Project 1 of Deep Learning: Classification, weight sharing, auxiliary losses

Yann Bouquet, 273827, yann.bouquet@epfl.ch, Master in Data Science,
Thomas Fry, 280922, thomas.fry@epfl.ch, Master in Computational Science and Engineering,
EPFL

***Abstract*—This project in Deep Learning is about comparing two digits visible in a two-channel image from the MNIST database and formatted with the prologue code from the class. In this report are described the building of different learning architectures and their numerous enhancements to reach a better prediction. Amongst them were implemented weight sharing, auxiliary losses, data augmentation, regularization, dropout, batch normalization and hyperparameters optimization. Finally an accuracy of 0.9.. was reached.**

I. Introduction

This work aims to implement deep networks such that, given pairs of 14 14 gray scale handwritten digit images, they predict for each pair if the first digit is lesser or equal to the second. In this process we want to analyse how weight sharing and auxiliary losses would help us increase the performance of such architectures.

II. Data Analysis

The MNIST database has handwritten digits images, with a total training set of 60,000 examples and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image. Those images have been formatted with an average pooling and then gathered into pairs, with the prologue code. Thus each image is a 14×14 pixels tensor. We make sure that around 50% of the pairs generated are in the first class so that no bias would influence the results of the networks. The class 0 represents the pairs where the first number is strictly higher than the second, the class 1 represents the pairs where the first number is inferior or equal to the second. On 100 generation of 1000 pairs, in average the ratio for class 1 is 0.547 (standard deviation of 0.0179), which is coherent due to the broad equality (a bit more pairs than in class 0). All experiments and performances described in this paper are estimated on a test set of 1000 pairs of images.

III. State of the Art

Current works present only architectures that recognize the figures (we call this kind of architecture 'Figure Recognition' or FR architectures) in the images but don't compare them. By observing the LeNet5 of Yann LeCun [3] and by adapting the architecture to our pairs input, we created a network that learns to recognize our pairs while an algorithm handles the figure comparisons. In this way we managed to reach an accuracy of 97.27% with a standard deviation of $\sigma = 0.36$ (cf Table: XIII). Considering this architecture as a reference and a goal, we want to create a network that takes as input the pairs of images and return directly the result class of the images comparison so that its performance would reach the performance of LeNet5 or outperform it (we call this kind of architecture 'Binary Comparison' or BC architectures).

IV. Our Baseline Model for Comparing the Images

For handling the comparison task, a more complete type of neural network was considered, the BC architecture. The idea of ours is to create two layer pipelines, each of which will take only one of the two images in a single channel. The structure of these pipelines are identical and would copy the structure of the LeNet5, so let's consider those pipelines as the FR modules of the architectures. We started by sharing the weights of the pipelines for one layer convolution [5]. For each image, the structure would give 10 outputs that would be concatenated together, before going into some fully connected layers. We called this architecture $2nets$ (referencing the pipelines structured as two LeNet5).

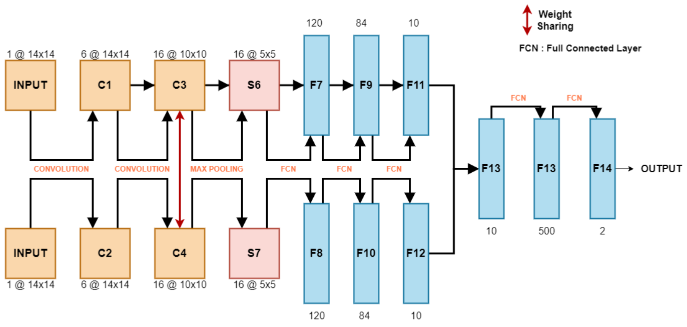

Fig. 1. Architecture of the 2Nets

V. Selection and Enhancement of the Architectures

This part deals with the progressive building of the final architecture, step by step. To help the model perform better, some justified enhancements are sought and then tested on 10 simulations, with the same number of epochs and the same batch size

Parameter	Symbol	Selected value
Number of epochs	n_{epochs}	50
Batch size	$\lvert B \rvert$	5

TABLE I
Fixed values of the parameters used for all performances.

During this experimental research, if the enhancement proves to be actually effective, it is kept for the next step. Otherwise the said enhancement is removed. In the appendix, all parameters set up for each result are referenced in Table XIII.

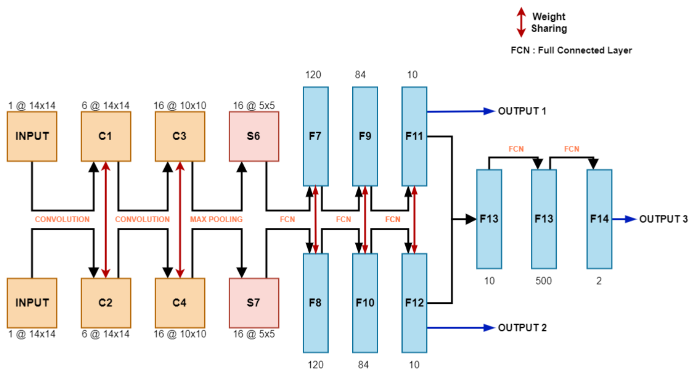

Fig. 2. Architecture of the 2Nets with weight sharing (called 2nets_ws) and auxiliary losses

A. *Weight sharing*

During the training process of the 2nets architecture we observed an over-fitting on the training set. Furthermore the pipelines operate the same way but each of them does on only half of the dataset as described in Fig. 1. In order to take advantage of the effect of both images, and to reduce the over-fitting phenomenon by decreasing the number of parameters of the model, some weights can be shared between different layers as described in the Fig. 2. The possibility to share, partially or entirely, weights rests upon the FR module of the architecture.

Models	Type	Weight sharing	Accuracy	Standard deviation
2nets	BC	✗	83.64	0.76
2nets_ws	BC	✓	86.53	1.45

TABLE II
EFFECT OF WEIGHT SHARING

Weight sharing was found relevant, for instance it increased by 3% the accuracy of the model $2nets$.

B. *Auxiliary loss*

Comparing the results of the weight sharing with the performance of the LeNet5 from III we can see that the 2nets with weight sharing (2nets_ws) operates poorly on the training sets. So the idea is to promote learning for the FR module by considering the loss that the modules would have during a figure recognition training process. Auxiliary losses are computed so that only the gradients of the FR module would be considered during the optimizer's step. These auxiliary losses are added with a weight, that equals 1. for now, to the final loss by considering the distributive property of the gradients.

Models	Type	Criterion	Auxiliary criterion	Accuracy	Standard deviation
2nets_ws	BC	CE	✗	86.53	1.45
2nets_ws	BC	CE	CE	94.47	3.12

TABLE III
EFFECT OF AUXILIARY LOSS

Auxiliary loss increased the $2nets\_ws$ accuracy by almost 8%. CE criterion stands for Cross Entropy.

C. *Optimizer*

The first optimizer to be tested was the Stochastic Gradient Descent (SGD). SGD was conclusive, except on some simulations where the accuracy of the BC architectures were around 50%, random guess. This suggested that sometimes the loss of the network was stuck in a local minimum, this is the reason another optimizer was introduced: Adam [2]. This algorithm is an extension of the stochastic gradient descent and adapts the learning rates according to the average of the first and second moments of the gradients.The idea here is to help the loss to converge faster, reach smaller minimal value and face those random guesses and also other situations where the network has difficulties to learn. This algorithm needs some parameters to work :

- α : learning rate
- β_1 : first order decaying parameter
- β_2 : second order decaying parameter (should be close to 1.)
- ϵ : small number for preventing a division by zero during the weights update

Documents say that $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are very good parameters that we don't need to modify. However we try to optimize the first decaying parameter to accelerate the training process. The first parameters we chose are : $\alpha = 7e-5$ and $\beta_1 = 0.5$ and gave us the following performance.

Models	Type	Optimizer	Accuracy	Standard deviation
2nets_ws	BC	SGD	94.47	3.12
2nets_ws	BC	Adam	91.10	3.30

TABLE IV
EFFECT OF ADAM OPTIMIZER

At first Adam optimizer made the accuracy drop by 3% on this specific 10 rounds of estimations, but in general it also made the accuracy more robust, always going beyond the 50% and converging.

D. *Dropout*

Many over-fitting phenomena were observed with the train accuracy was at 100% pretty much every time but the test accuracy was not high enough. So weight sharing wasn't sufficient for solving the over-fitting problem. To reduce such a behavior, dropout was inserted into the different layers of the network [4]. We tried to add dropout layers with $p = 0.5$ the probability of keeping the weight. Those dropout layers are only append to the fully connected layers in this way :

$$Linear- > ReLU- > Dropout$$

We decided to ignore some dropouts during the convolution layers as they don't involve a lot of parameters. Indeed, almost every weights are located in the fully connected layers.

Models	Type	Optimizer	Dropout	Accuracy	Standard deviation
2nets_ws	BC	Adam	✗	91.10	3.30
2nets_ws	BC	Adam	✓	97.28	0.42

TABLE V
EFFECT OF DROPOUT

Dropout improved the $2nets\_ws$ accuracy under Adam optimizer by more than 6%.

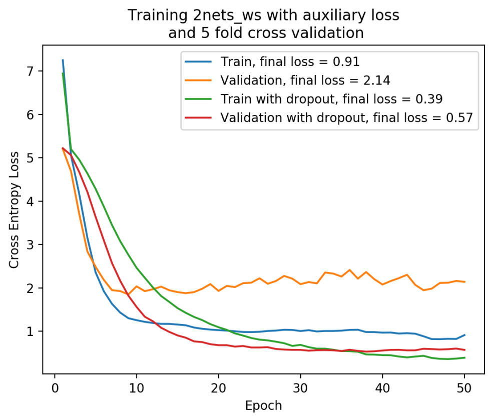

Fig. 3.

E. Batch normalization

Yet the train accuracy is 100%. Batch normalization was tried thanks to its slight regularization and robustness effects. We paired batch normalization [1] with dropout as :

$$Linear->BatchNorm->ReLU->Dropout$$

Similar to dropout, it adds some noise to each hidden layer's activations.

Models	Type	Batch norm.	Accuracy	Standard deviation
2nets_ws	BC	✗	97.28	0.42
2nets_ws	BC	✓	96.04	0.54

TABLE VI
EFFECT OF BATCH NORMALIZATION

Unfortunately the regularization of batch normalization costs a small decay in accuracy. That is why batch normalization was not kept for the next simulations.

F. Weight decaying for L2-regularization

To avoid any irregularly high weight and keep minimizing over-fitting, weight decaying was added to the loss.

Models	Type	Weight decay	Accuracy	Standard deviation
2nets_ws	BC	✓	97.28	0.42
2nets_ws	BC	✗	97.14	0.38

TABLE VII
EFFECT OF WEIGHT DECAYING

Weight decay improved lightly the already high accuracy of the model.

G. Data augmentation

1000 pairs of image might be too small a dataset for a deep learning task. One possibility to grow it artificially is to take some pairs, split them and switch their images between them in order to form new pairs. The accuracy can be significantly increased this way, however the drawback is that any simulation is much more time consuming to go through all the data, and the number of epochs should be increased in order to fully train the model. Moreover, because weight sharing has been added to the model, presenting switched pairs would be less effective.

Models	Type	DA	Accuracy	Standard deviation
2nets_ws	BC	✗	97.28	0.42
2nets_ws	BC	✓	97.02	0.54

TABLE VIII
EFFECT OF DATA AUGMENTATION (DA)

Regrettably data augmentation did not help raise the accuracy. As expected, because weight sharing spread the information of any image between layers, adding new pairs with already seen images do not have much effect. It might even pronounces overfitting, by showing several times the same images and thus the same patterns. Consequently data augmentation was not added to the training.

H. Hyperparameters optimization

We tried to optimize some hyperparameters for the model with dropout and weight decay using k_fold cross validation on the training set. The results were not satisfying as it is a very long process even we don't look at the best tuples of hyperparameters but we optimize them sequently. Indeed from a mean accuracy of 97.14% we reached a performance equals to 96.06% on the test sets.

VI. CONCLUSION: FINAL PERFORMANCE

A. Benchmark requirement

The basic requirement is a convnet with ≈ 70,000 parameters that can be trained with 25 epochs in the VM in less than 2 seconds and with ≈ 85% accuracy. We met this requirement with the model $required$ wearing 5.19×10^4 parameters. On 10 simulations with randomized dataset and weight initialization, after a training in 25 epochs, it achieved a mean test accuracy of 85.01%, with a standard deviation of 0.89 in a mean time of 1.95 seconds in the VM.

B. Optimal performance

Finally an accuracy of 97.28% was reached with our architecture described in V-D. Below are the tuned values of the hyperparameters that were used, and the fixed values of the other parameters.

Parameter	Selected value
Number of epochs	50
Batch size	5
Learning rate	6.0×10^{-4}
First order parameter	0.6
Second order parameter	0.999

TABLE IX
FIXED VALUES OF THE PARAMETERS USED FOR THE FINAL ARCHITECTURE.

References

[1] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[3] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[5] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks, 2015.

VII. Appendix

A. Pretraining

What if the model was trained only on figure recognition first and then on comparing the numbers? That was the idea that lead us to implement a pretraining where the model takes only into account the auxiliary loss. We implemented the training algorithm to enable to pretraining of the FR module of the models with auxiliary losses. However we didn't find any hyper parameters that would help a model to perform better thanks to a pretraining over the figures recognition.

Models	Type	Optimizer	Criterion	Auxiliary criterion	Weight sharing	Decreasing learning rate	Weight decay	Dropout	Image switching	Batch norm.	DA	Shuffle	Mean test accuracy	Standard deviation
2nets	BC	SGD	CE	✗	✗	✗	✗	✗	✗	✗	✗	✓	83.64	0.76
required	BC	Adam	CE	✗	✓	✓	✗	✗	✗	✗	✗	✓	84.56	1.50
2nets_ws	BC	SGD	CE	✗	✓	✗	✗	✗	✗	✗	✗	✓	86.53	1.45
2nets_ws	BC	Adam	CE	CE	✓	✓	✗	✗	✗	✗	✗	✓	91.10	3.30
2nets_ws	BC	SGD	CE	CE	✓	✗	✗	✗	✗	✗	✗	✓	94.47	3.12
2nets_ws	BC	Adam	CE	CE	✓	✓	✗	✓	✗	✓	✗	✓	96.04	0.54
net	FR	SGD	MSE	none	none	✗	✗	✗	✓	✗	✗	✓	96.82	0.53
net2	FR	SGD	MSE	none	none	✗	✗	✗	✓	✗	✗	✓	96.86	0.51
LeNet5	FR	SGD	MSE	none	none	✗	✗	✗	✓	✗	✗	✓	96.90	0.60
2nets_ws	BC	Adam	CE	CE	✓	✓	✓	✓	✗	✗	✗	✓	97.14	0.37
2nets_ws	BC	Adam	CE	CE	✓	✓	✗	✓	✗	✗	✗	✓	97.28	0.42

TABLE X
SUMMARY OF THE ACCURACIES IN INCREASING ORDER, OBTAINED ON THE VM

For the simulations with Adam optimizer, the batch size and the number of epochs correspond to the Table **??**, but the learning rate had to be adapted.

Architectures	Type	Number of parameters	Best accuracy	Standard deviation
2channels	BC	1.25×10^5	80.16	1.17
required	BC	5.19×10^4	84.56	1.50
2nets	BC	1.35×10^5	90.47	4.16
2lenet5	BC	7.47×10^5	92.53	0.98
2onechannel	BC	3.51×10^5	93.58	0.70
oneimage	BC	3.39×10^5	95.63	0.66
net	FR	2.18×10^5	96.82	0.53
net2	FR	3.86×10^5	96.99	0.28
LeNet5	FR	3.51×10^5	97.29	0.52
2nets_ws	BC	7.23×10^4	97.38	1.69

TABLE XI
NUMBER OF PARAMETERS FOR EACH ARCHITECTURE IN INCREASING ACCURACY ORDER, ON OUR COMPUTER

Models	Type	Optimizer	Criterion	Auxiliary criterion	Weight sharing	Decreasing learning rate	Weight decay	Dropout	Image switching	Batch norm.	DA	Mean accuracy	Standard deviation
required	BC	Adam	CE	CE	✓	✓	✗	✗	✗	✗	✗	84.35	0.89

TABLE XII
STRUCTURE OF THE REQUIRED CONVNET

This table shows results for every architecture and configuration that we registered on our project. Some of the architectures in this table are not explained in the main development of the project. Those architectures were created during our research process for this project and we think that including those architectures in the project may give interesting information for feature readers if they want to read the appendices of the project. The number of epochs is fixed to 50, the batch size is fixed to 5. The learning rates and other hyper-parameters are registered in the code that we provide with this report.

Models	Type	Optimizer	Criterion	Auxiliary criterion	Weight sharing	Decreasing learning rate	Weight decay	Dropout	Image switching	Batch norm.	DA	Shuffle	Mean test accuracy	Standard deviation
net	FR	SGD	MSE	none	none	✗	✗	✗	✓	✗	✗	✗	75.81	21.85
net	FR	SGD	MSE	none	none	✓	✗	✗	✓	✗	✗	✗	80.11	21.06
2channels	BC	SGD	CE	none	none	✗	✗	✗	✗	✗	✗	✗	80.16	1.17
2nets_ws	BC	Adam	CE	✗	✓	✓	✗	✓	✗	✗	✗	✗	80.84	1.47
2nets	BC	Adam	CE	✗	✗	✓	✗	✗	✗	✗	✗	✗	82.18	1.39
2nets	BC	Adam	CE	CE	✗	✓	✗	✗	✗	✗	✗	✗	82.76	3.64
2onechannel	BC	SGD	CE	none	✓	✓	✗	✗	✗	✗	✗	✗	83.26	1.37
oneimage	BC	SGD	CE	none	none	✓	✗	✗	✗	✗	✗	✗	83.67	1.14
2lenet5	BC	SGD	CE	none	✗	✓	✗	✗	✗	✗	✗	✗	83.77	0.78
2nets_ws	BC	Adam	CE	✗	✓	✓	✓	✓	✗	✗	✗	✗	83.82	1.34
2nets	BC	SGD	CE	✗	✗	✗	✗	✗	✗	✗	✗	✗	83.89	0.80
2nets	BC	SGD	CE	✗	✗	✗	✗	✗	✗	✗	✗	✓	84.04	0.86
2nets_ws	BC	Adam	CE	✗	✓	✓	✗	✗	✗	✗	✗	✗	84.35	0.83
2nets_ws	BC	Adam	CE	✗	✓	✓	✓	✓	✗	✗	✗	✓	84.76	1.54
2lenet5	BC	SGD	CE	none	✗	✓	✗	✓	✗	✗	✗	✗	84.85	0.72
2onechannel	BC	SGD	CE	none	✓	✓	✗	✓	✗	✗	✗	✗	85.25	1.05
2onechannel	BC	SGD	CE	none	✓	✓	✗	✓	✓	✗	✗	✗	85.75	1.26
2nets_ws	BC	SGD	CE	✗	✓	✗	✗	✗	✗	✗	✗	✗	86.45	1.52
2lenet5	BC	SGD	CE	none	✗	✗	✗	✓	✓	✗	✗	✗	86.86	0.75
2nets_ws	BC	SGD	CE	✗	✓	✗	✗	✗	✗	✗	✗	✓	86.87	1.85
oneimage	BC	SGD	CE	none	none	✓	✗	✓	✗	✗	✗	✗	87.50	0.69
oneimage	BC	SGD	CE	none	none	✓	✗	✓	✓	✗	✗	✗	87.80	0.64
2nets	BC	SGD	BCE	CE	✗	✗	✗	✗	✗	✗	✗	✗	89.49	7.80
2nets	BC	SGD	CE	CE	✗	✗	✗	✗	✗	✗	✗	✗	90.47	4.16
2nets_ws	BC	Adam	CE	CE	✓	✓	✓	✗	✗	✓	✗	✗	90.41	1.10
2nets_ws	BC	Adam	CE	CE	✓	✓	✗	✗	✗	✓	✗	✗	90.59	1.40
2nets_ws	BC	Adam	CE	CE	✓	✓	✗	✗	✗	✗	✗	✗	90.63	3.07
2nets_ws	BC	Adam	CE	CE	✓	✓	✗	✗	✗	✗	✗	✓	92.41	2.10
2lenet5	BC	SGD	CE	none	✗	✓	✗	✓	✗	✗	$\times 10$	✗	92.53	0.98
2nets_ws	BC	SGD	CE	CE	✓	✗	✗	✗	✗	✗	✗	✓	93.16	4.12
2onechannel	BC	SGD	CE	none	✓	✓	✗	✓	✗	✗	$\times 10$	✗	93.58	0.70
2nets_ws	BC	SGD	BCE	CE	✓	✗	✗	✗	✗	✗	✗	✗	94.26	3.69
2nets_ws	BC	SGD	CE	CE	✓	✗	✗	✗	✗	✗	✗	✗	95.25	1.69
2nets_ws	BC	Adam	CE	CE	✓	✓	✗	✓	✗	✓	✗	✗	95.50	0.60
oneimage	BC	SGD	CE	none	none	✓	✗	✓	✓	✗	$\times 10$	✗	95.63	0.66
net2	FR	SGD	MSE	none	none	✓	✗	✗	✓	✗	✗	✗	96.88	0.49
net2	FR	SGD	MSE	none	none	✗	✗	✗	✓	✗	✗	✗	96.99	0.28
2nets_ws	BC	Adam	CE	CE	✓	✓	✗	✓	✗	✗	✗	✗	97.02	0.58
2nets_ws	BC	Adam	CE	CE	✓	✓	✓	✓	✗	✗	$\times 10$	✗	97.02	0.54
LeNet5	FR	SGD	MSE	none	none	✓	✗	✗	✓	✗	✗	✗	97.08	0.51
LeNet5	FR	SGD	MSE	none	none	✗	✗	✗	✓	✗	✗	✗	97.14	0.47
LeNet5	FR	SGD	CE	none	none	✗	✗	✗	✓	✗	✗	✗	97.27	0.36
LeNet5	FR	SGD	CE	none	none	✓	✗	✗	✓	✗	✗	✗	97.29	0.52
2nets_ws	BC	Adam	CE	CE	✓	✓	✓	✓	✗	✗	✗	✗	97.35	0.35
2nets_ws	BC	Adam	CE	CE	✓	✓	✗	✓	✗	✗	✗	✓	97.38	0.46

TABLE XIII
SUMMARY OF THE ACCURACIES IN INCREASING ORDER, OBTAINED ON OUR COMPUTERS

For the simulations with Adam optimizer, the batch size and the number of epochs correspond to the Table **??**, but the learning rate had to be adapted. For the simulations with data augmentation, the number of epochs was increased to 150 in order to train fully the models.