Right from the beginning we worked with ladder networks, implemented with tensorflow on python. It uses the unlabelled data by comparing the pure unlabelled data with unlabelled data to which we some gaussian noise is added. The input that must be given to the program is the (training) data, the (test) data that should be labelled, the layer sizes, the cost factor per layer (how bad it is when the result of unlabelled data points is changed by the noise), the standard deviation of the noise, the number of training steps and the step from which on the step size should be reduced.

It turns out that it is good to put a high denoising cost on the first layer (1000), a lower one on the next layer (10) and afterwards stay with a small cost (0.1) – giving a much higher overall cost puts to much emphasis and decreases the accuracy dramatically (factor 10 -> only 40% accuracy). With a much lower cost, the unlabelled data can not alter the prediction at all at is useless.

The accuracy strongly depends on the chosen noise std. After trials with stds between 0.08 and 0.25, we settled for 0.2. The step after which the step size is decreased did not affect the results a lot in a reasonable range of parameters; we settled with step 15.

Almost 93% accuracy were obtained with various layer sizes and training steps, i.e. [128,2000,2000,2000,10] with 100 steps, but also [128,1000,1000,1000,10] and 200 steps. The similarity of the results demotivated us from trying more extensively with larger layer sizes. Also, the memory capacity on the euler cluster (20 GB) would soon become to small with our way of saving checkpoints and even higher layer sizes.

More steps did not seem to improve the result much more.

Thus, to achieve significantly better results and score above the high baseline we would probably need a new algorithm/approach. Since optimization of this approach already took quite some time, no new approach was tired.