Albert-Ludwigs Universität Freiburg
Institut für Informatik
Deep Learning Lab **Excercise Sheet 2**
Arce y de la Borbolla, José 13.11.2018

Deep Learning Lab
Excercise Sheet 2

13.11.2018

Abstract

This report shows the results of training a convolutional neural network with different hyperparameters, and trying to find the best set of hyperparameters, with the purpose of classifying the handwritten numerical digits of the MNIST data set.

1 Results

For this exercise, we used a Convolutional Neural Network with an architecture like the one shown in figure 1.

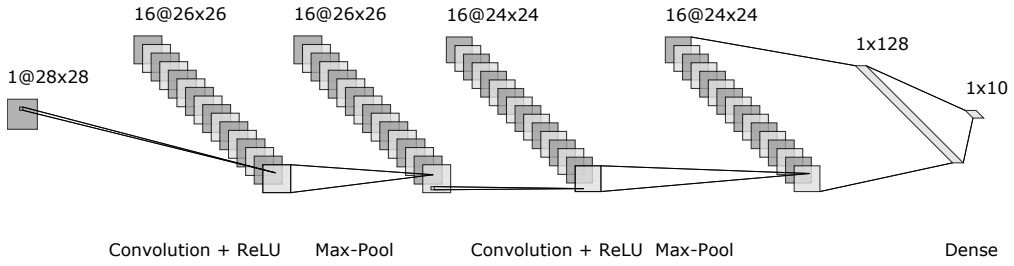

Figure 1: Convolutional Neural Network Architecture.

The filter size and number of filters per layer were the only design choices that changed between experiments with respect of the original architecture.

1.1 Implementing a CNN in Tensorflow

After modeling a Convolutional Neural Network using Tensorflow, and training it with the MNIST Dataset, we get the following performance data as a benchmark for subsequent tests.

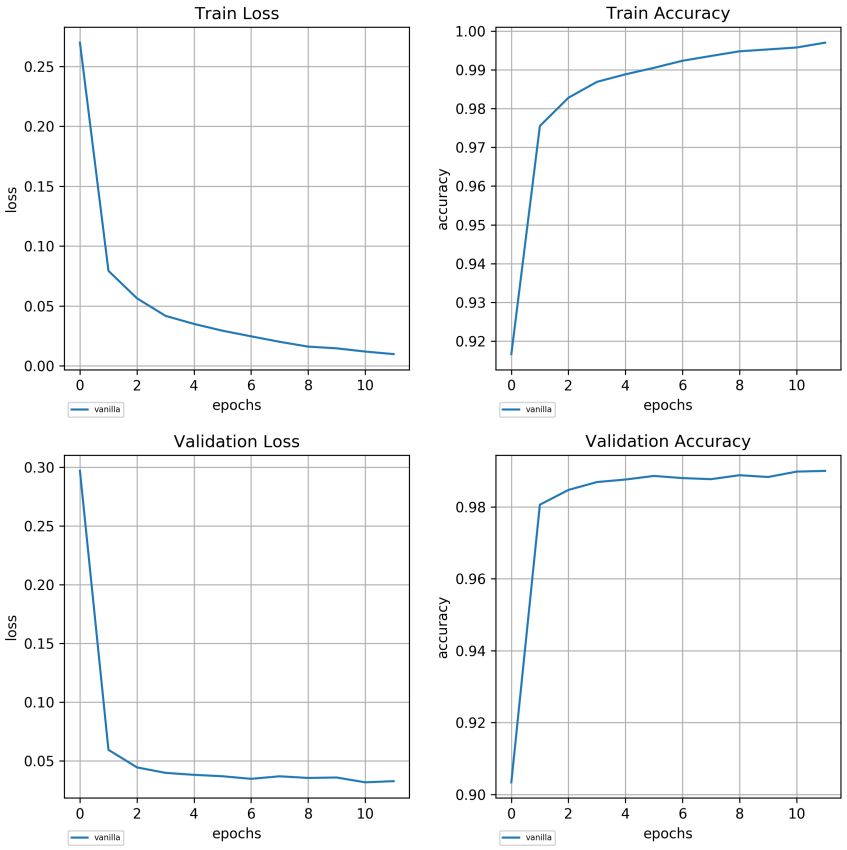

Figure 2: Test Network Training Data

For this first run, the hyperparameters used are shown in table 1.

Parameter	**Value**
Optimization Algorithm	*sgd*
Learning Rate	0.1
Batch Size	64
Number of Epochs	12
Number of filters per convolutional layer	16
Filter Size	3×3

Table 1: Initial Experiment Hyperparameters and design choices.

1.2 Learning Rate

Now that we know that our network works and is trainable, we begin experimenting with the hyperparameters by first selecting the learning rate from the set $lr = \{0.1, 0.01, 0.001, 0.0001\}$.

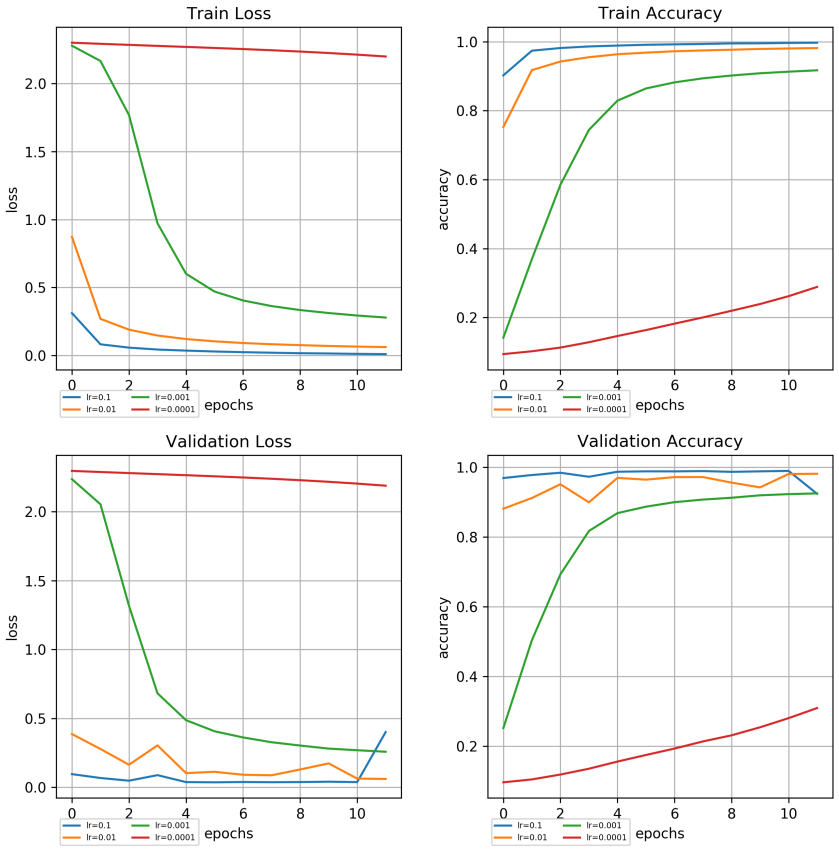

Figure 3: Performance with various learning rate values.

As it can be appreciated in the figure, the selection of the learning rate directly affects the training performance of the network. With really low values, it takes the network more epochs to properly learn the features from the training dataset, as the weights are updated by a very small fraction of the gradient, as shown by the red learning curves. On the other hand, too large of a learning rate can lead to oscillations around the local minimum or, in the worst case scenario, to not even converge.

1.3 Convolution Type

For the second experiment, we vary the size of the filters to be used by the two convolutional layers from the following set: $\{1, 3, 5, 7\}$. The number of filters per layer was kept to 16 in all tests.

The layer size and quantity of layers relates to the number of parameters that need to be adjusted during optimization. Thus, by selecting a filter size of 1, we can adjust a single parameter per filter, making the network adapt slower to the input data as features do not stand in single pixels. By choosing larger filter sizes, we also increase the number of parameters to train.

Choosing the filter size, as with any other parameter, is dependent on the data that wants to be processed by the network. For an image classifier, the filter size would depend on the input image resolution, and the size of the desired features to learn in each layer. For example, for high resolution images, it may make more sense to increase the filter sizes for the first layers to be able to decompose the image into significant features. In any case, it is a tunable design choice and should be selected depending on the results it yields for each application.

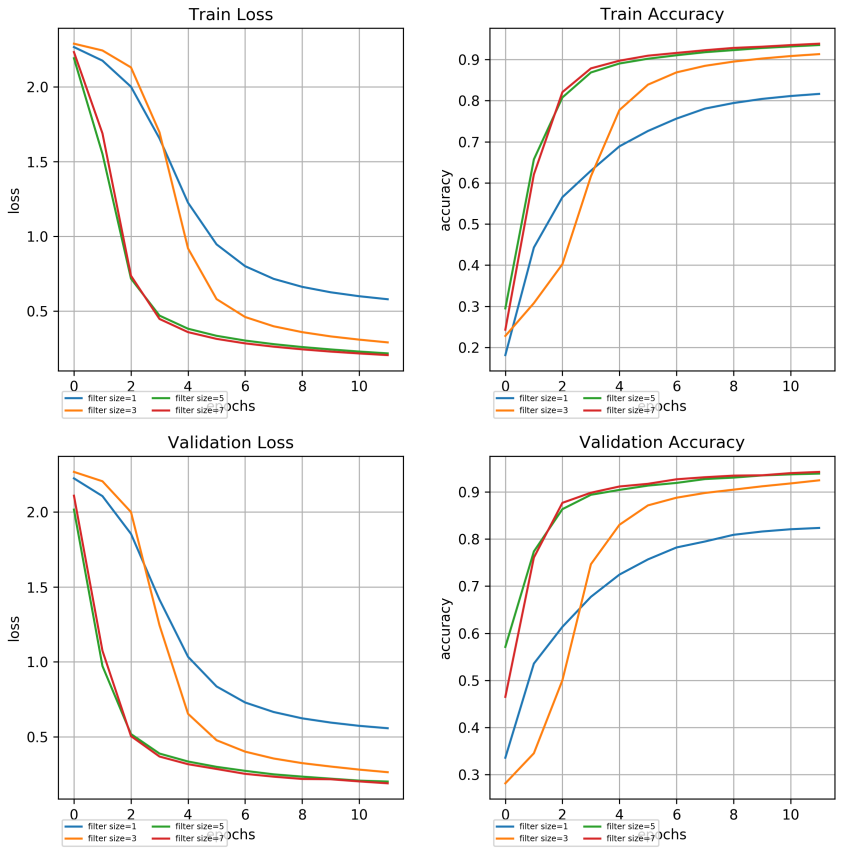

Figure 4: Performance with various learning rate values.

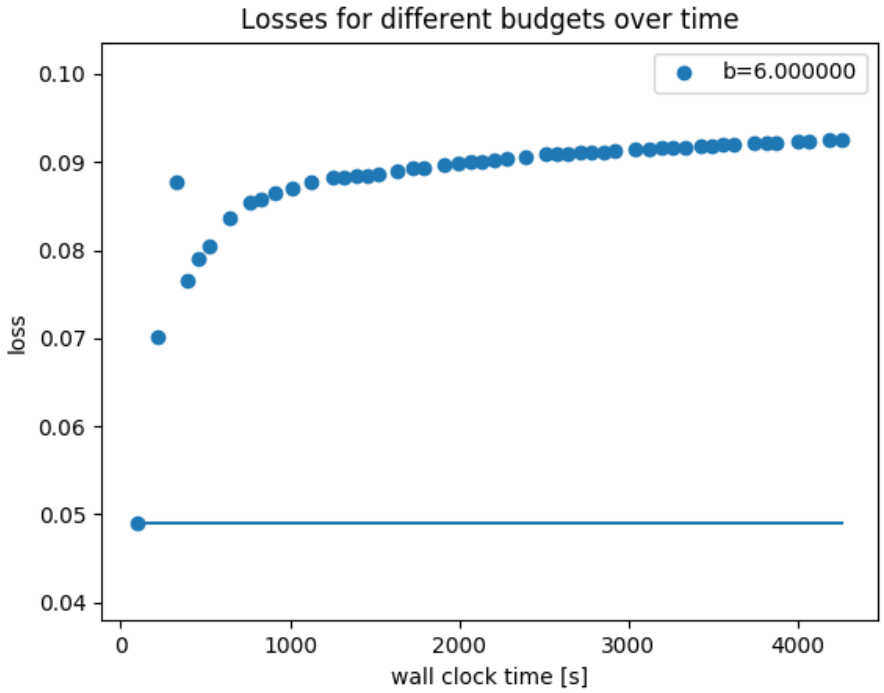

Figure 5: Random Search Losses.

1.4 Random Search

Finally, by using an automated algorithm, we can tune and fine the optimal hyperparameters for a certain model. In this particular case, the used algorithm was a random search in the sets and intervals shown in table 2.

Hyperparameter	Range/Set
Learning Rate	$[10^{-4}, 10^{-1}]$
Batch Size	$[16, 128]$
Number of filters	$[2^3, 2^6]$
Filter Size	$\{3, 5\}$

Table 2: Random Search Hyperparameters.

The loss values for each iteration can be seen in the following figure. It can be seen that the algorithm found the best solutions during the first set of the iterations, but due to it's random nature and the sensibility of the training performance to the hyperparameters, it soon drifted off of the original best iterations. Other algorithms actually tune the hyperparameters in a more systematic way, trying to improve the performance during each iteration.

After the random search algorithm finished, it returned the incumbent parameterization as the best it got during it's iterations. Such instance had the following hyperparameters:

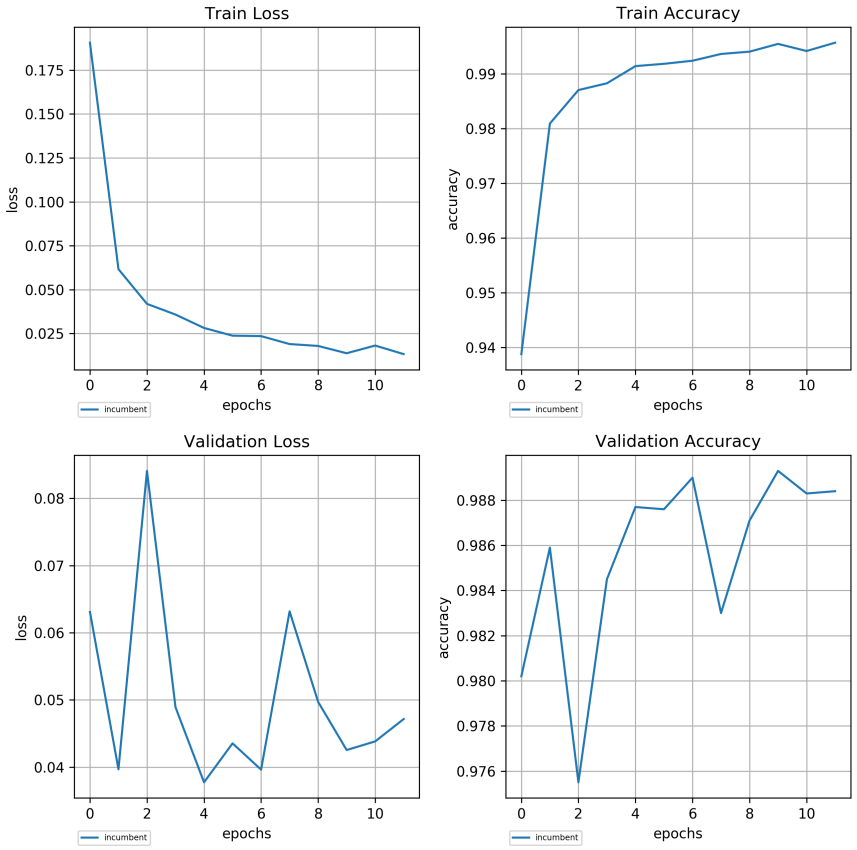

Figure 6: Random Search Losses.

Hyperparameter	Range/Set
Learning Rate	0.3454628959392778
Batch Size	41
Number of filters	10
Filter Size	3

Table 3: Random Search Hyperparameters.

The network was the retrained with those settings and had the following training and validation performance:

It can be seen that it achieved very high training and validation accuracies in just 12 epochs, and also had a great starting point after just a single epoch.

Conclusions

Hyperparameters and network design are, along with the selection and preparation of the dataset, the most critical tasks for machine learning, and there is a dependency of the parameters on the data to be learned by the neural network. There are techniques, algorithms and tools that help us deal with tuning the network to obtain better results for our machine learning solutions.