In the attached notebook you will see a slightly improved version of the Fashion MNIST classification using MLP we studied in class.

Note: In order to make experimentation faster and more interesting, we have decreased the size of the MLP Neural Network in the example.
(Advanced question: Check how smaller the network is now, in terms of the number of learnable parameters).

Preamble:
Run the notebook.
Make yourself familiar with the various parts of the code using the examples - it will be useful later. Don't be afraid to experiment and change parameters.

Important notes:

1. The display utilities are improved. For example, **`show_in_plot`** can handle "source" of types **Dataset** or **DataLoader** or **list of (image_tensor, label) tuples** .
2. It also automatically displays the labels and can display the results of the classifier if passed correctly (there is also a new surprise to make analyzing results easier).
3. The code now tries to use only the DataLoader wrapper and not access the original Dataset class directly. For your convenience, the Dataset and DataLoader relationships are documented inside the notebook.

Question 1: Display a confusion matrix for the classification results

Question 2: We will experiment with transforming the images in the dataset.

Note that when reading the train or test dataset, there is a parameter called `transform` .
It specifies an image transform performed on each image in the dataset before it is served to the consumer. The input image is passed through the transform and only the output is served.
The default one does nothing except change the format to what we need for PyTorch before serving it:

```
transform_unchanged = transforms.Compose([transforms.ToTensor()])
```

The “Compose” method can be used to chain several transforms one after the other, similar to “Pipeline” which we used in scikit-learn.

An example of a two-stage pipeline is given in the notebook:

```
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor()
])
```

Question 2a: Read about `RandomHorizontalFlip` and other functions in the `transforms` library.
Replace the default transform on the train data with `train_transform` and observe the change in the data behaviour.
Continue with the train and evaluation process. What has changed in the results?

Question 2b: Change the parameter from **p=0.5** to **p=1** and observe the change in data and results.

Question 2c: Change back to **p=0.5** , but also apply `train_transform` to the *test* dataset.
Try to predict - what do you think will be the result of this change?
Check the actual results and compare to your prediction.

Question 3: Why is the `shuffle` parameter set to **True** for *train* but **False** for *test*?
What will happen if we switch these values, or if we set both to **True** ?

Question 4:
The current list of 10 categories is:

```
{'T-shirt/top': 0,
  'Trouser': 1,
  'Pullover': 2,
  'Dress': 3,
  'Coat': 4,
  'Sandal': 5,
  'Shirt': 6,
  'Sneaker': 7,
  'Bag': 8,
  'Ankle boot': 9}
```

We can try to solve a simpler problem, with fewer more general categories:

```
{'TopWear': 0,
  'Lower/FullWear': 1
  'ShoeWear': 2,
  'Accessories': 3,
}
```

Each of the 10 original categories can be mapped to one of the 4 new categories.

Question 4a: Update the dataset to map the previous target classes to the new ones.
Note: There is an elegant way to do that. Read the documentation.

Now run the network again and observe the results.

Question 4b: The original values of dataset.targets might not have changed. How can we check that? Can you explain why?
How can we actually make sure the new targets have the right values?

Question 5:
Going back to the original dataset, we want to try applying a different transform and observing its effect on the results.
But this will be more difficult.
The suggested transform is “*Move all the images of type "**Sneaker**" two rows (pixels) up, <u>but only in the training set</u>*”. Keep the image size the same, you can for example add two blank lines at the bottom.

Question 5a: Can this be done using the standard tools that we saw this far? If not - explain what the problem is.
Question 5b: Try to solve the problem. It is not easy, and can be done with the help of an LLM - but there are several traps on the way.
Make sure you fully understand the code and know how to test that it does what it’s supposed to do.

Question 6 (Optional): Dropout – A Simple Trick with a Big Effect

In this question, we introduc e a new neural-network layer: **Dropout**.

Dropout randomly “turns off” a fraction of the neurons during training only. This forces the network to rely less on specific neurons and more on distributed representations, which can reduce overfitting.

6a. Add a Dropout layer to the MLP

Modify the MLP model so that it includes a nn.Dropout(p=0.5) layer before the last linear layer.
(For example: Flatten → Linear → ReLU → Dropout → Linear.)

Train the model again and compare:

• Training loss
• Test loss
• Training accuracy
• Test accuracy
• Confusion matrix

to the results of the original network without dropout.
What changed?

6b. Experiment with different dropout probabilities

Repeat the training with:

• p = 0.2
• p = 0.8

Record and compare the training/test accuracies.
Which values of p seem too weak? Which seem too strong?

6c. Important: The train/eval mode question

Evaluate the model without calling model.eval() before testing.
Then evaluate again using model.eval().

Explain the difference in results.
Why is it necessary to switch the model to evaluation mode when computing test accuracy?