UNIVERSITÉ DE GENÈVE
FACULTY OF SCIENCE

Report:
Deep learning final project (miniprojects)

Kevin Siebert*

Department of Informatics, Faculty of Science,
University of Geneva

Dated: December 17, 2022

Abstract

This is a report on the solution of the two mini projects proposed as a final project for the course Deep Learning (14X050). The solutions consists of this report and a GitHub archive containing the corresponding code (Github Archive).

Project 1

1.1 Introduction

The main goal of the first project is to compare the performance of basic neural network architectures. This is done with respect to a specific classification task where the network is supposed to predict the relationship (smaller or equal / larger) between two numbers of the MNIST dataset. A small selection from this dataset can be seen in fig. 1.

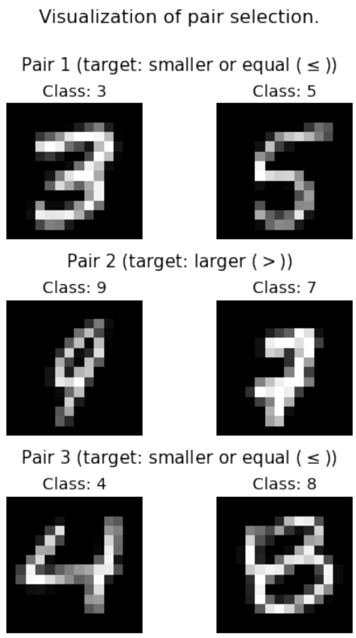


Figure 1: Small selection of inputs from the dataset where every row represents on input and the caption show the corresponding target classes.

*sKevin.Siebert@etu.unige.ch

1.2 Architectures

For the structure of the networks, four variable architectures where chosen. For all architectures there is one one version which starts with several convolutional and Batchnorm layers and is followed by linear layers, another which is fully convolutional and every architecture has the possibility to switch between several or one output neuron depending on whether one hot encoded labels are used.

The fully convolutional networks are created on the basis of the mixed networks at initialization through the function convolutionize which is based on the convolutionize function [**Fleuret2022**] defined in the lectures. They are not transformed after training but at initialization so they are trained as fully convolutional networks. From this point on not calling convolutionize will be the norm for project one. Whenever fully covolutional networks are used it will be explicitly specified.

One hot encoding is specified through the variable one_hot_encoding at initialization and is enabled for all additional functions by the same variable. From this point on one hot encoding will be the norm for project one. Whenever one hot encoding is turned of it will be explicitly specified.

All architectures use ReLU as their internal activation functions and Sigmoid as their output activation. All models are initialized with 64, 64 and 32 for the number of neurons in the main branch for the hidden Linear layers.

The first implemented architecture can be seen in fig. 2 it represents the "naive" approach where the images are passed through separate convolutional layers to extract their feature. The results are concatenated flattened and passed through fully connected linear layers/convolutional layers for the actual classification. This architecture will be called "simple network" (SN) from now on.

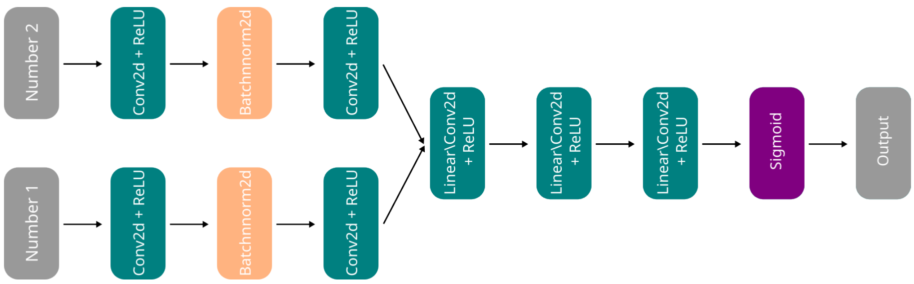


Figure 2: Diagrammatic visualization of the architecture for the simple network. (figure made using Inkscape)

Of course an architecture like fig. 2 wastes some learning potential by separating the feature extraction. Therefore an attempt on improvement is made through the architecture in fig. 3 from now on called "weight sharing network" (WSN). Here, we do the feature extraction by passing Number 1 as well as Number 2 (one after the other) threw the first two convolutional layers, then concatenate and continue.

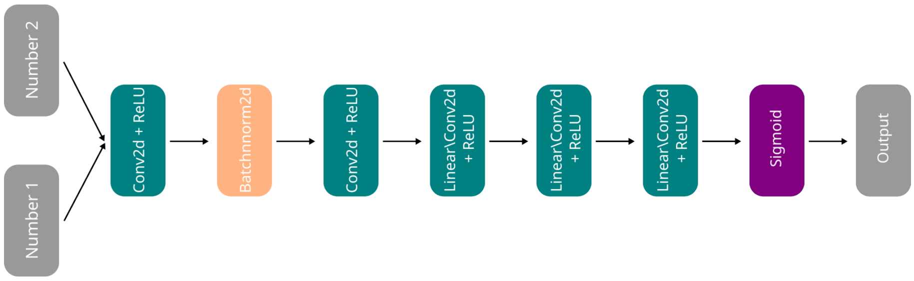


Figure 3: Diagrammatic visualization of the architecture for the weight sharing network.(figure made using Inkscape)

The following two architectures rely on the concept of auxiliary classifiers as described in [**Szegedy2014**]. The basic idea is to branch of the main architecture at some point to do the same or another classification task from the input. This part is also trained and the loss is added (usually multiplied by a some fraction smaller than one) to the loss of the main branch. This part is usually discarded when evaluating the model.

Figure 4 takes the most simple approach to this concept by branching of and trying to perform the same classification as the main branch and will therefore be called "Simple auxiliary classifier model" (SAUX) from now on.

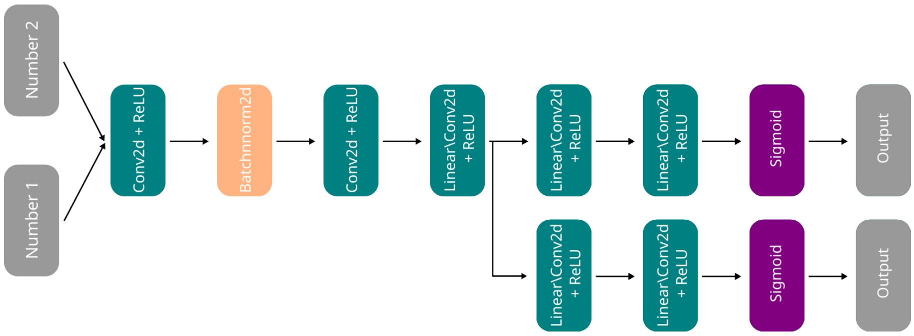


Figure 4: Diagrammatic visualization of the architecture for the simple auxiliary neural network.(figure made using Inkscape)

Another approach could be to use the information about the classes of the respective numbers from the MNIST. Figure 5 introduces this concept by having an auxiliary classifier branch just after the initial feature extraction to find the classes of the respective numbers. This architecture will be called "auxiliary classifier using classes" (CAUX) from now on.

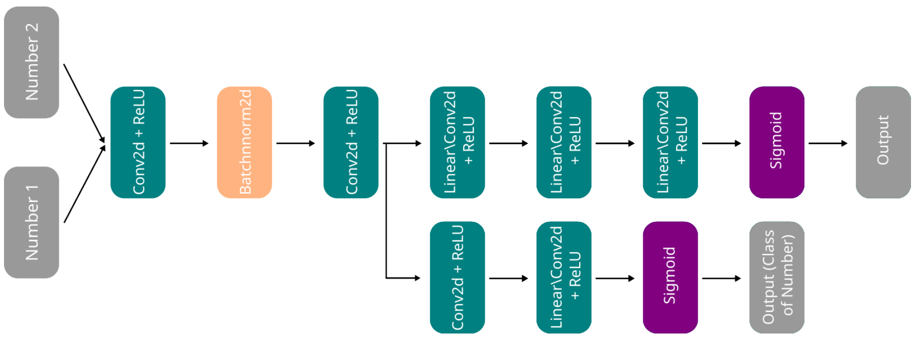


Figure 5: Diagrammatic visualization of the architecture for the auxiliary neural network using the numbers classes. (figure made using Inkscape)

For both auxiliary architectures the modification specified at the start of this section are also imposed on the auxiliary classifiers. Both architectures keep the structure of the WSN for the initial feature extraction.

1.3 Training

All the Architectures were trained for 100 epochs, which was repeated for 150 times while logging the test accuracy to calculate the mean values and standard deviation for the results of that experiment. The actual Training and testing is performed by the Teacher class which stores the train and test data and whose methods are the actual train and test function of the model. The whole process for the analysis is performed by the run_analysis function which returns the name of the model input, the mean and the standard deviation for use in further analysis or visualization. All models with one hot encoding where trained using the CrossEntropyLoss optimizer and all the models without one hot encoding where trained using the MSELoss optimizer.

All models where trained with a learning rate of 0.01 and a batch size of 50. These parameters remained fixed for the whole analysis in project one to keep the results as comparable as possible.

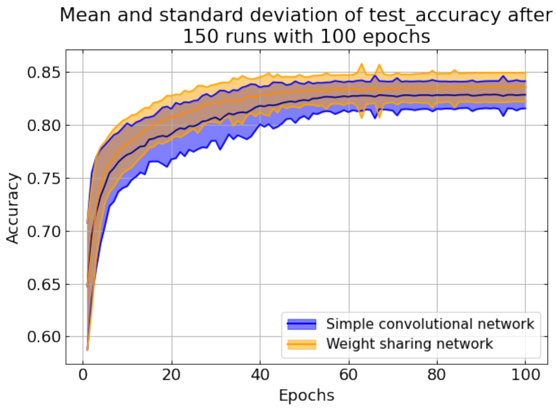
1.4 Results

As a starting point the different architectures are compared in their standard forms (one hot encoding and not fully convolutional) as specified in section 1.2. The results of this comparison can be seen in fig. 6. Figure 6a shows the WSN performing better on average than the SN. not only does it reach a higher value at the end of the training, it also as a steeper slope for the evolution of the test accuracy. This development is consistent with the ideas behind the WSN eplained in section 1.2.

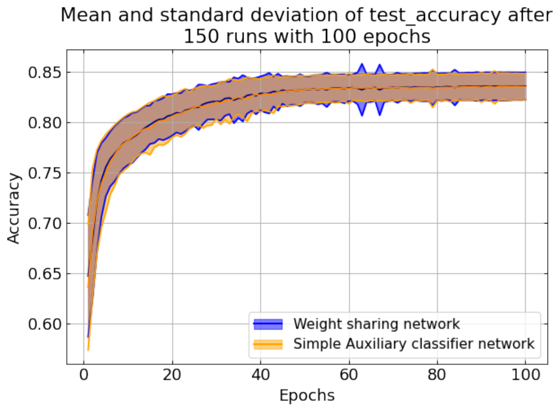
The WSN is then compared to the SAUX with the result of them having nearly indistinguishable performance in the comparison fig. 6b.

Figure 6c shows the comparison between SAUX and CAUX, where SAUX seems to perform better overall with both architectures starting out very similar but the SAUX plateaus less agressivly. This result is unexpected as one might guess the introduction of additional informattion might have let to a boost in performance.

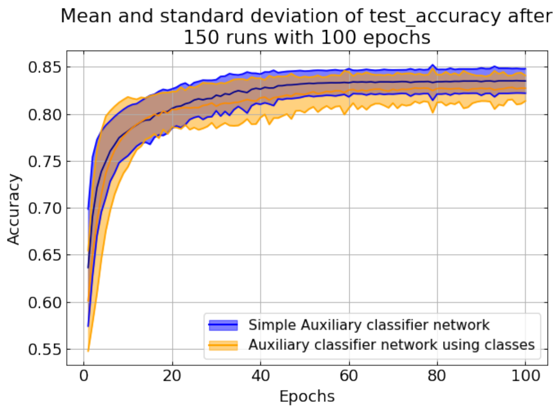
Project 2



(a) Simple network

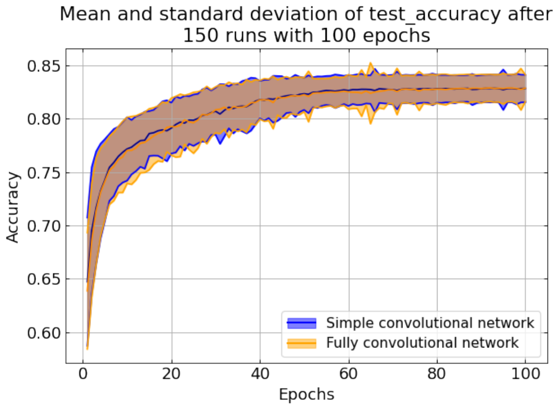


(b) Weight sharing network

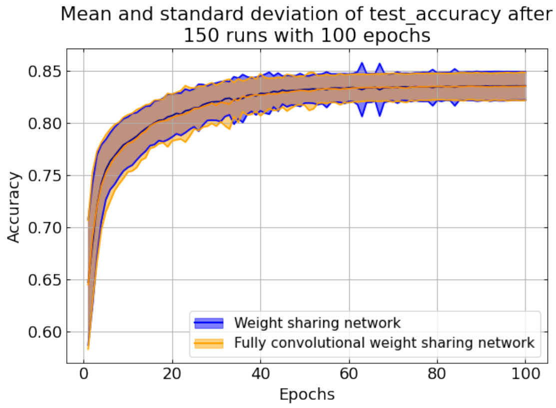


(c) Simple auxiliary classifier network

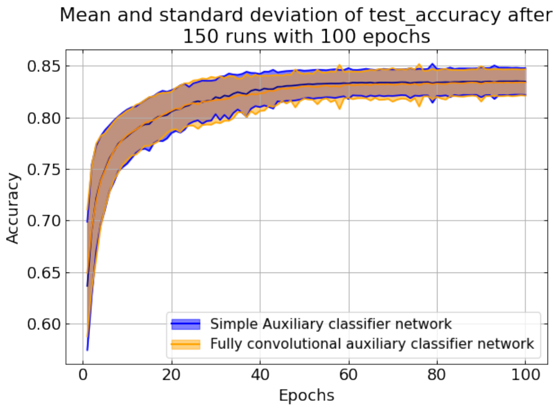
Figure 6: The mean and standard deviation of the test accuracy compared over the course of 100 epochs calculated for 150 trials. For the architectures in their standard form.



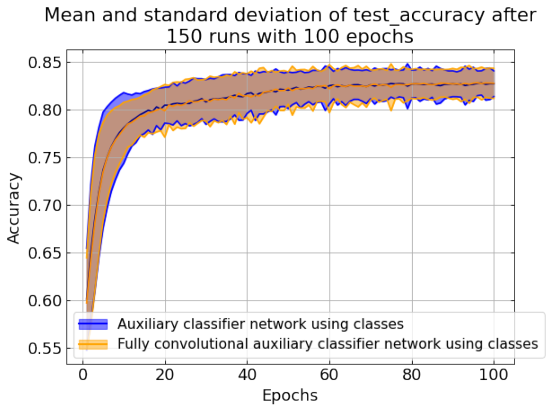
(a) Simple network



(b) Weight sharing network

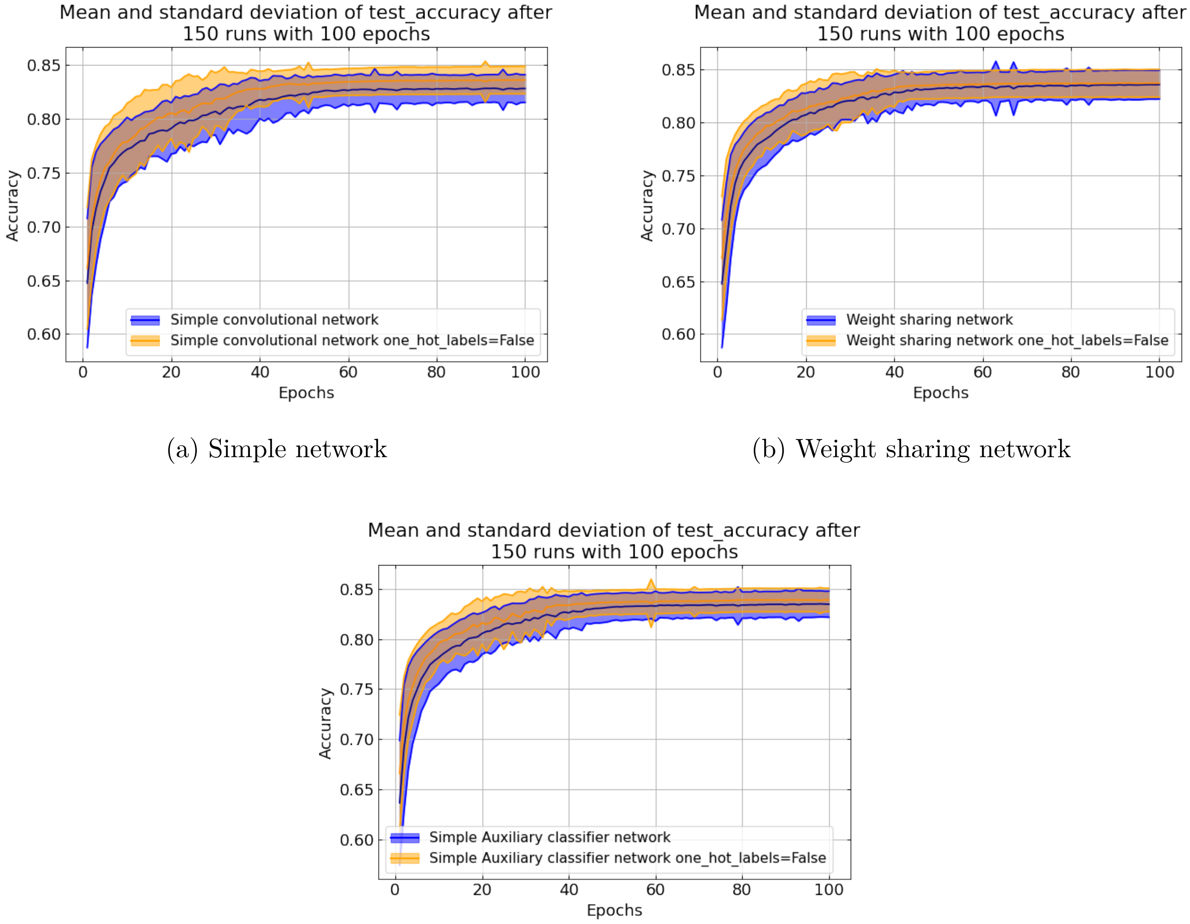


(c) Simple auxiliary classifier network



(d) Auxiliary classifier network (number classes)

Figure 7: The average and standard deviation of critical parameters: Region R4

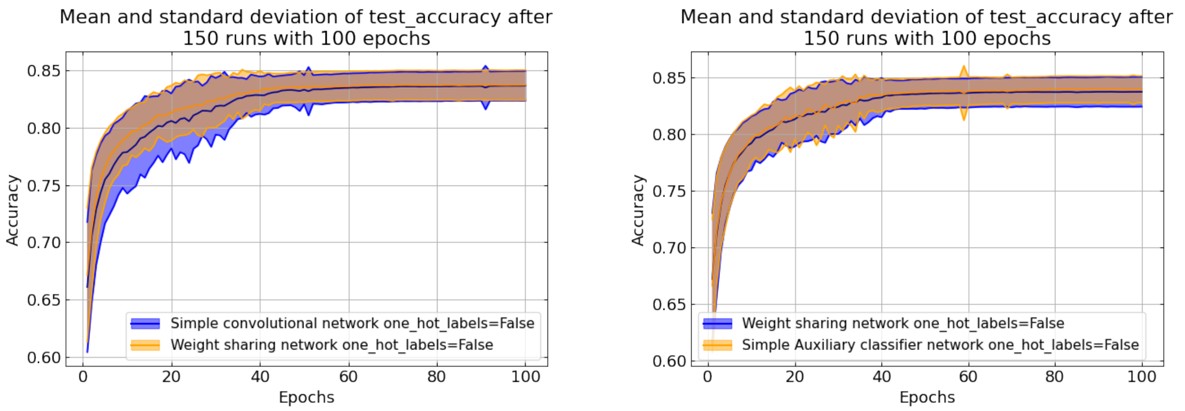


(a) Simple network

(b) Weight sharing network

(c) Simple auxiliary classifier network

Figure 8: The average and standard deviation of critical parameters: Region R4



(a) Simple network

(b) Weight sharing network

Figure 9: The average and standard deviation of critical parameters: Region R4