Introduction to deep learning, WS 2021/22

Problem 3

Description: Use the standard `Fashion-MNIST` dataset from the `torchvision` library to train models able to classify the images by their type.

Your task: Use the available train data to train A) multilayer perceptron and B) convolutional neural network described below to predict the classes. Test the models on the test part of the data and calculate the average accuracy over the test set for both models (accuracy = # correct / total instances).

Details of your task: You shall create a jupyter notebook in which you describe your steps (in plain English in markdown cells) and put your code (in the code cells) so that that your results can be reproduced when I (or anybody else) re-rerun the complete jupyter notebook.

Use of GPUs: For this task, you shall train the networks on GPUs. If you do not have any on your computer, use some of the free platforms providing the capability. I recommend Google Colab or Kaggle Notebooks. They are both for free, the environment looks very similarly to jupyter notebook (and you can upload/download), you can import pytorch and all other necessary libraries and use the GPU acceleration (Google Colab menu: Runtime -> Change runtime type -> Hardware accelerator GPU; Kaggle Notebooks menu in the right: Accelerator GPU). It is fairly intuitive to use and there are plenty of tutorials online explaining how to get setup and going.

Implementation rules: All work shall be done in Python using PyTorch and its API in particular the `torch.nn, torch.optim, torch.utils, torchvision modules.`

Teams: The same as for previous Problems.

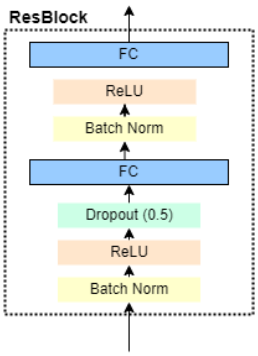
Submission: Submit the complete jupyter notebook via e-learning (only once for the team of two). ***The deadline is Sunday 2.1.2022 23:59***.

Questions: If you have any, please ask me right away (via mattermost).

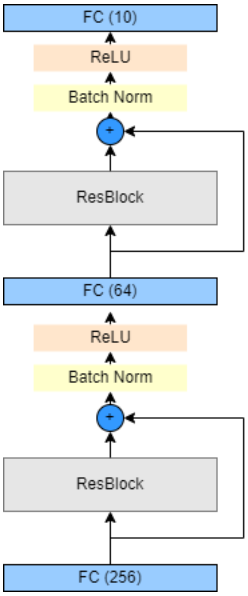
Disclaimer: The architectures suggested here below are not somehow cleverly tuned for the Fashion-MNIST task. They are here primarily for the exercise of translating the architecture description into a workable code. Nevertheless, I hope they will perform decently.

A) MLP architecture:

1. You shall implement a feed forward (MLP) with residual connections (ref. 1). Define the ResBlock in the image below as a custom layer (subclass of `nn.Module()`). The output dimensions of the fully connected layers shall be parameters that can be setup by the user.



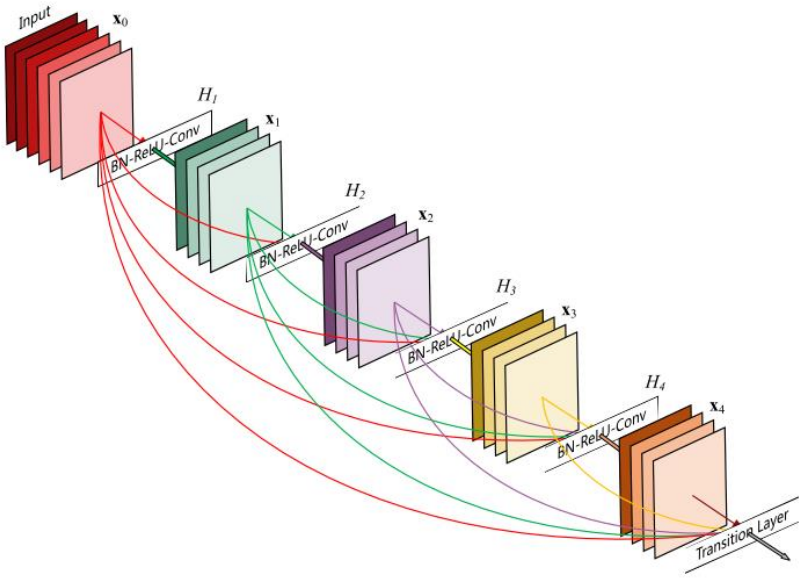
2. Use the ResBlock defined above to define an MLP network with the following architecture.



3. Use the training setup developed for Problem 1 – Part B (dataloader, cross-entropy loss, SGD optimizer, model training procedure) to train the MLP network.
4. Evaluate the accuracy of the network over the training and test data and comment the results also in comparisons to results obtained in Problem 1 – Part B.
5. Play with the values of the hyper-parameters, try different combinations and see how good/bad your model training is. Try to find a good combination of these to get good accuracy. What is the best you got?

B) CNN architecture:

1. You shall implement a DensNet (ref. 2). The idea develops further from ResNet but instead of passing the signal once around the ResBlock and adding it, in a DenseBlock we connect all layers together as illustrated at this schema from ref. 2:

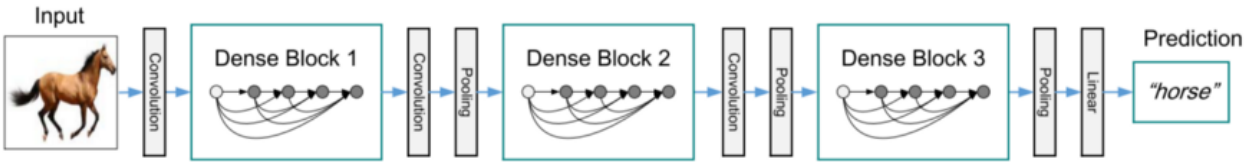


The features are not combined through a summation but by concatenating the channels (preserving the spatial size) so that the l[th] layer receives as inputs the concatenation of the outputs of all the preceding layers $[x_0, x_1, \ldots, x_{l-1}]$.

$$x_l = H_l([x_0, x_1, \ldots, x_{l-1}])$$

H_l is the composition of the following three consecutive operations: batch normalization, followed by a ReLU and 3x3 convolution. Define a DenseBlock as a subclass of `nn.Module()`. The number of layers within the DenseBlock should be a parameter that the user can set. Remember that the spatial dimensions shall be preserved throughout the DenseBlock!

2. The DenseNet connects multiple DenseBlocks putting *transition layers* between the blocks. Since the DenseBlock blows up the channel dimension and preserves the spatial dimension, it is the transition layers which perform downsampling and channel reduction. The transition layers consists of batch normalization, followed by 1x1 convolution to reduce the number of channels, and 2x2 average pooling with stride 2 to reduce spatial dimensions. Here is an example of the full architecture from ref. 2.



3. Using your DenseBlock defined above, create a DenseNet with the following architecture:
   The net shall contain 2 DenseBlocks each with 3 layers. Before the first DenseBlock the inputs are passed through 3x3 convolution (preserving spatial dimensions) with 4 output channels and average pooling with stride 2 (no padding) to reduce the size. The 1x1 convolution of the transition layer between the blocks reduces the number of channels back to 4. After the second DenseBlock use a global average pooling layer followed by a fully connected layer with 10-dimensional output that can be fed into the cross-entropy loss.
4. Answer the following questions:
   - What is the spatial dimension of the first DenseBlock and how many input channels and output channels it has?

- What is the spatial dimension of the second DenseBlock?
- What dimension is the input into the final FC layer (after the global average pooling)?

5. Train and evaluate the network on the train and test data as in part A), play with hyperparameters to get reasonable results and comment on what you achieved.

Hints:

- Remember that batch norm as well as dropout operate differently during the training of the model and a test time. Check the `train()` and `eval()` methods of `nn.Module`.
- When you want to train on GPU, both the model and data have to be transferred to the GPU memory. You can use the `to()` to transfer tensors and models back and forth between gpu and cpu. Here is an easy example to follow: https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html
- If anything is not clear (or not quite) or you simply want to discuss something, please write to mattermost! This is a learning exercise (though shall be evaluated). I want you to learn so if you are not sure, we can discuss. Much better than if you just get stuck and fail!

References:

1. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." *ArXiv:1512.03385 [Cs]*, December 10, 2015. http://arxiv.org/abs/1512.03385.
2. Huang, Gao, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks." *ArXiv:1608.06993 [Cs]*, January 28, 2018. http://arxiv.org/abs/1608.06993.