Week 6

Master Thesis 2020

Tobias Engelhardt Rasmussen (s153057)

DTU Compute

October 22, 2020

Outline

Using all digits of MNIST

Using full rank in the spatial dimensions and 20 hidden neurons in each

	Full data	Rank 15	Rank 100	Rank 200
# inputs	784	15	100	200
Total # parameters	3145	535	2,320	4,420
Testing acc. (%)	≈ 93	≈ 70	≈ 75	≈ 83

Even though we get more parameters than with the full input, the accuracy does not match the one of simply inserting the full data

Outline

Implementation

Implementation based on *Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications*:

- Tucker-2 of a convolutional layer (Input- and output)
- Tucker-1 of a convolutional layer (Output channel)
- Tucker-2 of a linear layer
- Tucker-1 of a linear layer

Tucker-2:

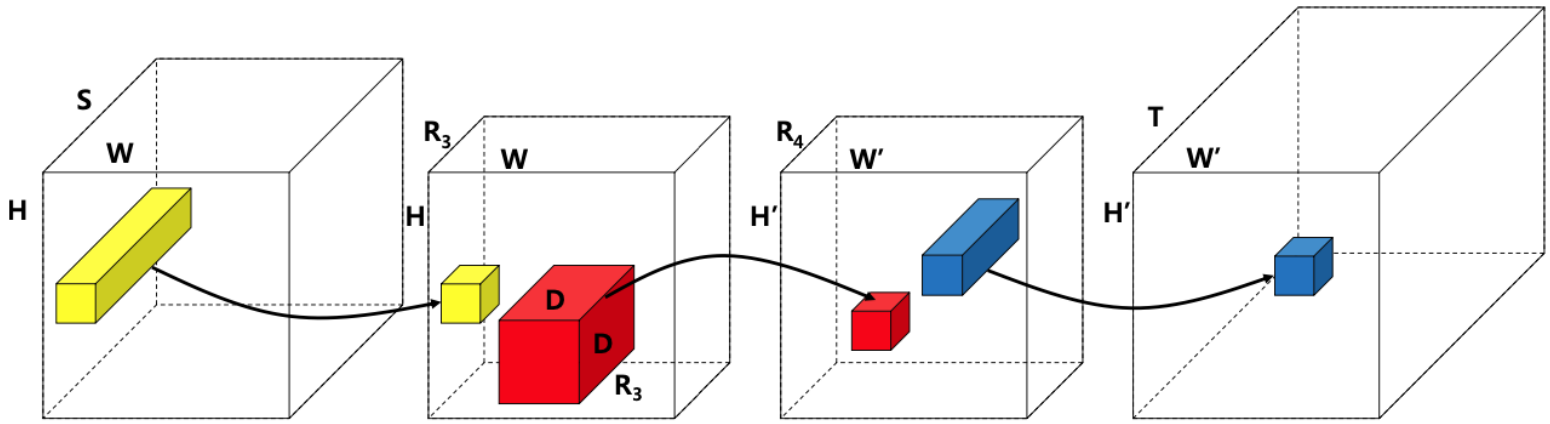
$$\mathcal{X} \approx \mathcal{G} \times_1 \boldsymbol{A} \times_2 \boldsymbol{B} \times_3 \boldsymbol{I}$$

Tucker-1:

$$\mathcal{X} \approx \mathcal{G} \times_1 \boldsymbol{A} \times_2 \boldsymbol{I} \times_3 \boldsymbol{I}$$

Tucker-2

Tucker-2 decomposition of a convolutional kernel:



To get the tucker-1 decomposition, only the output channels will be decomposed, hence only second half of the sequence will be used.

Linear layers

Linear layers are given as:

$$\boldsymbol{Y} = \boldsymbol{W}x + b$$

But with the decomposition $\boldsymbol{W} = \boldsymbol{AG}$ (Tucker-1):

$$\boldsymbol{Y} = \boldsymbol{AG}x + b$$

Is faster if done from the back because $\boldsymbol{G}x$ is only a vector length the rank.
With Tucker-2 the decomposition becomes $\boldsymbol{W} = \boldsymbol{AGB}^T$, hence:

$$\boldsymbol{Y} = \boldsymbol{AGB}^T x + b$$

Which is also faster, but where both input and output dimensions are considered.

Compression of the whole network

Net before:

```
Net(
  (conv1): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1))
  (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (dropout1): Dropout(p=0.2, inplace=False)
  (dropout2): Dropout2d(p=0.2, inplace=False)
  (l1): Linear(in_features=800, out_features=120, bias=True)
  (l2): Linear(in_features=120, out_features=84, bias=True)
  (l_out): Linear(in_features=84, out_features=10, bias=True)
)
```

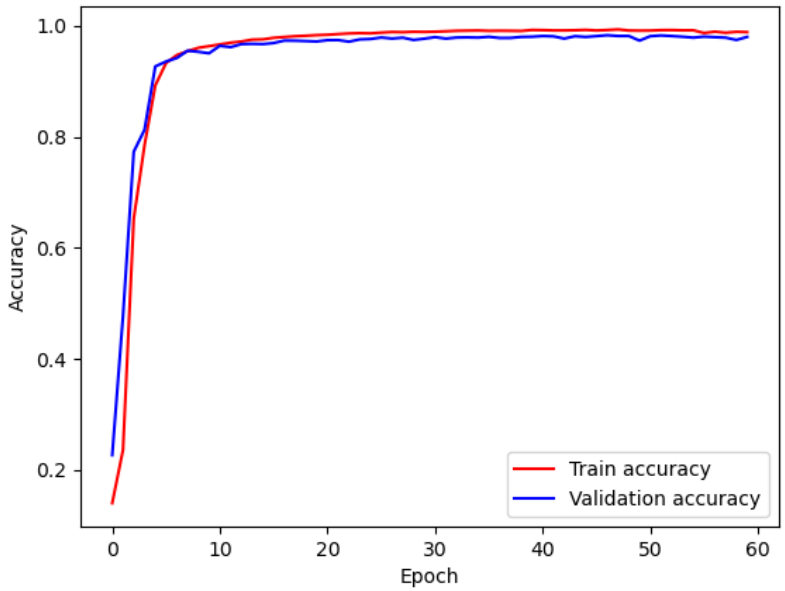
Net after:

```
Net(
  (conv1): Sequential(
    (0): Conv2d(1, 2, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
    (1): Conv2d(2, 16, kernel_size=(1, 1), stride=(1, 1))
  )
  (conv2): Sequential(
    (0): Conv2d(16, 8, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (1): Conv2d(8, 4, kernel_size=(5, 5), stride=(1, 1), bias=False)
    (2): Conv2d(4, 32, kernel_size=(1, 1), stride=(1, 1))
  )
  (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (dropout1): Dropout(p=0.2, inplace=False)
  (dropout2): Dropout2d(p=0.2, inplace=False)
  (l1): Sequential(
    (0): Linear(in_features=800, out_features=40, bias=False)
    (1): Linear(in_features=40, out_features=200, bias=False)
    (2): Linear(in_features=200, out_features=120, bias=True)
  )
  (l2): Sequential(
    (0): Linear(in_features=120, out_features=30, bias=False)
    (1): Linear(in_features=30, out_features=84, bias=True)
  )
  (l_out): Linear(in_features=84, out_features=10, bias=True)
)
```

Results

	Full network	Decomposed network after fine-tuning
# Parameters	120382	59560
Testing Acc. (%)	98.277	98.227
Mean Time (10,000)	1.807	1.945

So not neccearily faster for simple problems (lacks cache efficiency) but manages to get the same results with under half of the parameters!



(a) Traning of full network