Project 1 Deep Learning

Victor Dramé
Email: victor.drame@epfl.ch

Arthur Mellinger
Email: athur.mellinger-seron@epfl.ch

Arnaud Guibbert
Email: arnaud.guibbert@epfl.ch

Introduction

The goal of Project 1 was to test different architectures to compare two digits, each corresponding to a channel of a two-channel-image. More precisely, we aim at constructing a model which outputs $\mathbb{1}\{digit_1 \leq digit_2\}$ with high accuracy, few parameters, and reduced training time.

In section I. we dig into the structure of the data set. In section II., we discuss the approaches we adopted to engineer complicated architectures to address the classification task. Section III. then presents the results we obtain, and part we analyze them in section IV.

I. Data set

The data set consists of N samples, each consisting in a $[x_{train}, label_1, label_2, x_{test}, label_1, label_2]$ array. x_{train} and x_{test} are $2 \times 14 \times 14$-size images, with each channel representing a grayscale image. $label_1 \in \{0, 1\}$ is equal to $\mathbb{1}\{digit_1 \leq digit_2\}$, and $label_2 \in \{0, 1, \ldots, 9\}^2$ is the digit behind each channel.

II. Methodology

The peculiarity of the task is that we have two ground truths at our disposal to train our models, namely the binary target and the true digit. As a first step, we tried to build simple but efficient architectures to predict each of these two targets independently.

A. Intermediate architectures

Digit-recognition on data sets such as MNIST is a standard task, and classic CNN's such as Le Net perform well on it. However the image resolution of our data set is significantly lower than in MNIST, and reducing training time is one of our goals. Therefore we tried architectures involving only a few convolutional layers, we called our reduced MNIST architecture MNistCNN (see Fig. 5 in appendix). It features small kernel sizes, max pooling, batch-normalization, and a little dropout[1].

Instead of MNistCNN another implementation is using a ResNet with 2 or 3 residual blocks and about as many convolutional layers: we compare both architecture later on to see which one performs best.

Predicting the binary class of the two digits-image was also done using a classic CNN with two convolutional layers and a fully-connected one, featuring batch-normalization, max pooling and dropout[1]. It is represented in Fig. 4 in appendix, and called "Naive Net" because it performs the prediction task addressed in this project in the most simple fashion.

As for the activation function, we consistently used ReLU, although we gave SeLU a shot because of its famous ability to make the model learn faster. However since we had no limit on the number of epochs, we didn't make it our main weapon. We trained all of our models with cross-entropy loss, natural in classification tasks.

B. Advanced architectures

To include the additional information provided by the labelling of the digits, we needed to build a model combining both "MNist CNN" and "Naive Net". This model needs to feature an auxiliary loss (say $loss_1$) measuring how good the prediction for each digit is, in addition to the standard loss (say $loss_2$) measuring the accuracy of the prediction for the binary classification task. Then, the overall loss we want to minimize is $loss = \lambda loss_1 + (1 - \lambda) loss_2$, with $0 \leq \lambda \leq 1$ a hyperparameter to be optimized. This weighting process can be generalized to more auxiliary losses, as in the case of Lugia Net.

In order to enable the information given by each loss to flow back in the network efficiently, we needed to intertwine both of the parts discussed above in a smart way. We tried different approaches, combining concatenation of intermediate outputs of the "MNist" part to the "Naive" one, or summing them by crafting compatible out-dimensions in order to make it work. The "MNist" part has to be the same for both images (weight sharing), which is implemented by simply taking the two channels into the same network.

The architecture we spent the most time optimizing (oO Net) is depicted in Fig. 1. The lines represent information flowing between the arms of the network. CNN1 stands for a "MNist CNN"-like net with output dimension $N \times 2 \times 64$, CNN2 for a "Naive Net"-like net with output dimension $N \times 128$. FC1, FC2 and FC3 have respective output dimensions $N \times 2 \times 10$, $N \times 4$, $N \times 2$. The total number of parameters is about 70000.

A similar architecture is Lugia Net, see Fig. 1. It features 3 losses, with two losses for digit recognition: one after an inception module for digit prediction ("ResNextBlock" in the

[1] Dropout and batch normalization have been added to avoid respectively overfitting and gradient vanishing/exploding

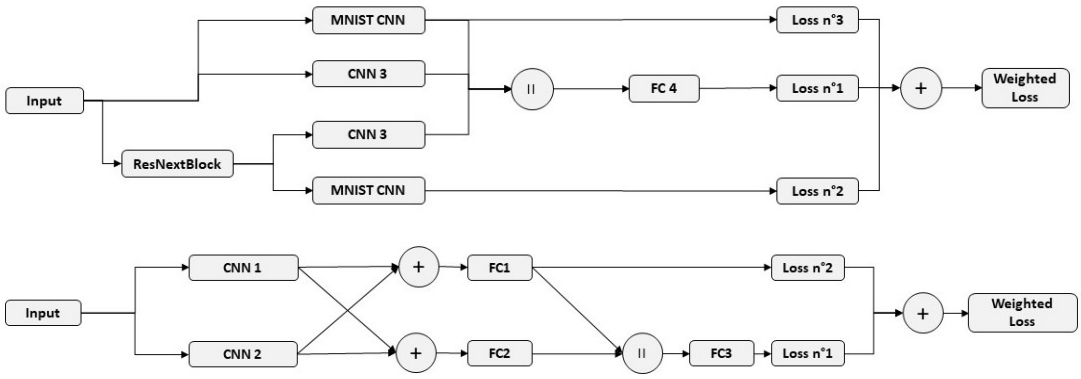

Figure 1: oO Net and Lugia Net architecture

The architectures of oO Net is presented at the top of the figure while Lugia Net one is presented at the bottom. Cells containing a + sign mean that the tensors are summed while cells containing a ‖ sign mean that they are concatenated.

graph), one after CNN3 for the same task. That way the network has even more flexibility. Information is shared by concatenation between the two lower arms.
CNN3 is a "Naive Net"-like CNN with input dimension $N \times 2 \times 14 \times 14$ and output dimension $N \times 8$. FC4 has input dimension[2] $N \times 36$ and output dimension $N \times 2$.

C. Training, testing

All of the results we present in this report are averaged over 15 runs. To perform these 15 runs we first generate two tanks of 5000 samples. For each run, we proceed as following. We extract randomly 1000 samples from the first tank to perform training, and 1000 samples from the second tank for testing. That way we make sure that they different. Besides, for the hyperparameter selection, a validation set is generated by extracting 1000 data points from the first tank such that they are different from the ones selected for the training set.

We decided to optimize the following hyperparameters:

- the weight given to each of the two losses in oO Net, $\lambda \in [0, 1]$;
- the size of the last hidden layer at the end of FC2 in oO Net, dubbed "embedded dimension" below. The reason for this interest is our lack of intuition on how much useful information the lower arm actually contains;
- the number of parallel blocks in the ResNext in Lugia Net and the number of residual blocks in the "MNIST part" of oO Net.

III. Results

A. Derivation of the best hyperparameters

- **Weighted loss.** From Fig. 7, we determine that the best weight loss for oO Net is [0.2,0.8]. Generally, it is better to give a little more weight to the loss minimizing the digit-recognition error. Setting it to 0 makes the performance drop.
- **Embedded dimension.** The embedded position didn't affect much the result. Therefore we chose an embedded dimension of 4 for oO Net, for the sake of minimizing the number of parameters.
- **Number of residual blocks.** For oO Net with a ResNet, the number of residual blocks doesn't seem to have much effect on the performance. Once again for the sake of low complexity, we decided to use only 1 resblock.
  For Lugia Net, having 2 parallel blocks in the inception module gives the best results as shown on Fig. 6.

[2]36 coming from the concatenation of two CNN3 of output dimension 8 and MNIST CNN dimension 20

B. Results on optimized models

With the previously derived hyper-parameters, we get the following results with 100 epochs, averaged over 15 runs. The accuracy presented is then the mean accuracy on the test set. The training time is for one epoch on an Intel iris+ graphics GPU. A summary of the performances is provided in Fig. 2 below.

Model	Params	Tr. time	Accuracy	std.
Big Naive Net	94'514	0.51s	82.41	1.46
Lugia Net	69'514	1.60s	95.72	0.91
oONet (CNN)	75'724	0.55s	96.95	0.69
oONet (ResNet)	49'852	0.48s	93.7	0.5

Table I: Results of the main models

C. Other remarks

Although SeLU gave better results than ReLU after 10 epochs, the difference vanished after a few more epochs. In the meantine, the networks took longer to train with it than with ReLU.

Dropout was found to be most efficient when put in the few last layer before the end of the networks. Dropping out half of the nodes after the first convolution layer sometimes

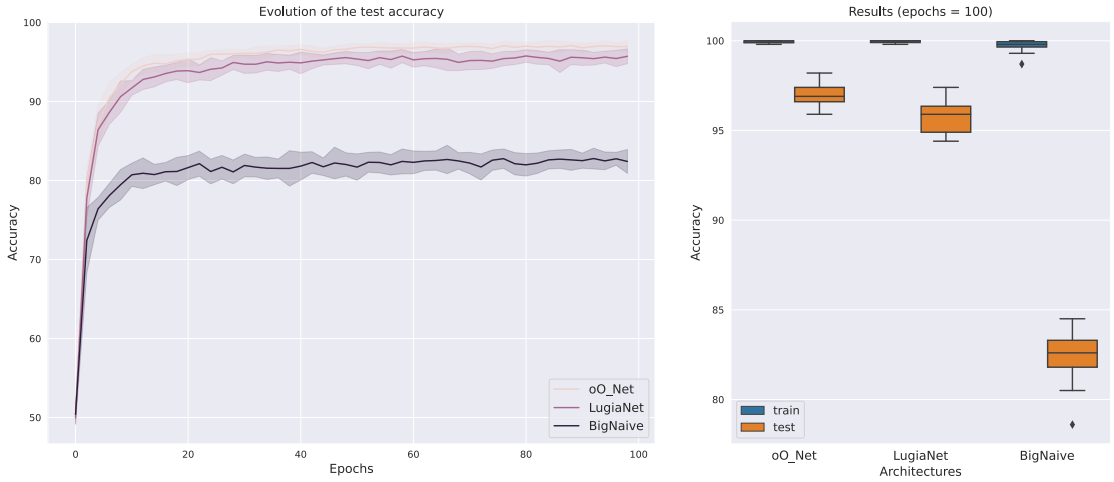

Figure 2: Final results on the train and test set. On the left figure the evolution of the accuracy on the test set is computed wrt to the number of epochs, while the right one is a boxplot of the final accuracy after 100 epochs.

made the test accuracy decrease. Generally, using a lot of parameters often resulted in overfitting. It is particularly evident in Big Naive Net with a 20% accuracy difference between train and test errors.

An insight on the impact of the number of epochs is provided in Fig. 2. The accuracy stabilizes in "Big Naive Net" (using only the binary loss) after about 30 epochs, while it keeps growing in more sophisticated models.

IV. Discussion

A first interesting perspective on these results is given by our own human performance on the binary class prediction. Over 100 trials, we got the results right 96 times, meaning 96% accuracy ($\pm$ a fairly large margin given the small test set). oO Net as well as Lugia Net give results of the same order of magnitude, which can thus be considered quite good.

V. Conclusion

A. *Influence of the auxiliary loss*

Compared to Big Naive Net which only uses binary targets, models with roughly the same number of parameters using both losses perform significantly better ($>$10% gain in accuracy, more stability). This is because learning to recognize the digits is very useful to solve the problem. Indeed, giving maximum weight to $loss_2$ leads to a drop of the performance, while giving maximum weight to $loss_1$ achieves almost as good performances as for balanced weights.

B. *Residual networks*

Lugia Net and oO Net have roughly the same perfomances, this could be explained by the fact that the ResNextBlock implemented in Lugia Net are minimalist. A way to explore further the project would be to implement more complex ResNextBlock and see the result.

VI. Appendix

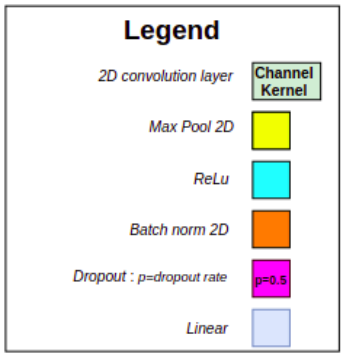

Figure 3: Legend

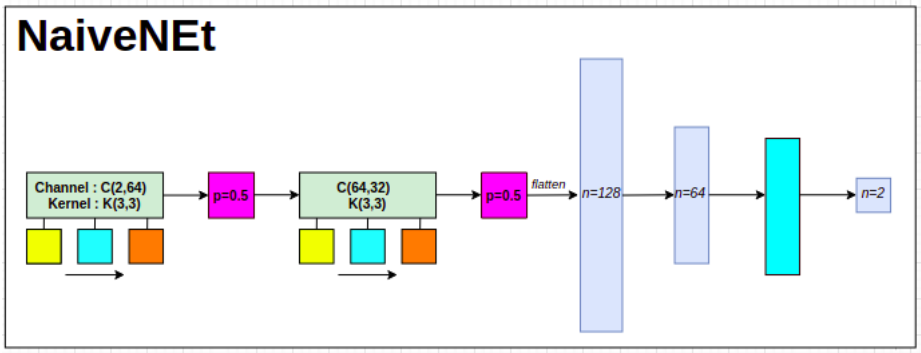

Figure 4: Naive_Net

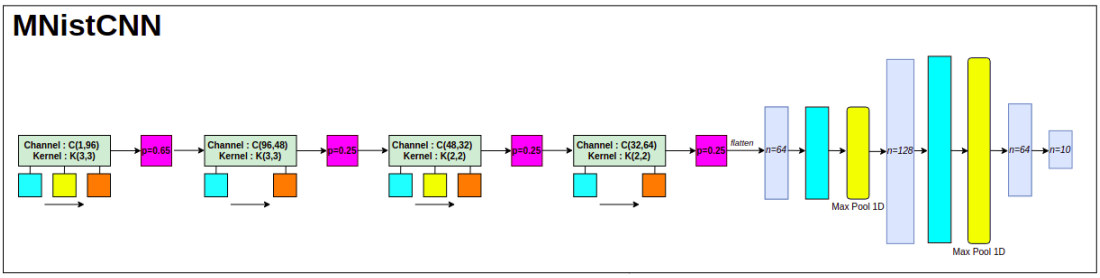

Figure 5: our architecture MNistCNN

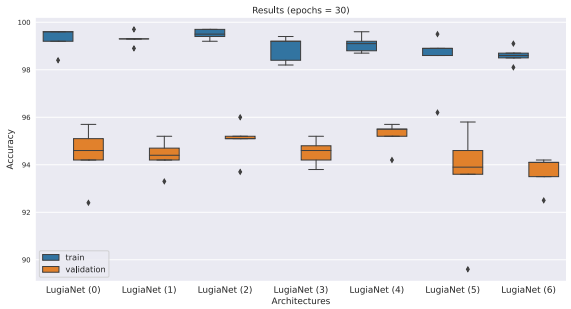

Figure 6: Accuracy of Luiga_Net for different number of res block

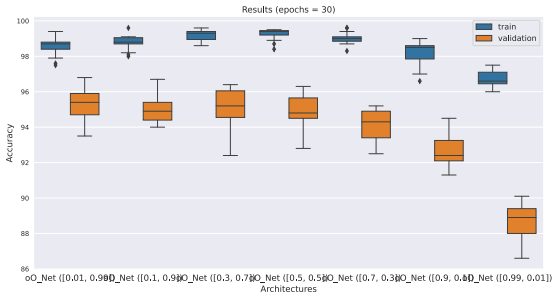

Figure 7: Accuracy in function of loss weight

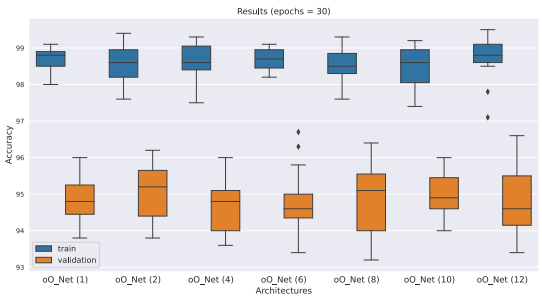

Figure 8: Accuracy in function of the embedded dimension of Naive_net

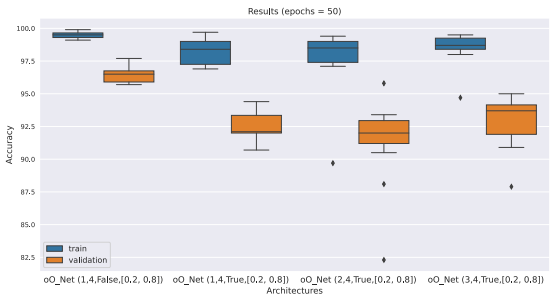

Figure 9: Accuracy comparison if we use a Resnet