Deep neural network parameter understanding
AI4

Anne Groth

January 2019

1 Motivation

The overall goal of this project is to acquire more knowledge of how to do recognition in images using convolutional neural network. This then explicit means that knowledge about the architecture of a deep neural networks also will be researched.
To understand architecture of the convolutional deep neural network, different parameters will be analyzed in order to get a knowledge of the importance and influence of the given parameter.
The accuracy function is used to interpret the performance of the neural network and from this optimize the network.
A part of this project will also be to get familiar with a framework used for training of the the deep neural network. And use this framework to train on a already existing data set.
The project will be exploratory. Meaning that the purpose of this project is not result oriented, but the point is to get an understanding of the influence of the parameters.

2 Tasks

The tasks of this project will be understanding the optimizing process of the different parameters and test the performance of these on a given data set.
The first task will be to study the architecture of the convolution neural network. From this get an idea of the parameters there can be changed in order to optimize the performance. The parameters there will be analyzed is the different layers, filters, pooling methods, learning rates and possibly other important parameters there will be acquired knowledge to during the project period. Besides this the initialization methods will be investigated. These parameters will first be analyzed theoretically individually to understand their influence, afterward they will be used practically in a deep neural network.

2.1 Optimization of hyper parameters

The goal of training a network is to have the output of the network so close to the true output as possible.

It is wanted to analyze the hyper parameters batch size, solver, learning rate, weight initialization and activation function. From these parameters get an idea of the how they can be changed in order to optimize the performance.

2.1.1 Batch size

The batch size is the amount of data there is sent in parallel. With more computer power more data can be handled in parallel, and thereby it will take a shorter amount of time. A small batch size will make the divergence of the weights fluctuates much more, than a big batch size because it evaluates based on fewer samples. Also the weights will be updated more often with a smaller batch size. Having a large batch size will require a lot of memory space in the computer. The batch size is often chosen to be a 2n̂ value in order to use the memory more efficiently.[1]

2.1.2 Solver

There are different methods of optimizing the search in order to find the a better weights faster. These are called solvers. The solver it a methods of how to update the weights as favorable as possible, so that the performance of the neural network would increase. The methods studied in this project use the strategy back-propagation. In back-propagation the result of the output compared to the truth. The error is from here back propagated trough the network. This is done be using the partial derivatives also described as the gradients.

The solver lead the weights a step in the direction that would make the accuracy higher, based on the batch of data. The size of this step is given by the parameter Learning Rate.

To achieve the goal; having the outputs as close as the truth as possible. An expression of the mean output of a batch can be described by Equation 1 as a function of the weights.[2]

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w) \tag{1}$$

Where Q is the output, n is the number of instances in the batch, i indicates the specific instance. It is wanted to find the set of weights that result in the output being equal to the truth of the instances.

[1] https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-i-hyper-parameter-8129009f131b

[2] https://en.wikipedia.org/wiki/Stochastic$_g radient_d escent$

The classifier SDG (Stochastic Gradient Descent) uses a linear combination of the gradients in order to approaching the weights witch maximize the accuracy. This is done by setting the weights equal to the weights and then subtract a given factor of the gradient. This is shown in Equation 2.

$$w := w - \eta \Delta Q(w) \tag{2}$$

The factor (η) is the learning rate and is described in the next paragraph.

2.1.3 Learning rate

This parameter describes the size of the steps the solver takes toward the weights that gives an better results. A low learning rate will cause the system to learn slow, but also to converge to the most optimal minima. While a high learning rate will take bigger steps and might approach the best weights faster, but it can deviate from a global minima, and possible find a local minima. An example is shown in Figure 1, where the high learning rate takes a step there is too big, and there by misses the global minimum.

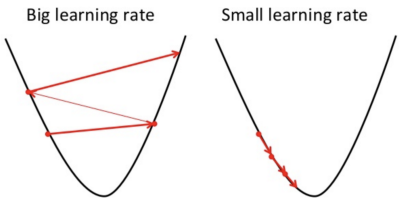


Figure 1: Loss curve for high and low learning rate[3]

2.1.4 Weight initialization

For being able to optimize the network it has to have some values to be optimized, therefore the weights has to be initialized. If the weights are not in initialized cleverly vanishing gradients and exploding gradients can occur. Vanishing happens when the the influence of the weight dies because of a disappearing small value in a given activation function. Exploding gradients happens when the initialized weights is too big which leads too big gradients, this can lead to excess of minima.[4]

It is also a possibility to use pretrained weights. This is primary for the convolutional layers. It might be that the weights, there have already been trained on another data set, extracts the same features as the ones needed. And thereby make a good starting point for the weights.

2.1.5 Activation function

The activation function is the method of activate the inputs in a neuron in a given way. The advantages about using a activation function is that it is possible to make the classifier non-linear.[5]

[4] https://medium.com/usf-msds/deep-learning-best-practices-1-weight-initialization-14e5c0295b94

[5] http://cs231n.github.io/neural-networks-1/

One of the commonly used activation functions is ReLU which is as described in the equation below.[6]

$$f(u) = max(0, u) \tag{3}$$

2.2 Optimization of architecture specific parameters

In this chapter the influence of the different parameters regarding the architecture of the convolutional neural network is studied. Three different types of layers are described. Afterward a short note on which parameters there have been chosen to vary is added.

2.2.1 Fully connected

An example of a fully connected layer is illustrated in the Figure 2. This is a layer where all neurons are connected between two layers.
Every connection contains a weight and all neurons uses an activation function.

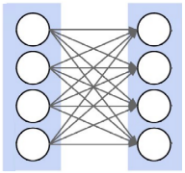


Figure 2: Fully connected layer

2.2.2 Convolutional

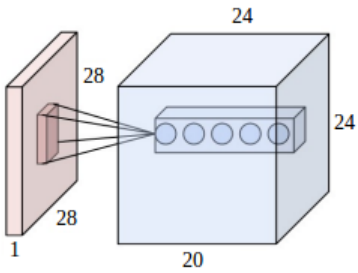


Figure 3: Convolutional layer

A convolutional layer, uses a **kernel** containing weights to slide over the image. The input to the next layer is the output the weights multiplied by the pixel values in the image. The slide the distance, ie the amount of pixels skipped before the core is multiplied on the pixel values, is called the **stride**. The depth of the output is equal to the **amount of filters** there is chosen as a output. It can be beneficial/necessary to apply a padding to the image. If no padding applied the pixels near the boarder will not be weighted as high as the other. And with padding it is possible to make sure the output dimension is valid.
An example is shown in Figure 3. In order to be sure that dimensions match to a valid output size Equation 4 can be used.

$$outputSize = \frac{inputSize - filterSize + 2 \cdot zeroPadding}{Stride} + 1 \tag{4}$$

For an example of a input size of 28x28, a kernel size of 5x5, a stride of 1 and zero padding:

$$outputSize = \frac{(28x28) - (5x5) + 2 \cdot 0}{1} + 1 = (24x24) \tag{5}$$

[6] https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning

2.2.3 Pooling

A pooling layer is as given in the name a layer that pools values. An example is shown in Figure 4. There is no weights to be remembered in this layer, this is pure down or up scaling of data. The advantages of the down pooling layers is to reduce the spatial size, and there by the amount of weights which have to be trained in future layers. There are different methods of pooling, one is max pooling. Where the highest value within the pooling window is passed on as input to the next layer.

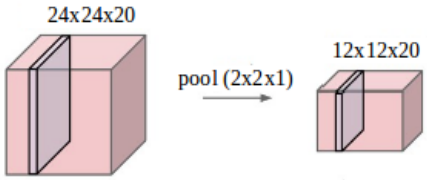


Figure 4: Pooling layer

2.2.4 Parameters varied to observe influance

From the knowledge gathered about the architecture specific parameters have the following parameters been chosen to change in order to observe the impact:

- Kernel size
- Stride
- Amount of convolutional filters
- Amount of convolutional layers
- Pooling layers

3 Method

In order to analyze the parameters is the mnist data set used[7]. This is a data set containing images of handwritten ciphers from 0-9. There is 1000 test images of each ciphers training 6000 training images.

Is has been arbitrarily chosen to use the architecture LeNet, this contains the three different layers which are earlier described. This can be seen in Figure 5.

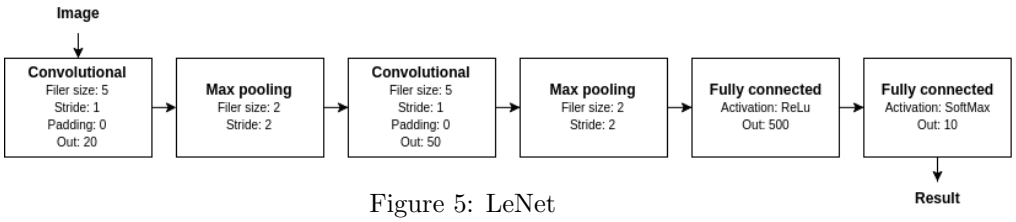


Figure 5: LeNet

For training the model on the images, is two different frameworks is tried out:

- **Tensorflow**
  With Keras[8] from Tensorflow is the model build up.

[7] http://yann.lecun.com/exdb/mnist/
[8] https://keras.io/

Keras is a library to Python containing many possibilities to build a model, tune all parameters, preprocess the dataset and ect.. All of the possibilities is listed at the documentation at the webpage.

- **Caffe**
  Using the API Digtigs[9] is Caffe testet.
  The interface allows easily changing of the parameters:
  Training epochs, Snapshot interval, Validation interval, Random seed, Batch size, Batch Accumulation, Solver type, Learning Rate.
  Along with the Data Transformations options:
  Subtract Mean and Crop Size
  Besides this three standard networks are saved and have a easy access in the API : *LeNet, AlexNet, GoogLeNet*

Both of the frameworks met the requirements to be able to do parameter optimization. It was chosen to use Tensorflow during this project. The choice is based on the possibilities of creating a program for training multiple different parameters as desired in one program. The freedom is also an advantage when doing mass experiments.

The network is build up with the library Keras as a Sequential model with the LeNet architecture.
There are chosen a initial value for all the parameters, thus one can be varied at a time. These initial values are shown in Table 1.

In order to be sure that it is the factor that have an impact on the result, and not the random initialization, is the 4 same seeds for each parameter. In this way the model has the same starting point in all experiments.

Parameter	**Initial values**
Hyper parameters	
Epochs	100
Batch size train	32
Batch size val	16
Learning rate	0.5
Solver	SGD
Activation	Relu
Architecture specific parameters	
Conv. 1 filters	20
Conv. 2 filters	50
Kernel size	5
Stride	1
Pooling window	2

Table 1: Initial parameter values

The created python script outputs a text file with the parameters below from all epochs in one training:

$$epoch, accuracy, validationaccuracy, loss, validationloss \tag{6}$$

This is done for 4 specific seeds. Which results in 4 documents containing these values for the 100 epochs. This is done for all parameters tried out.

A Matlab script has been created to be able to analyze the data. This loads the text file, and takes the mean of all four seeds. From this graphs of the accuracy

[9] https://github.com/NVIDIA/DIGITS

and loss have been made. It was chosen to show the graph of the validation accuracy in this report as this can describe the performance of the network as wanted.

4 Optimizing

In this section all of the result are analyzed. It was not all of the changed parameters which resulted in a significant change. These results are only describes and not shown in graphs.

4.1 Optimization of hyper parameters

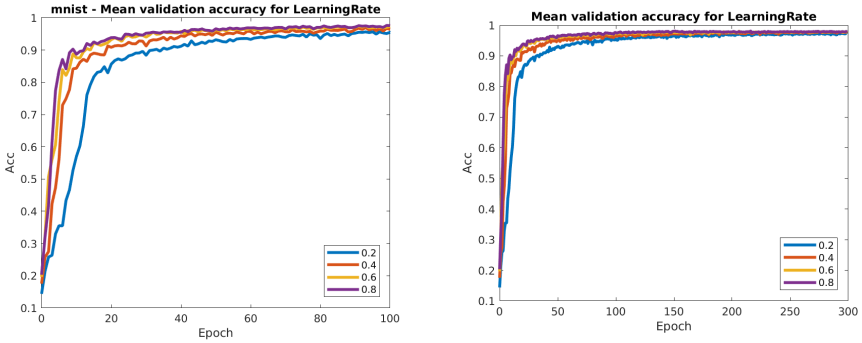
4.1.1 Batch size

The performance of different batch sizes was a bit incorporated by the performance on the computer used for the experiments. It was observed on the experiments which succeeded, that the batch size did not have a significant impact on the final accuracy.

4.1.2 Solver

From this experiment there was no significant difference, all of the graphs looked approximately the same. If more time was available it might have been nice to look into tuning other parameters than only the learning rate. To get a better insight in how the different solvers have their different advantages.

4.1.3 Learning rate



(a) 100 epochs

(b) 300 epochs

Figure 6: Accuracy for a different learning rates

It can be seen that with a higher learning rate, the the accuracy raises faster. This was expected from the theory, but it was also expected that the lower learning rate might result in a higher accuracy after more epochs. Therefore an other training with more epochs has been made. And it can be seen from figure 6b that it approaches the same accuracy.

4.1.4 Weight initialization

It was tried out with different initialized values. It has been observed that with only negative values, the network will not learn anything. Also normal distributed random weight initialization around 1 is not making the network learn, this might be due to exploding gradients.

4.1.5 Activation function

The activation functions used was: relu, softsign, tanh and linear.
There was no significant differences with these activation functions. It have been expected that the linear activation function would have resulted in a worse result, but not in these experiments. This could have been due to the simplicity of the data set.

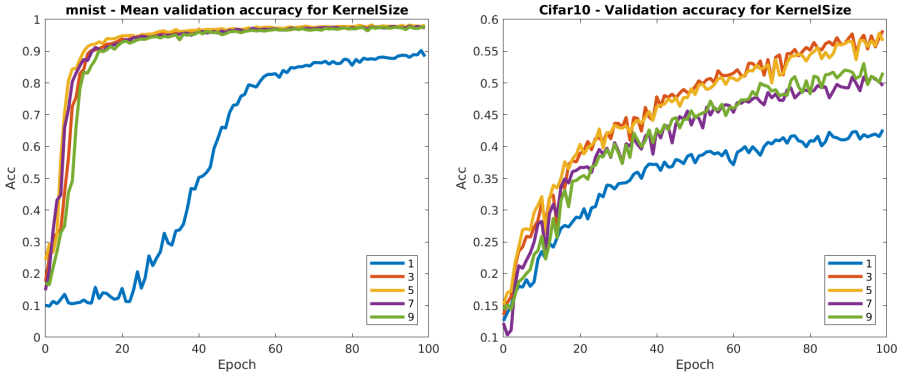
4.2 Optimization of architecture specific parameters

During the experiments it was discovered that the architecture specific parameters did not have a significant impact on the results. As the mnist dataset contains simple images, with the same characteristic lines for all classes, it was considered that the convalutional layers did not have that big an impact on the performance.

Therefore it was chosen to redo all the experiments using another data set. The cifar10 data set was chosen. This contains 10 classes with 1000 test images and 5000 training images each. The classes are as follows:

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

4.2.1 Kernel size

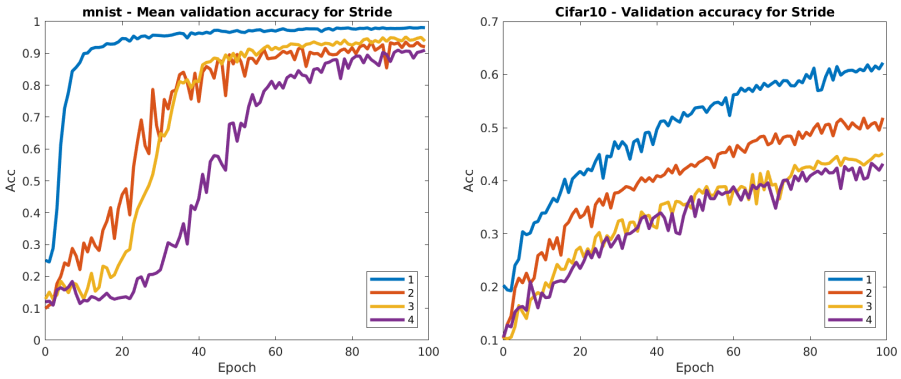


(a) mnist (b) cifar10

Figure 7: Accuracy for a different kernel sizes

There was only a significant change in the kernel size equal one in the mnist data set. The lower performance might be due to a kernel equal one is only a scale of the pixel values and make no further advantage for the result.
In the cifar10 data set it can be seen that the different kernel sizes has an impact on the performance, it can also be seen that a larger kernel not necessarily lead to better results.

4.2.2 Stride

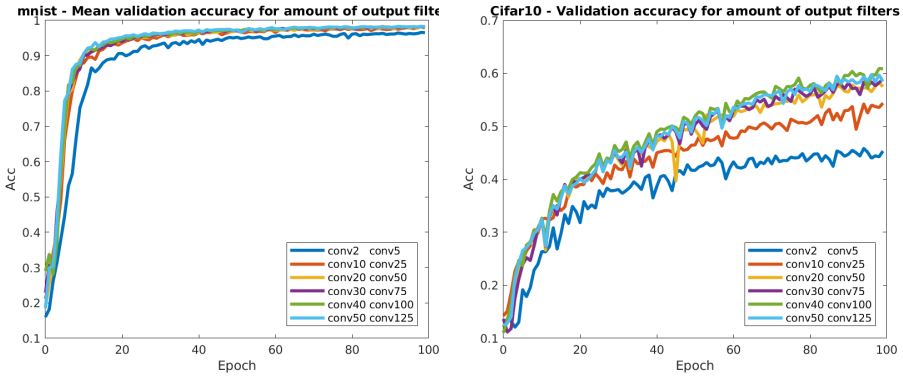


(a) mnist (b) cifar10

Figure 8: Accuracy for a different strides

From the figure it can be seen that the stride have an impact on both data sets. It can be seen that a larger stride leads to a lower accuracy. Which makes sense as more data is discarded.

4.2.3 Amount of convolutional filters



(a) mnist (b) cifar10

Figure 9: Accuracy for different amount of filters in the convolutional layers

It is observed that the amount of convolutional filterers have a larger impact on the cifar10 data set. The cifar10 dataset is more complex than the mnist, and the classification might rely more on the features extracted than the mnist.

4.2.4 Amount of convolutional layers

In this training process has the is the performance of a network with 2 , 1 and none convolutional layers compared. This result of this comparison degenerating equally to the amount of filters per layer. There is a small change from no layers and none between one and two layers for the mnist. And the performance of the cifar10 increases with the amount of convalutional layers.

4.2.5 Pooling

Four different versions of the pooling layers have been trained. These are the original two layers of max pool, only with one layer of max pooling, no pooling layers and also two layers of pooling where the pooling method is average (L2-norm) instead of max pooling. There where no significant change in the performance. Neither on the mnist nor the cifar data set. From which it can be seen that it can be computational favourable to down scale.

5 Conclusion

Different parameters for training a neural network have been analyzed.
It might have been more describing to use a more complex data set than the one originally used, also for the hyper parameter optimization.
Another factor which have been nice to observe is the training time, and from this being able to know the impact on this, as well as the loss and accuracy.