EE-559
Deep Learning
Spring 2021

Project 1
Classification, weight sharing, auxiliary losses

Authors
Jon Kqiku
Mert Soydinc
Alejandro Noguerón

Professors
François Fleuret

November 20, 2021

1 Introduction

In this project, we have addressed the problem of classification on the MNIST dataset. The input is composed by to images of said data set, and the network has the task of deciding whether the first digit is lesser or equal than the second one. In particular, we want to investigate how do Fully Connected Components compare to Convolutional Layers in performance, to what extent does weight-sharing affect the performance of the network, and whether the use of an auxiliary loss -the classification of the number in this case- improves performance.

To be able to assess this questions individually, we have designed 8 networks, 4 networks with fully connected components and 4 Siamese [1] networks with convolutional layers. The 4 networks correspond to a simple network, a network with weight sharing, a network that uses the auxiliary loss and a network that uses both weight sharing and auxiliary loss. See the table in section 3 for details.

2 Methods

To decide on an architecture for our networks, we first started optimizing the simple architecture, that is, without weight sharing nor auxiliary loss. By trying different simple architectures, we arrived to the models shown next.

Network	Layer	Layer Type	Network	Layer	Layer Type
FCC	1	Linear(512)	Siamese	1	Conv ($16\times12\times12$)
	2	ReLU		2	MaxPool
	3	Dropout		3	Dropout
	4	Linear (256)		4	BatchNorm
	5	BatchNorm		5	ReLU
	6	ReLU		6	Conv ($32\times4\times4$)
	7	Linear (32)		7	BatchNorm
	8	BatchNorm (32)		8	ReLU
	9	ReLU (32)		9	Conv ($64\times2\times2$)
	10	Concatenation		10	BatchNorm $\rightarrow$ReLU
	11	Linear (2) $\rightarrow$ReLU		11	Subtraction
				12	Linear (128)
				13	BatchNorm $\rightarrow$ReLU
				14	Linear (64)
				15	BatchNorm $\rightarrow$ReLU
				16	Linear (32)
				17	BatchNorm $\rightarrow$ReLU
				18	Linear (2)
				19	BatchNorm $\rightarrow$ReLU

We implemented this models creating one class for each of FCC and Convolutional model, both inheriting the class `torch.nn.Module` of Pytorch, In each of modules, we added the options to use an auxiliary loss and to share weights, to be set at when instantiating the class. When weight sharing is not active, we use two instances of the initial part of the model (before concatenation/substraction, one for predicting each number) instead of running the model for both of the letter inputs. For the auxiliary loss, we add a second output to the model originating from the output of layer 9 in the FCC and layer 10 in the siamese networks, we then pass this output to our loss criterion using the classes of the two input letters

as labels. By doing so we help the previous layers by giving them an additional push to recognize letter classes. For the For details, see the file *models.py*. To train the model, we used PyTorch's implementations of the Cross Entropy Loss as loss function (for both the classification loss and the auxiliary loss), and of the Adam optimizer with a learning rate of 0.01, parameters that we found suitable for both models.

3 Results & Discussion

To properly evaluate each model, we trained each of the models for 10 times, each time for 25 epochs and resetting the parameters, and after each time we tested its performance. In the following table, we have plotted the average -over each of the 10 training rounds- test and train error rate for each model.

Architecture	Auxiliary Loss	Weight Sharing	Train Error Rate (SD)	Test Error Rate (SD)
FCC	×	×	0.17 % (0.4)	17.45% (0.82)
FCC	×	✓	0.06% (0.13)	13.84% (0.67)
FCC	✓	×	0.08% (0.22)	18.11% (1.06)
FCC	✓	✓	0.02% (0.06)	13.94% (0.76)
Siamese CNN	×	×	1.65% (1.85)	15.10% (0.95)
Siamese CNN	×	✓	2.45% (0.98)	14.10% (1.18)
Siamese CNN	✓	×	2.9% (4.02)	15.6% (2.25)
Siamese CNN	✓	✓	2.53% (1.58)	12.8% (1.72)

Architecture Type We can immediately see that for the size of networks, the performance is good in both cases. However, in the testing set, the performance of the Siamese CNN is consistently better than the performance of the FCC network, even if just by a few percentual points.

On the other hand, on the training set, the error rate of the FCC network is close to 0, whereas the error rate of the Siamese CNN architecture is around 2%. This shows that the FCC architecture does not have a lot of room for increasing its capacity, as it is already close to overfitting (something we also found while choosing the architecture). The Siamese CNN network thus shows less overfitting, higher accuracy, and the potential to increase its capacity (which we did not do due to hardware limitations).

Auxiliary Loss and Weight Sharing Weight sharing clearly improves both FCC, and Siamese CNN. On the other hand, the auxiliary loss does not show such a strong effect, and in any case, seems to worsen a bit the performance of both architectures when used by itself. However, the two networks that use both weight sharing and auxiliary loss have top performance in their respective architecture, with the Siamese CNN drastically improving performance.

This results stress the differences between the two architectures, and how different techniques will have a different effect on each of them. However, results are consistent with literature in the sense that these techniques generally improve performance, and that CNN architectures outperform FCC in computer vision tasks.

3.1 Training

To finish the results section, we will plot the accuracy and loss evolution for each of the 8 models we trained (not the average but only 1 round). See figure 1.

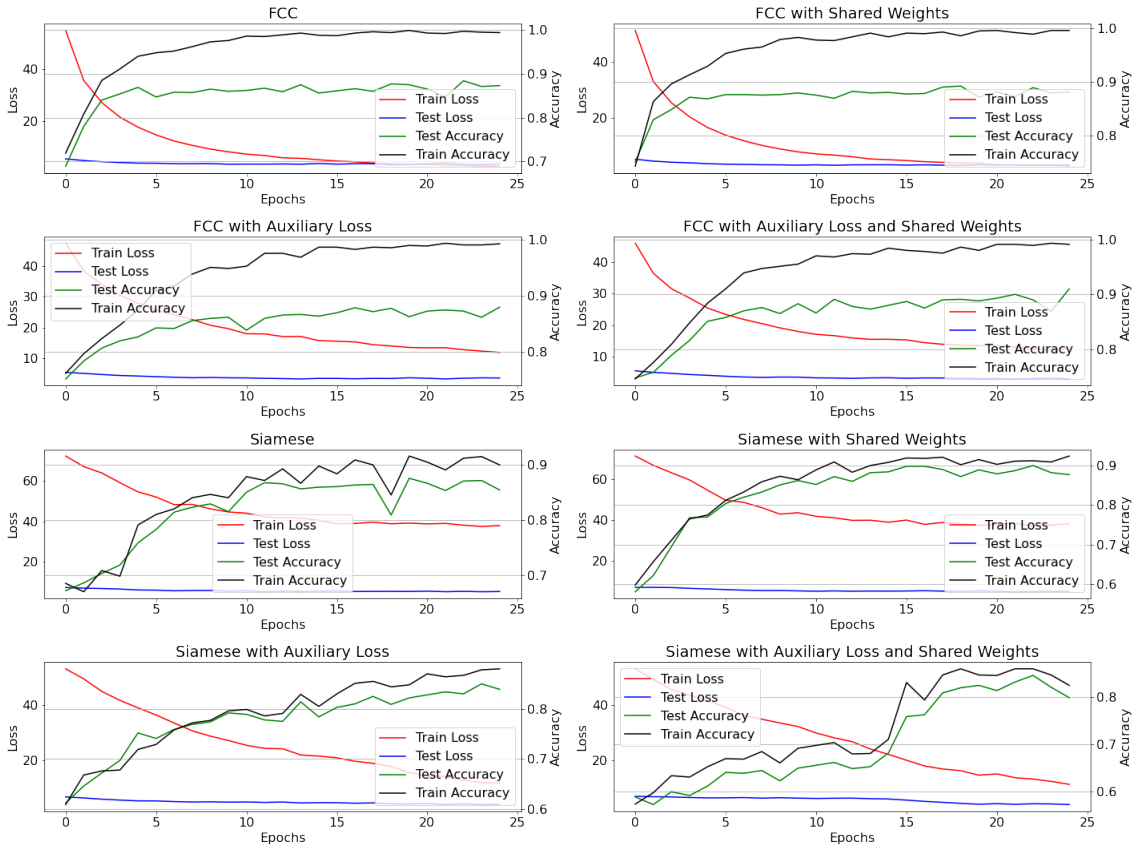

Figure 1: Loss and accuracy plots during 25 epochs of training for each of the 8 tested models.

We can see that FCC models reach their peak relatively early compared to Convolutional Siamese models. This again shows that FCC is more prone to overfitting especially without weight-sharing.

4 Conclusion

We have assesed the performance of Convolutional type networks and Fully Connected Components (or dense networks) on the task of differentiating whether one data point of the MNIST dataset belongs to the same class as a different data point (classigication task).

Our results agree with literature in deep learning. In general, convolutions architectures outperform dense architectures, and the use of techniques like weight sharing and an auxiliary loss can improve the performance of a network. We also see that auxiliary loss without weight sharing causes worse outcomes probably due to the fact that it is harder for the gradient to affect deep architectures without weight-sharing.

References

[1] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.