Exercise 10

Deadline: 07.07.2017, 2:15 pm
Regulations: You should hand in the exercises in groups of two or three persons. Please send a *compressed* (!) directory or file containing your solutions including all graphics, descriptions and source code to *thorsten.beier@iwr.uni-heidelberg.de*. The subject line of this email should start with [MLCV17][EX10] followed by the full names of all group members. Please cross-reference your code files in your writeup, such that it is clear which file has to be run for each exercise.

1 Convolutional Neural Fabrics

CNNs are very successful but selecting the optimal architecture for a given task is still an open problem. In [4] this problem has been addressed. Saxena and Verbeek introduce the concept of "Convolutional Neural Fabrics" which allow to optimize over an exponential set of architectures simultaneous. Furthermore existing CNN architectures can be visualized as a (maybe bifurcated) path in such Fabrics as depicted in figure 1

1.1 Visualize Famous CNN architectures as Fabrics (10P)

Visualize the following architectures as Fabrics as in the lecture (similar to figure 1).

- Holistically-Nested Edge Detection [6]
- Deep Residual Learning for Image Recognition (ResNet) [1]
- Densely Connected Convolutional Networks [1] (DenseNets) [2]

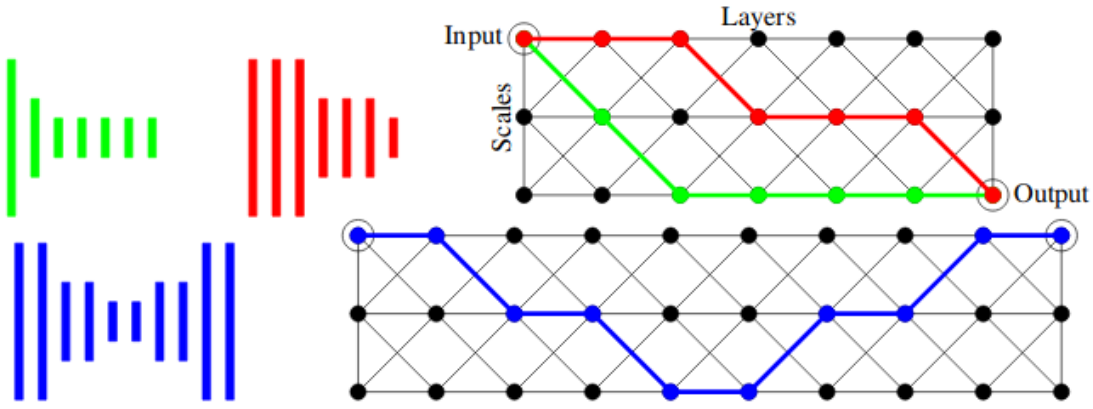

Figure 1: Fabrics embedding two seven-layer CNNs (red, green) and a ten-layer deconvolutional network (blue). Feature map size of the CNN layers are given by height. Fabric nodes receiving input and producing output are encircled. All edges are oriented to the right, down in the first layer, and towards the output in the last layer. The channel dimension of the 3D fabric is omitted for clarity.

Figure 1: Figure taken from [4]

[1] `https://github.com/liuzhuang13/DenseNet`

2 CNN Frameworks

In this exercise we will work with the CIFAR-10 dataset [2]. We will use big GPUs [3] to train a neural network.

The architecture we use is a VGG network [5] (VGG-16). You can choose the framework you want to use from the following:

- PyTorch
- Keras
- TF-Slim

The GPU Machines (Ubuntu) can be reached via ssh:

```
hgsgpu01.iwr.uni-heidelberg.de
```

To see the (memory) usage of the the 8 GPUs type

```
nvidia-smi
```

in a terminal. This should output something like:

```
student_1@hgsgpu01:~$ nvidia-smi
Thu Jun 29 10:55:51 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.51                 Driver Version: 375.51                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Quadro P5000        Off  | 0000:04:00.0     Off |                  Off |
| 26%   42C    P0    43W / 180W |      0MiB / 16273MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Quadro P5000        Off  | 0000:06:00.0     Off |                  Off |
| 30%   48C    P0    43W / 180W |      0MiB / 16273MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Quadro P5000        Off  | 0000:07:00.0     Off |                  Off |
| 27%   43C    P0    43W / 180W |      0MiB / 16273MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Quadro P5000        Off  | 0000:08:00.0     Off |                  Off |
| 29%   46C    P0    42W / 180W |      0MiB / 16273MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   4  Quadro P5000        Off  | 0000:0C:00.0     Off |                  Off |
| 28%   44C    P0    44W / 180W |      0MiB / 16273MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   5  Quadro P5000        Off  | 0000:0D:00.0     Off |                  Off |
| 31%   48C    P0    43W / 180W |      0MiB / 16273MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   6  Quadro P5000        Off  | 0000:0E:00.0     Off |                  Off |
| 27%   43C    P0    43W / 180W |      0MiB / 16273MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   7  Quadro P5000        Off  | 0000:0F:00.0     Off |                  Off |
| 27%   42C    P0    43W / 180W |      0MiB / 16273MiB |      1%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

To select a GPU with low memory / cpu usage type the following in a termial before you run your script:

```
export CUDA_VISIBLE_DEVICES="5"
```

This would select the GPU with index 5 (starting at zero).

[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

[3] You will have access to the best GPUs of the HCI. 8 x NVIDIA P5000

Thorsten Beier
thorsten.beier@iwr.uni-heidelberg.de

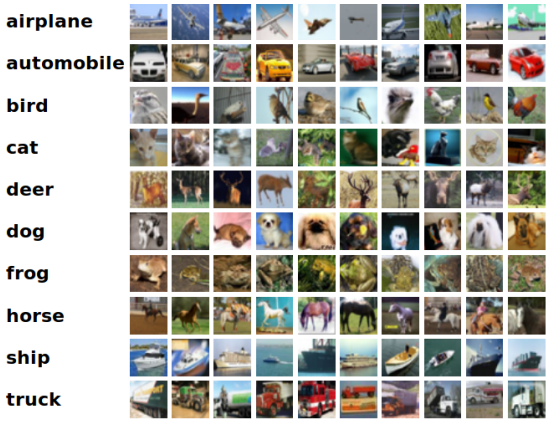

Figure 2: Classes in CIFAR-10, and 10 random images from each class

2.1 Implement the Architecture

Implement the VGG-16 network in PyTorch, Keras or TF-slim. Feel free to copy-paste existing code to set up the architecture. Use ADAM [3] as optimizer.

2.2 Train your network with CIFAR-10

Load the CIFAR dataset and train your network with it. Since the training is done with gradient descent one needs to specify a learning rate. Plot loss / training curves for different learning rates. How well does the network generalize on the test dataset?

2.3 Bonus

Visualize the weights of the convolutional kernels. Comment on the kernels.

2.4 Hint

- torchvision has easy access to CIFAR-10:
  `http://pytorch.org/docs/master/torchvision/datasets.html`
- torchvision has easy access to models (even pretrained models)
  `https://github.com/pytorch/vision#models`

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[4] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. *CoRR*, abs/1606.02492, 2016.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[6] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. *CoRR*, abs/1504.06375, 2015.