Project 1: Introduction to Deep Learning

Using a Feed Forward Neural Network (FFNN)

1 Objective

The goal of this project is to introduce students to PyTorch and help them understand the performance of shallow and deep neural networks in classification tasks.

Specifically, students will:

- Learn how to use PyTorch for model engineering, building, and training neural networks.
- Compare the performance of shallow and deep networks.
- Experiment with different hyperparameters, optimizers and architectures.
- Explore overfitting and apply normalization techniques such as dropout and batch normalization.

2 Submission Rules

- Groups consist of 3 students.
- Each group must submit a zip file containing the following:
  - A report (maximum 10 pages) describing the approach, experiments and results, including tables and plots.
  - The Juypyter notebook(s) and code (e.g., libraries with classes and functions written by you) to solve the tasks.
    - **Best practice**: Add comments and headers (Markdown) sections to understand what you are doing. They will i) help you tomorrow to understand what you did today and ii) help us to interpret your solution correctly.
    - **The notebook needs to be executed**: code and results must be visible so that we can interpret what you have done and what the results look like.
    - **Must run**: the code must work if we need to run it again.
    - **Submission format**: Include the notebook file (`.ipynb`) and an HTML export for easier review.
- Each group must upload the zip file to the teaching portal via Moodle before the deadline.

3 Dataset: CICIDS2017 Dataset Description

The CICIDS2017 dataset, developed by the Canadian Institute for Cybersecurity (CIC) at the University of New Brunswick, serves as a comprehensive benchmark for the evaluation of intrusion detection systems (IDS) and intrusion prevention systems (IPS). This dataset addresses the limitations of previous datasets by providing a realistic representation of modern network traffic that includes both benign activity and a variety of common cyberattacks.

More information about the CICIDS2017 dataset can be found at Kaggle - CICIDS2017 and on the website University of New Brunswick - CICIDS2017 pages.

3.1 Dataset Characteristics

- **Timeframe:** Data collection spanned five days, from 9 a.m. on Monday, July 3, 2017, to 5 p.m. on Friday, July 7, 2017. Each day focused on specific traffic patterns:
  - **Monday:** Exclusively benign traffic.
  - **Tuesday to Friday:** A mixture of benign traffic and various attack scenarios carried out both in the morning and in the afternoon.
- **Attack Types:** Your dataset contains a subset of the original attacks. You have the following attack vectors:
  - Benign
  - PortScan
  - DoS Hulk
  - Brute Force
- **Data Format:** The dataset consists of a CSV file with labeled data streams. Each flow is labeled with a timestamp, source and destination IPs, source and destination ports, protocols and attack types.

3.2 Key Features

We have selected a subset of the original features to make it easier for you to analyse. The selected features are:

- **Flow Duration**: Total duration of the network flow.
- **Flow IAT Mean**: Mean inter-arrival time between packets in the flow.
- **Fwd PSH Flags**: Count of PUSH (PSH) flags in forward direction.
- **Bwd Packet Length Mean**: Average packet length in the backward direction.
- **Bwd Packet Length Max**: Maximum packet length in the backward direction.
- **Flow Bytes/s**: Number of bytes per second in the flow.
- **Down/Up Ratio**: Ratio between the number of packets in the backward direction and the number of packets in the forward direction.
- **SYN Flag Count**: Count of packets for which the SYN flag is set.
- **Fwd Packet Length Mean**: Average length of the packets in the forward direction.
- **Fwd IAT Std**: Standard deviation of inter-arrival times between the forward packets.
- **Packet Length Mean**: Mean packet length over the entire flow.
- **Fwd Packet Length Max**: Maximum length of the packets in the forward direction.
- **Subflow Fwd Packets**: Number of packets in forward direction within a detected sub-flow.

- **Flow Packets/s**: Number of packets per second in the flow.
- **Label**: Classification label indicating the type of traffic (e.g., Benign, PortScan, etc.).
- **Total Fwd Packets**: Total number of packets sent in the forward direction.
- **Destination Port**: Port number of the destination host.

4 Tasks

Students must create a Machine Learning pipeline based on a Feed Forward Neural Network (FFNN) that performs all required steps. Each task describes the steps to follow.

4.1 Task 1: Data Preprocessing

The first task in developing a Machine Learning pipelines starts with a good preprocessing step. Start by preprocessing the dataset. **Report the most important steps** and intermediate results:

- Remove missing values (NaN) and duplicate entries.
- Ensure data consistency and correct formatting.
- Split the dataset to extract a training, validation and test sets (60%, 20%, 20%).
- Focus on the **training and validation partitions**. Check for the presence of outliers and decide on the correct normalization.
- Now, **focus on the test partition**. How do you preprocess the data? Is the preprocessing the same as for the training partition?

4.2 Task 2: Shallow Neural Network

Design your first Neural Network. Start by creating a single-layer neural network with the following characteristics:

# Hyperparameter	value
# Layers	1
# Neurons per Layer	{32, 64, 128}
Activation	Linear
Weight Initialization	Default
Batch Size	64
Loss Function	Cross-Entropy
Optimizer	AdamW
Learning Rate	0.0005
Epochs & Early Stopping	100 or shorter based on the designed criteria
Regularization	None

After you have trained the models, evaluate the performance of the model using the datasets to answer the following questions.

- Describe how you carried out the training process. Describe the code
- How does the loss curve evolve during training on the training and validation set? Image of the plot
- How do you select the best model across epochs? Which model do you use for validation and test?
- What is the overall classification performance in the validation and test datasets and considering the different classes? Indicate the results.
- Why is the performance of the model so poor?

Change the model with the best performance (optimal number of neurons) by changing the activation function in *ReLU* and evaluate the effects.

4.3 Task 3: The impact of specific features

As you learned in the lecture, biases in data collection can carry over to the model and become *inductive biases*. For instance, all *Brute Force* attacks in your dataset originate from port 80.

- Is this a reasonable assumption?
- Now replace port `80` with port `8080` for the **Brute Force** attacks in the **Testset**. Use the previously trained model for inference: Does the performance change? How does it change? Why?

Now remove the feature **port** from the original dataset (**uses this dataset from now on** for the entire lab) and repeat all preprocessing steps.

- How many *PortScan* do you now have after preprocessing? How many did you have before?
- Why do you think *PortScan* is the most affected class after dropping the duplicates?
- Are the classes now balanced?

Now repeat the training process with the best architecture found in the previous step

- How does the performance change? Can you still classify the rarest class?

To improve performance and take into account the imbalance between classes, you should use a *weighted loss*. Use the parameter `weight` of the class `CrossEntropyLoss`, which expects a tensor of the class weights. To estimate the weights, you can use the *sklearn* function `compute_class_weight` and set `class_weight` to "balanced". Repeat the training process with the new loss:

- How does the performance change per class and overall? In particular, how does the *accuracy* change? How does the *f1 score* change?

4.4 Task 4: Deep Neural Network

Now extend the architecture by adding multiple layers to design your first Deep Learning architecture.

4.4.1 Design the first Deep Learning

Design your first Deep Learning Feed Forward Neural Network with the following characteristics:

# Hyperparameter	**Value**
# Layers	2 to 5
# Neurons per Layer	{2, 4, 8, 16, 32} (layers can have different sizes)
Activation	ReLu
Weight Initialization	Default
Batch Size	64
Loss Function	Cross-Entropy
Optimizer	AdamW
Learning Rate	0.0005
Epochs & Early Stopping	50 or shorter based on the designed criteria
Regularization	None

Compare different architectures and:

- Identify the best-performing architecture.
- Plot and analyze the losses.
- Evaluate the performance of the validation and test set.

4.4.2 The impact of batch size

Use the best hyperparameter identified in the previous step and experiment with different batch sizes. In particular, use as batch size: {1, 32, 64, 128, 512}.

- Does the performance change? And why? Report both the validation and test results.
- How long does it take to train the models depending on the batch size? And why?

4.4.3 The impact of the Activation Function

Use the best hyperparameter identified in the previous steps and experiment with different activation functions.

In particular, use as activation functions: *Linear*, *Sigmoid*, *ReLU*.

- Does the performance change? Why does it change? Report both the validation and the test results.
- Explain why and how the different activation functions affect performance in this architecture

4.4.4 The impact of the Optimizer

Finally, evaluate here how the optimizers affect the classification performance, training time and loss trend.

The evaluated optimizers are: Stochastic Gradient Descent (SGD), SGD with Momentum (0.1, 0.5, 0.9) and AdamW.

- Is there a difference in the trend of the loss functions?
- How long does it take to train the models with the different optimizers? And why?

Evaluate the effects of the different learning rates and epochs for the best optimizer and architecture found above.

4.5 Task 3: Overfitting and Regularization

This last task is based on a Feed Forward Neural Network with 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

NOTICE: Each # Neurons per Layer describes ow many neurons must be present in each layer. For example: 1st layer 256, 2nd layer 128 etc.

- What do the losses look like? Is the model overfitting?

Now apply normalization techniques (dropout, batch normalization) and play with the regularization of the weights (AdamW's *weight decay*).

- What impact do the different normalization techniques have on validation and testing performance?