Solution for Project #1
Deep Learning, HHU 2016

Thomas Germer **Michael Janschek** **Patrick Brzoska**

1 Code and Architecture

The code is provided as GitHub repository and is available under

`https://github.com/99991/DeepLearningProjects/tree/master/projects`

To test various neural network configurations, run

`https://github.com/99991/DeepLearningProjects/blob/master/projects/test/test_runner.py`

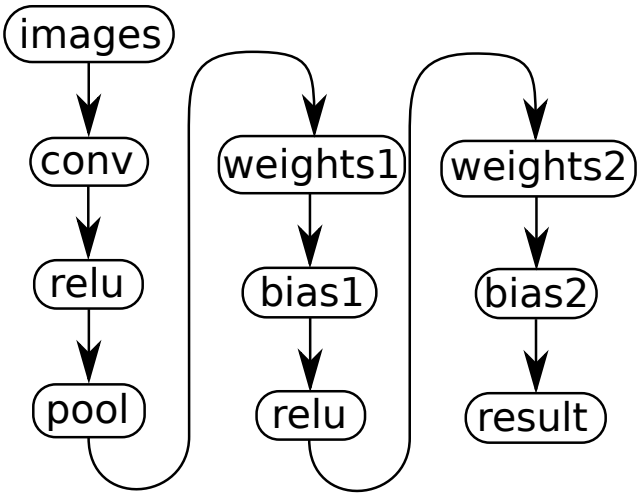
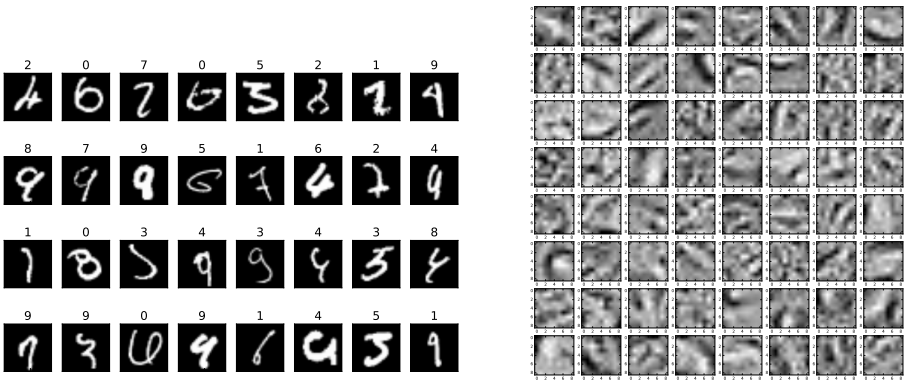


Figure 1: The network



(a) wrong predictions (b) visualization of CNN learning

Figure 2: visualizations for questions 3 and 7

2 Questions

1. Figure 1 describes the architecture of our CNN. We chose a small and therefore fast to train network, while still achieving high accuracy. We also tried larger networks through adding another conv-relu-pool layer and more hidden layers, with no improvement in accuracy but an increase in training time. The ReLu activation function was chosen because it performed better than the tanh activation function.

2. The CNN was trained with standard parameters using softmax regression (`tf.nn.softmax_cross_entropy_with_logits`). This method applies a softmax nonlinearity to the output of the network and calculates the cross-entropy between the normalized predictions and a 1-hot encoding of the label. Various Optimizers were tested, with the Adam-Optimizer converging faster and to a higher accuracy compared to Momentum-Optimizer and Gradient-Descent-Optimizer.

3. The CNN had an accuracy of up to 99.45%, depending on the parameters. Figure 2a shows a selection of misclassified images.

Figure 3: learning rate

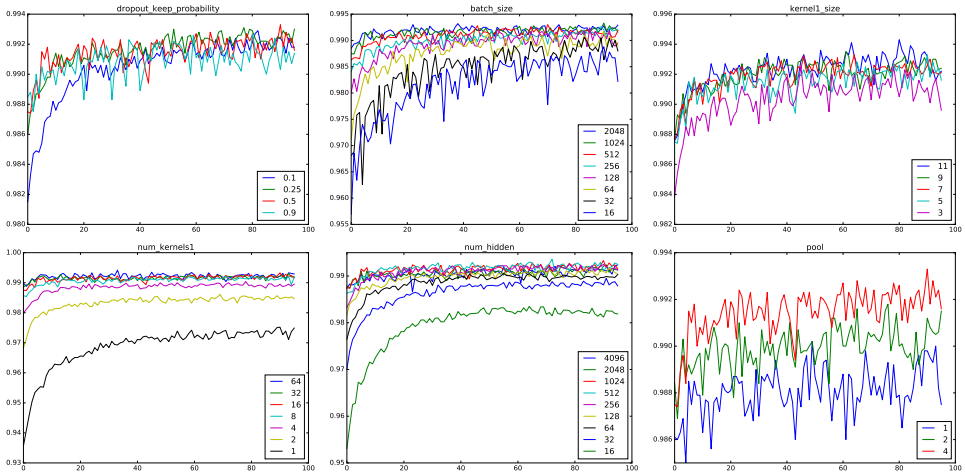


Figure 4: parameter comparison

4. While running solely on a CPU, setting the seed parameter for otherwise randomized initializations will give the same result for each run. The MNIST loader, which randomizes the batches per default, can be seeded via `np.random.seed`, while `tf.truncated_normal` and `tf.nn.dropout` offer seed parameters. However, the high degree of multithreaded calculations on the GPU results in non-deterministic behaviour.

5. For the evolution of accuracy with respect to the number of iterations, see 8. Training time on one million samples varied between 132 and 723 seconds on a GeForce GTX 1070, depending on the selection of the parameters.

6. Choosing a sensible learning rate is imperative. While a very small learning rate might take a long time to converge, a high learning rate might not reach the maximum accuracy or converge at all. As seen in figure 3, choosing a rate of 0.001 is preferable to both 0.01 and 0.0001.

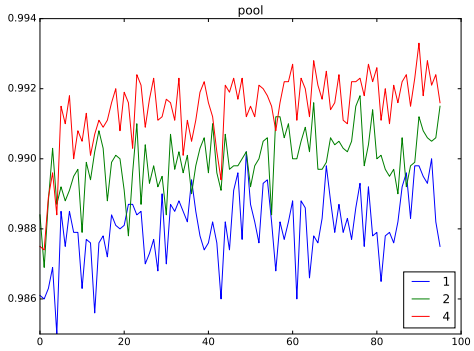
7. A CNN basically trains a feature extractor which is then used to classify any given data, preferably of the same type as the training data, in our case images. Figure 2b shows a visualization of these features.

8. Figure 4 details the accuracy in respect to various parameter changes. For the batch size and the number of kernels, a higher value results in better accuracy. While this is also the case for the kernel size, the calculation is more costly. The calculation time needed on a CPU for a kernel size of 5 doubles compared to the kernel size of 3. For the max pooling, the best results are achieved with a value of 4. This seems to be the maximum, though, since additional tests have shown that an even higher value of 7 or even 14 have a much lower rated result. The dropout keep probability shows the best results for a value of 0.25.
9. Table 1 shows two small configurations with still "good" accuracy.

Table 1: Minimal configurations

Parameter	Accuracy 97.4%	Accuracy 99%
num_hidden	64	512
num_kernels1	4	16
learning_rate	0.001	0.001
regularization_factor	0.0001	0.0001
batch_size	2048	512
dropout_keep_probability	0.25	0.25
seed	666	666
kernel1_size	3	5
test_interval	100	100
num_batches	20001	2001
pool	4	4

10. Table 5b depicts the different pooling kernel sizes and its consequences. A kernel size of 1 is the equivalent to no pooling.



(a) the plot

Pooling Kernel Size	Seconds	Accuracy
1	261.9	0.8438
2	255.7	0.8668
4	214.0	0.7905
7	211.8	0.7401
14	209.5	0.454

(b) the table

Figure 5: pooling comparisons