EE-559: Deep Learning - Project 1

Riccardo Cadei
riccardo.cadei@epfl.ch
SCIPER: 321957

Riccardo Fradiani
riccardo.fradiani@epfl.ch
SCIPER: 322979

Niccolò Polvani
niccolo.polvani@epfl.ch
SCIPER: 321215

***Abstract*—In this study we compare different Deep Neural Networks to predict the inequality among 2 gray-scale images representing handwritten digits from MNIST database. Advantages of weight-sharing and auxiliary loss are also discussed. By training the models on a training set of 1,000 couples of images we got a test error rate equal to 2.96%.**

I. Introduction

This project's aim is to evaluate the performance of different Deep Neural Networks predicting the inequality among 2 gray-scale images representing handwritten digits from MNIST database. In particular, we want to study the impact on this task of weight sharing and the use of auxiliary losses. With "weight sharing" we refer to the practice of allowing the same parameters to modulate different parts of the processing in a given architecture. This technique significantly reduces the number of parameters in a model, possibly leading to a faster convergence of the loss function but also penalizing the overall flexibility of the architecture. On the other side, with "auxiliary loss" we refer to GoogLeNet paper [1] by Szegedy et al. Its purpose is to tackle the issue of vanishing gradient and it does so by attaching a small network to the output of some of the module layers, producing early classification predictions which are then added to the final loss with a certain weight. In particular, this trick can help to accelerate the training during the early cycles when weights are randomly initialized.

II. Dataset and Problem

The dataset consists in pairs of 14 x 14 pixel images in grayscale (unique channel) taken from the MNIST dataset [2]. For each pair we have 2 information: the *target* and the *classes*. The *target* is a binary label equal to 1 if the digit represented by the first image is less than or equal to the second one; and the *classes* are the values of the digits themselves. The goal is to create a model to correctly predict the *target* of a pair of images. The training set size and the test set size are equal to 1,000 pairs each.

A. Data Augmentation

A first idea that we considered to help the training, in particular for the Deeper networks, was Data Augmentation. Using an on-the-fly algorithm and taking advantage of the knowledge of the *classes* in the training samples, we generated new pairs of images from the existing ones. In particular, every time that we wanted to extract a pair from the training set, we rather considered 2 images among the $1,000 \cdot 2$ available and we computed their $target$ inequality from their $classes$. This allowed us to increase the training size by 4,000 times from 1,000 to $2,000^2 (= 4,000,000)$ without any additional storing usage, giving us the possibility to train even more complex and deeper architectures.

III. Models

We defined and compared different architectures to address this problem. In particular, we started with the 4 architectures described in Section III-A to model directly the *target* inequality without considering the *classes*. So we evaluated pro and contra of these models and we proposed a variant of the most promising one using *Weight Sharing*.

A. Architectures

1) `MLP`: A Multi-Layer Perceptron with 4 hidden layers with the following number of nodes: 350 / 250 / 200 / 20 using ReLU as activation function after each hidden layer. Its total number of trainable parameters is 279,562.

2) `ConvNet`: A Convolutional Neural Network with 4 layers: 2 convoloutional and 2 linearly fully connected. The convolutional layers are structured as follow:

Input→Convolution(*kernel size, padding*)→Batch Normalizaion →ReLu→Max Pooling(2x2)→Output

with *kernel size* respectively equal to 5 and 3, and *padding* equal to 3. The first fully connected layer is preceded by Batch Normalization and followed by ReLU activation function, while the last layer is just preceded by Dropout. The total number of trainable parameters is 77,530.

3) `DeepConvNet`: A Convolutional Neural Network structured in 10 blocks, followed by Average Pooling(14x14), DropOut and a final Fully Connected Layer. Each block is equal to the second convolutional layer present in `ConvNet` but without Max Pooling. The total number of trainable parameters is 2,700,834.

4) `ResNet`: A state-of-the-art architecture used as a baseline for the other models. It is a Residual Neural Network structured in 10 blocks where each residual block is composed by 3 convolutional layers, using Batch Normalization and ReLu as the activation function. The structure of a block is represented in Figure 1. The series of blocks, as for the `DeepConvNet`, is then followed by an Average Pooling, Dropout and a Fully Connected layer. The total number of trainable parameters is 2,746,626.

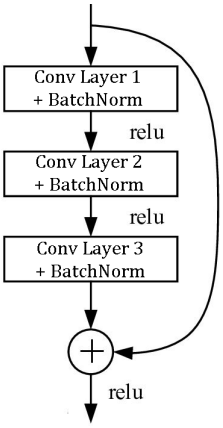

Figure 1: Structure of a Residual Block in the ResNet

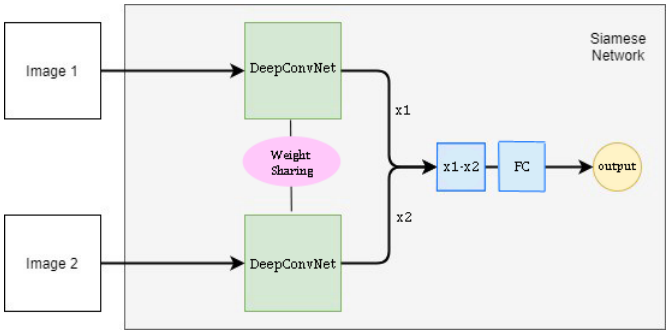

Figure 2: Structure of the Siamese network

B. Weight Sharing

After building the architectures presented in Section III-A, we then proceeded to train them and test their performance. Given the results shown in Section V, `DeepConvNet` seemed to be the most promising model so we proposed a variant of the latter using weight sharing. In particular, we implemented its corresponding `Siamese` network. This consists in a duplication of `DeepConvNet`, with each duplicate taking as input one image from the couple given as input. These two networks process the two one-channel images individually but sharing all the parameters. The two outputs of the networks are then subtracted and passed through a Fully Connected layer to obtain a single final output which is used for classification. The structure of our `Siamese` network is represented in Figure 2.

IV. Training

We trained each model using 90% of the training set and we used the remaining 10% as validation. In particular, we optimized the Binary Cross Entropy Loss (defined in Section IV-A) using Adam optimizer (described in Section IV-B) updating the parameters using batches of 50 samples. Finally we saved the model parameters which generated the lowest validation error during the training.

A. Loss function

We decided to train the models minimizing the empirical Binary Cross Entropy Loss, defined as:

$$L = -\sum_{i=1}^{2} t_i log(p_i)$$

where t_i and p_i are (respectively) the *target* value and the network output for class *i*.

- *Auxiliary Loss:* Taking advantage of the knowledge of the *classes*, we also implemented an auxiliary loss (as described in Section I) for the deeper architectures (i.e. `DeepConvNet` and `Siamese`), analyzing if it could lead to any performance improvement. In particular, we modified the `DeepConvNet` by adding an auxiliary block every 5 convolutional layers. Each of these blocks is composed by Average Pooling (which reduces the images dimension from 14 x 14 to 7 x 7) and a Fully Connected layer, which outputs a classification for both the *classes* of the input couple of images (i.e. $2 \cdot 10$ *classes*). The Cross Entropy Loss between these outputs and the real *classes* is then computed. A new final loss is then computed combining all the auxiliary losses (Cross Entropy Loss) and the loss on the final binary prediction (Binary Cross Entropy Loss) using by default the weights 0.6 (0.3 for each of the two *class* prediction) and 0.4, obtained using Grid Search.
For the `Siamese`, a further Auxiliary Loss was considered at the end of the two individual branches of the network.

B. Optimizer

We optimized the Binary Cross Entropy Loss (and its variant using the Auxiliary Loss) using Adam optimizer [3]. In particular, we set by default the exponential decay rate for the first and second moment estimates to 0.9 and 0.999, we set $\epsilon = 1e{-}8$ and we implement some hyper parameter tuning on the *learning rate* and *learning rate decay* for each specific model.

V. Results

The results of all the experiments described in the previous sections are reported in Table I and II. In Table I the models are described by rows, and the performances reported in the last three columns refer to the mean error rate ($\pm$ standard deviation) among 10 experiments (without data augmentation and trained for 25 epochs each). In the first section of the Table (first 4 rows) are reported the 4 architectures implemented without any improvement. The best performances on the test set are obtained by the `DeepConvNet` (error rate $\approx 21\%$), however they are not significantly better than the simpler `MLP` and `ConvNet` models . It is also interesting to observe that even more complex model like `ResNet` is not able to properly learn the task. It is evident that all the models are suffering from overfitting and some adjustments need to be made. The second section of the same table (last three rows) is focused on 3 variants of `DeepConvNet`: using Auxiliary Loss (see Section IV-A), Weight Sharing (see Section III-B) and the two combined. As expected both solutions led to

Model	N. Param.	Aux. Loss	Learning Rate	Weight Decay	Train Error	Validation Error	Test Error
MLP	279 562	False	0.0001	0.1	0.0546 ± 0.04335	0.2239 ± 0.0548	0.22 ± 0.0205
ConvNet	77 530	False	0.0001	0.1	0.1174 ± 0.0199	0.252 ± 0.0322	0.2404 ± 0.0102
DeepConvNet	2 700 834	False	0.001	0.001	0.0852 ± 0.0315	0.1899 ± 0.0466	**0.2131** ± 0.0164
ResNet	2 746 626	False	0.0001	0.1	0.1942 ± 0.1407	0.3010 ± 0.1028	0.3259 ± 0.0728
DeepConvNet	2 826 294	True	0.001	0.001	0.0686 ± 0.0606	0.201 ± 0.0558	0.197 ± 0.0310
Siamese	2 699 968	False	0.001	0.001	0.0736 ± 0.0193	0.13 ± 0.0365	0.1496 ± 0.0164
Siamese	2 762 698	True	0.001	0.001	0.0444 ± 0.0078	0.119 ± 0.0430	**0.1221** ± 0.0146

Table I: Results without data augmentation, using 25 epochs per training and considering 10 experiments per model

Model	N. Param.	Aux. Loss	Learning Rate	Weight Decay	Train Error	Validation Error	Test Error
MLP	279 562	False	0.0001	0.1	0.0470 ± 0.0052	0.0799 ± 0.0173	0.0890 ± 0.0065
ConvNet	77 530	False	0.0001	0.1	0.1174 ± 0.0126	0.12 ± 0.0200	0.1333 ± 0.0155
DeepConvNet	2 700 834	False	0.001	0.001	0.0422 ± 0.0022	0.0499 ± 0.01	**0.0493** ± 0.0037
ResNet	2 74626	False	0.0001	0.1	0.0911 ± 0.0105	0.0766 ± 0.0152	0.097 ± 0.0145
DeepConvNet	2 826 294	True	0.001	0.001	0.0229 ± 0.0055	0.0366 ± 0.0208	0.0373 ± 0.0005
Siamese	2 699 968	False	0.001	0.001	0.0396 ± 0.0168	0.0600 ± 0.0173	0.0643 ± 0.0283
Siamese	2 762 698	True	0.001	0.001	0.0140 ± 0.0084	0.0166 ± 0.0152	**0.0296** ± 0.0106

Table II: Results with data augmentation, using 200 epochs per training and considering 3 experiments per model

improved the performances (especially weight sharing), and the best error rate is obtained when combining them together (error rate $\approx 12\%$). In Figure 3 the evolution of the loss for `DeepConvNet` and `Siamese` with Auxiliary Loss training (both on the training and on the validation set) is showed. Even if the losses are computed in two different ways, is clear how the training using weight sharing and auxiliary loss increases convergence speed and reduces overfitting. However, in both cases the trend of the loss assessed on the validation test continues to be quite fluctuating. Finally, we decided to test again all the seven models just mentioned taking advantage of the data augmentation described in Section II-A. The results are summarized in Table II. As in Table I the models are described by rows, and the performances reported in the last three columns refer to the mean error rate ($\pm$ standard deviation) among 3 experiments (with data augmentation and trained for 200 epochs each). What is evident from this second table is that all the models got a strong benefit when trained on the augmented dataset (the error rate reduces up to 5 times) and all the overfitting pathologies almost disappear (training error equal to the test error). Even the complex ResNet is able to learn the task (test error rate $< 10\%$) but the best performances are still reached with `DeepConvNet`. In particular the `DeepConvNet` using weight sharing (i.e. `Siamese` network), auxiliary loss and data augmentation is able to reach an error rate equal to $2.96\% \pm 1.06\%$.

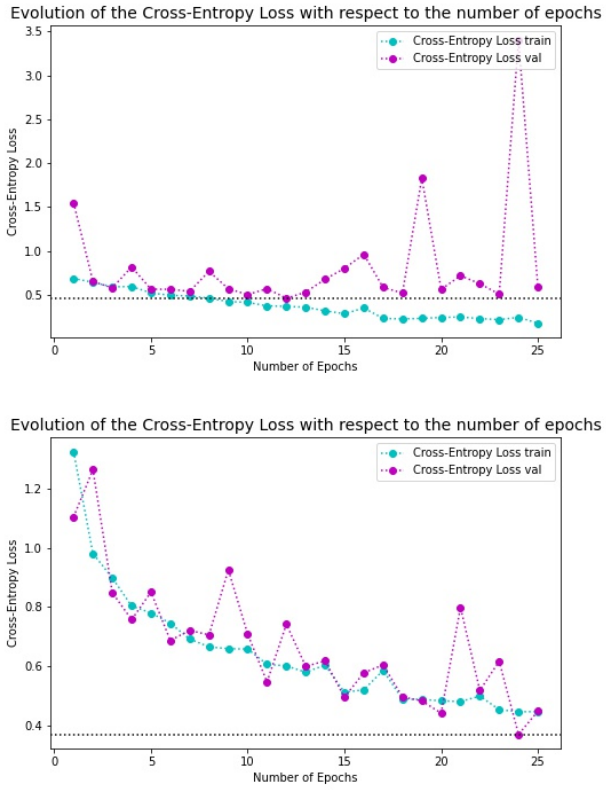

Figure 3: Evolution of the loss during training for `DeepConvNet` (above) and `Siamese` + Auxiliary Loss (below)

.

VI. Conclusion

In this study we compared different Deep Neural Networks to solve a binary classification task on MNIST dataset. Advantages in using Weight Sharing and Auxiliary Loss are discussed and confirmed by experiments. Further improvements are reached making use of data augmentation. The best model reaches a test error rate equal to $2.96\% \pm 1.06\%$.

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[2] B. L. B. Y. LeCun, Y. and Haffner, "Gradient-based learning applied to document recognition," 1998.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.