# Homework 2: Neural Networks with TensorFlow

## Large-Scale Data Analysis

Christian Igel

**Submission Deadline:**
*19 May 2020, 10:00*

## 1 Standard Neural Network (15 pts.)

Consider the following Keras model:

```
tf.keras.models.Sequential([
  tf.keras.layers.Dense(64, activation='sigmoid', input_shape=(1,), name='
    hidden1'),
  tf.keras.layers.Dense(1, activation='linear', name='output')
])
```

1. **(5 pts.)** Add a layer (`tf.keras.layers.Dense`) with 32 neurons and sigmoid activation function directly before the layer with the name "output".

   Show the code in your report.

2. **(5 pts.)** Rewrite the model (including the new layer) using the functional functional API (e.g., see the notebook `LSDA2020_DL_MNIST_with_different_Keras_APIs.ipynb` or `https://www.tensorflow.org/guide/keras/functional`). Show the code in your report.

3. **(5 pts.)** Add a shortcut connection from the input layer to the last layer. That is, the activation of the last layer should be a weighted combination of the outputs from the previous layer (the one you added) and the external input (plus the bias).

   To clarify:

   - The input and output dimensionality of the model are not supposed to change, i.e., the network should still implement a mapping $\mathbb{R} \to \mathbb{R}$.
   - Yes, the additional input to the last layer should get a trainable weight.

   You may need to concatenate the output of layers, which can be achieved using `tf.keras.layers.concatenate`.

   Report the number of trainable parameters of the resulting model. Show the code in your report.

## 2 Convolutional neural network for traffic sign recognition (40 pts.)

We consider the data from the German Traffic Sign Recognition Benchmark [5]. First the data will be briefly introduced. Then a reading assignment is given before the actual tasks are described.


Figure 1: Examples from the traffic sign data set.

**The Traffic Sign Recognition Data** Recognition of traffic signs is a challenging real-world problem of high industrial relevance. Traffic sign recognition can be viewed as a multi-class classification problem with unbalanced class frequencies, in which one has to cope with large variations in visual appearances due to illumination changes, partial occlusions, rotations, weather conditions, etc. However, humans are capable of recognizing the large variety of existing road signs with close to 100 % correctness – not only in real-world driving situations, which provides both context and multiple views of a single traffic sign, but also when looking at single images as those shown in Figure 1. Now the question is how good a computer can become at solving this problem.

**Reading** Carefully read the following frequently cited paper,[1] which is freely available online:

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

You will need algorithmic details of the described method for answering question below.

**Simple CNN** Consider the notebook `LSDA2020_Traffic_Signs_Assignment.ipynb` available on Absalon. It trains a convolutional neural network (CNN) for classifying traffic signs. It uses the same data as the competition described in [5].

The notebook is supposed to run with GPU support, for example, using *Google Colaboratory*. Even then, executing it will take some time.

1. **(15 pts.)** Add batch normalization to the model using `tf.keras.layers.BatchNormalization`. Please show in the report all places where you added it (include the layer before and after for each batch normalization layer).

   It is an open discussion where to do the batch normalization relative to the non-linearities (activation functions). Here you have to do it as described in [3].

2. **(15 pts.)** Report the number of *trainable* parameters of the model after you added batch normalization. Explain the number of parameters for each layer *individually*.

   The batch normalization layer has trainable parameters that were not explained in the lecture. Explain these trainable parameters (i.e., write what they do) in the report based on [3].

3. **(10 pts.)** Convolutional neural networks are highly complex models. To reduce the risk of overfitting and in order to be able to learn difficult tasks with a high input variability, many training examples are needed. *Data augmentation* is a way to enlarge the training data set. The training data set is extended by artificial input generated from the available data. For example, a new training image can be generated by rotating, flipping, and/or shifting an available inout image. In the context of neural networks, this type of data augmentation can be traced back to

[1] The learning goal of this part of the assignment is gaining experience in reading original scientific Deep Learning articles.

[1]. It has been used successfully for CNNs, for instance already in the influential work by [4]

Inspect the notebook `LSDA2020_Traffic_Signs_Assignment.ipynb`. Answer the following questions briefly in your report: Which transformations are applied to the input images? Why is a transformation conditioned on the label?

Please add at least one additional transformation. Show the corresponding code in your report and briefly explain why you think that this is a reasonable augmentation.

## 3 Experimental architecture comparison (30 pts.)

The task is to empirically compare different CNN architectures. Take the notebook `LSDA2020_Traffic_Signs_Assignment.ipynb` as a starting point.

You are supposed to investigate one of the following questions (you can choose freely):

- Does the perfomance degrade when you swap batch normalization and non-linearity?
- Does adding dropout (using `tf.keras.layers.Dropout`) help?
- What happens to the performance if the inputs are converted to grayscale images and a single input channel is used?

Training a neural network is a random process (random weight initialization, random mini-batches). Thus, single trials are not sufficient to establish that a modification generally improves the learning for a given task. Proper statistical evaluation is required. The least you can do is to repeat the training and evaluation of the baseline and your alternative system a couple of times and compare the mean/median performance of the approaches. For a scientific study, you would establish the statistical significance of your findings using a (typically non-parametric) statistical significance test. For this assignment, it is sufficient to perform three independent trials for each architecture (you are encouraged to do more, but that is optional).

Training time may be an issue, and longer training can give you better results. **Thus, start with the experiments in good time before the deadline.** Fix a maximum number of epochs for your experiments. This should be *at least* 400 epochs. Remember to report this number, because the budget may influence the ranking of methods. Visualize the training progress.

In the report, you are supposed to describe what you did, present the results, discuss the results, and draw very careful preliminary conclusions. The report should contain a plot (remember proper axes labels etc.) showing the training progress vs. epochs.

## 4 Keras training and testing mode (15 pts.)

Models and layers in Keras have a Boolean flag that can put them in training and testing mode by specifying `training=True` and `training=False`, respectively. The higher level functions, such as `tf.keras.Model.fit`, set these flags automatically.

Please explain why two different modes are necessary

1. when using dropout (see the slides or section 7.12 in `https://www.deeplearningbook.org/contents/regularization.html` [2]), and
2. when using batch normalization [3] .

## 5 Challenge (optional)

*This part is optional.* Human performance on the traffic sign recognition task on the test set is 99.22 %, the best algorithm in [5] achieves 99.46,% using an ensemble of CNNs.

Now we challenge you: How good can you get a CNN to perform on this task? For example, you could try dropout; adding neurons, layers or connections to the given architecture; better training augmentation; starting with a pretrained model; test time augmentation; better preprocessing; and/or changing the optimizer and its hyperparameters.

The constraints are:

- The solution should be a single CNN, not a ensemble.
- The solution should be reproducible, that is, if you repeat the training a few times, you should get a solution of similar quality.
- Training the network should not take longer than a couple of hours on *Google Colaboratory*.
- The code should fit in a notebook not longer than twice the one handed out.
- The notebook should be self-contained, only common libraries may be imported (e.g., Scikit-image or PILLOW would be OK). The fewer external libraries the better.
- You may look for inspiration online. However, please cite all sources of inspiration you used.

If you consistently get better than 99 %, your solution is good. If you consistently beat human performance, your solution is very good. If you consistently reach or beat 99.46,% your solution is excellent. Let me know if you get a solution that is good or better!

## References


[1] H. S. Baird. Document image defect models. In *Structured Document Image Analysis*, pages 546–556. Springer, 1992.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[5] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.