Homework 05

IANNWTF 21/22

Submission until 28th Nov 23:59 via https://forms.gle/n6ERdhYx3uBPzuGn9

Welcome back to the fifth homework for IANNWTF. By now, you should already be familiar with implementing MLPs in tensorflow and playing around with them. This week you will basically do the same thing, just that instead of a MLP we will ask you to implement a CNN. However, you should notice that with a great library like tensorflow this shouldn't proof to be a lot more difficult. In fact, it is super easy.

As always, there are hints (denoted by the superscript numbers) at the end of the document to help you get started. Do not read them immediately, but think about it first, and only go to them if you're stuck. Sometimes instructions might be a bit vague and that usually is intentional. We leave most things up to you specifically to enable the respective learning experiences which we deem necessary. But sometimes there are also mistakes from our side, so don't be shy to ask if something is unclear for you or doesn't make sense.
Also, feel free to copy code snippets from your last week's homework or the courseware!

1 Data set

As CNNs are very fashionable, in this week we would like you to work with the Fashion MNIST Dataset.

Note that this is very similar to the MNIST dataset that you have already seen in notebooks on courseware. The images have the same shape and there are the same number of classes.

1.1 Construct a Data Pipeline

Make yourself familiar with the dataset and construct a Data Pipeline. Perform all the preprocessing steps that you deem necessary. [1][2]

2 Model

Although the basic procedure of defining a CNN Model is the same, you will need to use entirely different layers this week. We want you to experiment a bit with how to structure your CNN but here are some general hints:

- You may want to alternate between convolutional layers and pooling layers. [3]
- Find the middle ground between the depth of the network and the number of filters used in each layer. Maybe consider the idea of the receptive field when deciding on the parameters. [4]
- Find the right parameters for the layers.[5]
- As this is an image classification task, make sure to have a classifier architecture following your convolutional and pooling operations. [6]

3 Training

Start by training your network for 10 epochs using a learning rate of 0.1. You can again copy most of the training procedure from previous scripts. [7] You may use all the optimization and regularization techniques that you wish and find applicable.

To pass this task, achieve at least an accuracy of 85% on the test dataset. [8]

4 Visualization

Visualize accuracy and loss for training and test data using matplotlib.

5 How-To Outstanding

The general rule again applies: To get an outstanding, your solution should be usable as a sample solution to other students. That means, the code must be instructive for other students in that it is clean, nicely structured, well commented, and efficiently using the methods introduced in Courseware. **Think of it like this: Your solution should be understandable by someone that may struggled with the homework. So always provide contex! That can be done as said by visualizing, structuring, commenting and also very important: If you have arguments in function / classes, provide explanation of them and information about their type e.g. "np.array, list, int" etc.!**

Additionally, we have the following requirements:

- Reach an accuracy of at least 90% on the test set.

- Compute the receptive field size of the final layer of your convolutional Neural Network. Provide step by step computations with explanations to follow what and why you are doing these computational steps. Does your receptive field size cover the whole input region / a sufficiently large enough region of the input to perform the task? In doubt, incorporate your findings to adjust your network architecture to boost up the accuracy.
- Optional: As later you will have to deal with more complex architectures to actually analytically compute the receptive field size, you may want to try and automatize the process. Feel free to come up with your own algorithm / way of doing so or for example make use of this library.

Happy coding!

Notes

[1]You definitely should onehotify the target and apply shuffling and batching. You may also think about normalizing the inputs

[2]You can pretty much recycle your old Data Pipeline or specifically the Data Pipeline from the MNIST notebook on Courseware. Just be aware that this time we want to work with the actual images and thus want to keep their structure. No reshaping into a one-dimensional tensor is necessary.

[3]For example, use one or two convolutional layers followed by a max pooling operation

[4]You should be able to solve the task with 5 layers only. Also, you should witness an increase in accuracy already after the first training epoch. On Colab with GPU support, an epoch doesn't need to take more than a minute.

[5]A kernel size of 3, no stride and 'same' padding usually is a good starting point. For filter sizes, we didn't use more than 64 filters per layer, but feel free to experiment (be aware of lengthy training times though)

[6]Apply Global Average Pooling followed by a Dense layer with the number of units depending on your number of classes. Use a softmax activation.

[7]For this task you again have to use tf.keras.losses.CategoricalCrossentropy()

[8]Again, with the right architecture this should be doable in 10 epochs and shouldn't take too long on Colab. If using Colab make sure to have your runtime settings set to GPU.