

Q: How does the performance change per class and overall? In particular, how does the accuracy change? How does the f1 score change? Overall, after introducing a weighted loss, we have a small decrement in the accuracy in training, validation and test, that is reported in the following table:

Effect on Accuracy	**Before**	**After**
Training	95.2229	92.6044
Validation	95.1268	93.1019
Test	94.8598	92.2786

Table 5: Effect of the introduction of a weighted loss on the accuracy

Looking at the specific classes, instead, we have small changes in precision and recall values for the first three classes, while for class 3 we have a much higher recall, 0.94 in test and 0.97 in validation, with a drop in precision, that goes from 0.50 to 0.22 in the tests. The f1 score remains almost the same in both experiments.
The complete Test Classification Reports, for both cases, are reported as follows:

	precision	recall	f1-score
0	0.96	0.97	0.97
1	0.78	0.96	0.86
2	1.00	0.88	0.94
3	0.50	0.27	0.35
accuracy			0.95
macro avg	0.81	0.77	0.78
weighted avg	**0.95**	0.95	0.95

(a) Test Classification report before the introduction of a weighted loss

	precision	recall	f1-score
0	0.99	0.92	0.95
1	0.75	0.96	0.85
2	0.94	0.92	0.93
3	0.22	0.94	0.36
accuracy			0.92
macro avg	0.73	0.94	0.77
weighted avg	**0.96**	0.92	0.93

(b) Test Classification report after having introduced a weighted loss

5 Task 5: Deep Neural Network

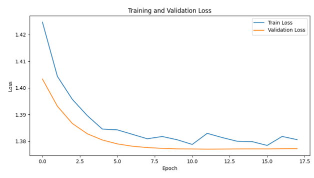
5.1 Design the first Deep Learning

The fifth task required to extend the previously created Shallow Neural Network to create a Deep Learning Feed Forward Neural Network. Our group decided to analyze the following architectures:

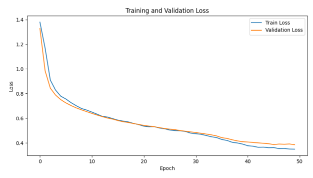
Table 7: Architectures analyzed for Deep Feed Forward Neural Network

Name	**Layer structure**
arch_1	[2, 2, 2]
arch_2	[4, 16, 8]
arch_3	[16, 16, 8, 4]
arch_4	[4, 4, 16, 8]
arch_5	[16, 16, 8, 8, 4]
arch_6	[4, 4, 16, 16, 8]

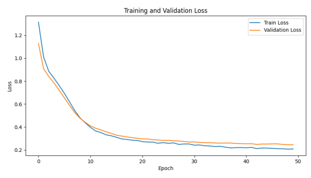
Q:Plot and analyze the losses. Do the models converge? The models all converge as shown but analyzing the accuracy we notice that arch_1 is the worst with 34% accuracy, arch_2 and arch_5 have 88% accuracy, arch_4 has 82% accuracy, arch_5 has 85% accuracy while arch_3 91%, also the f1 -core for class 3 is better with arch_3.



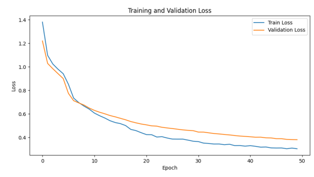
(a) arch_1



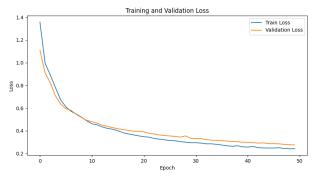
(b) arch_2



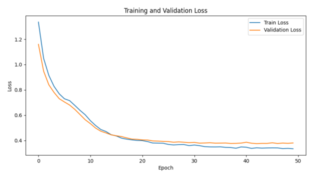
(c) arch_3



(d) arch_4



(e) arch_5



(f) arch_6

Figure 3: Training loss curve for 6 architectures

Q:Calculate the performance in the validation set and identify the best-performing architecture. How do you select one? The best-performing architecture is the third one, with *4 layers and [16, 16, 8, 4] neurons.* We chose it by running the code different times (and so on different dataset splits) and by noticing that it was the most consistent architecture and often also the one with the best performances. It kept every time an accuracy of about 90-92% in training, validation and test and it was also very good in recognizing every class, even class 3 that is the rarest one.

5.2 The impact of Batch Size

We study the impact of different batch sizes: {4, 64, 256, 1024}, we use the `arch_3` described above.

Q:Does performance change? And why? Report the validation results. With smaller batch size, the model tends to have better generalization than with greater batch size, this is why smaller batch size has better performances in general. We can see the the smaller the batch size is the better the f1-score is for class 3.

(a) Validation - batch size = 4

	precision	recall	f1-score
0	0.98	0.95	0.97
1	0.80	0.93	0.86
2	0.93	0.94	0.93
3	0.48	0.88	**0.62**
accuracy			0.95
macro avg	0.80	0.92	0.84
weighted avg	**0.96**	0.95	0.95

(b) Validation - batch size = 64

	precision	recall	f1-score
0	0.97	0.93	0.95
1	0.73	0.93	0.82
2	0.98	0.87	0.92
3	0.25	0.91	**0.39**
accuracy			0.92
macro avg	0.73	0.91	0.77
weighted avg	**0.95**	0.92	0.93

(c) Validation - batch size = 256

	precision	recall	f1-score
0	0.99	0.83	0.90
1	0.39	0.93	0.55
2	0.95	0.86	0.91
3	0.18	0.98	**0.31**
accuracy			0.84
macro avg	0.63	0.90	0.67
weighted avg	**0.93**	0.84	0.87

(d) Validation - batch size = 1024

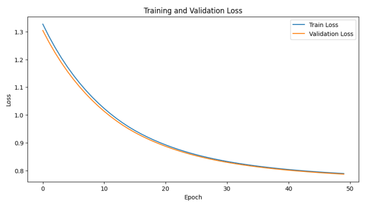
	precision	recall	f1-score
0	0.89	0.99	0.94
1	0.00	0.00	0.00
2	0.96	0.87	0.91
3	0.00	0.00	**0.00**
accuracy			0.90
macro avg	0.46	0.47	0.46
weighted avg	**0.83**	0.90	0.86

Table 8: Validation results for different batch size

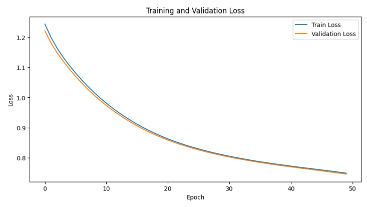
Q:How long does it take to train the models depending on the batch size? And why? The smaller the batch size, the longer the training takes, because *the gradient is updated more frequently* and *each epoch takes longer to process.*

5.3 The impact of the Optimizer

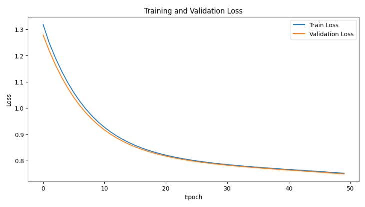
Q:Is there a difference in the trend of the loss functions? SDG is slow to converge, SDG with momentum converge faster and AdamW converge even faster.



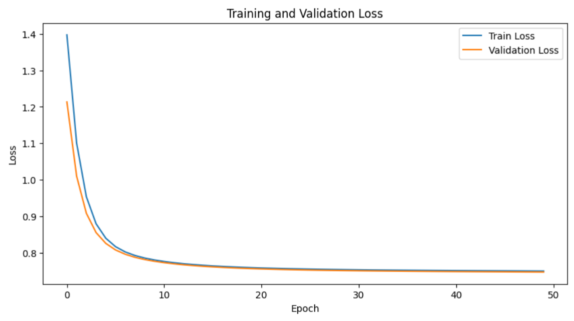
(a) SDG



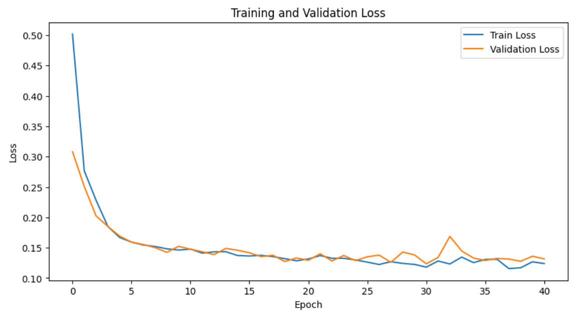
(b) SDG Momentum 0.1



(c) SDG Momentum 0.5



(d) SDG Momentum 0.9



(e) AdamW

Figure 4: Training loss curve for 5 optimizers

Q:How long does it take to train the models with the different optimizers? And why? SDG needs more epochs because the convergence is slow but the epochs are quite fast because they do not demand to much computation, while AdamW needs less epochs to converge but each epochs takes longer because it requires more calculation.

Now the focus is put on the best optimizer, as we can see in 9, the best optimizer is *AdamW*, since it lead to the best performances in terms of accuracy.

	SGD	SGD Momentum=0.1	SGD Momentum=0.5	SGD Momentum=0.9	AdamW
Train accuracy	75.16	75.16	75.15	89.93	96.39
Validation accuracy	75.14	75.14	75.14	90.21	96.52
Test accuracy	75.18	75.18	75.19	89.63	96.53

Table 9: Accuracy for the different optimizers

Q:Evaluate the effects of the different learning rates and epochs. Report the test results for the best model. The following learning rates are tested with `AdamW` : `[0.0001, 0.0005, 0.0008, 0.001, 0.005, 0.008, 0.01]`. A smaller learning rate leads to a slower convergence, a higher rate converge faster but is also less stable. Also, feature 3 is difficult to learn because it is the least represented, with a small lr the model doesn't learn feature 3. The best model is the one with learning rate $lr = 0.001$, that lead to the results presented in 10.

The following epochs size are tested : `[30, 40, 50, 60, 70, 80, 90, 100]`. With greater epochs we have a better generalization until we reach `80`, then the the validation accuracy does not increase, we just learn the data and not the pattern anymore. We increase the weighted f1-score. Of course, the training time increase with `num_epochs`.

	precision	recall	f1-score
0	0.99	0.90	0.94
1	0.79	0.93	0.85
2	0.99	0.87	0.93
3	0.13	0.98	**0.24**
accuracy			0.89
macro avg	0.73	0.92	0.74
weighted avg	0.97	0.89	**0.92**

Table 10: Validation - AdamW with lr = 0.001

	precision	recall	f1-score
0	0.97	0.95	0.96
1	0.74	0.92	0.82
2	1.00	0.87	0.93
3	0.33	0.93	**0.49**
accuracy			0.94
macro avg	0.76	0.92	0.80
weighted avg	0.95	0.94	**0.94**

Table 11: Validation - AdamW with epochs = 80

6 Task 6: Overfitting and Regularization

The last task of the laboratory is based on a Feed Forward Neural Network that has the following characteristics:

# Hyperparameter	**Value**
# Layers	6
# Neurons per Layer	{256, 128, 64, 32, 16}
Activation	ReLu
Weight Initialization	Default
Batch Size	128
Loss Function	Cross-Entropy
Optimizer	AdamW
Learning Rate	0.0005
Epochs & Early Stopping	50
Regularization	None

Figure 5: The architecture of the FFNN

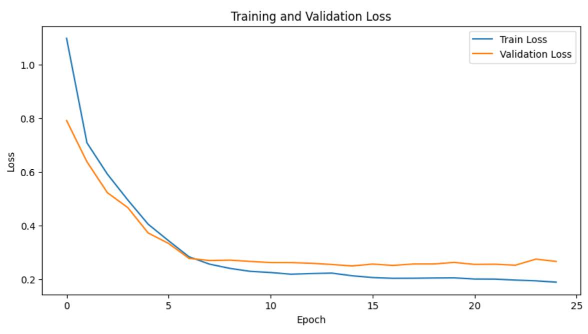
Q: What do the losses look like? Is the model overfitting? The losses, both the training one and the validation one, converge in a few epochs and follow also a similar trend (Figure 6a). The model is not overfitting: it is possible to see this fact by looking at the test accuracy (Table 12), which is very close to the training and the validation accuracy.

	Accuracy
Training	92.84
Validation	93.44
Test	92.70

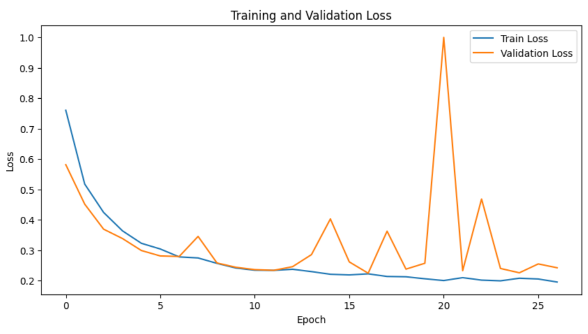
Table 12: Accuracy with the architecture in Figure 5

Now we tried to add some normalization techniques, like AdamW's weight decay, the dropout (with probability $p = 0.3$) and the batch normalization.

Q: What impact do the different normalization techniques have on validation and testing performance? In each case, the performances are not so far from the previous case. Looking at the losses, they are always very similar to the ones in Figure 6a except for the validation loss with Batch normalization that is very unstable, as can be seen in Figure 6b.



(a) Training and Validation loss with the architecture in Figure 5



(b) Training and Validation loss with Batch Normalization

The validation and test metrics obtained in the different cases are reported in Table 13 and Table 14.

(a) With no normalization

	precision	recall	f1-score
0	0.99	0.93	0.96
1	0.79	0.93	0.85
2	0.86	0.96	0.91
3	0.42	0.91	0.58
accuracy			0.93
macro avg	0.77	0.93	0.82
weighted avg	**0.95**	0.93	**0.94**

(b) With Dropout

	precision	recall	f1-score
0	0.99	0.91	0.95
1	0.75	0.93	0.83
2	0.87	0.95	0.91
3	0.31	0.91	0.46
accuracy			0.92
macro avg	0.73	0.93	0.79
weighted avg	**0.94**	0.92	**0.93**

(c) With Batch Normalization

	precision	recall	f1-score
0	0.99	0.93	0.96
1	0.76	0.93	0.84
2	0.89	0.96	0.92
3	0.42	0.91	0.57
accuracy			0.94
macro avg	0.76	0.93	0.82
weighted avg	**0.95**	0.94	**0.94**

(d) With Weight Decay

	precision	recall	f1-score
0	0.99	0.92	0.95
1	0.77	0.93	0.84
2	0.84	0.96	0.90
3	0.37	0.91	0.53
accuracy			0.92
macro avg	0.74	0.93	0.80
weighted avg	**0.94**	0.92	**0.93**

Table 13: Validation results for different normalization techniques

(a) With no normalization

	precision	recall	f1-score
0	0.99	0.89	0.94
1	0.75	0.96	0.85
2	0.82	0.96	0.88
3	0.24	0.87	0.38
accuracy			0.91
macro avg	0.70	0.92	0.76
weighted avg	**0.94**	0.91	0.92

(b) With Dropout

	precision	recall	f1-score
0	0.99	0.89	0.94
1	0.74	0.96	0.84
2	0.81	0.96	0.87
3	0.32	0.87	0.46
accuracy			0.91
macro avg	0.71	0.92	0.78
weighted avg	**0.93**	0.91	0.92

(c) With Batch Normalization

	precision	recall	f1-score
0	0.98	0.93	0.96
1	0.77	0.96	0.85
2	0.92	0.93	0.93
3	0.33	0.85	0.48
accuracy			0.94
macro avg	0.75	0.92	0.80
weighted avg	**0.95**	0.94	0.94

(d) With Weight Decay

	precision	recall	f1-score
0	0.99	0.88	0.94
1	0.77	0.92	0.84
2	0.84	0.89	0.86
3	0.18	0.98	0.30
accuracy			0.89
macro avg	0.69	0.92	0.73
weighted avg	**0.94**	0.89	0.91

Table 14: Test results for different normalization techniques