

# Deep Learning for Image Analysis

## DL4IA – Report for Assignment 3

Student Linus Falk

April 25, 2023

### Introduction

Third assignment in the course Deep learning for image analysis

## 1 Classification of hand-written digits using a Convolutional Neural Network

**Exercise 1.1** In this exercise we implemented the same neural network as in assignment 2. But this time we used the PyTorch library and GPU support for training. The task was to compare the performance in accuracy and training time. The weight were initialized in the same way as in assignment 2 and training was done with the same hyperparameters, see table:1. As we can see in table 2 the accuracy performance is very similar since the architecture and the training methods is the same. The difference between the training times are on the other hand noticeably shorter for the PyTorch version. This is much thanks to the GPU support, but also more effective data handling then the "homeCooked" version has. In 1 we can see signs that the model is starting to overfit. The test loss has stopped to improve while the training loss on the subset (6000 samples) of the training set is still improving. The model is overfitting to the training data and if we would continue to train we would we decrease in performance on the test set.

Hyperparameters	
BatchSize	30
Epochs	60
lr	0.01
Optimizer	SGD

Table 1: example

	"HomeCooked NN"	PyTorch
Accuracy (%)	97.26	97.22
Time (sec)	289	22.28

Table 2: example

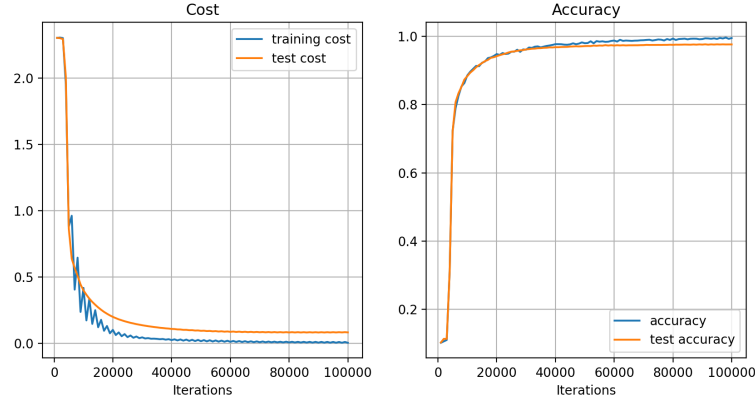


Figure 1: Training history: Assignment 2 Torch version, Accuracy: 97.22%

**Exercise 1.2** In this exercise we construct a convolutional neural network with the PyTorch library according to the instructions. We compare the accuracy and number of weights between the Fully connected and the convolutional NN in table: 4. Looking at the training history we can conclude that this network doesn't show the same signs of overfitting. The convolutional neural network has improved accuracy with fewer trainable weights. By using the CNN we improve the accuracy by preserving the spatial information and reduce the risk of overfitting by having fewer trainable parameters.

Hyperparameters	
BatchSize	100
Epochs	60
lr	0.005
Optimizer	SGD

Table 3: example

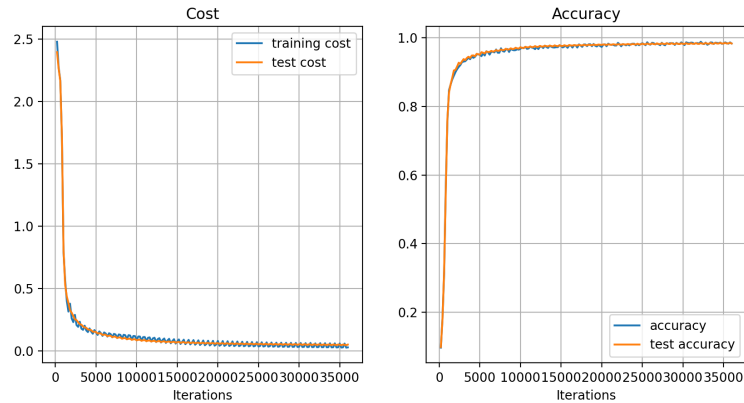


Figure 2: Training history Exercise 1.2, Accuracy: 98.35%

Architecture	Accuracy (%)	Number of weights
FF	97.22	109386
CNN	98.55	21688

Table 4: example

**Exercise 1.3** Here we are asked to swap place of the maxpooling layer and take the maxpooling before the activation function instead. This will result in fewer connections to the activation function making the time for calculating the activations fewer and therefore reducing the training time which we can see in: Table 5. If we would use a more complicated activation function like the *hyperbolic tangent* the time difference would increase. The worse accuracy in the swapped layer case can be caused by the fact that we are passing on less information to the activation function.

	CNN without swapped layers	CNN with swapped layers
Accuracy (%)	98.55	96.87
Time (sec)	202	135

Table 5: Result Exercise 1.3

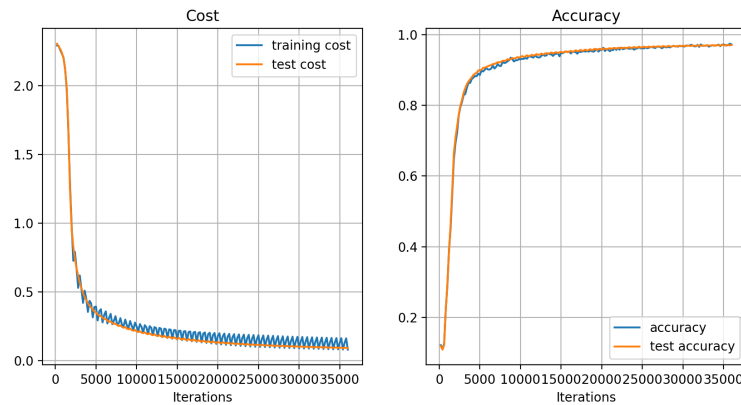


Figure 3: Training history Exercise 1.3

**Exercise 1.4** To investigate how the choice of optimizer effects the training time, we are here asked to use the optimizer: *ADAM* (with default parameters) instead of *SGD* and compare how much faster the training becomes. *ADAM* is in many cases the best choice of optimizer and we can conclude from the result of our test in Table: 7 that we need fewer epochs to achieve the same (or better) accuracy as *SGD* when using *ADAM*. We can see that the training accuracy, in figure: 4 improving slightly more in the last iterations compared to the test set, indicating that were starting to get some overfitting.

Hyperparameters	
BatchSize	100
Epochs	15
Optimizer	Adam
lr	0.001
betas	(0.9, 0.999)
eps	1e-8
weight_decay	0

Table 6: Hyperparameters Exercise 1.4

	CNN trained with SGD	CNN trained with ADAM
Accuracy (%)	98.55	98.69
Time (sec)	202	73

Table 7: Result Exercise 1.4

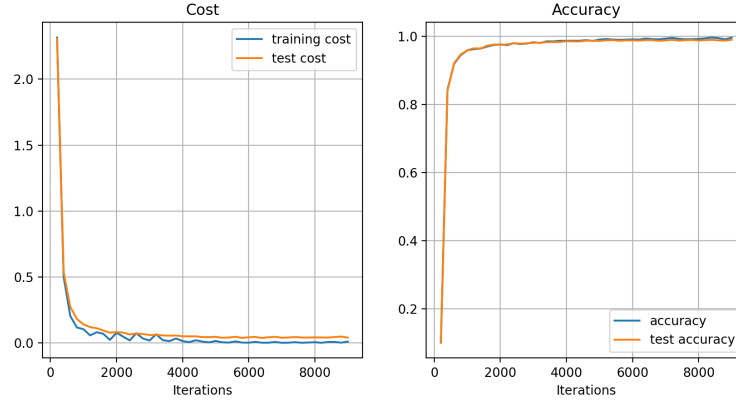


Figure 4: Training history Exercise 1.3

**Exercise 1.5** In this exercise we try out at least 3 different ways to improve our model by changing for example architecture, learning rate or using different types of regularization methods. The result is presented below with tables of changes for each model and performance measure. The result from the best model is also presented with a confusion matrix of the test set.

**Model 1:** We increase the last, fully connected layers to 50 nodes to improve see if we can improve the model with more learnable parameters. To reduce the risk of overfitting we also include a dropout layer after this layer with a probability,  $p=0,25$ .

Hyperparameters	
BatchSize	100
Epochs	10
Optimizer	Adam
lr	0.003
<b>Result:</b>	98.8%

Table 8: Hyperparameters Exercise 1.5, Model 1

**Model 2:** Included batch normalization layers between all layers which can improve the training by normalizing all activation. In this case it didn't give major improvement since we didn't have a problem with vanishing or exploding gradients which it is most commonly used to improve.

Hyperparameters	
BatchSize	100
Epochs	10
Optimizer	Adam
lr	0.003
<b>Result:</b>	98.89%

Table 9: Hyperparameters Exercise 1.5, Model 2

**Model 3:** Here we change the learning rate strategy by changing the learning rate such that  $i$  depends on the iteration number ( $i$ ) with the function:

$$\gamma^{(i)} = \gamma_{\min} + (\gamma_{\max} - \gamma_{\min})e^{\frac{i}{2000}} \quad (1)$$

We set  $\gamma_{\max} = 0.003$  and  $\gamma_{\min} = 0.0001$  and update it throughout the training.

Hyperparameters	
BatchSize	100
Epochs	10
Optimizer	Adam
lr	0.003 - 0.0001
<b>Result:</b>	99.04%

Table 10: Hyperparameters Exercise 1.5, Model 3

**Model 4:** Here we simply combine model 1 and model 3. Increasing the number of nodes to 200 in the final fully connected layer, add a dropout layer after that and then use the changing training rate through the training. Even though we use the drop out layer we can see some indication of overfitting in the training history, looking at the accuracy. The training accuracy continue to improve but the test accuracy stays pretty much the same. Since this was the best performing model we also take a look at the confusion matrix. Here we can see that the most miss-classification is done on the actual digit 9, often being miss-classified as a 4, which is understandable considering the shape of the two digits.

Hyperparameters	
BatchSize	100
Epochs	10
Optimizer	Adam
lr	0.003 - 0.0001
<b>Result:</b>	99.11%

Table 11: Hyperparameters Exercise 1.5, Model 4

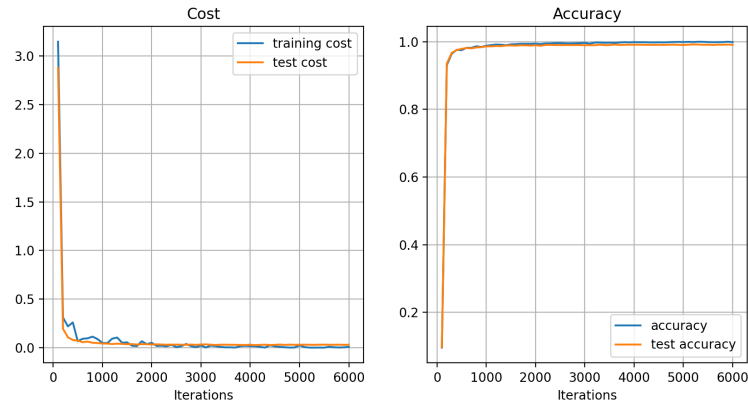


Figure 5: Example of caption

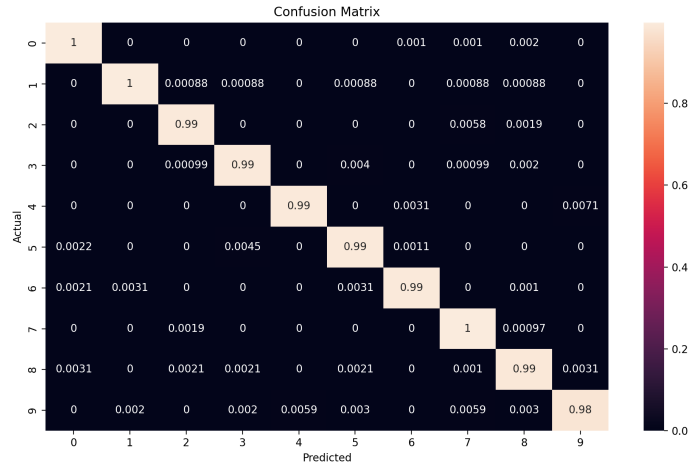


Figure 6: Example of caption

## 2 Semantic segmentation of Biomedical images

**Exercise 2.1** Modifying our previous neural network by removing the fully connected layers replacing it with a  $1 \times 1$  convolutional layer, we will now tackle a segmentation problem and evaluate it with Sørensen–Dice coefficient as performance measurement. In Figure: 7 we can see the result from the segmentation of a randomly picked test image. The segmentation models worst performance on the test set is presented in Figure: 8. Here we can see that it seems like the specimen is ending abrupt in the image, showing a lot of dark areas, presumably the background. Since this case is not well represented in the training set it is not so surprising that it performed badly on this image.

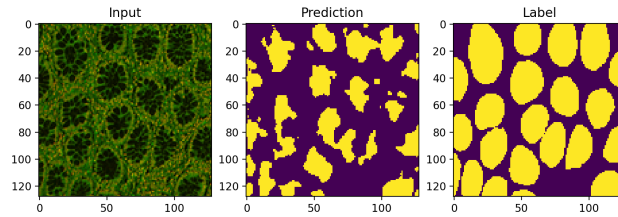


Figure 7: Segmentation of test-image: image\_05.png

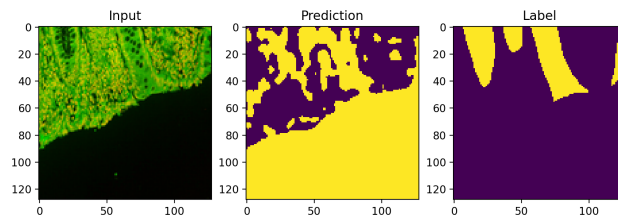


Figure 8: Segmentation of test-image: image\_11.png

<b>Hyperparameters</b>	
Epochs	150
Optimizer	Adam
learning	0.003
<b>Result:</b>	0.773%

Table 12: Hyperparameters Exercise 2.1

**Exercise 2.2 Model 1:** Using batchnormalization

<b>Hyperparameters</b>	
Epochs	150
Optimizer	Adam
learning	0.003
<b>Result:</b>	0.858%

Table 13: Hyperparameters Exercise 2.1, Model 1

## References

- [1] Asimov, Issac (1942). Runaround