Tutorial 06: Introduction to Keras (20P + 3P)

Keras is a high-level neural network library that runs either **TensorFlow**, Theano or CNTK (low-level libraries) as its backend. The main purpose of Keras is fast and easy prototyping by being modular, minimalistic and extensible. In this tutorial you will be given an introduction to the usage of Keras which also will be used in future exercises.

For training the models of this exercise you should split your data into a **train**, **validation** and a **test** partition. The test set is usually given by loading the dataset using the Keras API. You should split the training set into a train and a validation partition either by hand or by using methods provided by Keras. You should adjust your network by using the train and validation set and do the final evaluation on the test set. For both datasets used in this tutorial the validation and the test set should contain about 10 000 samples.

Please submit your code/answers by 2nd Dec 23:59 to manuel.milling@informatik.uni-augsburg.de and thomas.wiest@informatik.uni-augsburg.de. You can submit your solutions alone or in teams of 2 (please indicate all names with the submission).

1 Keras Tutorial (0P)

Read through the tutorial at https://keras.io/ for installation instructions and to get a basic feeling of the API. If you use Google Colab (recommended) you should be able to use Keras straight away.

2 MNIST Classification using Dense Layers (5P)

In Exercise 3 and 4 you implemented a basic neural network to classify handwritten digits using fully connected layers only. Now we will do the same again, using Keras. Implement a *Sequential* that uses 3 *Dense* layers of which the first two use *ReLU's* as activation functions and *Softmax* for the last layer. Achieve a test accuracy of at least 95%. If you have trouble with overfitting try to reduce the number of parameters. Experiment with batch size, epochs, optimizer and learning rate to best fit the data.

Hints:

- Use `model.compile` to configure your model for training
- Use `model.fit` to train your model on the dataset

- Use `model.evaluate` to check its performance on the test set

See https://keras.io/models/model/ for more details.

3 MNIST Classification using Conv Layers (8P)

Create another MNIST classifier that corresponds to a "regular CNN architecture". Start out with (a) *2D Convolutional* layer(s) followed by down sampling and a maximum of two final *Dense* layers. Keep in mind that a *Convolutional* layer expects width, height and channel dimensions. See https://keras.io/layers/convolutional/ for more details. Achieve a test accuracy of at least 95%.

4 Parameter Count (2P)

Calculate the number of trainable parameters of your model in exercise 3. Show the calculations for each layer you used in your implementation. You can confirm your total result with `model.summary`.

5 Choice of Architecture (2P)

Which of both architectural approaches (Exercise 2 and 3) would be better suited for large images (e.g. 512×512 pixels) and why? *(1-3 sentences)*

6 Down Sampling Methods (3P)

How do pooling layers and strided convolutions reduce the spatial dimensions of an image? *(1-3 sentences for each)*

7 Bonus: CIFAR10 (3P)

Implement a neural network architecture for the CIFAR10 dataset. Achieve at least 75% accuracy on the test set. You might want to have a look at regularization techniques (e.g. DropOut, BatchNormalization, L2 regularization etc.). Why is it more difficult to achieve a high accuracy on CIFAR10 than on MNIST? *Note: This exercise is a bonus, i.e., it is not necessary to solve this exercise to obtain 100 % on the sheet.*