Machine Learning - Homework 10 Report

Lukas Johannes Ruettgers (2023403372)

December 20, 2023

1 Introduction

Convolutional Neural Networks (CNNs) mimic the way the human retina and prefrontal cortex process its visual observations. Their architecture relies on the heuristic that observations are incrementally interpreted from local features to global relations among more abstract representations of these local features and achieved tremendous success in downstream computer vision tasks during the last century. The prototypical CNN consists out of a convolutional stage that repeatedly aggregates local patterns with kernel convolutions into denser embeddings, process these embeddings through a non-linear activation function and finally subsample the data by computing the maximum or average value for each region. This convolutional layer then culminates in a final layer of fully connected weights which seek to obtain the desired output from the embedded feature space.

2 Experiment Objective

This experiment examines the impact specific hyperparameters play with regard to the classification performance of CNNs. To that end, the experiments avail to the *MNIST Digits* dataset, which contains 60,000 grayscale 28×28 images of handdrawn digits. Specifically, I want to investigate:

1. what advantage the ELU() activation function has over the classical ReLU() choice,
2. whether the increase of the receptive field associated with a higher dilation proves useful for the classification of the MNIST digits,
3. and how strongly the classification performance and overfitting depends on the fully connected layers after the convolution stage.

To approach these three questions, I implemented a CNN that receives kernel size, stride, padding, dilation, bias, input and output dimensions, activation functions, layer depth and the size of the fully connected layers at the end as input parameters and dynamically constructs its layer structure based on these input parameters. Then I created a baseline model by selecting parameters that achieved mediocre classification performance on the MNIST Digits dataset. These parameters were the following:

- Number of convolutional layers: LAYER_DEPTH=4,
- Number of channels at each point in the convolutional stage: CHANNEL=[1, 4, 16, 64, 128],
- Kernel size in convolutional layers: KRNLSIZE=3,
- Kernel size in max pooling layers: POOLKRNLSIZE=2,
- Padding in convolutional layers: PAD=1,
- Dilation in convolutional layers: DIL=1,
- Stride in convolutional layers: STRIDE=1 (Note that the stride in pooling layers equates the kernel size by default),
- Bias: BIAS=True,
- Activation function after each convolution and after each fully connected layer: ACT=ReLU,

- Width of the fully connected layer between the last convolutional layer and the output layer: FCLAYERWIDTH=128.

By the nature of the dataset and the classification task into 10 possible digit classes, the input size was chosen to be 28 and the output size was a vector of size 10, whose values were normalized to $[0, 1]$ with a final softmax layer. For each of the three questions above, I adjusted these baseline parameters and hence obtained the following three models:

1. **ELU:** Baseline model with ELU() activation function in each layer instead of ReLU().
2. **Dilated:** Baseline model with dilation = 2 instead of dilation = 1 in each convolutional layer. The number of convolutional layers was decreased to 3 because the output size reached $128 \times 2 \times 2$ in the third layer, which is already smaller than the used kernel size.
3. **Stronger FC:** Baseline model with twice the FCLAYERWIDTH (256 instead of 128).

2.1 Implementation Details

2.1.1 Convolutional Neural Network

To evaluate CNNs for small alternations in hyperparameters, I implemented a CNN that received all the parameters listed above as input and configured its structure dynamically based on the specified quantities. To ensure that kernels match the input dimensions at each layer, some heuristics were applied to ensure that the size (both width and height since we have quadratic images)

$$d_{out} = \frac{d_{in} + 2 \cdot \text{padding} - \text{dilation} \cdot (\text{kernelsize} - 1) - 1}{\text{stride}}$$

was indeed an integer, where d_{in} is the input size and d_{out} the output size accordingly. If d_{out} was not an integer, then the module ignored some pixels at the corner of the inputs, which may be acceptable for the raw input image, but crucial at deeper layers were the corner values also can represent meaningful abstractions. To ensure more fairness between the models, I hence included these aforementioned heuristics, which included, but were not limited to, increasing the padding and slightly adjusting the kernel size or the dilation. In practice, only the dilation model availed to this dynamic adjustment and increased the padding by one in the second max pooling layer.

2.1.2 Training

The dataset was split into 10 equally sized chunks. One of them was randomly selected as a cross validation and training dataset respectively. The remaining eight chunks were concatenated to the training dataset. Each model was trained with the Adam optimizer and 80 training epochs. In each epoch, the entire training dataset was split into 100 batches and one gradient descent step was performed on the cross entropy loss between the model's prediction on this entire batch and a one hot encoding of the ground truth labels. For each batch, the prediction accuracy was computed as the average softmax value at the ground truth index of the prediction vector. In contrast to mapping the highest value to 1 and all other values to 0, this method further accounts for the uncertainty of the model's prediction, in particular with regard to the aleatoric uncertainty inherent to the dataset, since some digits are even hard to classify for a human. This batch accuracy was averaged over all batches in one epoch. After each epoch, the model's accuracy on the cross validation dataset was computed in the same way and both values along with the average cross entropy loss were logged to the `results.txt` file in this folder.

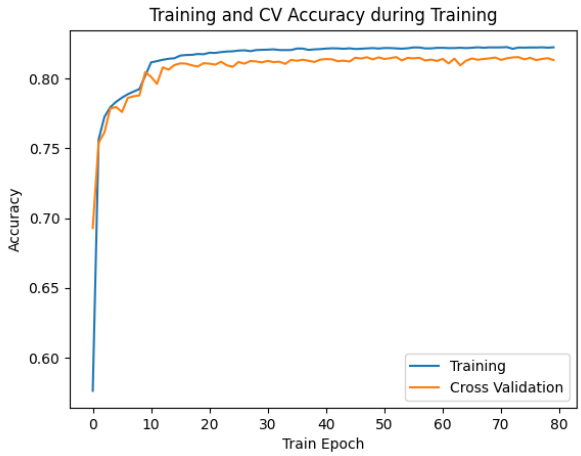
2.1.3 Testing

The accuracy on the test dataset was computed similarly. Moreover, the computation of the confusion matrix followed the same spirit, as each row $i, 0 \leq ileq9$, in the confusion matrix represents the average softmax vector that the model predicted for a digit i.

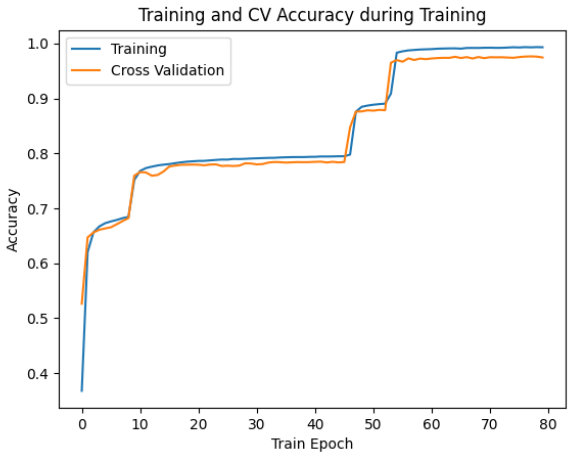
2.1.4 Reproducibility

To ensure the reproducibility of the experiment results, all random number generators from torch, numpy and the internal random package were fixed. Moreover, I tried to force torch and CUDA to use only deterministic algorithms by setting the environment variable `CUBLAS_WORKSPACE_CONFIG` according to the API Reference Guide of cuBLAS (1). However, I wasn't able to perform any operations on the GPU if I set this variable. For this reason, there was still a small amount of randomness in the execution when running the code on a GPU, but I ran it multiple times to observe the strength of the fluctuation.

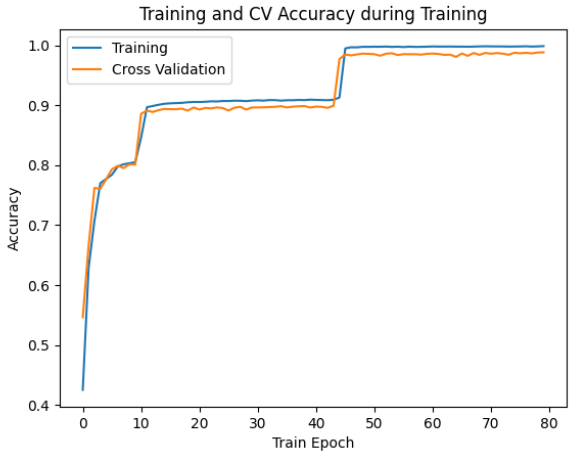
3 Results



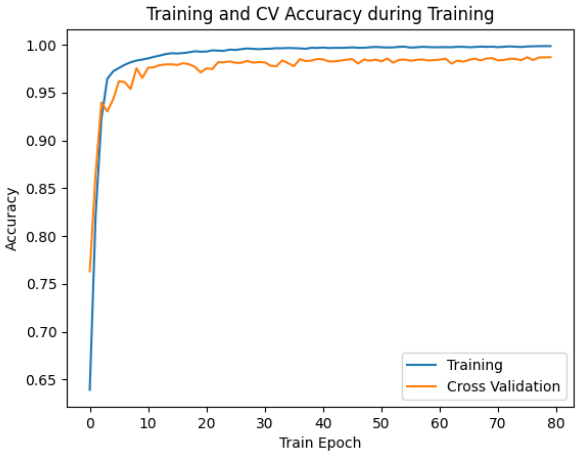
(a) Baseline



(b) Dilated



(c) Stronger FC



(d) ELU

Figure 1: Learning curve during training that reports the classification accuracy for both the training and the cross validation dataset.

Because the accuracy seemed the most intuitive and understandable metric, I decided to plot the learning curves over this metric. The baseline model has a relatively smooth learning progress, as the accuracy on both the training and the cross validation dataset demonstrates (Fig. 1a). However, it converges at an accuracy of around 81% (Fig. 2). As the confusion matrix (Fig. 3a) indicates, this is mainly due to the inability of the model to correctly classify the digits 4 and 6, while it achieves almost perfect classification results on the other digits.

The increased receptive field of the dilated model allows the kernels in the convolutional layers to capture more global patterns from the beginning. As noted in the previous section, this additional dilation had to be traded against a smaller number of convolutional layers. Nevertheless, the model seems to profit from this trade, as Fig. 1b depicts. While the learning curve looks similar for the first half of training, the

model eventually manages to capture essential visual attributes of the digits 4 and 6 and the accuracy steeply increases at both epochs respectively. Unfortunately, this performance is not always achieved. The non-deterministic algorithms used in the CUDA library seem to be sensitive to randomness in an extent that the model's accuracy fluctuates from 80% to 97%. This happens although the seeds for torch, numpy and the internal random module were fixed. Fig. 2 hence includes the best, the worst and the median model's performance for the dilated model. The learning curves for the two other models are also included in the `Figures` folder. While the worst model manages to classify only 8 out of 10 digits correctly and obtains a confusion matrix that resembles the one of the baseline model, the median model progresses to good classifications on 9 out of 10 digits. Its confusion matrix is displayed in Fig. 3b. As the confusion matrix depicts, the model manages to almost perfectly classify all digits from 1 to 9. Surprisingly, it fails to distinguish 0es from other digits and culminates in a decision rule that leaves the softmax vector at the first index always 0, which results in the almost uniformly random classification distribution for digits 0. Another surprising observation is that the model assumes a resemblance between the digits 2 and 0, while most humans would confuse the digit 0 rather with 8 or 9. With regard to the cross-entropy loss function, this might indicate that the models' erroneous predictions on other digits would increase to drastically if the model shifted its inductive bias more towards 0. The information purity thus remains highest if the model does not even try to predict 0 at all, because all other digits receive a strong prediction accuracy.

Model	Training Accuracy	CV Accuracy	Test Accuracy
Baseline:	0.8224	0.8139	0.8116
Dilated (Worst)	0.8192	0.8037	0.8010
Dilated (Median)	0.8954	0.8815	0.8822
Dilated (Best)	0.9933	0.9747	0.9765
Stronger FC	0.9985	0.9881	0.9872
ELU	0.9987	0.9871	0.9872

Figure 2: Accuracies of the four models on the training, cross validation (CV) and test dataset. Because of the relatively high performance fluctuation of the dilated model, both the weakest, the strongest and the median performance is displayed. This data was extracted from `results.txt`.

0.9892	0.0004	0.0007	0.0004	0.0004	0.0054	0.0004	0.0020	0.0005	0.0004
0.0001	0.9960	0.0015	0.0001	0.0001	0.0008	0.0001	0.0001	0.0001	0.0010
0.0000	0.0005	0.9922	0.0000	0.0000	0.0000	0.0000	0.0063	0.0009	0.0000
0.0007	0.0000	0.0017	0.9946	0.0000	0.0024	0.0000	0.0000	0.0000	0.0005
0.0983	0.1000	0.0996	0.0983	0.0983	0.0983	0.0983	0.1007	0.0983	0.1099
0.0000	0.0001	0.0000	0.0002	0.0000	0.9979	0.0000	0.0000	0.0018	0.0000
0.0998	0.0967	0.0966	0.0966	0.0966	0.1262	0.0966	0.0966	0.0976	0.0966
0.0000	0.0025	0.0028	0.0006	0.0000	0.0022	0.0000	0.9853	0.0004	0.0061
0.0000	0.0017	0.0001	0.0093	0.0000	0.0196	0.0000	0.0017	0.9631	0.0044
0.0009	0.0009	0.0009	0.0011	0.0009	0.0080	0.0009	0.0055	0.0027	0.9780

(a) Baseline

0.0000	0.0035	0.2523	0.0103	0.1365	0.1237	0.0728	0.1323	0.1159	0.1527
0.0000	0.9865	0.0088	0.0009	0.0001	0.0008	0.0001	0.0014	0.0002	0.0013
0.0000	0.0004	0.9872	0.0003	0.0000	0.0015	0.0000	0.0067	0.0037	0.0001
0.0000	0.0008	0.0116	0.9647	0.0000	0.0075	0.0000	0.0049	0.0057	0.0047
0.0000	0.0005	0.0060	0.0000	0.9790	0.0000	0.0041	0.0017	0.0001	0.0086
0.0000	0.0017	0.0015	0.0025	0.0015	0.9748	0.0066	0.0022	0.0083	0.0009
0.0000	0.0002	0.0012	0.0000	0.0025	0.0046	0.9900	0.0000	0.0015	0.0000
0.0000	0.0016	0.0142	0.0002	0.0065	0.0027	0.0000	0.9688	0.0009	0.0051
0.0000	0.0017	0.0108	0.0067	0.0003	0.0011	0.0030	0.0020	0.9720	0.0025
0.0000	0.0033	0.0020	0.0017	0.0107	0.0049	0.0016	0.0061	0.0055	0.9643

(b) Dilated (Median)

0.9944	0.0000	0.0000	0.0000	0.0000	0.0000	0.0052	0.0000	0.0004	0.0000
0.0000	0.9962	0.0032	0.0005	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001
0.0000	0.0000	0.9954	0.0000	0.0000	0.0000	0.0000	0.0046	0.0000	0.0000
0.0017	0.0033	0.0027	0.9718	0.0000	0.0103	0.0000	0.0000	0.0068	0.0033
0.0000	0.0012	0.0035	0.0000	0.9753	0.0000	0.0067	0.0011	0.0019	0.0103
0.0000	0.0000	0.0000	0.0053	0.0000	0.9887	0.0042	0.0000	0.0018	0.0000
0.0020	0.0000	0.0000	0.0000	0.0000	0.0033	0.9930	0.0000	0.0018	0.0000
0.0015	0.0025	0.0072	0.0000	0.0004	0.0020	0.0000	0.9827	0.0004	0.0031
0.0024	0.0000	0.0018	0.0036	0.0000	0.0022	0.0001	0.0017	0.9882	0.0000
0.0017	0.0000	0.0000	0.0000	0.0052	0.0033	0.0000	0.0017	0.0033	0.9848

(c) Stronger FC

0.9944	0.0016	0.0000	0.0000	0.0000	0.0000	0.0022	0.0017	0.0001	0.0000
0.0000	0.9959	0.0039	0.0000	0.0001	0.0000	0.0000	0.0000	0.0000	0.0001
0.0000	0.0033	0.9924	0.0007	0.0000	0.0000	0.0000	0.0036	0.0000	0.0000
0.0017	0.0000	0.0065	0.9775	0.0000	0.0093	0.0000	0.0010	0.0040	0.0000
0.0000	0.0017	0.0017	0.0000	0.9887	0.0000	0.0017	0.0019	0.0000	0.0041
0.0000	0.0000	0.0000	0.0024	0.0000	0.9919	0.0054	0.0000	0.0000	0.0003
0.0000	0.0000	0.0000	0.0000	0.0016	0.0032	0.9935	0.0000	0.0017	0.0000
0.0000	0.0029	0.0038	0.0000	0.0000	0.0000	0.0000	0.9887	0.0016	0.0030
0.0018	0.0025	0.0005	0.0057	0.0048	0.0073	0.0020	0.0014	0.9705	0.0034
0.0000	0.0000	0.0001	0.0017	0.0098	0.0033	0.0000	0.0060	0.0026	0.9764

(d) ELU

Figure 3: Comparing the confusion matrices of the four CNN models. The index (starting from 0) of each row indicates the ground truth label, the column index represents the prediction of the model. This data was extracted from `results.txt`.

On the other side, both the model with the stronger fully connected layers and the model with the ELU activation function achieve strong results of 98.72% on the testing dataset (Fig. 2). The implications of this high accuracy are two-fold. It first demonstrates that our choice of the non-linear function has

a strong impact on how accurately we can approximate the unknown perfect classifier function f. Since ReLU is computationally simpler than ELU for negative inputs, this underscores the trade-off between efficiency and accuracy that both functions mostly differ in. On the other side, even without dilations, more powerful non-linear functions or other alternations in the hyperparameters, the latent features that are obtained after the convolution stage still carry enough information for a stronger fully connected layer to make correct predictions. Intuitively, the additional linear weights may help to obtain a better linearized approximation of the non-linearity that the model does not manage to explain with the ReLU activation function. Fortunately, the additional weights do not cause the model to overfit the training data, as the cross validation scores still remain almost as good as the scores on the training dataset (Fig. 2). In particular, the model with the stronger fully connected layers does not lack behind the ELU model regarding the test and cross validation accuracy. Frankly, this is not what I expected before conducting the experiments since I expected the model to overfit the testing data more. However, this might indicate that the latent space after the convolutions transform the observations to abstractions that still carry meaning which is too general to memorize the training data.

Interestingly, the learning curve of the ELU model smoothly and quickly approaches almost perfect prediction accuracy (Fig. 1d), while the learning curve model with the stronger fully connected layers exhibits similar abrupt improvements of the prediction accuracy in 10% steps as the dilated model (Fig. 1c). This suggests that the model suddenly reaches strong decreases in the landscape of the loss function. This may be due to the sudden decrease of the gradient of the ReLU function from 1 to 0 for values smaller than 0, while the ELU gradient remains continuous in this region.

4 Conclusion

To conclude, these experiments visualize the effect of a non-linear activation function with non-continuous gradients on the loss landscape during training and underscore the strength of both fully connected layers and computationally more costly non-linear activation functions like ELU to better approximate the visual patterns in the data.

References

[1] NVIDIA Corporation & affiliates. cuBLAS API Reference Guide. https://docs.nvidia.com/cuda/cublas/index.html#cublasApi_reproducibility, 2023. Accessed: 2023-12-20.