EE-559 Deep learning project 1

Francesco Ballestrazzi
EPFL
Lausanne, Switzerland
francesco.ballestrazzi@epfl.ch

Mateo Lopez
EPFL
Lausanne, Switzerland
mateo.lopez@epfl.ch

***Abstract*—This report presents the results of different deep neural networks architectures that compare two numbers represented as images. The results of our experiments shows that two parallel convolutional neural networks that use auxiliary losses and weight sharing can obtain an accuracy of 97.74% on test data.**

I. Introduction

The goal of this project is the evaluation of performance of different deep neural networks on a given task, that consists in the comparison of two digits visible in a two-channel image. In particular we are going to focus on the impact of weight sharing and auxiliary losses on the training process and on model's accuracy.

In the following sections, we first describe models and methods tested, and then the achieved results.

II. Dataset

The dataset used is a subset of MNIST[5]. The network need to take as input tensors of shape (2,14,14) that corresponds to two MNIST images and predict if the first digit is lesser or equal to the second. Training and test sets consist of 1000 such pairs of digits. The training set has been further splitted to obtain a validation set used to evaluate the model before the testing phase. Training-validation ratio used is 80:20.

A. Data preprocessing

Before training, the whole dataset is standardized by subtracting the mean and dividing by standard deviation. Test data has been standardized with mean and standard deviation of training data.

No data augmentation technique has been considered as the MNIST dataset is really well formatted and it present almost always images of the same scale, orientation and contrast, and therefore it doesn't really benefit the model's results to add such preprocessing.

III. Models and methods

All the following models have been implemented in the Pytorch[7] library and executed on CPU.

As this is a classification problem and the output classes are mutually exclusive, we used cross-entropy loss and the Softmax activation function[1] for the last layer output.

For all other layers, both ReLu[6] and Tanh activations have been tested, but the former gave best results and then have been used in the reported results.

The Adam[3] optimizer has been used for two main reasons, its adaptive learning rate and its ability to converge pretty fast. Due to the limited computational power these two properties were necessary to reduce training time. After a small grid search on initial learning rate, 0.005 have been used as it reached the best results.

The weights used for the testing of each model corresponds to the weights that, during training, gave the best validation accuracy.

Batch size used for all executions is 32.

If nothing else is said about it, the methods and models have all their hyperparameters set to default Pytorch values.

Since the classes in the dataset were well balanced we decided to use accuracy to evaluate the results of the different architectures. Accuracy is defined as the number of correctly classified samples and the total number of them. All the presented results are averaged over 10 runs, where data and weight initialization are randomized, and standard deviation is reported as well.

A. Baseline model

As a starting model we decided to focus on convolutional neural networks (CNNs) as they are well known to work well on image classification tasks[4].

The base model consists of a simple convolutional neural network. The fact that the two-channel images given as input are instead two distinct greyscale images have not been considered in this first phase, and the model treats the input as a single two channels image.

The first part, the one with convolutional layers, is responsible for the feature extraction in the images, and the second part, a dense neural network classifier, is responsible for the classification (see Fig.1).

For the baseline model we obtained an accuracy of $78.71\%(\pm 1, 11)$

1) Dropout: As the baseline model tends to over-fit pretty quickly, we added dropout[8] layers in the classifier, between each linear layer output and the activation function. Dropout rates are, respectively, 0.3 and 0.2. Higher dropout rates and dropout in convolutional layers have been tested as well, but resulted in too strong limitations that prevented the network from learning, and thus have been removed.

2) Batch normalization: Although data had been already standardized in the pre-processing phase, we decided to add batch normalization layers[2] both in the convolutional layers and in the classifier. The BN layers are in both cases located after the activation functions, position that gave us the best results.

The use of dropout and batch normalization, together, helped a lot to reduce over-fitting and to improve results, reaching an accuracy of $82, 17\%$.

B. Siamese network

As the structure of the baseline model we used is quite standard in a image classification task, we decided to use it as a module for the other models, with small changes in the layers shapes.

The Siamese architecture (see Fig.2) consist in two "parallel" baseline models (each one taking as input an image) that are followed by another dense neural network classifier, with the same structure as the one used in the baseline model. Ideally the two baseline models should classify the digits images in the correct label (a number in range 0-9) and the classifier should tell if the first digit is lesser or equal to the second.

Concerning the last layer of the two parallels baseline models, no Softmax layer has been used, as better results were obtained without it.

From this base structure of the network we developed and tested four variants, that originate from the combination of weights sharing and auxiliary losses.

1) Weight sharing: By weight sharing we mean that the two baseline models share the same weights. This is implemented by creating only one network and make the forward pass twice (once per image) on it. The Siamese model with weight sharing performed really well compared to the baseline model, resulting in a 90.43% accuracy

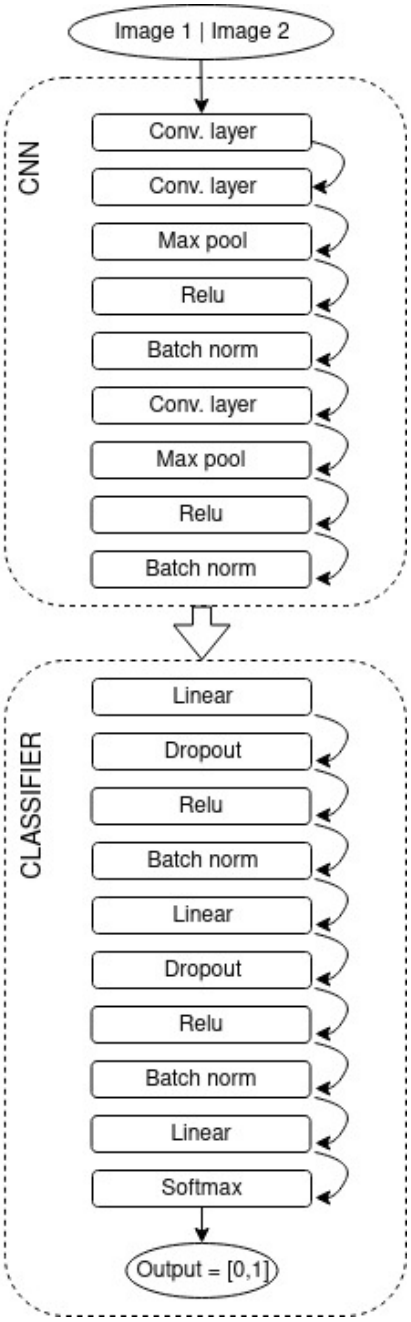

Fig. 1: Baseline model structure

2) Auxiliary losses: Since the class labels of the digits were available in the dataset, we decided to take advantage of that information to further improve our results. Each baseline network returns a tensor of shape 10, the number of possible digits in the dataset. By computing loss on the output of such networks and adding it to the main loss (the one on the final output) it is possible to force the two networks to learn to classify the digits correctly. Without auxiliary losses the risk is

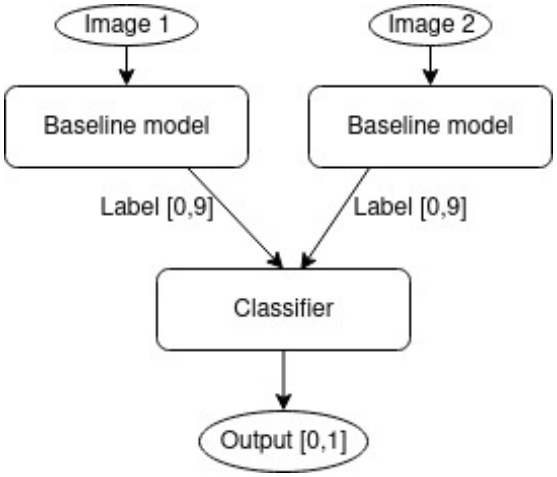

Fig. 2: Siamese network structure

to have two baseline networks that learn features not connected with the task.

In our project, we used cross-entropy loss for both networks, and the final loss was simply the sum of the three.

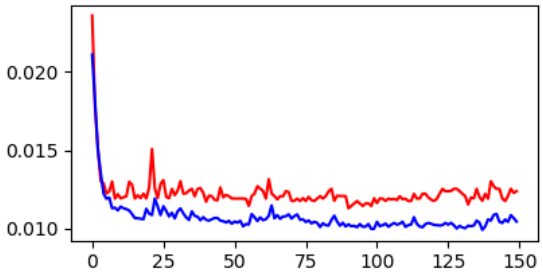

Fig. 3: Train (blue) and validation (red) error for the Siamese network with weight sharing and auxiliary losses, on 150 epochs

.

Adding auxiliary losses incredibly improved our results. Both models, with and without shared weights, benefited from this additional information and gained $\approx 7\%$ in accuracy.

IV. Results and discussion

The results obtained for the different networks are reported in Table I. The two models that achieved the best results are the ones that exploit auxiliary losses.

We are quite surprised to not observe a bigger change due to the use or not of weights sharing. This parameter doesn't seem to influence too much the results and this is quite unexpected.

Even if 97.7% accuracy can be a satisfying accuracy, we are well aware that better results could have been achieved through some more hyper-parameters tuning. The use of CPU for training kept us from really test a wide grid search to try more parameters combinations.

Model	Weights sharing	Aux. loss	N. params	Test accuracy	Std. dev.
Baseline	-	-	69530	$82,17$	$\pm 1,686$
Siamese	Yes	No	71119	90.43	± 2.881
Siamese	Yes	Yes	71119	97.74	± 0.455
Siamese	No	No	141361	90.63	± 1.616
Siamese	No	Yes	141361	97.62	± 0.780

TABLE I: Results for different architectures

Thanks to this project, we had the opportunity to deeply explore Pytorch and the contents presented in the course, and thus it has been a positive experience.

References

[1] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[2] Sergey Ioffe and Christian Szegedy. "Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv e-prints". In: (2015).

[3] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[5] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[6] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[7] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. 2019.

[8] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.