EE-559 Deep Learning: Project 1

Raphael Uebersax
School of Engineering (STI)
EPFL
Lausanne, Switzerland
raphael.uebersax@epfl.ch

Pedro Bedmar López
School of Computer Science (IC)
EPFL
Lausanne, Switzerland
pedro.bedmarlopez@epfl.ch

Jonas Perolini
School of Engineering (STI)
EPFL
Lausanne, Switzerland
jonas.perolini@epfl.ch

I. Introduction

Deep neural networks are constantly gaining in importance in recent years and many different architecture have emerged. One typical application field of deep learning is image classification which consists in predicting the input image's class. A well known benchmarking dataset used in image classification is MNIST and many researchers have optimized deep architectures to obtain the best results on it. The goal of this project is to implement different deep network architectures such that, given as input a series of 2x14x14 tensors, corresponding to pairs of 14x14 grayscale images from a down-scaled version of the MNIST dataset, they predict for each pair if the first digit is lesser or equal to the second. Besides, class labels are provided for both the pair of digits and the result of the comparison.

This project will look at performances that can be reached with typical deep architectures as well as the influence of some key strategies for such a classification task. In particular, the experiments are designed to assess performance improvements achieved through weight sharing and auxiliary losses. Additionally, one architecture is then further analysed to assess performance improvements obtained with more specific techniques such as batch normalization and dropout.

The report is organized as follows. First, in section II, the considered deep architectures are introduced. The general methodology for performance assessment is then detailed in section III. The results are presented in section IV and discussed in section V. Finally, section VI concludes the report.

II. Model architecture

A. Models

In order to perform the aforementioned classification task, four deep architectures are proposed. All of them are inspired from well known architectures and have been modified to fulfill the task at hand. To do so, four *Siamese* networks (see Figure 1) are implemented with different types of layers. The idea is to push the network to first classify each image individually using different types of convolutional networks, before comparing both digits with a multi-layer perceptron of one hidden layer. The size of the hidden layer remains the same accross all different models and is chosen to be 72 as it represents each possible combination of two digits between 0 and 9. Furthermore, after each convolutional layer for all four models, batch normalization is performed to help the gradient propagation and speed up training. Next, the different architectures are presented. It is worth noting that the number of parameters indicated for each model is calculated for the models including weight sharing. Moreover, the description will focus solely on the upper and lower branches of the *Siamese* networks as the last two linear layers were introduced above.

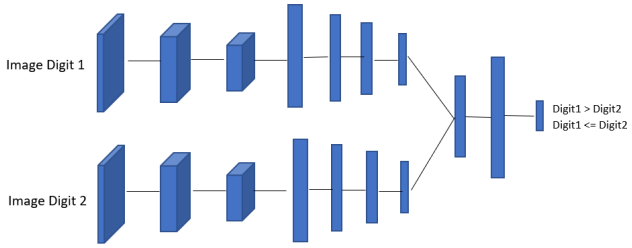

Fig. 1: Example of siamese architecture for this project

- **LeNet5 (62520 parameters):** Both, the upper and lower branch are constituted of two convolutional layers followed by three linear ones.
- **LeNet5 Fully Convolutional (63092 parameters):** Based on the previous model, it substitutes linear layers by convolutional ones.
- **Net (15460 parameters):** This architecture is a simplified version of LeNet5. It uses two convolutional layers with less channels followed by two linear ones for the image classification task. The goal is to have a computationally less expensive model and to be able to assess its influence on the performance.
- **ResNet (58244 parameters):** Based on the original ResNet model, this network is composed of ResNetBlocks which are repeated several times while keeping the number of channels and the dimension of the image constant. Three ResNetBlocks have been considered for this project.

B. Variations

Next, the training strategies as well as regularization techniques implemented in the deep architectures are presented, and hypothesis on their respective influence on the network are emitted.

- **Auxiliary losses:** It consists in introducing intermediate losses designed to train sub-parts of the network. In this specific problem, the intermediate step can be seen as the process of training the network to first use the pair of images to classify the digits (one loss for each image) before comparing them (one loss for the final comparison). Adding this strategy will help the propagation of the gradient in early layers which is expected to improve the classification results. Furthermore, performance should improve as auxiliary losses allow to consider not only the labels of the final classification (only information used without auxiliary losses), but also the class labels of the pair of digits.
- **Weight sharing:** When using weight sharing, both branches of the Siamese network share all parameters which simplifies the training and speeds up computation. The model's parameters also has twice as much training images as both input images train the same parameters. It is thus expected to improve performances.
- **Batch normalization:** Batch normalization controls the value of the activations' statistics. This is very helpful as the following layers do not need to adapt to their drift which results in faster training. This is done by adding the estimated mean and re-scaling it with the variance estimated on the batch at training.
- **Dropout:** Dropout is a regularization techniques that randomly removes units of each training sample during the forward pass and puts them back during testing. This results in increased independence between units.

III. Methodology

In order to asses performances of each model architecture as well as influence of auxiliary loss and weight sharing, quantitative results are obtained with a grid search over those methods. To ensure a "fair" comparison between the architectures, for each model, the learning rate minimizing the loss on the training set is approximated via a grid search. Each model is then trained with the Adam optimizer over 30 epochs, using a mini-batch size of 100 samples and a cross-entropy loss criterion. It is worth noting that all parameters are initialized with PyTorch's default initialization. The performances for every model on the testing dataset are then reported using the mean and standard deviation over ten fold repetitions. For each fold, the training sets have the same 1000 samples but are shuffled to add randomness. However, it is important to keep in mind that for the sake of comparison, all models are trained with the exact same training sets by fixing the random seed manually.

After assessing performances of all four model, an in-depth analysis of the $LeNet5$ model is carried out to evaluate the effects of batch normalization after all convolutional layers and of dropout. This analysis is done on the model with auxiliary losses always enabled. Besides, the same training parameters and learning rates as in the previously explained grid search are used.

IV. Results

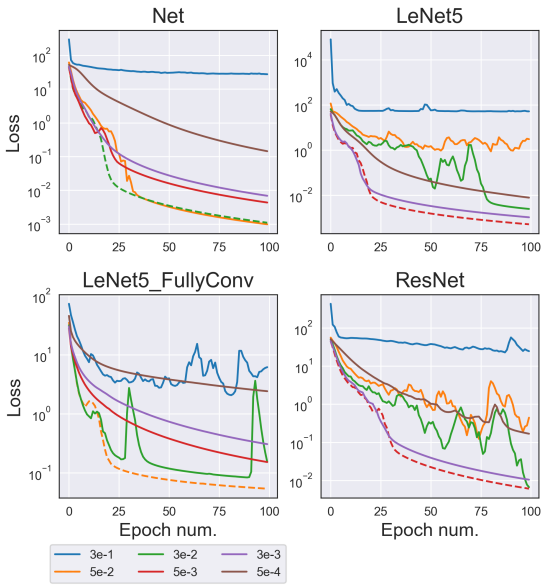

Fig. 2: Learning rate grid search for each model. The optimal learning rate used for the comparison is represented in dotted line

TABLE I: Error analysis for all four models.The best results for each model are marked in bold and the overall best results in red.

Model	Weight sharing		Auxiliary loss	
			No	Yes
Net	No	Mean	17.7	8.719
		Std. dev.	1.217	1.12
	Yes	Mean	15.91	**6.909**
		Std. dev.	1.627	**0.914**
LeNet5	No	Mean	16.459	6.149
		Std. dev.	1.921	0.91
	Yes	Mean	13.63	**6.08**
		Std. dev.	1.161	**0.8**
LeNet5_FullyConv	No	Mean	17.2	5.109
		Std. dev.	1.061	0.737
	Yes	Mean	13.49	**3.629**
		Std. dev.	1.127	**0.748**
ResNet	No	Mean	17.469	6.98
		Std. dev.	0.5696	0.625
	Yes	Mean	14.97	**5.159**
		Std. dev.	1.666	**0.579**

TABLE II: In-depth error percentage analysis of LeNet5. The best overall results are marked in red.

Weight sharing	Batch norm	Dropout	Mean	Std. dev.
No	No	No	9.619	2.737
		Yes	10.12	1.637
	Yes	No	6.149	0.91
		Yes	8.59	0.79
Yes	No	No	7.02	1.512
		Yes	7.32	1.005
	Yes	No	**6.08**	**0.802**
		Yes	6.71	0.879

V. Discussion

Results of the learning rate grid search in figure 2 enabled to retrieve valuable information about the optimal learning

rate and the number of epochs required to train the model properly. The dotted line in each sub-figure corresponds to the final choice of the learning rate for each model. The number of epoch is a trade off between computational cost and performances. It was thus decided to choose the minimal number of epoch after which the loss decay slows down and stabilizes, synonym of low performance improvements. Furthermore, to ensure fair comparisons between the model, the same number of epoch must be used. Accordingly, it was decided to use 30 epochs for every model ensuring that the elbow of the loss curve is reached.

Results of the model comparison can be found in the table I comparing all the possible combinations of weight sharing on/off and auxiliary losses on/off over the testing set. In all different networks, with weight sharing on/off, it can be seen that the performances are drastically improved by the use of auxiliary losses. Indeed, with auxiliary losses, the error percentage for the final classification is between two and three times lower (roughly 10%) depending on the model than without it. As previously mentioned, adding auxiliary losses for classifying correctly the images gives a lot more information to the network to perform its final task. By closer inspection, the network tries to classify the digits of each image first before comparing them whereas without auxiliary losses, it not classifying the digits but extracts other features. Thus, this results corroborates the initial hypothesis emitted in section II. However, it is important to keep in mind that to be able to apply auxiliary losses to a network, information about intermediary layers is necessary which is not the case for every application.

On the same table, the effect of weight sharing on the classification errors can be noticed. Once again, the results confirms the performance improvement hypothesis emitted in section II. Indeed, for all four models, weight sharing not only reduces the number of parameters but also improves accuracy (about 1 to 3%) on the testing set. This results from the fact that weight sharing allows each layer to benefit from information of both initial images rather than only one. Although these improvements are much less than with auxiliary losses, this method does not require any knowledge of intermediary layers. It is also worth noting (see I) that combining weight sharing with auxiliary losses further improves the network's accuracy.

Having assessed the effect of auxiliary losses and weight sharing, it is now possible to analyse and contrasts performance of each model architecture. First, comparing Net and $LeNet5$ gives valuable insight on the effect of increasing the number of parameters for the task at hand. As a reminder, Net and $LeNet5$ have similar architectures but different number of parameters: 15460 and 62520 respectively. It can be seen that for the relatively simple classification task of this project, the network does not require an outstanding number of parameters to obtain good results. The Net model is already able to perform the task relatively well with less than 7% classification error. However, deepening the network is usually beneficial when computational cost at training is not an issue. As expected, the deeper architecture (i.e. $LeNet5$) outperforms slightly ((1%) the shallow one (i.e. Net).

Then, benefits of a fully convolutional network can be analyzed in table I. The $LeNet5\_FullyConv$ outperform the classical $LeNet5$ architecture by decreasing the classification error from 6.08% to 3.63%. The advantage of fully convolutional networks is their ability to take bigger inputs as arguments and re-use computation to get a prediction at different location. However, this project also shows that it can improve performance of a typical convolutional network. This probably results from faster training due to the replacement of dense layers by convolutional one as well as from the additional batch normalization steps.

Moreover, table I shows that even though $ResNet$ is usually used for complicated and large training data, its performance on a simple classification task is also slightly better than $LeNet5$. However, it is worth noting that its time at training is much slower even though the number of parameters is approximately the same.

Lastly, having compared the different models, the effect of batch normalization and dropouts can be analyzed on the $LeNet5$ model. Results of this analysis can be found in table II. While batch normalization after convolutional layer helps increasing the performances of the network, dropout deteriorates the accuracy. Improvements using batch normalization can be explained by the faster training ability of the network. The drop in accuracy when using dropout seems to suggest that the training set is close to the testing set. Indeed, since dropout is a regularization techniques that adds randomness to the network to avoid overfitting for better generalization, if the testing set is close to the training set, it can be unfavorable.

VI. Conclusion

Image classification being a very useful and widely used method, knowing advantages of different types of deep neural network can come in very handy. This project compared performances of four different well known neural network architectures applied to the simple classification task of comparing digits on two images. All architectures obtained overall good classification results, but the $LeNet5$ fully convolutional network was the best performing one with 3.629% error on the testing data. Furthermore, the experiments confirmed the general hypothesis that the networks would perform better using auxiliary losses and weight sharing. This work however, doesn't allow to generalize the results and are valid for the task at hand. Indeed, the classification task was especially well suited for auxiliary losses since it could easily be divided into two sub-tasks (i.e. extract digits from images and than compare them). Similarly, as the input was a pair of similar images it was propitious for weight sharing (when using a siamense network). Lastly, positive effect of batch normalization as well as negative effect of dropout for this task have been analyzed for a typical convolutional network from the "LeNet"-family. Further work could be to assess the effect on performance of more advanced regularization techniques such as L1-L2 regularization or also by weighting the auxiliary losses.