APAI2025 - LAB02  
**DNN Definition and Training**

short line

*Authors: Davide Nadalini, Ilenia Carboni, Alberto Dequino, Luca Bompani, Lorenzo Lamberti, Francesco Conti*

*Contacts: d.nadalini@unibo.it*

***Links:*** [***GitHub Link (code)***](https://github.com/EEESlab/APAI25-LAB02-DNN-definition-and-training)

# **Summary**

1. PyTorch definition of a NN model;
2. Count network’s parameters and MAC operations;
3. Data loader for Fashion-MNIST;
4. Code for testing a neural network on Fashion MNIST dataset;
5. Code for training a neural network on Fashion MNIST;
6. Save and load model’s trained weights;

**Submission deadline: Oct 17th 2025**

# **How to deliver the assignment**

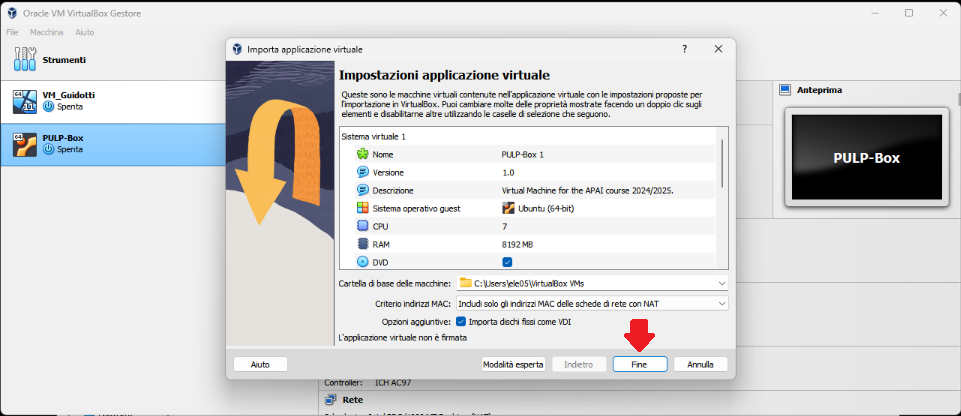
You will deliver ONLY THIS TEXT FILE, no code

* Download this file.
* Fill in the required results.
* Export to pdf format.
* Rename the file to: LAB<number\_of\_the\_lesson>\_APAI\_<your\_name>.pdf
* Use Virtuale platform to load ONLY your .pdf file

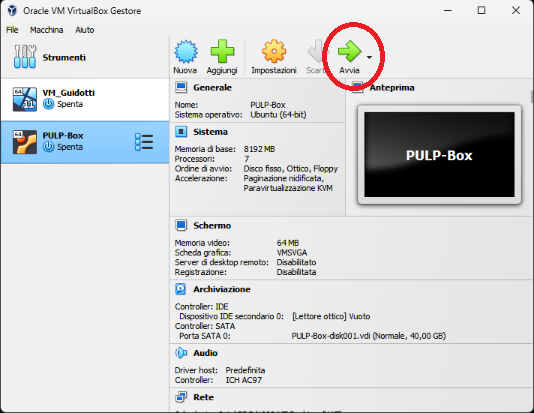
# **LAB STARTS HERE**

# **0. (ONLY LAB1) Access to the local VM**

* On the lab’s PCs, open the file explorer and go to This PC, C:/VM\_APAI
* Double click on PULP-box.ova
* VirtualBox opens, just click on “Fine”

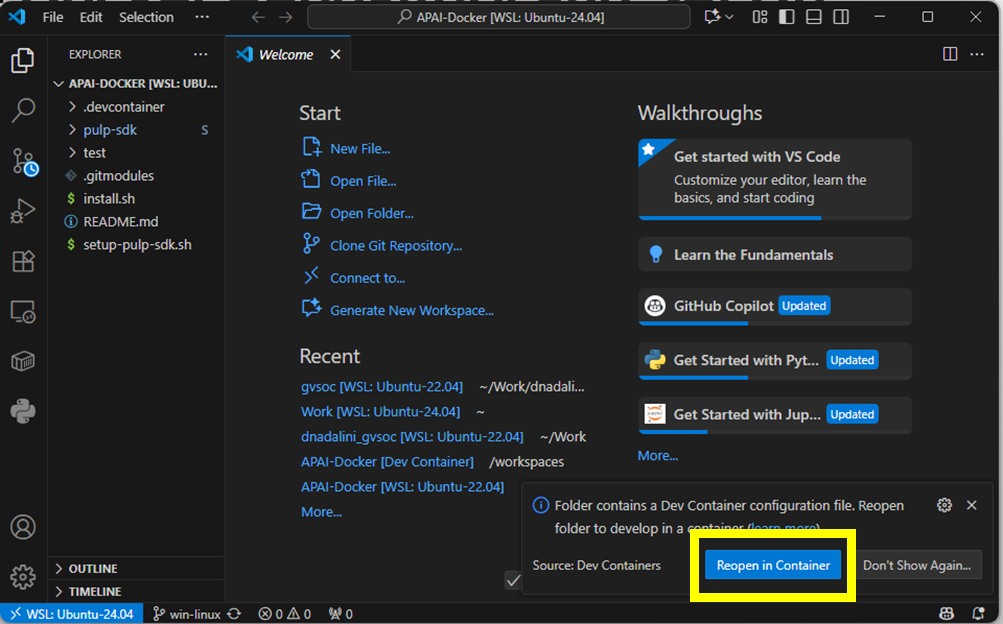


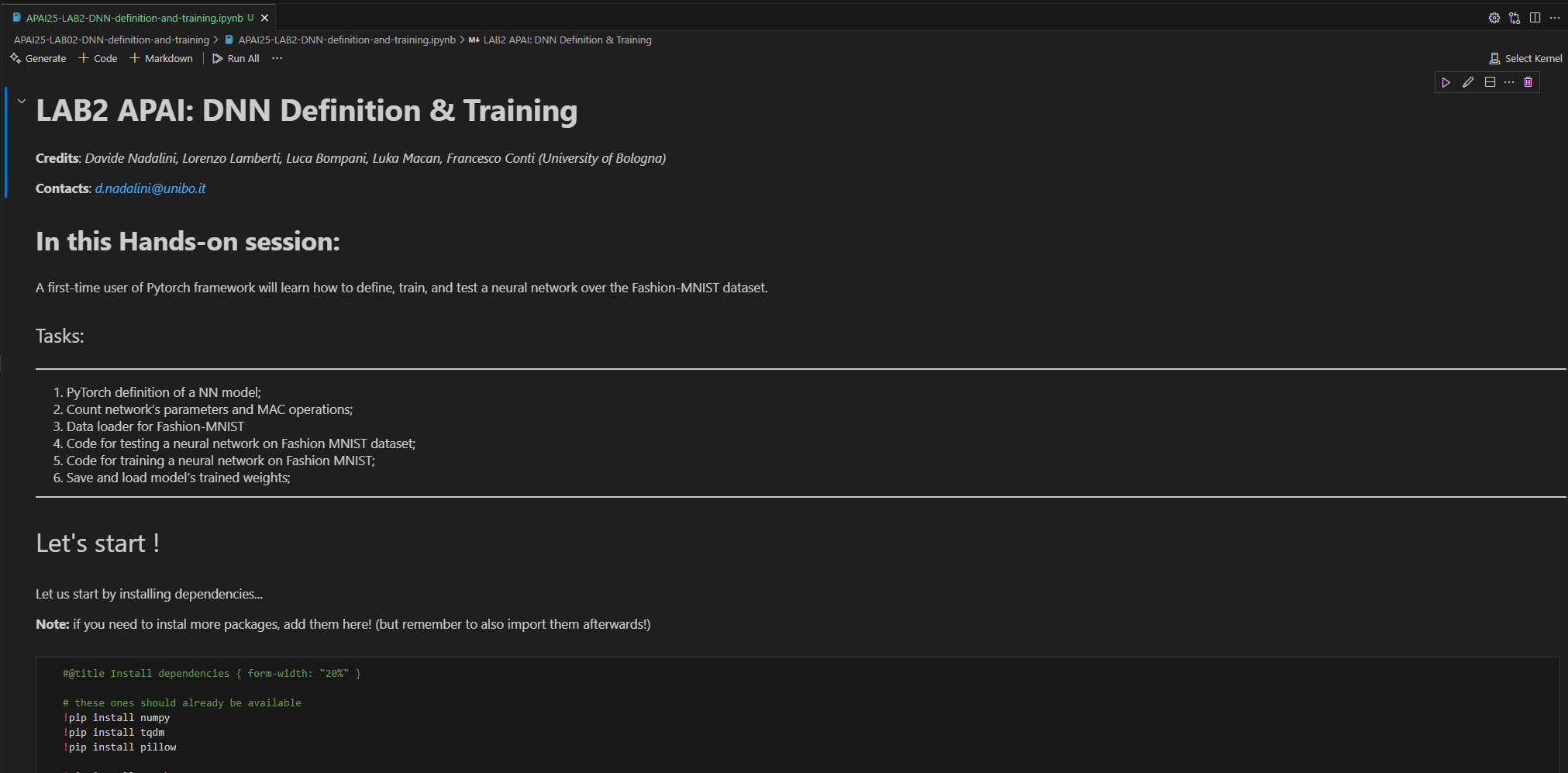
* Wait for the VM to be imported
* Open the VM with “Avvia”



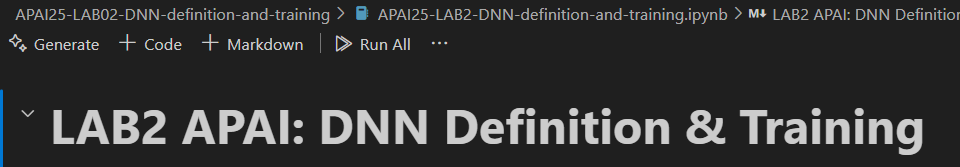
* Password is ‘pulp’

# **1. (ALL) Open Docker Container with VSCode**

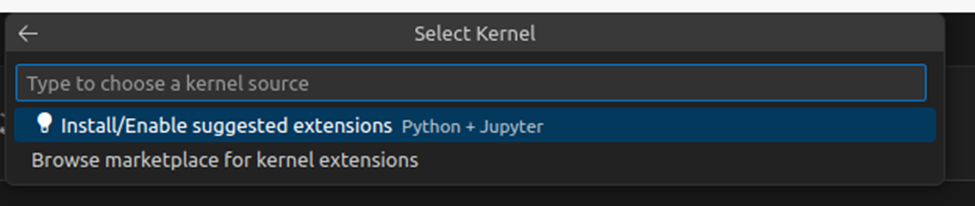
* Open a terminal (right click – open a new terminal)
* Go in the APAI-Docker folder: cd APAI-Docker
* Open Docker with VSCode: code .
* Select the pop-up option “Reopen in container”
* Clone GitHub repository of today’s lab: git clone https://github.com/EEESlab/APAI25-LAB02-DNN-definition-and-training
* cd APAI25-LAB02-DNN-definition-and-training
* Click on the file to open it in your VSCode



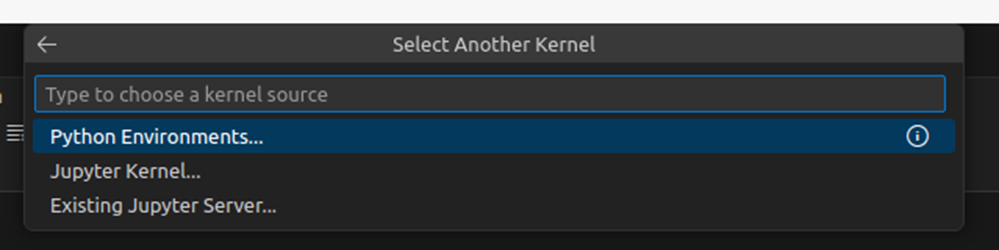
* Tu run every cell, do as in the test that you tried when you set up your Docker, that is: first, click on “Run all”



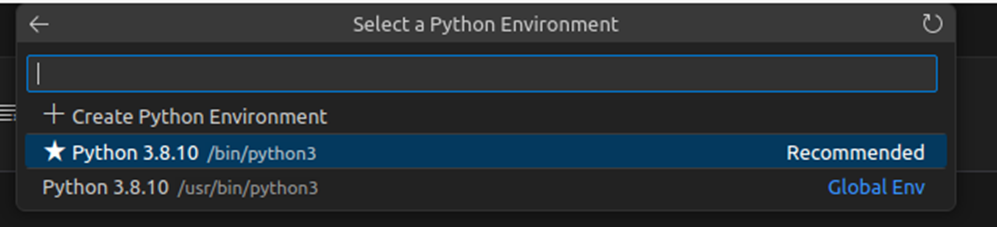
* In case you don’t have it, install the suggested extensions



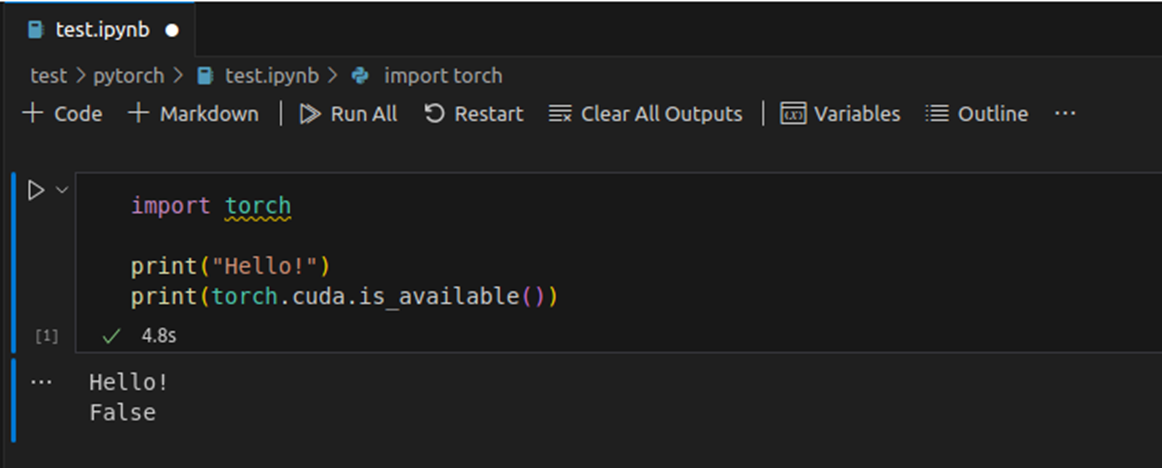
* Then, select the execution kernel



* That is your python



* Now, you should be able to execute even single blocks of code, as in the test (with the arrow on the upper-left corner of every block

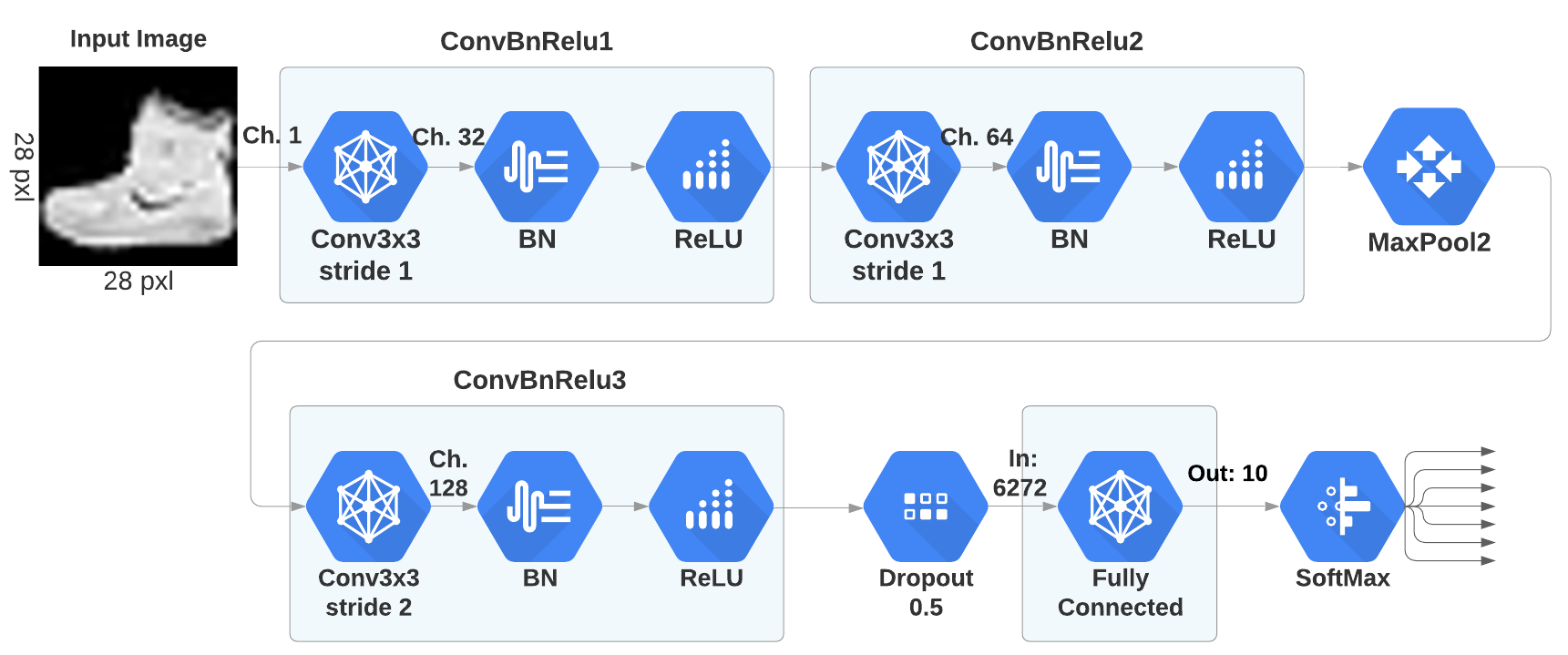


# **Task 1: CNN topology definition (PyTorch)**

*Resources:* [official PyTorch tutorials and documentation](https://pytorch.org/tutorials/beginner/basics/intro.html#how-to-use-this-guide), [Pytorch tutorial: how to define a NN](https://pytorch.org/tutorials/recipes/recipes/defining_a_neural_network.html)

*Description:* PyTorch definition of a CNN topology, following [Fig](#sieudgp015ti)[ure 1](#zigpi15szgpg).

*Output:* Code of a NN definition, following [Code 1](#22a5kywha16z) structure.



*Figure 1: The Custom CNN topology.*

|  |
| --- |
| **class CNN(nn.Module):  def \_\_init\_\_(self):  super(CNN, self).\_\_init\_\_()  conv1 = ...  conv2 = ...   def forward(self, x):**  **…**  **return output**  **net = CNN.to(device)** |

*Code 1.*

# **Task 2: Count network’s parameters and MAC operations**

*Resources:* [PyTorch Info](https://github.com/TylerYep/torchinfo), [PyTorch Operations counter](https://github.com/Lyken17/pytorch-OpCounter), [PyTroch FLOPs counter](https://github.com/sovrasov/flops-counter.pytorch)

*Description:* calculate the number of model’s parameters. Calculate the Multiply-Add-Operations (MACs). You can use some external pre-implemented functions from github (see refs).

*Output:* model size and MAC operations of the NN defined.

# **Task 3: Define dataset loaders**

*Resources:* [quickstart pytorch training](https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html), [data\_tutorial](https://pytorch.org/tutorials/beginner/basics/data_tutorial.html)

*Description:* create dataset and dataloader for Fashion-Mnist for both training and validation sets. We will use these in the training loop. Make use of torchvision.datasets and torch-utils.data.DataLoader

*Output:* a dataset loader for training and validation sets of Fashion-MNIST.

*Sub-Tasks:*

1. download fashion mnist with torchvision with torchvision.datasets
2. define 2 transformations that we will apply to the dataset: transform data to tensor, and normalization of input pixel data
3. define the dataset object
4. define the dataset loader with torch.utils.data.DataLoader. batch=128.
5. Sanity check: you must get 60 k training images, and 10 k validation images

# **Task 4: Testing a neural network in PyTorch**

*Resources:* [quickstart PyTorch](https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html),

*Description:*  build a testing loop in order to calculate accuracy of our NN

*Output:* Code of the testing loop (following Code 3 structure)

*Sub-tasks:*

* Define an accuracy metric as (num\_correct\_predictions/total\_n\_predictions)
* write testing loop, following Code 3 structure.
* Note: Remember to set network to evaluating with net.eval()

Example:

|  |
| --- |
| **def validate(net, dataloader, accuracy\_function, loss\_function):  n\_images = len(dataloader.dataset)  num\_batches = len(dataloader)  net.eval() *# set network to eval mode*  test\_loss, correct = 0, 0  with torch.no\_grad():  for batch\_idx, data in dataloader:  inputs, labels = data[0].to(device), data[1].to(device)**  ***# Compute prediction (forward input in the model)***  ***…***  ***# calculate accuracy***  **…**  ***# calculate testing loss***  **…    *# print accuracy and loss*   print(f"Test Error: \n Accuracy: {(100\*correct):>0.1f}%, Avg loss: {test\_loss:>8f} \n")** |

# **Task 5: Training a neural network in PyTorch**

*Resources:* [quickstart pytorch training](https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html), [Learning pytorch with examples](https://pytorch.org/tutorials/beginner/pytorch_with_examples.html)

*Description:* training a neural network on MNIST dataset

*Output:* Code of a training loop, following Code 2 structure

Sub-tasks:

* Define a Cross Entropy [Loss Function](https://pytorch.org/docs/stable/nn.html#loss-functions)
* define an SGD [optimizer](https://pytorch.org/docs/stable/optim.html)
* write the training loop function, which takes the arguments listed in Code 2
* plot training and validation loss over epochs. this is a good practice to track the training process.

*Tips:*

* use .to(device) on both model and inputs, in order to use the GPU at training time. This will be much faster than using CPU!
* Use net.train() to set network to training mode before starting the main loop

Example:

|  |
| --- |
| **def train(net, dataloader, loss\_fn, optimizer):  net.train()  for epoch in range(epochs):  for batch\_n, (input, label) in enumerate(dataloader):  size = len(dataloader.dataset)  X, y = X.to(device), y.to(device)   *# Compute prediction (forward input in the model)*  ...  *# Compute prediction error with the loss function*  ...   *# Backpropagation*  optimizer.zero\_grad()  ...   *# optimizer step*  ...   *# print: training loss and accuracy***  **...**  ***# Test network on validation set***  **...**  ***# print: validation accuracy, validation loss***  **...** |

# **Task 6: Save and load model’s trained weights**

*Resources:* [*save and load PyTorch model*](https://pytorch.org/tutorials/beginner/basics/saveloadrun_tutorial.html)

*Description:* You must be able to save and load correctly your model, as you will use it for future LAB sessions.

*Output:* model size and MAC operations of the NN defined

Sub-tasks:

* Save current model weights
* Load pre-trained weights to an non-initialized NN definition.
* Use the Testing function previously defined to test the new model with pre-trained weights, and get ~90% accuracy.