Project 1 - Classification, weight sharing, auxiliary losses

Axel Dinh Van Chi[1] **and Nino Herve**[1]

[1]EE-559 Deep Learning, Ecole Polytechnique Fedérale de Lausanne

For this project, the objective is to compare different architectures of neural networks whose purpose are to recognize if one digit image is higher than an other digit image. Two main categories of architectures have been studied here. Networks that treat a pair of images as one image with two channels (one channel corresponds to one image) or networks that analysis each image separately before combining the information into one output (siamese network). Additional features such as weight sharing and auxiliary losses were also studied to assess if any improvement can be achieved when adding them.
All the code is available here.

Introduction - Setup

Data. The data originated from the MNIST database. For each training, 1000 pairs of digit images were combined. A sample was labeled as 1 if the first digit was lesser or equal than the second digit. Finally, all samples were normalized before training. Class labels (value ranging from 0 to 9) of individual images were preserved for later use (see Auxiliary losses). For all models, the Cross Entropy loss was used to train. The formula is reminded here :

Loss.

$$CrossEntropy = \frac{1}{n}\sum_{i=1}^{n} -\log\frac{e^{A_i}}{\sum_{j=0}^{c} e^{A_j}}$$

where A_i is the output of the target class and c are the different classes. For the final output, the classes are either 0 or 1 while the auxiliary outputs range from 0 to 9.

Training. In order to train all of our models, we used the Adam optimizer along with the Cross Entropy loss. Firstly we needed to find hyperparameters, to do so, we used a 5-fold cross-validation on 25 epochs over different configurations, and selected the model with the best final accuracy. Secondly, to evaluate the performance of the models, we trained each of them 15 times on 50 epochs to assure convergence. Before each training round, the model's weights and the dataset were reinitialized. The learning rates (lr) used are shown into the different tables or figure.

First strategy - Naïve networks

A first strategy consisted in combining the pair of images into one image and put the whole picture into a simple learning model. Thus each pair were transformed into an image with two channels, where each channel contained a single digit. Two types of simple networks were implemented to predict labels. A multi-layer perceptron (MLP) and a convolutional neural network (ConvNet). For each kind of model, layer sizes and other hyperparameters were optimized using a 5-fold cross-validation. In order to avoid comparing gigantic structures with tiny ones, only models with a number of parameters between 90 000 and 110 000 were tested.
MLP architectures were build with two hidden layers of various sizes (Figure 1). The ConvNet structure was employed as a first weight-sharing example. Kernel sizes were always 3x3, with 0 padding and a stride of 1. We used two convolutional layers with flexible number of channels, each followed by a batch normalization and a max pooling (Figure 1).

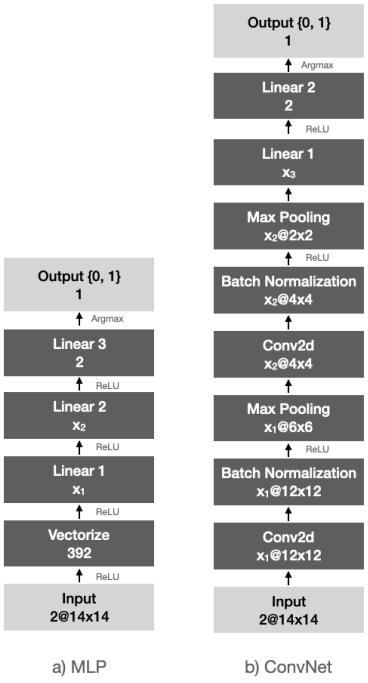

Fig. 1. *MLP and Convnet structures*: x_i values in the diagram correspond to the number of output features for Linear layers, or kernel sizes for other layers. They were optimized using 5-fold cross-validation. All other values were constant across models.

The optimal hyperparameters obtained are summarized, together with the top test accuracies, in Table 1.

Second strategy - Siamese Networks

Another, more intuitive, architecture is to have individual network branches for each image, and then combine the two

	MLP	ConvNet
Hyperparameters	Linear 1 = 200 Linear 2 = 100 $lr = 10^{-4}$	conv 1 = 128 conv 2 = 32 Linear 1 = 400 $lr = 10^{-4}$
Test accuracies	$0.807(\pm 0.015)$	$0.834(\pm 0.011)$

Table 1. *Best scores, with associated parameters, of MLP and ConvNet.* Parameters were optimized using 5-fold cross-validation on 25 epochs.

branches in order to compare observations (Siamese network, see figure 2). In other words, first analyse each image separately, before comparing the results.

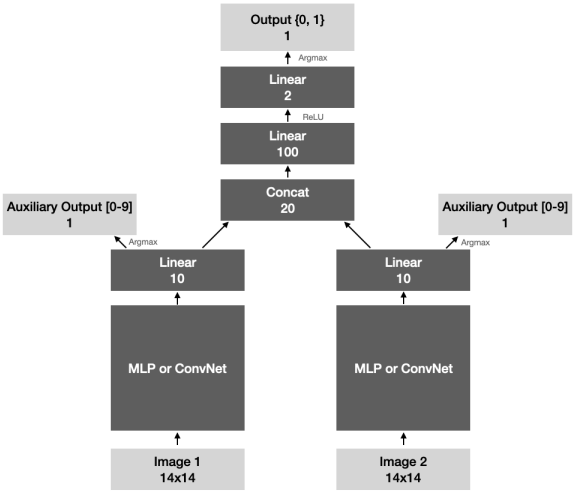

Fig. 2. *Siamese architecture.*

The branches can either be implemented by an MLP or a ConvNet. The best parameters for theses branches were again found with 5-fold cross-validations and are displayed in Table 2.

	with MLPs	with ConvNets
Hyperparameters	Linear 1 = 150 Linear 2 = 100 $lr = 10^{-4}$	conv 1 = 128 conv 2 = 32 Linear 1 = 50 $lr = 10^{-4}$
Test accuracies	$0.810(\pm 0.013)$	$0.842(\pm 0.011)$

Table 2. *Best scores, with associated parameters, for siamese network with MLP or ConvNet.* Parameters were optimized using 5-fold cross-validation and 25 epochs. lr = learning rate

Since each branch has the same purpose, and that there is no reason that a category of images would rather go through one branch than the other, it makes sense to share the weights of these two branches. Therefore, a weight sharing option (WS) was implemented for the siamese model. Figure 3 shows accuracy results for different siamese models with and without weight sharing. The first observation is that networks implemented with ConvNets tend to achieve higher accuracies. Small improvements can also be seen when the weight sharing option is activated. Although the weight sharing improvement is not very big for ConvNet, it is worth mentioning that sharing weights considerably decreases the number of parameters of the model, hence the model takes less space in memory. This can be a feature of interest, especially for big models, where the number of parameters can easily reach millions.

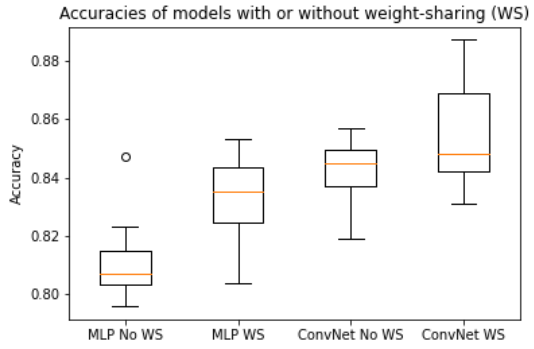

Fig. 3. *Mean accuracy over 15 runs for MLP/ConvNet siamese models with or without weight sharing.* Epochs = 50.

Considering that the model analyses each image, it is also possible to help the learning by adding to the final loss, auxiliary losses that optimize how well the branches perform before comparison. For example, it might be useful for the network to first identify the two digits in the sample and then compare the predicted numbers. Adding some classification losses (in this case Cross Entropy applied on image classes) at the end of the branches, forces the network to train on a digit recognition task in parallel. The final loss therefore becomes:

$$Loss = OuputLoss + \lambda * \sum_{i=1}^{2} AuxiliaryLoss(i)$$

where $OutputLoss$ is the loss on the binary output, $AuxiliaryLoss(i)$ are the losses on the digits and λ is the auxiliary coefficient. This way, if we give a big auxiliary coefficient to the model, it will tend to reduce as much as possible the loss on the digit recognition task. λ was optimal for $\lambda = 100$ (see Figure 4).
We can note that the accuracy tends to cap around 0.95 when $\lambda > 1$, this leads us to say that the digit recognition is important for the task we want to achieve. This is intuitive as it is easy to solve our task if we are given the values of the 2 digits.

Best Networks

This section summarizes all major models implemented during this project. Figure 5 shows the impact of training on the test accuracy.
It can be seen that the siamese networks tend to learn better than the naive models. Whats more, adding auxiliary losses outperforms the previous best model by around 10% accuracy. This could mean that optimizing for digit recognition is a good learning strategy for neural networks in this particular setup (which is quite intuitive). Test accuracies of each of these models are presented in Table 3.

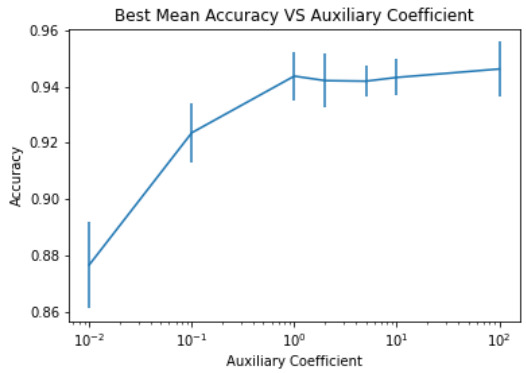

Fig. 4. *Mean accuracy for different auxiliary coefficients.* Weight sharing was activated. Best accuracy was achieved for a auxiliary coefficient of 100. epochs = 50

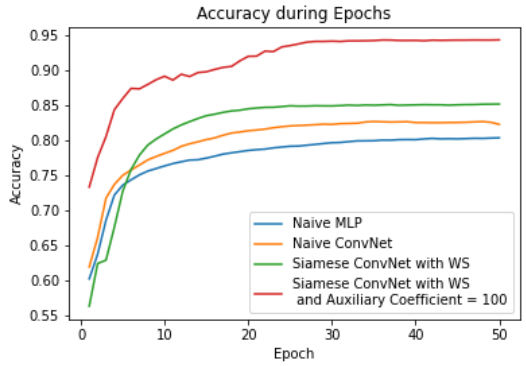

Fig. 5. *Mean test accuracy during training (15 runs) for different models with their optimal hyperparameters.*

Conclusion

In the project, we showed that using a simple MLP might lead to poor performance. But, by taking a deeper look on the task and the dataset, it is possible to get much better models.

One can make use of convolutional layers, in order to create features which depend more on the locality of the input, or use a model to solve many smaller tasks, as we did here in the Siamese network. Those examples of weight sharing allow to greatly decrease the number of parameters of the model, which can solve storing or deployement issues of models for example.

Finally, we saw that it is possible to train a model on auxiliary tasks, which can highly help the model with the final task.

There are several things we could have implemented to go further. For example, we could have done more pre-processing on the dataset, we could have tried our implementations on different optimizers, or even use some schedulers, which are known to lead to better performances. This is left to future improvements.

Model	Test Accuracy	# parameters
Naive MLP	$0.807(\pm 0.015)$	98902
Naive ConvNet	$0.834(\pm 0.011)$	92050
Siamese ConvNet no weight sharing	$0.842(\pm 0.011)$	93214
Siamese ConvNet and weight sharing	$0.856(\pm 0.011)$	47758
Siamese ConvNet, weight sharing and auxiliary loss	$0.946(\pm 0.010)$	47758

Table 3. *Mean with standard deviation of the best accuracies achieved for 15 runs. There hyperparameters are detailed in Table 1, and Table 2. Except for the model with auxiliary losses where $lr = 10^{-2}$, all the other models use $lr = 10^{-4}$.*