UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.
IWR Interdisciplinary Center for Scientific Computing

Deep Vision
Summer Semester 19
Prof. Björn Ommer
ommer@uni-heidelberg.de

Exercise sheet 8: Classification using convolutional neural networks

Due on *24.06.2019*, 11am.
Timo Milbich (timo.milbich@iwr.uni-heidelberg.de).

Task 1: Classification using CNNs (10P)

In previous exercises we already used the `pytorch` framework to train a neural network classifier. Now, we implement a convolutional neural network using the various layers introduced in the deep vision lecture. Further, we analyse the impact of normalization, augmentation and optimization on the training process and testing performance of our model.

1. *Defining a network.* Implement the following network as a `torch.nn.Module` module. In particular implement its `__init__` and `forward` function: (1P)

   (a) Convolutional layer: 6 filters of size 5×5 with padding 0 and stride 1.
   (b) ReLU layer.
   (c) Max-Pooling layer: size 2×2 and stride 2
   (d) Convolutional layer: 16 filters of size 5×5 with padding 0 and stride 1.
   (e) ReLU layer.
   (f) Max-Pooling layer: size 2×2 and stride 2
   (g) Fully connected layer: 120 neurons.
   (h) ReLU layer.
   (i) Fully connected layer: 84 neurons.
   (j) ReLU layer.
   (k) Fully connected layer: 10 neurons.

2. *Analysing network characteristics.* Determine the following two characteristics for each layer of the network you implemented in (1) (in particular for the 2 convolutional, 2 pooling and 3 fully-connected layers.):

   - Number of learnable parameters per layer and in total reported as scalar values. Please pay attention to the bias parameters which also contribute to the number of learnable parameters. (1P)
   - The feature map size of each layer reported as the shape of a three-dimensional tensor for the outputs of the convolutional layers and as the shape of a vector for the outputs of the fully-connected layers. (1P)

Hints:

- The feature map size is computed using a formula that recursively uses the result from the previous lower layer in order to compute the value for the consecutive higher layer. The formula is influenced by the input image resolution, the stride, the padding and the window size (i.e. kernel size) of the convolution.
- For this question you can assume the input image shape for the network is 64x64x1. The image has one color channel and the spatial resolution is square, thus the spatial resolution of every convolutional feature map is square as well.

3. *Training a CNN.* Train a CNN classifier based on the network (1) and using Cross-Entropy Loss on the training-set of the CIFAR10 dataset (image size $32 \times 32 \times 3$). Train your model using a batchsize of 16, SGD with learning rate 0.001 and momentum of 0.9 for 10 epochs. Report the accuracy (i.e. percentage of correctly classified images) of your model on the test-set of CIFAR10. (3P)

   Hints:

   - For data-loading you can use the `pytorch` built-in dataset library `torchvision.datasets.CIFAR10`. By setting the flag `download=True`, the dataset is automatically downloaded. Don't forget to activate data shuffling while training!
   - During training set your model to train mode (e.g. `model.train()`) and for evaluation set your model to eval model (e.g. `model.eval()`).

4. *Input normalization and augmentation.* Train a model using the same setup as in (3). However, this time normalize your input images channel-wise using $\mu = 0.5$ and $\sigma = 0.5$. Train a second model using the same input normalization and additional data augmentation. To augment the images use random horizontal flipping and random cropping with `padding=4`. Both input normalization and augmentation is easily realized with the `torchvision.transforms` library. Train each model for 10 epochs and compare their accuracy with the result of the baseline model (3). (1P)

5. *Setting the learning rate.* Consider the training setup of (4) (i.e. (3) using input normalization and training augmentation). Train different models using the learning rates $\{0.1, 0.01, 0.001, 0.0001\}$. Train each model for 10

epochs and report the testing accuracy after each epoch of training. Compare and discuss the differences of your models in both final performance and performance progress throughout training. (1P)

6. *Batch Normalization.* Extend the network architecture (1) with a batch normalization layer after each convolutional layer. Based on this network, train a model following the training setup of (5) with learning rate 0.001 for 10 epochs. Report and discuss your results compared to the baseline model (3). (1P)

7. *Prominent nets.* Train a model using the MobileNet architecture. You can use the the attached network definition `mobileNet_cifar10.py` which is adapted to the image size of CIFAR10. Given 5 epochs of training, try to achieve maximal performance by finding the best combination and configuration of the techniques used in (1)-(6). Report your best setup and result. (1P)

Note: Submit exactly one ZIP file and one PDF file via Moodle before the deadline. The ZIP file should contain your executable code. Make sure that it runs on different operating systems and use relative paths. Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. The PDF file should contain your written code, all figures, explanations and answers to questions. Make sure that plots have informative axis labels, legends and captions.