Deep Learning Project 1

Jacob Bamberger (328027), **Kristians Cernevics** (293870), **Mariella Kast** (271602)

November 29, 2021

1 Introduction

Image classification is one of the fundamental applications of neural networks that is used to blur the lines between the computer vision and human vision. This is the process of identifying unique aspects in the image and allocating digital pixels to the corresponding classes. Although the relationship between the data and the classes must be known beforehand to train a neural network, the learning is done by the network itself.
We investigate and compare different neural network architectures in order to more accurately predict which of the two images of the digits is larger than the other. First, we discuss and justify the design of our networks in and then we present the results of few of our selected networks. We pay special attention to implementation of auxiliary loses and weight-sharing in order to improve both accuracy and speed of our networks.

2 Design of network architectures

Base Line models
Our investigation starts by building two straight-forward networks to establish a baseline for accuracy. We name the first one *LinearNet*, as it consists of two linear layers with ReLu activation functions. The second one is inspired by LeNet [2] as it consists of two convolutional layers (kernel size = 3) with max pooling (kernel size = 2) and further processing with two linear layers, where all activations are done with ReLu. Thus, the *ConvNet* treats the two input images as two channels that are simultaneously processed.
Improved design
As suggested by the project description, we explore two design strategies to improve on these baseline models: auxiliary loss and weight sharing.

The former follows naturally from identifying a 'subtask' in our classification task, which in our case is predicting the classes of the two images. From *ConvNet* we thus arrive at *AuxNet*, by first forcing a hidden layer to consist of 20 units (10 for each image), and then defining the auxiliary loss from the task of predicting both classes accurately. Let y be the output of the class predictions, i.e. a vector of size 20 and x be the final network prediction, our total loss function for *AuxNet* is then

$$Loss = CELoss_x + \alpha(CELoss_{y[:10]} + CELoss_{y[10:]}), \tag{1}$$

where the α is the weight of the auxiliary loss. Intuitively, this auxiliary loss will push the network towards a better prediction of the numbers in the images, which will then facilitate the task of the final linear layer of outputting which of the numbers is larger.

The second suggestion for improvement is weight sharing, which leverages symmetries of the input in the framework of *Geometric Deep Learning* [1]. For example, the 2d convolutional layer is derived as a translation equivariant layer for signals on grids, where translations are the symmetries in the problem, and this is the advantage of *ConvNet* over *LinearNet*. However, *ConvNet* treats the signal as a two dimensional signal on a grid (i.e. channel size 2), so the symmetries exploited are the ones of a simple grid. Realizing that we have two distinct signals leads to the idea of treating both images separately and implementing *ParallelNet*. Finally, the last symmetry that we exploit is permutation of the two images. Indeed, permutation of the images should yield opposite prediction (unless the digits are equal). We can ensure this by a weight sharing between the processing of both images, leading to *SiameseNet* which is illustrated in 1.

Combining both ideas to fully exploit the prediction task's structure lead to the best performing models. Our improved networks are instances of a *FullNet*, which consists of a first block that predicts probabilities of digit classes, and a

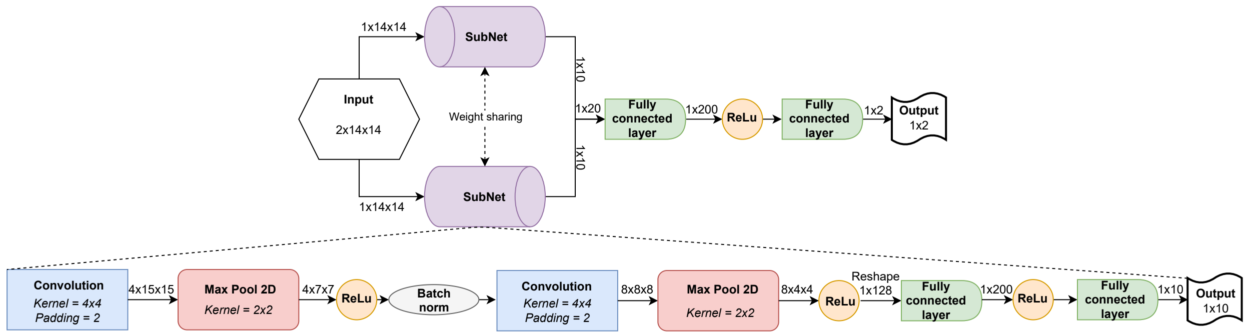

Figure 1: Scheme of the SiameseNet architecture

second block that computes the final output, as see in 1. ReLU is used as an activation function between layers unless specified otherwise. In increasing order of symmetry exploitation, we explored the three following networks:

- *NaiveNet*: There is only one subnetwork, which naively processes both images at the same time. This is just an extension of *AuxNet*.
- *ParallelNet*: There are two subnetworks, where each takes one image as an input and processes it. The images are thus processed separately and no weights are shared
- *SiameseNet*: The subnetwork takes one image as an input and processes it. The same subnetwork is used for both images, i.e. both images share the weights of the subnetwork.

Clearly, we expect *SiameseNet* to be the best architecture and will mainly utilize the other two options to demonstrate the advantage of the weight sharing.

Design choices for activation functions
We use ReLu activation functions as they were easier to train, with one exception: We use a softmax to process the class predictions of the subnetwork. Intuitively, the class predictions should be probabilities, so it appears natural to apply a softmax, coincidentally this is also the operation that the cross-entropy loss uses. We do not apply activation functions to an output before passing it to the (auxiliary) Cross Entropy Loss, as pytorch's Cross Entropy Loss applies softmax to the predictions, so a second activation function would be redundant.

3 Results and discussion

Clearly, there are many possibilities to vary network hyperparameters and other training parameters. In order to remain concise, we picked reasonable parameter choices after an initial trial phase and kept them identical for all model architectures. We train all our models for 25 epochs with a minibatch size of 50. Results are stated as means $\pm$ standard deviations, estimated over 25 runs with weights re-initialized according to the standard pytorch initialization. We use cross-entropy loss and stochastic gradient descent with a learning rate of 0.1 to train our networks. All our models take less than 5s per training run unless stated otherwise.

In Figure 2, we present the results for our different architectures. For the *LinearNet* , we managed to achieve a test error of $19 \pm 0.5\%$, which was largely independent of the number of hidden units. Further increase of the number of linear layers did not yield a noticeable improvement, while *ConvNet* yielded slightly lower test error. Unsurprisingly, we found that the largest influence on the test error reduction for *ConvNet* was the number of channels in the convolutional layer and we settled on 30 and 60 channels in the first and second layer respectively. For comparison, we also use a slightly smaller version of *ConvNet* that is identical to *AuxNet* in structure, but does not make use of the auxiliary loss.

The success of AuxNet (test error reduction of 5 %) clearly shows that the utilization of auxiliary losses can be a very effective method in cases, where either some intermediate result or additional constraints can be used. Our final *AuxNet*, uses $\alpha = 2$ as a weight of the auxiliary loss, which was optimized over a parameter search. Most notably for all networks discussed so far, we observe 0% Training error, which indicates strong overfitting. One possible explanation could be the Bias-Variance tradeoff, i.e. that we overparametrized our networks. However, while a reduction in size increased train error, it did not yield improvements in the test error.Thus, we did not manage to alleviate this problem by further modifying the parameters of our network. A possible explanation is that our training set does not include "noise" or mislabeled data, hence the observed low training error does not lead to reduced test accuracy.

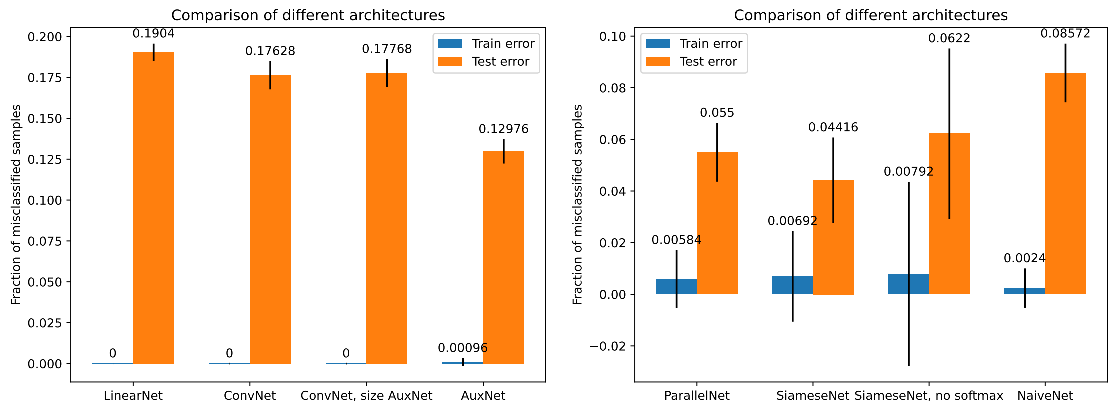

Figure 2: Comparison of the error of different networks

Of course, further improvements could potentially be achieved with more rigorous testing of additional parameters. Structural changes such as adding batch normalization (both before or after activation function), utilization of dropout (p=0.5) or the inclusion of a residual block to the *ConvNet* saw no visible improvement to the test error, so that we kept the relatively simple structure.
In the right plot in Figure 2, we present the results for the networks whose schema is shown in Figure 1. On first glance we observe an improvement in test error, as well as an increase in train error compared to the base line models. Starting with NaiveNet, it achieves a further 4% improvement in test error due to the additional linear layers and the use of the softmax after the class prediction. The hypothesis for these improvements is that softmax models probabilities, and thus synergises well with the cross entropy loss. Additionally, one linear layer in AuxNet might not have been enough to model the 'greater than' operator. As expected, treating both images separately, as done in ParallelNet, yields further improvement to the test error. Indeed, predicting the class of an image should not require knowledge of the other image. We can thus achieve high accuracy of guessing the correct digit class and use this information for the final comparison between the two digits. Finally, the SiameseNet, which takes advantage of weight sharing on top of the auxiliary loss gives the best prediction (4.4% test error) - despite only having half as many the parameters as ParallelNet or NaiveNet. In a sense, weight-sharing allows us to reduce the number of parameters as the same operations are done to both images and, hence, also potentially speeds up convergence. To test our hypothesis that you should use softmax to model class probabilities, we designed a SiameseNet which uses ReLu in the class prediction instead. The performance is indeed worse, in particular the standard deviation of the test error increase (2.5 %). A further study of auxiliary loss weight to find an optimal value was done and is presented in Figure 3 in the Appendix. The optimal weight for the auxiliary loss appears to be between 0.5 and 0.9. This is notably different than the optimal weight found for the AuxNet, showing that optimizing parameter for a specific network architecture is beneficial. Obviously, running an extensive hyperparameter study (e.g. number/size of hidden layers, kernel sizes...) could further improve our network. For this project, we decided to focus on exploring as many different architectures as possible instead.

4 Conclusion

In conclusion, we investigated multiple architectures of neural networks with an emphasis on utilization of weight-sharing and auxiliary losses in order to improve the performance of digit classification. Our best network achieved test accuracy of 4.4% within 25 epochs of training in $5s$. We saw that, for example, including auxiliary losses from guessing each individual digit before making the final guess of the comparison significantly improved the performance of the network. However the comparison between NaiveNet and AuxNet, both using the idea of auxiliary losses, shows that implementing a paradigm in a principled way can further improve results. Similarly, building a Siamese network that utilizes weight-sharing led to even larger improvements in our classification accuracy. For further work, we should also mention additional parameters that could have potentially been optimized to increase the accuracy even further, but were not explored in this report - different criterion and optimizer, learning rate, L1/L2 regularization. Another interesting aspect we would have liked to explore is the impact of weight sharing on convergence speed. In summary, we saw that great care (and sometimes luck) is needed in the optimization of the parameters of the neural networks in order to achieve optimal performance.

References

[1] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

[2] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

A Appendix

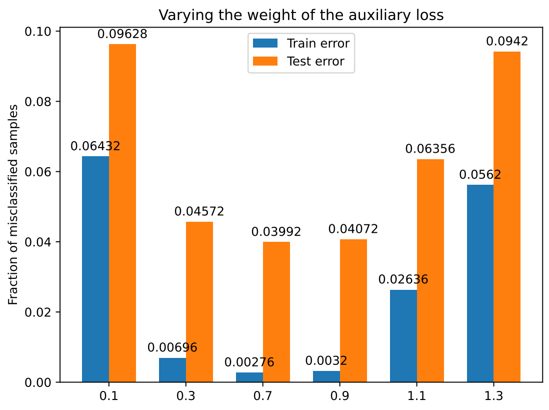

Figure 3: Parameter study for the weight of the auxiliary loss