EE-559: Deep Learning

Mini Project 1: *"Classification, weight sharing, auxiliary losses"*

Jelena Banjac, Pavlo Karalupov, Aslı Yörüsün

May 22, 2020

1 Introduction

Image recognition is one of the hottest topics in the field of Deep Learning for Computer Vision. In this project, our aim is to implement and test different architectures to compare two 28x28 pixels handwritten digits from MNIST dataset which are down-sampled to 14x14 pixels for purposes of this project. After preprocessing we generated pairs of digits and used max pooling to receive a two-channel image (which corresponds to pairs of 14x14 gray-scale images).

We developed models for the purpose of studying the impact of weight sharing as well as impact of using auxiliary losses to help the training of the main objective. These models rely on the implementation of a simple convolutional neural network.

2 Architectures

We examined different deep neural network models, namely, *weight sharing*, *weight sharing and auxiliary losses*, *no weight sharing*, *no weight sharing and auxiliary losses*, and a *simple convolutional neural network* to predict for each pair if the first digit is lesser or equal to the second. In each of them, we basically used fully-connected layers as well as convolutional architectures for extracting the features so as to see the impact of weight sharing. For the building blocks of architectures explained below, convolution layers are represented as *number of channels, kernel size, stride, padding* and fully-connected layers as *number of channels.* All our model architectures and relevant dataset can be seen in our github repository.

2.1 Simple Convolutional Neural Network

As a very first step, we built a simple neural network, which is a building block for our solutions architectures below, with two convolutional layers for feature extraction and a fully-connected layer to classify our target values, digits from 0 to 9, by using a cross-entropy loss. As seen from the Fig.1, this model has by far the greatest accuracy result as we use this network just to predict value of a digit and compare them in the script. As this network is only recognizing digit but not comparing them, it is no surprise that it gives the best result.

2.2 Weight Sharing

In our weight sharing solution, we implemented a Siamese Neural Network Architecture by using the simple convolutional neural network (one per digit) defined above by using a cross-entropy loss and an additional fully connected layer for the comparison of digits. As it can be seen from the forward pass definition in the model architecture, we reused same module for both sub-networks, so they have the same configuration with the same parameters and weights which implies that our siamese network contains two *identical* sub-networks.

2.3 Weight Sharing and Auxiliary Loss

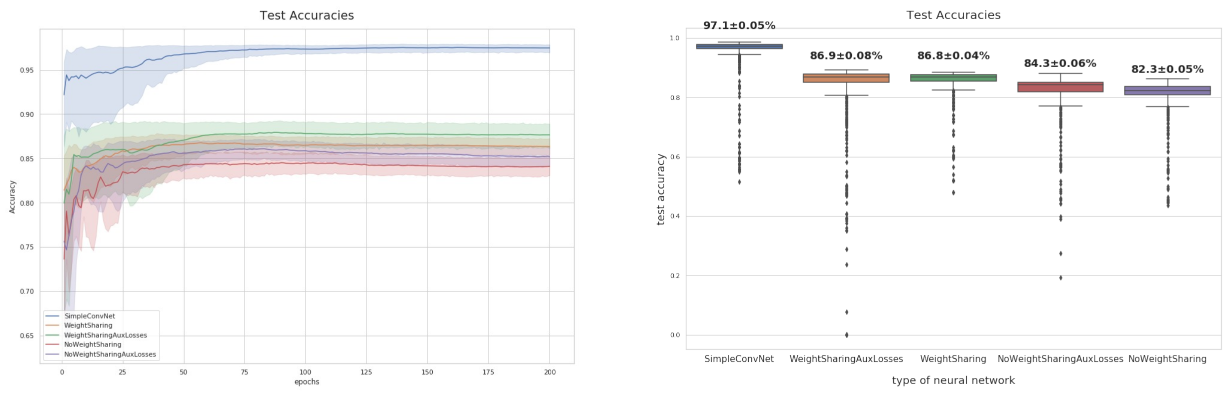
Following the weight sharing solution, we wanted to observe the improvement of the results by using the labels of the digits in each pair in batches during the training. In addition to the architecture of the weight sharing defined above, we also used the output of two sub-networks to classify the digits. Then we calculated the total loss of this method by simply adding the cross-entropy losses for the classification of each digit on top of the original loss of the digit comparison.

2.4 No Weight Sharing

The architecture of the model "No weight sharing" differs from the weight sharing solution in terms of using different modules for both sub-networks so there is no parameters and weights shared in sub-networks, they are optimized separately.

2.5 No Weight Sharing and Auxiliary Loss

Following the no weight sharing solution, we wanted to observe the improvement of the results by using the labels of the digits in each pair in batches during the training. In addition to the architecture of the



(a) confidence intervals of accuracies for each model (b) boxplot with standard deviations of each model

Figure 1: Performance estimates for each model architecture estimated through 15 rounds. Settings: Adam optimizer, no data augmentation, learning rate 0.001, 200 epochs

no weight sharing defined above, we also used the output of two sub-networks to classify the digits. Then we calculated the total cost of this method by simply adding the cross-entropy losses for the classification of each digit on top of the original cost of the digit comparison.

3 Results

For each of the solutions described above we have run experiments by using optimizers such as Adam, SGD, and RMSProp. We have run each method 15 times with 200 epochs in each run by choosing the learning rate as 0.001 and batch size as 1000. We evaluated each methods' test accuracy with the mean of the samples and their standard deviations around the mean.

As it can be seen from the Fig.1 that the **Simple Convolutional Neural Network** has the highest test accuracy of binary digit comparison with **97.1±0.05%**.

For all the other model architectures we focused on comparing the effects of adding weight sharing and/or the effects of using auxiliary losses. In that sense, we found that the test accuracy for the *Weight Sharing* model as of 86.8±0.04% performs better than the *No Weight Sharing* model as of 82.3±0.05%. In addition to that, we also saw that the test accuracy of the *Weight Sharing Auxiliary Losses* model as of 86.9±0.08% performs better than that of *No Weight Sharing Auxiliary Losses* model as of 84.3±0.06%. Although the test accuracy of these models are lower than that of the Simple Convolutional Neural Network, between them **the architecture with weight sharing and auxiliary losses gives better performance**.

Following the previous experiment done with weight sharing and auxiliary losses, we would like to see the effect of using image augmentation which is a very powerful technique used to artificially create variations in existing images to expand an existing image data set assuming that this pre-processing eventually will increase the accuracy of the model learning.

We applied two image augmentation methods sequentially. They include *torchvision.transforms*: *RandomCrop()* which crops the given PIL Image at a random location with padding equals to 4 and *RandomHorizontalFlip()* which horizontally flips the given PIL Image randomly with a default probability of 0.5.

It can be seen from the Fig.2 that the effect of the image augmentation on test accuracy performance is mostly seen in the *No Weight Sharing* model with 3.1% difference, while the least increase is occurred in *Simple Convolutional Network* model. In general, applying image augmentation increases the model accuracy as we observed in our experiment.

In Fig.3, we compared the performances of each model by applying the most used optimizers: Adam, SGD, RMSProp. What we observed from our experiment was that Adam performs the best for the majority of architectures, the second best optimizer is SGD and the least well performed between these three optimizers is RMSProp. Their performance can change depending on the machine learning problem. We found that for our case, Adam optimizer has the best performance.

4 Conclusion

Amongst the architectures we implemented to compare digits in a two-channel image, we observed that the **weight sharing**, and **auxiliary losses** independently and together improve the models' performances. Furthermore, Adam optimizer performs best in comparison to SGD and RMSProp for this problem. Last but not least, we also saw that applying a data augmentation improves the accuracy of all the models.

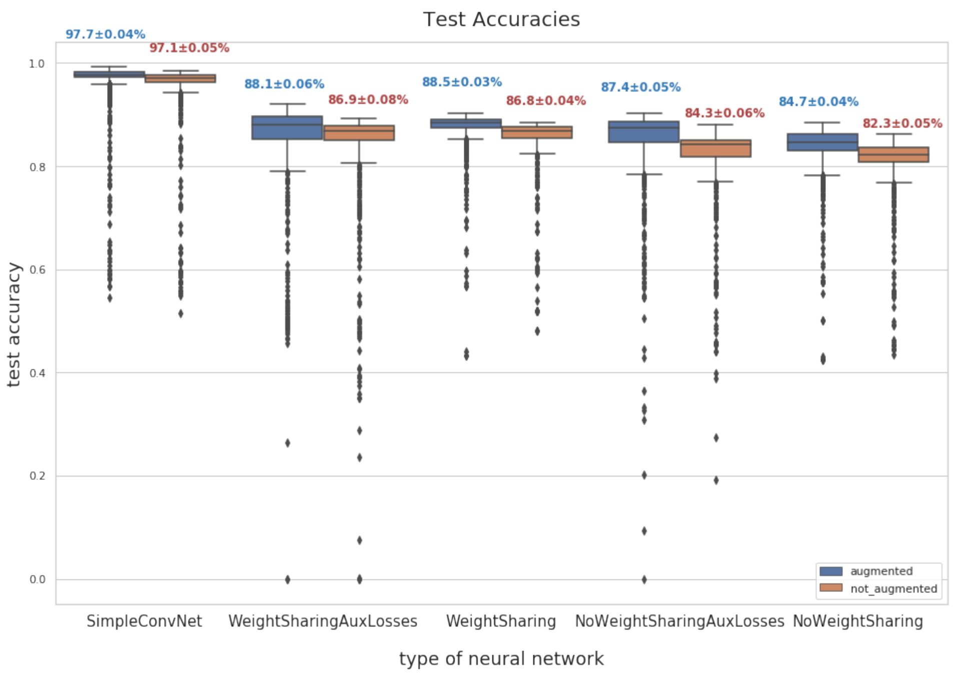


Figure 2: Performance comparison estimated through 15 rounds for each architecture including data augmentation (blue) vs. not including data augmentation (orange). Other settings: Adam optimizer, learning rate 0.001, 200 epochs

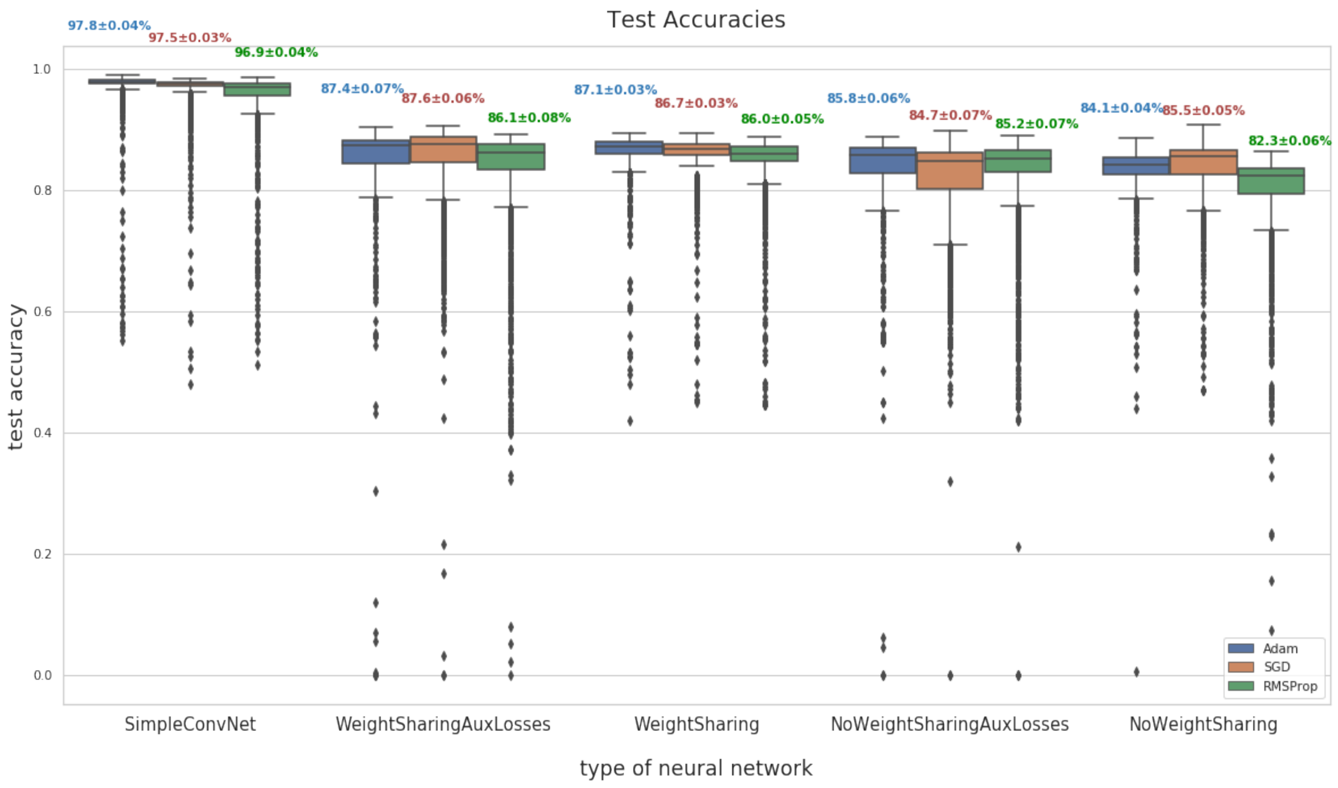


Figure 3: Performance comparison estimated through 15 rounds for each architecture compared between three different optimizers: RMSProp (green), SGD (orange), and Adam (blue). Other settings: data augmentation, learning rate 0.001, 200 epochs