Project 1 - Deep Learning EE559

1 Introduction

This projects consists on trying different architectures for comparison of two hand-written digits; both with and without weight sharing and auxiliary loss implementations. The aim here is to understand the advantages provided by these two techniques for the model performance.

To do so, we implemented several architectures with roughly $7 \cdot 10^4$ parameters and an auxiliary output. We then tried several auxiliary factors (including zero) to see whether using auxiliary loss improved the performances or not. We also assessed whether or not sharing the weights of the convolutions between the two input pictures results in better accuracies.

2 Models

The models aim at identifying whether one input digit is greater or lower than the other one. It is thus a classification problem with two classes. The output layer of the network consists of two units, each indicating the confidence of the network that the input belongs to one of the classes. We used an activation function after every layer, which was always *ReLu*, except for the last activation function before the main and the auxiliary outputs, which was a *Softmax*. In the names of the models, the number following the letter 'C' refers to the number of convolutional layers. The number that follows letter 'L' refers to the number of linear layers after the convolutions.

- **C1L2**: Uses a convolution with kernel size 3 that identifies 72 features. It then uses a max pooling layer with kernel and stride of 3, followed by a linear layer that turns its input of size $2 \cdot 4^2 \cdot 72 = 2304$ into an output of size 20 and another that gets them down to our two outputs. This is in a sense the simplest convolutional network that one could implement with the constraints of having $7 \cdot 10^4$ parameter, a convolutional layer and a max pooling, and a second to last layer of size 20 for our auxiliary loss purpose.
- **C2L2**: Uses now two convolutional layers with kernel size 3 that identify 22, then 110 features; each coupled with a max pooling of kernel and stride 2. Two linear layers were then added to turn our input from $2 \cdot 2^2 \cdot 110 = 880$ to 20 and then from 2 to 2 outputs.
- **C2L3**: Has two convolutional layers of kernel size 3 that identify 32 and then 64 features, that are separated by max pooling layers of kernel size 2 and stride 2. The main difference lies in the additional linear layer added. This network thus has a linear layer that goes from size $2 \cdot 2^2 \cdot 64 = 512$ to 60 after the convolutional and max pooling part; and another one that turns the 60 inputs into 20, and a final one that turns them into 2.
- **C3L2**: Contains three convolutional layers with kernel size 3 that identify 12, then 24, and then 48 features, as well as two linear layers that go from $2 \cdot 4^2 \cdot 48 = 1536$ to 20 units, then from 20 to 2 units. A max pooling layer of kernel and stride size 2 was added only after the third convolutional layer.
- **C4L2**: Has four convolutional layers with kernel size 3 for the first two, and size 2 for the rest. They identify 9, then 18, 36 and finally 72 features. They are followed by two linear layers that reduce their input from $2 \cdot 3^2 \cdot 72 = 1296$ to 20, and then from 20 to 2 output units, similarly to the other models. Max pooling of kernel size and stride 2 is present only after the second convolutional layer.

	Models				
Settings	C1L2	C2L2	C2L3	C3L2	C4L2
Baseline	70 642	70 702	74 814	72 466	68 230
Weight Sharing	58 392	44 182	53 428	51 624	34 146

Table 1: Number of parameters in each model

2.1 Auxiliary Loss

Auxiliary losses try to help the network with training by acting as a regularizer. At some point, the network lets us extract an alternative output. The goal of such alternative output is to classify the two digits rather than compare them. It associates an output to each digit, thus, this auxiliary output is of shape $[batch, 2, 10]$, and contains the predictions made by the network. It uses the actual value of the digits in the pictures.

From them, we can create an auxiliary loss that represents how correctly the models manage to predict each number. This loss is computed using the auxiliary output obtained through a linear layer and a softmax right after the last convolutional layer, then multiplied with a pre-factor and added to the main loss. That way, the model is more penalised when it predicts the digits wrong and gets feedback about that, which should lead to better overall accuracy for the number comparison task. We have used three different pre-factors for the auxiliary part of the loss: 0, 0.1 and 1. Using a factor of 0.1 seems to be the best solution since it provides an auxiliary loss that is in the same order of magnitude as the main one, only slightly smaller.

2.2 Weight sharing

The aim of weight sharing is to use the same weights for both input pictures, in the convolutional part of each model. Indeed, since we want to extract the same information from the two pictures in the first part of the processing, it seems logical to perform the same computations on both of them. The models using weight sharing implementations are characterized by a 'WS' suffix in the code. Sharing weights avoids double tuning of parameter for a same task and should thus accelerate training and improve convergence. Weight sharing has been implemented by performing convolutions on both pictures using the same convolutional module, which has half as many output channels as the same convolutional layer in the corresponding model without weight sharing. Notice that in the models which implement weight sharing, this also applies to computations for the auxiliary outputs, whenever they're activated.

2.3 Training

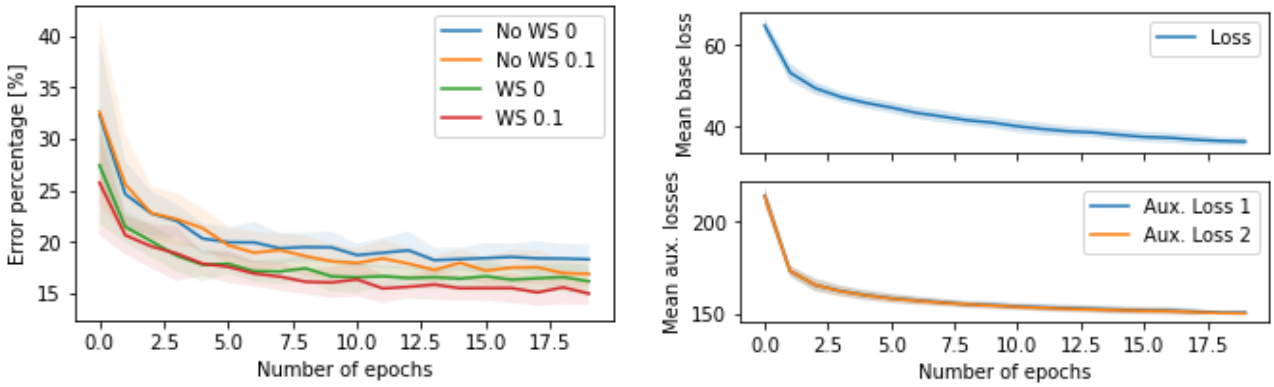
All models were trained using Adam algorithm, a learning rate of 10^{-3}, mini batches of size 10, and our results were averaged over 20 runs of 20 training epochs each. The data was shuffled before each epoch, and checkpoints saved after each epoch for each run.

3 Results

Performance of each model have been evaluated under different conditions. Indeed, they have been tested with and without weight sharing or auxiliary loss, with both, or none. Note that auxiliary loss have been tried every time with factors 0.1 and 1, corresponding to its weight within the total loss. It represents the strength of the regularisation. Results are presented in Table 2.

	Models				
Settings	C1L2	C2L2	C2L3	C3L2	C4L2
Baseline	19.2±5.54	16.8±0.85	18.4±2.26	17.9±1.48	18.3±1.42
Auxiliary 0.1	18.1±1.29	16.7±1.56	17.9±2.19	16.9±1.06	16.8±1.26
Auxiliary 1	19.3±2.13	16.5±1.51	15.9±1.56	17.3±1.83	15.6±1.62
Weight Sharing only	17.9±1.12	16.6±7.15	16.3±1.57	17.0±2.01	16.1±1.53
Auxiliary 0.1 and Weight Sharing	19.7±6.02	14.9±1.20	15.4±1.49	15.5±1.43	14.9±1.11
Auxiliary 1 and Weight Sharing	20.7±5.71	16.0±1.09	15.2±1.60	16.2±1.54	14.4±1.58

Table 2: Models' average error [%] performance on 20 runs of 20 epochs each. (See *Training* section for additional details.)



(a) **(b)**

Figure 1

(a) Mean error (solid lines) and variance (shaded region) for the C4L2 model. We tried the model with and without weight sharing (respectively, 'WS' and 'NO WS' in the legend) and with auxiliary losses either deactivated or with a 0.1 pre-factor (respectively, '0' and '0.1' in the legend).

(b) Decrease of the mean loss and auxiliary loss over 20 different runs when training model C4L2 with weight sharing and a factor of 0.1 for the auxiliary loss. Note that the auxiliary loss isn't multiplied by 0.1 in the plot. The shaded area is small, indicating small standard deviations.

4 Discussion

In general, error converges around 17% for all baseline models, as seen in Table 2. C4L2 is the best architecture here but overall, we get similar performances for all models with more than one convolutional layer. C4L2 seems to be sensitive to weight sharing and auxiliary loss, compared to C1L2, whose performance vary a lot. This result is expected for weight sharing since it is only applied to convolutional layers and C4L2 contains more of them. Auxiliary loss effect does not seem to correlate with the structure of the model. Note however that this may be due to the fact that we always extract output at the same point of the network, so its structure does not really influence the auxiliary output. It may be interesting to further explore this idea and try to put the auxiliary network at different locations. Error as a function of epochs are shown for model C4L2 in Fig.1. By fixing the model choice, we can observe an improvement of accuracy with auxiliary loss (both with and without weight sharing). Note however that the effect is smaller than expected. Similar conclusions can be drawn for weight sharing. A faster convergence was expected in addition to that, which cannot be observed here. This may be due to the fact that computations for the two inputs are not done in parallel, and thus, times are added together. This may be a point for further improvements.

5 Conclusions

Overall, weight sharing and auxiliary loss slightly improved accuracy of models, independently of their structure. However, we are convinced that effects of these implementations could be increased with further researches. As mentioned before, parallelization of computations for the inputs when using weight sharing, different architectures, or different auxiliary outputs may contribute to such progress.

Note that accuracy may also be improved by using processes such as batch normalization or data augmentation. Deeper network may not induce big improvements as we have seen that adding convolutional layers does not greatly reduce the error. Another idea would be to change the way auxiliary loss was implemented. This could include adding layers, extracting value at another point of the main network or using a different shape of output.

To conclude, weight sharing and auxiliary loss are functional and promising techniques. We showed that they actually have potential for improving both the run time and performance of a neural architectures. However, some aspects of our implementations can still be improved in order to enhance their effects and benefit from them.