APAI Lab1

DNN Definition & Training

***Credits**: Davide Nadalini, Lorenzo Lamberti, Luca Bompani, Alberto Dequino, Francesco Conti. (University of Bologna)*

***Contacts**: d.nadalini@unibo.it, f.conti@unibo.it.*

How to deliver the assignment:

DEADLINE: 4/10/2024 (midnight)

Instructions:

- Use Virtuale platform to load your file: link
- update only the .ipynb file, named as follows: LAB1_APAI_yourname.ipynb

Important: the notebook must be pre-run by you. Outputs must be correct and visible when you download it

Links to COLAB exercise:

GitHub:
https://github.com/EEESlab/APAI24-LAB01-DNN-definition-and-training/blob/master/APAI24-LAB1-DNN-definition-and-training.ipynb
Solution is coming after the deadline

Lab session summary:

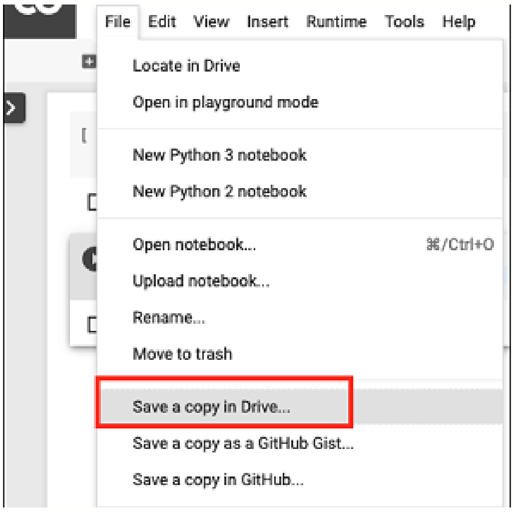
1. PyTorch definition of a NN model;
2. Count network's parameters and MAC operations;
3. Data loader for Fashion-MNIST;
4. Code for testing a neural network on Fashion MNIST dataset;
5. Code for training a neural network on Fashion MNIST;
6. Save and load model's trained weights;

Setup:

0. Open the link to launch [COLAB]() from the [GitHub repository]()

1. Create your own COLAB copy of the notebook!

- In Google Colab, use the menu: *File > Save a copy in Drive*

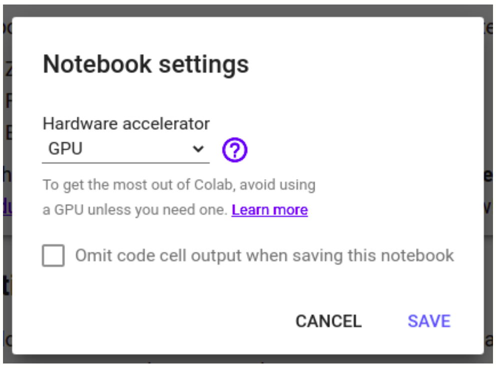


Then, log in with your own Google account.

Note: in case you don't complete the assignment, Colab keeps your file in your Google Drive!

2. Activate/deactivate GPU: *Runtime -> Change runtime type*

- ***Note**: If you use for too much time the GPU, your account will be limited to CPU for 24h.*



Task 1: CNN topology definition (PyTorch)

Resources: official PyTorch tutorials and documentation, Pytorch tutorial: how to define a NN
Description: PyTorch definition of a CNN topology, following Figure 1.
Output: Code of a NN definition, following Code 1 structure.

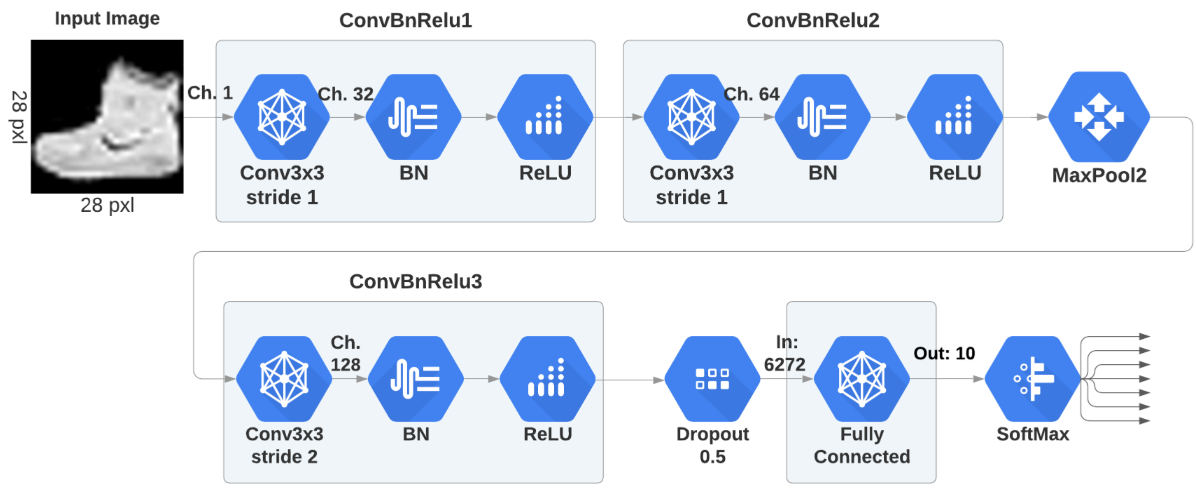


Figure 1: The Custom CNN topology.

```
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, PyTorch Operations counter, PyTroch FLOPs counter
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, data_tutorial

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](),
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, Learning pytorch with examples
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
- define an SGD optimizer
- 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]()
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.