Group 10, Assignment 1

Bulkens, Hecken, Kisin, Schmidle, Streitberger

May 16, 2023

1 Willingness to present:

Willing to present both the paper review (2.1) as well as the coding exercises (2.2)

2 Exercise 2.1

2.1 Review on the paper cuDNN: Efficient Primitives for Deep Learning (Chetlur et al. 2014)

The authors propose the library cuDNN, which provides a hardware-specific implementation of basic computational operations for parallel computing for deep learning, especially for CNNs. cuDNN is built to be integrated into deep learning frameworks such as Caffe, Torch etc., to enhance their performance by optimizing them for different hardware architectures. As cuDNN handles the optimization for the hardware, developers of frameworks can dedicate more time to high-level issues as they do not have to handle the time-consuming optimization when using cuDNN.

The authors give an overview of the implemented operations, especially their implementation of convolutions which lowers the convolution into a single matrix multiplication that is executed in multiple parts to decrease the memory latency. They show that cuDNN increases the performance of a DNN in comparison to plain Caffe, and they also show the hardware portability by comparing the performance on different GPU architectures. Furthermore, they give an insight into the integration of cuDNN into Baidu and Caffe.

As the paper states, no such libraries existed when the paper was published. Therefore cuDNN was a significant contribution to the deep learning community as their optimization allows for the development of bigger models. We strongly accept the paper. The last ten years have shown that cuDNN has become the industry standard and is well integrated into two of the most prominent ML frameworks (Tensorflow and PyTorch) and others.

3 Exercise 2.2

3.1 Discussion

We trained the implemented neural network with different learning rates. The following figures show the training and validation loss as well as the test accuracy for each used learning rate.

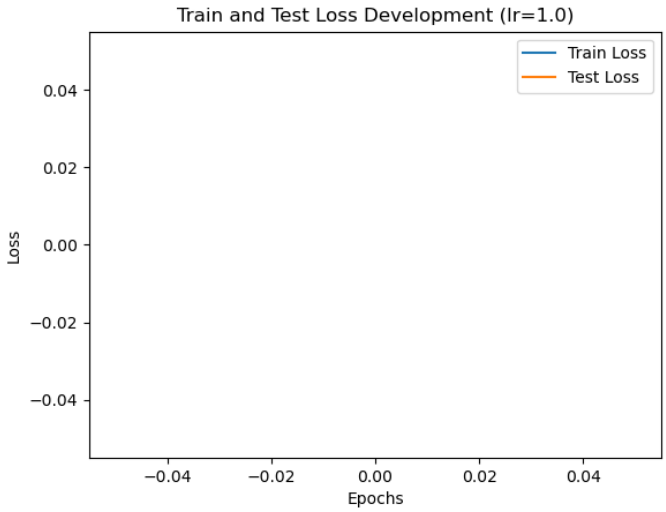

Figure 1: Loss (Learning Rate 1.0)

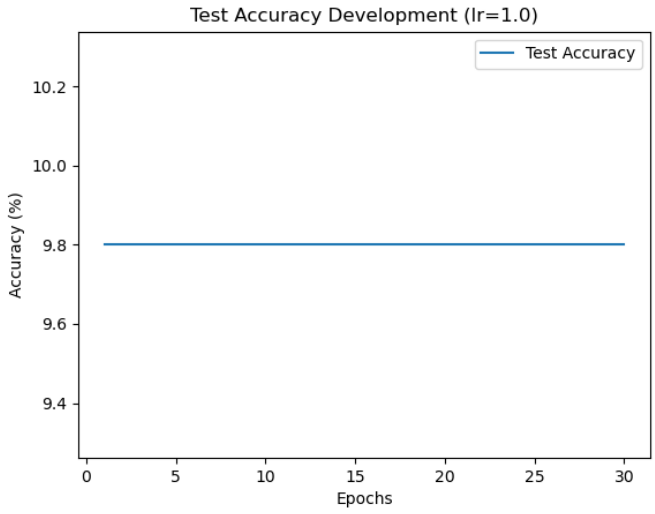

Figure 2: Accuracy (Learning Rate 1.0)

A learning rate of 0.1, as shown in image 3 and 4, has heavy fluctuation throughout the epochs. This is caused by a too high learning rate jumping over the loss landscape without settling in any minimum.

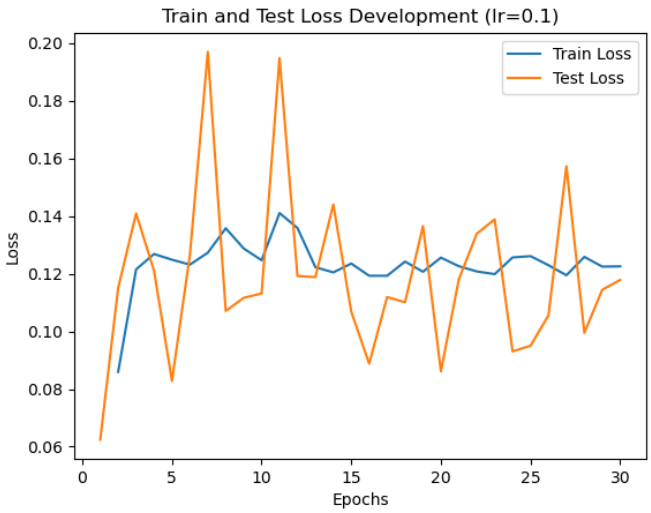

Figure 3: Loss (Learning Rate 0.1)

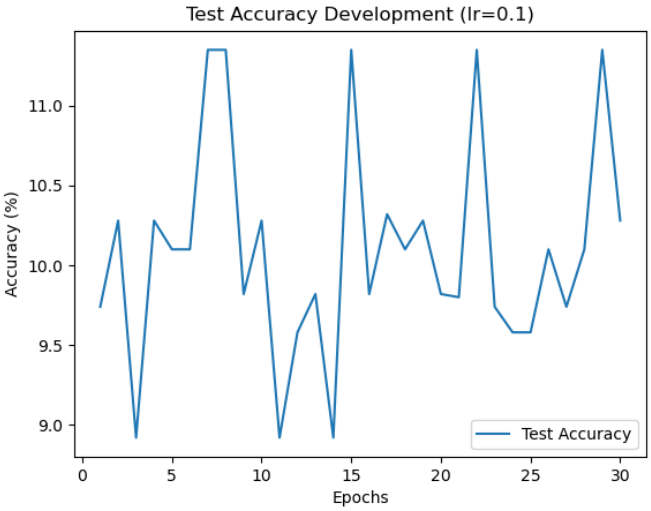

Figure 4: Accuracy (Learning Rate 0.1)

A learning rate of 0.01, as shown in image 5 and 6, convergences very quickly. It seems to reach the local minimum at about after 10-15 epochs, both considering the loss and the accuracy and stagnates afterwards. The initial drop in loss is very steep, but the step size seems reasonable since the loss convergences afterwards.

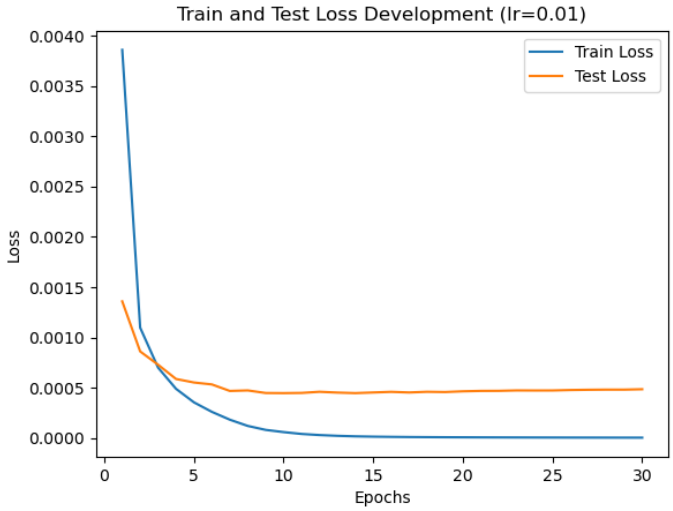

Figure 5: Loss (Learning Rate 0.01)

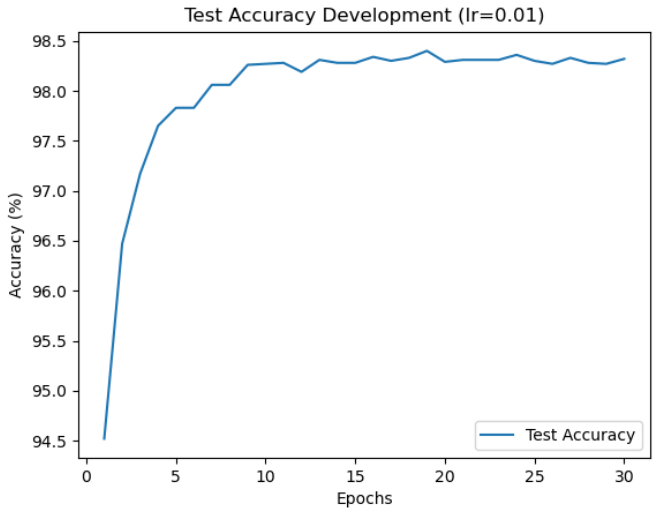

Figure 6: Accuracy (Learning Rate 0.01)

A learning rate of 0.005, as shown in image 7 and 8, convergences very slowly relative to the bigger learning rates. It seems to reach the local minimum right about after 20 epochs, both considering the loss and the accuracy and stagnates afterwards. Before convergence, the accuracy seems to go up and down a little bit but finally reaches the minimum.

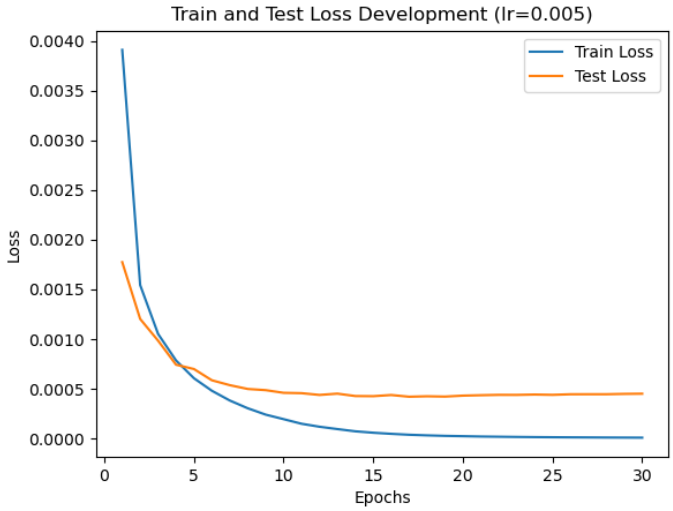

Figure 7: Loss (Learning Rate 0.005)

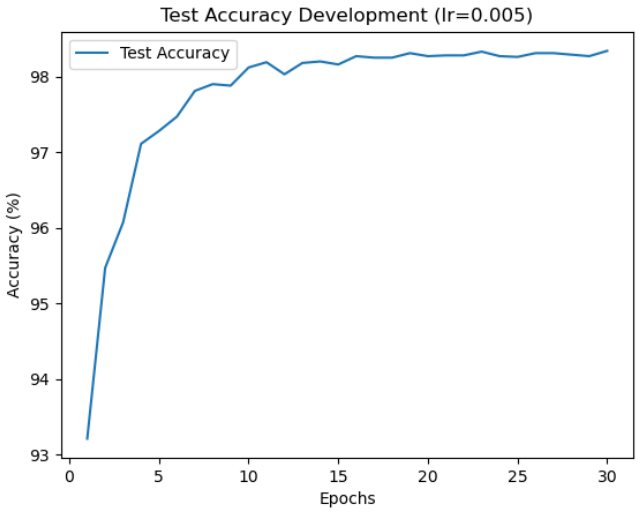

Figure 8: Accuracy (Learning Rate 0.005)

A learning rate of 0.001, as shown in image 9 and 10, convergences very slowly relative to the bigger learning rates. It seems to reach the local minimum right at the end of 30 epochs, and maybe even needs a little bit longer. Before convergence, the loss seems to go down very smoothly and shows no fluctuation at all due to the very small gradient steps.

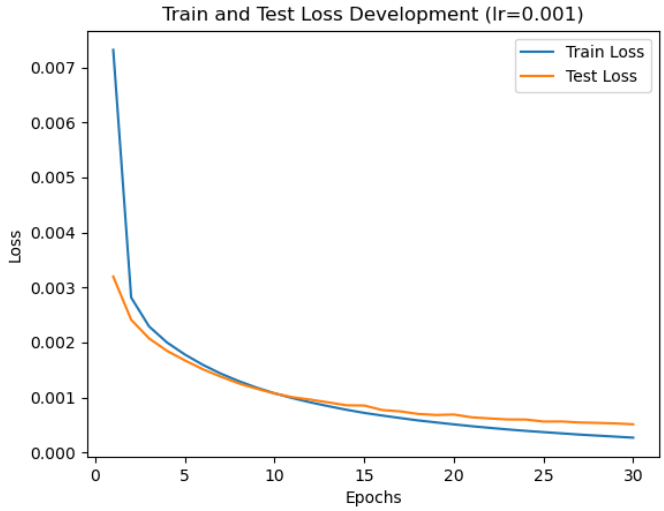

Figure 9: Loss (Learning Rate 0.001)

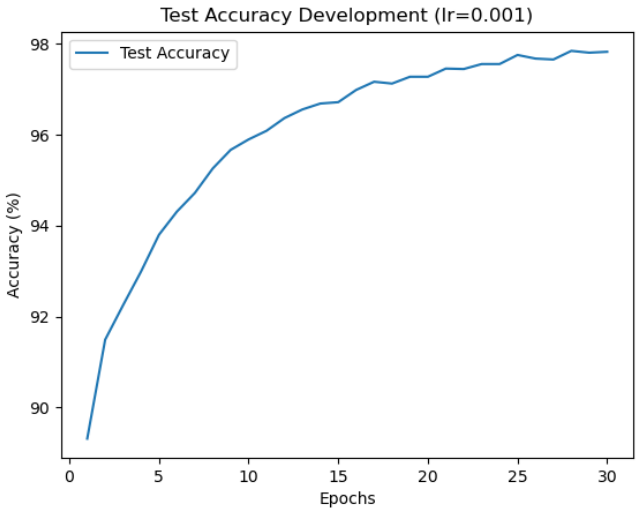

Figure 10: Accuracy (Learning Rate 0.001)

The results of the different learning rates illustrate the importance of the learning rate for the convergence of a model. If the learning rate is too high, the model can not converge as it overshoots the minimum. A learning rate that is too low leads to a slow training and the possibility for the model to get stuck in a bad local minimum.