Project 1 - Classification, weight sharing, auxiliary losses

Maxence Jouve, Paul Griesser, Valérian Rey
EPF Lausanne, Switzerland

***Abstract*—Deep Learning methods are now widely used for Computer Vision related tasks. The goal of this project was to familiarize us with techniques used for images classification. Given two 14 x 14 digits images obtained from the MNIST dataset (down-sampled using average pooling with a kernel of size 2), we aimed to compare the digits and output which one is bigger. We had access to a boolean target (0 or 1) telling us which digit is bigger and also to the class of each digit (an integer between 0 and 9). We will first present the general pipeline we used to train and evaluate our models, we will then look at the first class of models we tried that did not take into consideration the fact we had two independent images, and finally, we will see our second class of models that treated each image independently.**

I. General Pipeline

The following pipeline was used for training and evaluating all our models.

A. Initial data processing

We always generated 1000 pairs of digit images with the target and their respective classes for the training and the testing sets. Moreover, we used our method $normalize\_data$ (in the $utils.py$ file) on our data. This method computes the $mean$ and the $standard\ deviation$ of the images in one set. It then subtracts the $mean$ and divides by the $standard\ deviation$ to obtain images with $mean = 0$ and $standard\ deviation = 1$.

B. Training the model

To train our models, we always used $batch\ size = 64$. We realized it offered good performance and training stability. Moreover, we used the PyTorch SGD optimizer. We tried different optimizer parameters, however, we ended up using the following ones for all our models as they always ensured convergence of the loss:

- starting $learning\ rate = 0.1$.
- $momentum = 0.9$.
- $gamma = 0.97$. This parameter was used to decrease the learning rate. After each epoch, the learning rate was multiplied by $gamma$. This ensures to start with a big enough $learning\ rate$ to avoid converging to local minima but the more we train, the more the $learning\ rate$ gets closer to 0 in order not to overshoot good minima.

Finally, we used the $CrossEntropyLoss$ as our loss function.

C. Optimize hyperparameters

Some of our models have external parameters that should be optimized. For example, we use two types of auxiliary losses. The auxiliary loss is computed and then added to the primary loss. In this case, we have: $global\ loss = primary\ loss + \alpha * auxiliary\ loss$. This parameter α must be optimized properly. To achieve this, we used cross-validation with K folds. For each possible α value, we train the model K times on K-1 different fold and evaluate it on the remaining 1 fold. Thus for each α we obtain a good estimate of the accuracy of our model and can choose the best parameter.

D. Model evaluation

To evaluate our models, we generated at least 10 different training and test sets (each corresponding to a different round). Then we trained the model using the best parameters found earlier and evaluated the accuracy. Finally, we averaged the accuracies obtained to obtain a mean accuracy as well as the accuracy standard deviation for each model.

II. First class of models

The first class of models we tried can be considered as naive. Indeed, we did not take advantage of the structure of the input (two independent images) and considered having an input with 2 channels. We tried to increase the depth and also add an auxiliary loss similar to the one in GoogLeNet [1]. The later is obtained by applying the loss function on an intermediary result that is trained with the same target as the final output.

A. Baseline model

The baseline model has 2 convolutional layers. We tried a different number of output channels, different numbers of hidden nodes in the fully connected layer, and different activation functions but we kept the following model for achieving the best results:

Conv2d(2, 32, 3, 1) → ReLU → Conv2d(32, 64, 3, 1) → ReLU → max_pool2d(kernel_size = 2) → Linear(1600, 128) → ReLU → Linear(128, 2)

B. Models with 3 convolutional layers

Then we tried to increase the depth of our network by adding one convolutional layer. Moreover, as the network gets deeper, we tried to add an auxiliary loss to improve the training.

1) Without auxiliary loss:

After trying different output channels and hidden nodes, we obtained the following architecture:

Conv2d(2, 32, 3, 1) → ReLU → Conv2d(32, 64, 3, 1) → ReLU * → Conv2d(64, 128, 3, 1) → ReLU → max_pool2d(kernel_size = 2) → Linear(2048, 128) → ReLU → Linear(128, 2) → softmax

2) With auxiliary loss:

This time the model has an auxiliary loss. After the second convolutional layer noted with * in the previous paragraph, we added the following fully connected part: max_pool2d(kernel_size = 2) → Linear(1600, 128) → ReLu → Linear(128, 2) → softmax

The architecture of this model can be seen in figure 1.

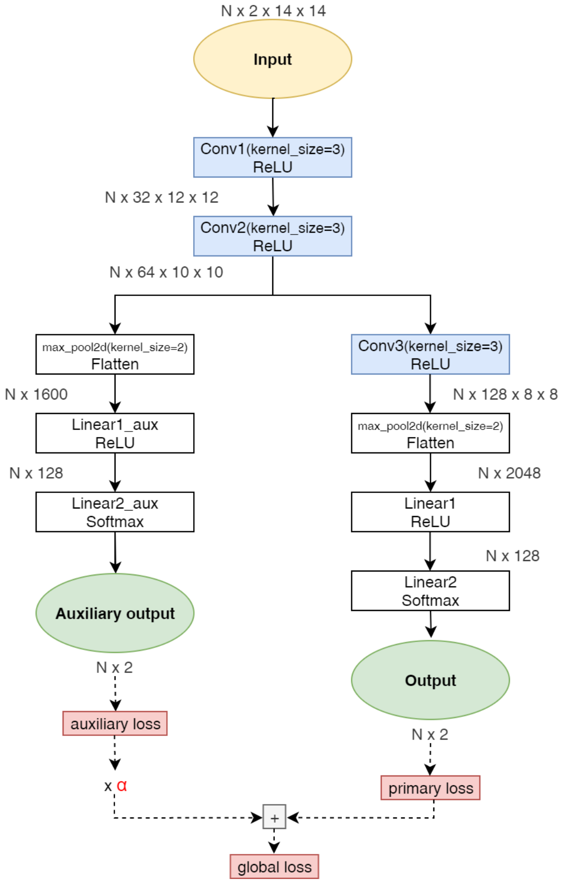

Figure 1. Network with an auxiliary loss similar to the one in GoogLeNet.

We thus obtained an auxiliary loss that is added to the primary loss with a factor $\alpha = 0.5$ (optimized with cross-validation). This auxiliary loss allows the training to be more stable as the first two convolutional layers are now less affected by problem encountered when training deeper networks such as vanishing gradient.

C. Model with 4 convolutional layers

We then further increased the network by adding another convolutional layer.

1) Without auxiliary loss:

We obtained the following best network:

Conv2d(2, 32, 3, 1) → ReLU → Conv2d(32, 32, 3, 1) → ReLU ** → Conv2d(32, 64, 3, 1) → ReLU → Conv2d(64, 128, 3, 1) → ReLU max_pool2d(kernel_size = 2) → Linear(1152, 128) → ReLU → Linear(128, 2) → softmax

2) With auxiliary loss:

This time we added the following fully connected component after the ** in the previous paragraph:

max_pool2d(kernel_size = 2) → Linear(800, 128) → ReLU → Linear(128, 2) → softmax

For this we have $\alpha = 0.9$.

D. Results

Model	Mean test accuracy	Accuracy std
Baseline network	81.64	0.8871
3 conv. layers	81.89	1.3781
3 conv. layers + auxiliary loss	82.38	0.7033
4 conv. layers	81.33	1.8208
4 conv. layers + auxiliary loss	81.83	1.0411

Table I
MODELS AND THEIR ACCURACY PERFORMANCE

The experiences, as well as training plots, can be seen in the notebook $Experiences\_Part\_1.ipynb$.

III. SECOND CLASS OF MODELS

Since the 2 input channels represent the same thing (14x14 digit images), we decided to use a Siamese network to make them go through the same layers. We then use a single fully connected layer to make the comparison of the digits. We also use the digit labels to improve the training, by using an auxiliary loss at the end of the Siamese network, at the place where the digits are predicted (thus making a straightforward use of the digit labels). We added this loss, multiplied by a factor α, to the final loss computed at the final output of the network.

A. Testing different values for α

We used the same scheme as defined in section 1.C. The final test accuracies for each value of alpha is summarized in the next table. Note that these accuracies are obtained when training only on 80% of the training set (and testing on the remaining 20%) and are thus not as good as during our final model evaluation. Note that for $\alpha = 0$, it corresponds to a Siamese network that did not use any auxiliary losses. We thus realized the latter improves performances. As a result, all our remaining tests have auxiliary losses with $\alpha = 0.4$ because for higher values the learning would sometimes fail to converge.

α	0	0.1	0.2	0.3	0.35	0.4
Final accuracy	79.0	90.7	92.9	91.6	94.1	95.1

Table II
TESTING DIFFERENT PARAMETERS

B. *Testing different learning schemes*

We tried different ways of learning the whole network by setting the $requires\_grad$ value of some parameters to $False$ so that they are not affected by the gradient descent.

1) Train digit prediction (the Siamese network) only, then digit comparison (the last fully connected layer) only. The digit prediction accuracy was quite good on the test set (91.6%) and the final test accuracy is already very good (94.5%).
2) Train digit prediction only, then train everything together. This worked a little bit better than the first scheme, giving a little bit better test digit accuracy (92.3%) and improving the final test accuracy by a little (94.8%).
3) Train everything together since the beginning. With this scheme, we had an even better accuracy than with the second one (95.6%). In the rest of the section, we'll only talk about this scheme.

C. *Trying different depths*

We tried to use 2, 3, and 4 convolutional layers in the Siamese network. We had very similar final accuracies between these experiments (close to 95%). We decided to avoid using a too deep network since the original image is only 14 x 14 and each convolutional layer reduces the size of the images. From the plots available in the notebook $Experiences\_Part2.ipynb$ (all of the experiences of this section can be found there), we can see that the deeper the network is, the longer it takes to converge (from about 15 epochs for the one with 2 convolutional layers to about 25 epochs for the one with 4 convolutional layers). We decided to use the one with 3 convolutional layers.

D. *Final Testing*

Since the variance was still quite high, we made the final test on 25 different trainings of the network, with more epochs (100) and with a gamma closer to one than before (0.98). The average final test accuracy obtained was 95.42% and its standard deviation over the 25 runs was 0.9284.

IV. DISCUSSIONS & CONCLUSION

The first observation that can be made from our experiments is that taking advantage to the structure of the data can lead to significant change. Indeed the first classes of model performed poorly compared the the second class of models that considered the input as two independent images rather than one input having 2 channels.

Furthermore, we learnt that using auxiliary losses has several benefits. On the one hand, auxiliary losses make the training better and more stable as the information does not need to flow from the final layer of our model. It can be seen in the results from the first class of models, where it increased the mean test accuracy and lowered its standard deviation. However, the main drawback is that the resulting network has more parameters and needs more computation resources to be trained. On the other hand, we realized that adding an auxiliary loss trained using digit classes improved performance by a lot, because it makes use of an additional source of ground truth information to help the training. For a more complicated task we could have combined the two different types of auxiliary losses, in order to both fully utilize the information at our disposal and to allow a deeper architecture.

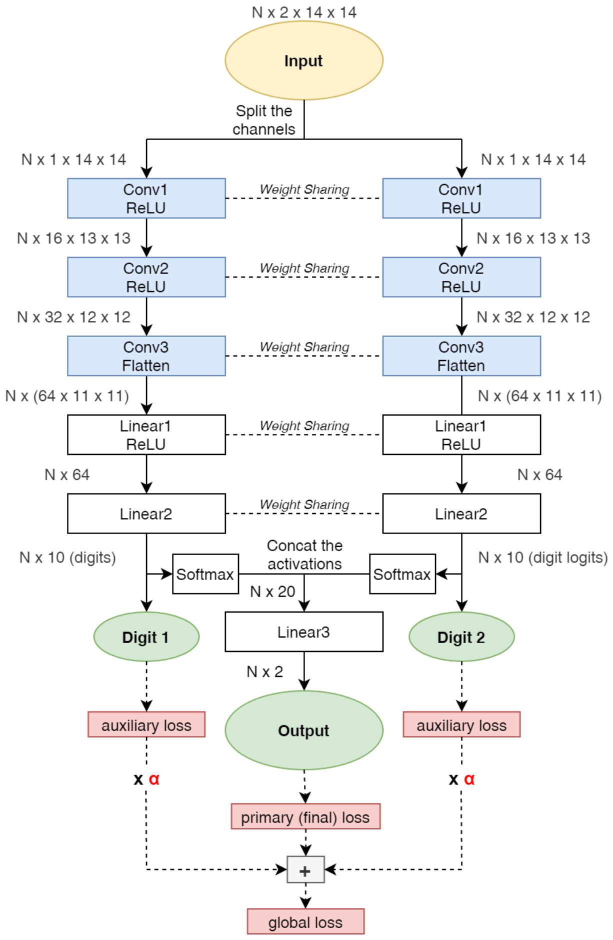

Figure 2. Siamese Network with auxiliary loss

REFERENCES

[1] Y. J. P. S. S. R. D. A. D. E. V. V. A. R. Christian Szegedy, Wei Liu, "Going deeper with convolutions," 2014.