EE-559 Deep Learning

Convolutional Neural Network and improvements for MNIST comparison

Using classification, weight sharing, auxiliary losses

1 Introduction

MNIST is a database of handwritten digits represented on grayscale images, broadly used in fields such as deep learning and pattern recognition methods.

The aim of this project is to implement multiple network architectures able to predict from a pair of images whether the first digit is lesser or equal to the second. Moreover, the attention is set on analyzing the different networks performances, and in particular on the improvement that can be achieved through weight sharing or the use of an auxiliary loss. Therefore, three types of model are explored in this report: a baseline convolutional network, a network using weight-sharing and a network combining weight-sharing and auxiliary loss, all of them being fully implemented with PyTorch.

The training and test set are composed of 1000 pairs each of 1 x 14 x 14 grayscale images, leading to tensors of 2 x 14 x 14. In addition, the dataset provides the class to predict, meaning 1 if the first digit is lesser or equal to the second and zero if not, but also the classes of the digits, namely the digit itself.

2 Architectures

2.1 Baseline convolutional network

To compare the improvement of both weight-sharing and auxiliary loss, a simple convolutional neural network (ConvNet) is used as the basis for the other implementations. The architectures presented in the following subsections use the same components albeit with slight modifications to allow for weight-sharing and/or auxiliary loss. All architectures are summed up in table 1.

The base ConvNet takes as input a pair of images of dimension 2x14x14, and feeds the pair through a convolutional layer, then the fully connected layers, treating the pair as two channels with features dimension 14x14. Its output has dimension 2, which corresponds to the two classes {0, 1} of the classification objective (whether the first digit is smaller or equal to the second one). Note that this base ConvNet does not train to recognize the digit in the image at any point.

2.2 Network with weight-sharing

To implement weight-sharing, the same base network is used on both images separately, with the main difference being its output being now of dimension 10, corresponding to the classes of the digits. The outputs of the ConvNet module are concatenated and the two classes {0, 1} of the classification objective are predicted through a final MLP module of 3 layers. This allows both images to use the same weights in each layer as shown in Fig. 1.

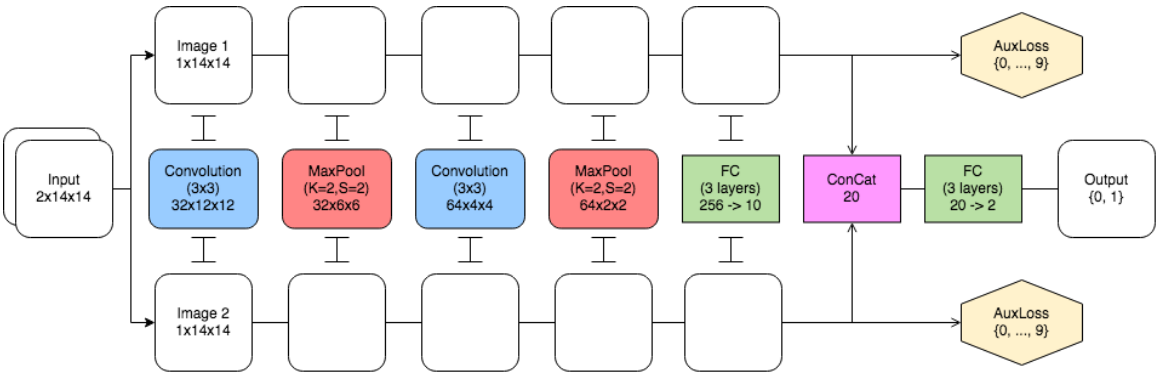


Figure 1: ***Diagram of the final network.*** *First, each image is reduced to 10 classes (corresponding to the digits) through a ConvNet module. From this output an auxiliary loss for each image is computed. Finally, the two outputs of the ConvNet module are concatenated and fed to a final MLP module which outputs the two classes {0,1}*

2.3 Network with weight-sharing and auxiliary loss

Finally, an auxiliary loss is implemented to improve performance by also training the network to recognize the digits of each pair. This is done by returning a prediction of the digit after the ConvNet module, and computing an auxiliary loss on train classes {0, ..., 9} which is added to the final loss computed on the train targets {0, 1}. A representation of the model is shown in Fig.1.

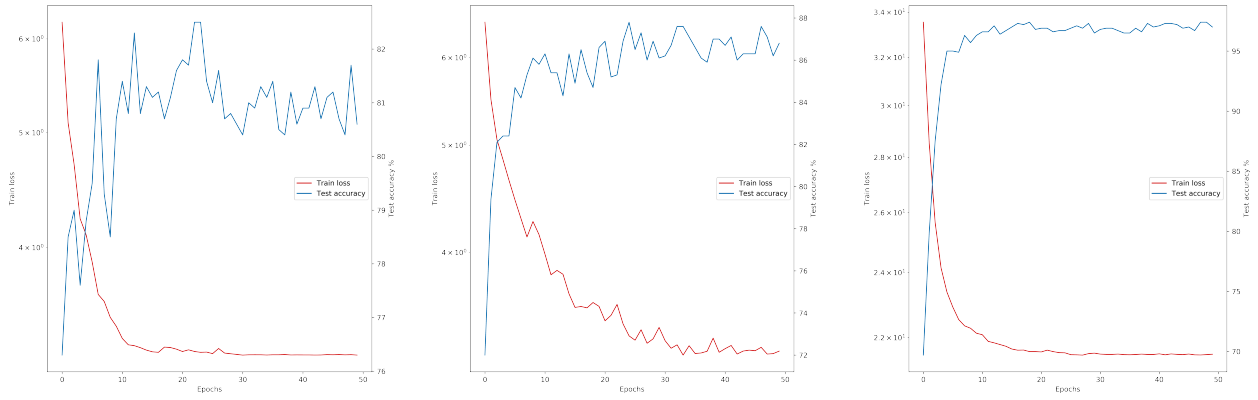
	ConvNet	Weight-sharing	Weight-sharing + Aux. loss
Input dim.	2 x 14 x 14	1 x 14 x 14	1 x 14 x 14
ConvNet module			
32 x 12 x 12	Conv2d(2, 32, k=3)	Conv2d(1, 32, k=3)	Conv2d(1, 32, k=3)
32 x 6 x 6	max_pool2d(k=2, s=2)	max_pool2d(k=2, s=2)	max_pool2d(k=2, s=2)
32 x 6 x 6	relu()	relu()	relu()
64 x 4 x 4	Conv2d(32, 64, k=3)	Conv2d(32, 64, k=3)	Conv2d(32, 64, k=3)
64 x 2 x 2	max_pool2d(k=2, s=2)	max_pool2d(k=2, s=2)	max_pool2d(k=2, s=2)
64 x 2 x 2	relu()	relu()	relu()
256	view(-1)	view(-1)	view(-1)
200	Linear(256, 200)	Linear(256, 200)	Linear(256, 200)
200	relu()	relu()	relu()
200	BatchNorm1d(200)	BatchNorm1d(200)	BatchNorm1d(200)
200	dropout(0.20)	dropout(0.20)	dropout(0.20)
100	Linear(200, 100)	Linear(200, 100)	Linear(200, 100)
100	relu()	relu()	relu()
100	BatchNorm1d(100)	BatchNorm1d(100)	BatchNorm1d(200)
100	dropout(0.20)	dropout(0.20)	dropout(0.20)
2 / 10 / 10	Linear(100,2)	Linear(100,10)	Linear(100, 10)
2 / 10 / 10	softmax()	softmax()	softmax()
Final MLP module	-	out_im1, out_im2	out_im1^*, out_im2^{**}
20		cat(out_im1, out_im2)	cat(out_im1, out_im2)
150		Linear(20, 150)	Linear(20, 150)
150		relu()	relu()
150		BatchNorm1d(150)	BatchNorm1d(150)
150		dropout(0.20)	dropout(0.20)
50		Linear(150, 50)	Linear(150, 50)
50		relu()	relu()
50		BatchNorm1d(50)	BatchNorm1d(50)
50		dropout(0.20)	dropout(0.20)
2		Linear(50, 2)	Linear(50, 2)
2		softmax()	softmax()
Output dim.	2	2	$2, 10^*, 10^{**}$

Table 1: ***Networks architecture and operations.*** ** Aux. loss for image 1, ** Aux. loss for image 2*

3 Training methodology

The same methodology is used to train the three networks and assess performances. Training is performed over 50 epochs where input is randomized at each epoch in batches of size 100. Cross entropy is used as loss criterion, which is the standard choice for classification tasks. Weights optimization is performed with the classical SGD algorithm with two different learning rates ($eta = 9 \cdot 10^{-2}$ in the first part of the training ($=3/5 \cdot nb\_epochs$) and $eta_2 = 10^{-3}$, in the second part) and a L_2 penalty ($weight\_decay = 5 \cdot 10^{-4}$). For ConvNet and Weight sharing models, the two output classes are compared the train target, from which loss, gradients are computed and weights are optimized. For the third model with auxiliary losses, as the main objective is not to train the network to recognize the digit, these auxiliary losses are weighted during training with $\alpha = 0.75$, and added to the main loss which is weighted with $\gamma = 1$. From this final loss, weights are optimized. Each model is tuned with the grid search method to find the best hyper-parameters (eta_1, eta_2, α, γ). For tuning, performance is computed on a validation set corresponding to 20% of the train set. An example of training loss and test accuracy behaviors during training is shown in Fig.2 for each network.

To assess performance estimates, after training, the number of errors was computed on the test set from which the architecture accuracy was then calculated. Performance for each architecture is estimated over 15 runs.



(a) ConvNet *(b) Weight Sharing* *(c) Weight Sharing and auxiliary losses*

Figure 2: ***Training procedure.*** *Values of train loss and test accuracy at each epoch during training. Compared to the base network, we see that the use of weight sharing and auxiliary losses yields to a more stable training procedure.*

4 Results

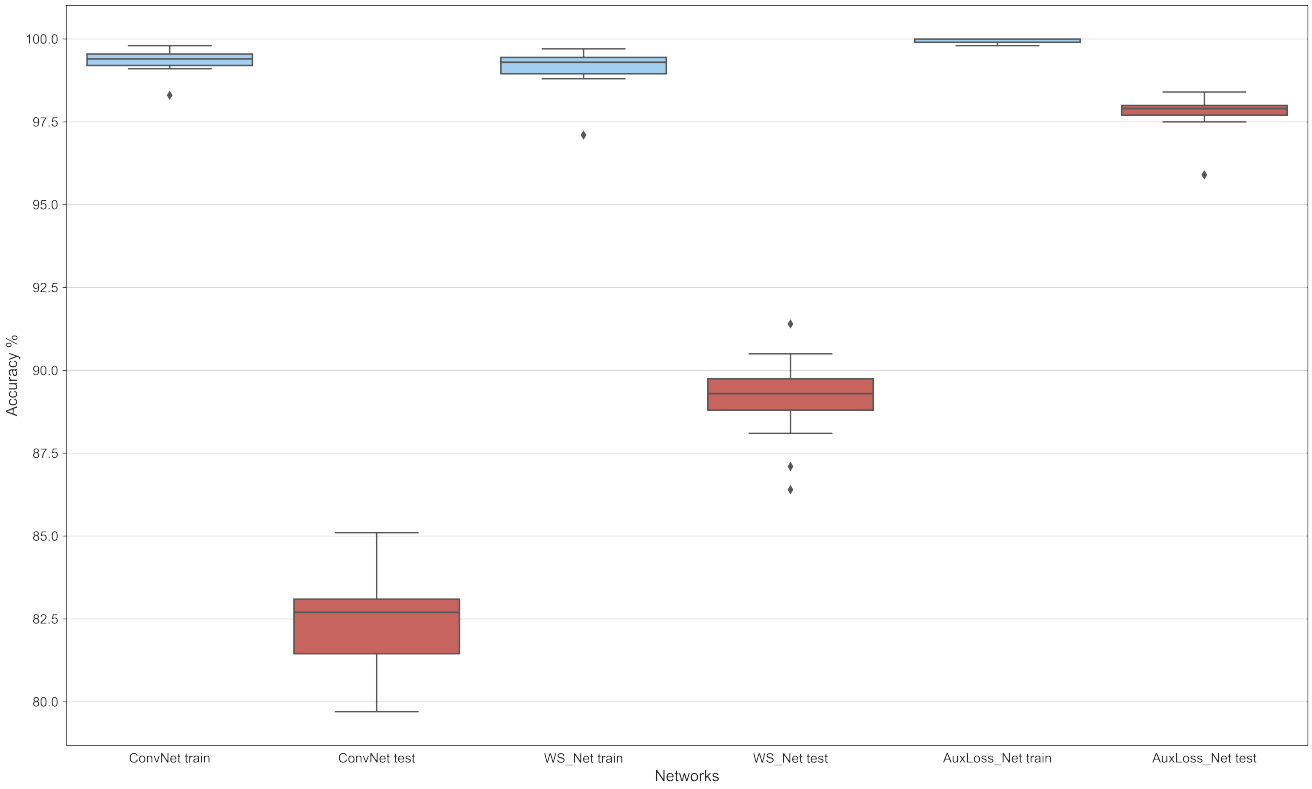


Figure 3: ***Models comparison boxplot.*** *This figure shows how the models perform at comparing digits. Estimates are calculated over 15 rounds. Results on the test set are the following :*
ConvNet $81.37\% \pm 1.37\%$, ***Weight-sharing*** $88.68\% \pm 1.57\%$, ***Weight-sharing+Aux. losses*** $97.91\% \pm 0.36\%$

5 Discussion

Our results yield a solid evidence that the combined use of weight sharing and auxiliary losses is effective at improving deep neural networks for classification tasks on the MNIST database and may also be useful for more complex visual recognition or computer vision tasks. The models were chosen in order to see that improvement, they have quite simple and similar structures in order to really focus on the gain obtained by implementation of weight sharing and auxiliary loss. At the end, with the final model, we reach $97.91\% \pm 0.36\%$ accuracy for the main objective compared to $81.37\% \pm 1.37\%$ accuracy obtained with the basic convolutional network, which is quite a big improvement for only few additional layers, showing again evidence that auxiliary losses and weight sharing are really effective features to have in mind when designing deep neural network architectures.