Mini-Project 1 - Comparing network architectures for image processing using auxiliary losses and weight sharing

Timothee Jaubert (261816), Simon Roquette (246540), Anton Soldatenkov (314433)
Deep Learning EE-559, EPF Lausanne, Switzerland

***Abstract*—Although neural networks are very powerful tools to complete computer vision tasks, they can be data-inefficient and expensive to train. The choice of the right architecture can improve both performance and parameter efficiency. Here, we explore different techniques to improve efficiency, in terms of accuracy and computation time. The task of this project is to predict, from two handwritten 14x14 grayscale digit images, if the first digit is lesser or equal to the second. In addition to the boolean target of the main task, we have access to the class of each digit. Different strategies will be explored to solve this problem conditioned by whether or not we include additional insights about the nature of the input data and the task we aim to solve.**

I. Problem description and and general pipeline

A. Dataset

We generate a train and a test dataset each composed of a thousand pairs of 14x14 handwritten grayscale images obtained from the MNIST dataset. The task is to predict for each pair if the first digit is lesser or equal to the second. We have access to the main target (truth value of the inequality) and the class of each digit, an integer between 0 and 9. We train on 1000 samples, and test on 1000 other samples.

B. Learning parameters

We chose to have two dimensional output activated by softmax instead of a single one with sigmoid even though this is a binary classification task, as we have experienced slightly better results this way. In this way we see the outputs as probabilities of being in a class or the other and use *Categorical Cross Entropy* loss. We investigated both Adam and SGD optimizer for all models, each with learning rates around their usual default values. After each model's architecture was defined, we conducted a gridsearch with Kfold (K=5) cross validation on the training set to find the best optimizer, learning rate and batch size for each model.

As the aim of the project was not to find the best possible model, but rather investigate the impact of auxiliary losses and weight sharing on a given model, we did not make an extensive grid search exploration, neither did we try to make too many parameters vary. We didn't try too hard to add many layers such as Dropout, BatchNormalization, or even vary on the kernel size of convolution layer, but rather keep it simple, to focus on the main objective of this mini-project.

We only used LeakyRelu instead of regular Relu to avoid the problem of dying Relu ([1]).

C. Evaluation

The main evaluation metric is the accuracy of our networks evaluated on the test set. We compute as well the variance of these accuracies as we conduct the experiments several times (10 for grid search, 20 for the main experiment) and provide the accuracy mean. We also decided to indicate the number of parameters and the training time per model, as they can be decisive in case of several models reaching the same accuracy.

Disclaimer : computing training time was done on a GPU Nvidia RTX 2070. Our scripts / loading in GPU memory are probably far from optimal. Also, in training time we take into consideration the time it takes to initialize the model and load its weights on GPU. On models with quick backpropagation, the loading time can be important relative to the "actual" training time.

II. Code Structure

The code is split in several parts: First, we load and normalize the data after computing their mean and standard deviation. The definition of the models follows, we will present our implemented architectures in the next section. Next is our training function, which takes a model and train it according to its type (simple model or auxiliary loss). After the training function comes the evaluating function which takes the resulting network and evaluates it's efficiency on the test set. An optional grid search cna be conducted on each given model, indicating which optimizer, learning parameters and batch size should be used to optimize the training of the model. Finally, an experiment is run on each model : 20 training with shuffled training data. It returns the evaluation on test data : the mean accuracy, and prints additional information such as the accuracy's standard deviation, the training and evaluation time. A box plot of these experiments is generated on the evalution accuracies.

III. Models

A. Primary models

We started from two ordinary CNN (convolutional layers for feature detection followed by linear layers for classification): model_1 and model_2 which have the same architectures but differ in depth and size. We implemented as well a MLP with two hidden layers, the model_0 to have some baseline results.

model_1 :
Conv2d(2, 16, 2, 1) $\rightarrow$ leakyRelu $\rightarrow$ Conv2d(16, 32, 2) $\rightarrow$ leakyRelu $\rightarrow$ max pool2d(kernel size = 2) $\rightarrow$ leakyRelu $\rightarrow$Linear(1568, 16) $\rightarrow$ leakyRelu $\rightarrow$ Linear(16, 2)

model_2 :
Conv2d(2, 32, 2) $\rightarrow$ leakyRelu $\rightarrow$ Conv2d(32, 64, 2) $\rightarrow$ leakyRelu $\rightarrow$ max pool2d(kernel size = 2) $\rightarrow$ Conv2d(64, 128, 2) $\rightarrow$ leakyRelu $\rightarrow$Linear(3200, 64) $\rightarrow$ leakyRelu $\rightarrow$ Linear(64, 12) $\rightarrow$ Dropout $\rightarrow$ Linear(12, 2)

To improve the learning of model_2, we branched an auxiliary output after its second convolutional layer as suggested in ([2]), which runs through a classifier similar as the one of model_1. This intermediate prediction is used to compute an auxiliary loss (using the same target as the final output) which is then added to the final loss. This technique is explores in model_2_AUX1.

B. *Secondary models*

For the primary models, we acted as if we had no information about the task to perform, nor the nature of the input, by only predicting a binary output. In secondary models, we will take into account the additional information at disposal : the input is composed of two images of the same nature and the task relies on the understanding and the relationship between their meanings. This insight gain legitimates a different approach to solve the task : the input will be split in two images which will run through the same layers independently (a feature detector and a classifier), loosely speaking to capture their features. This is what is referred to as weight sharing : we use the same weights/layers to compute the feature extraction task, and then further layers take each of these images features and learn to compare them. We implemented this weight sharing method on our previous models to get model_1_ws and model_2_ws.

Finally, taking advantage of the availability of the classes of the two digits in each pair, we branch out the digits prediction for an auxiliary loss, to get model_1_ws_AUX2 and model_2_ws_AUX2.

Our results are given in table I. First, we observe that despite its simpleness, the model_0, a naive MLP, performs surprisingly well if we take in account the small training time. We did not spend time on searching to improve it, but we suppose the task could be achieved with respectable performance via this model if better conceived.

Comparing model_1 and model_0 in their simplest version, we do not observe a significant accuracy improvement, only few percent are gained for an increase of the computational time.

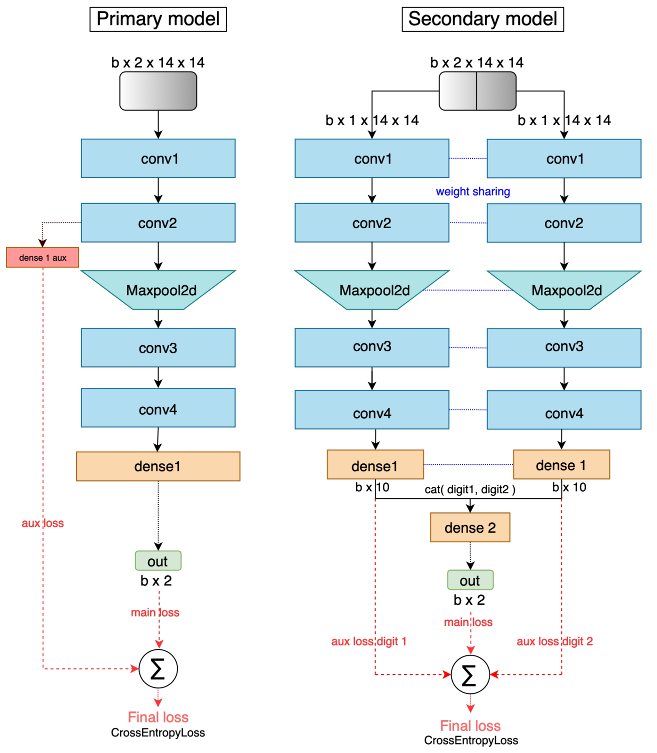

Figure 1. Primary and secondary models architectures. In dotted red, possible implementations of auxiliary losses

IV. Results discussion

Model		# param	Train acc (Var)*	Train time**
model_0	-	58 690	80.58% (0.9)	3.1 s
model_1	-	13 154	81.95%(2.2)	5.1 s
	WS	6 770	81.75% (1.1)	7.1 s
	AUX2	54 686	90.54% (1.2)	9.5 s
	WS+AUX	31 508	96.09% (1.0)	11.0 s
model_2	-	113 506	82.5%(0.6)	8.1 s
	WS	56 114	84.64% (1.0)	11.5 s
	AUX1	119 780	83.00% (0.4)	9.6 s
	AUX2	171 116	89.93% (1.5)	12.8 s
	WS+AUX	139 028	**96.52%** (0.4)	15.1 s

Table I
Model Performance Comparison *WS: weight sharing, AUX1: auxiliary loss as used in primary models, AUX2: auxiliary losses as used in secondary models*

** Measured on 20 training with data shuffle*
*** Train time with 1000 samples, 25 epochs on RTX 2070 GPU*

As expected, each model reaches its best accuracy with weight sharing and digit auxiliary losses. It is however interesting to note that despite their important difference in complexity ($20'820$ vs $337'216$ parameters) both model reach similar accuracy (over 96%).

The variances have similar values, except for the weight sharing for which they are slightly higher.

For models using AUX2 loss, we can evaluate the model on two tasks : digits recognition, and predicting the order

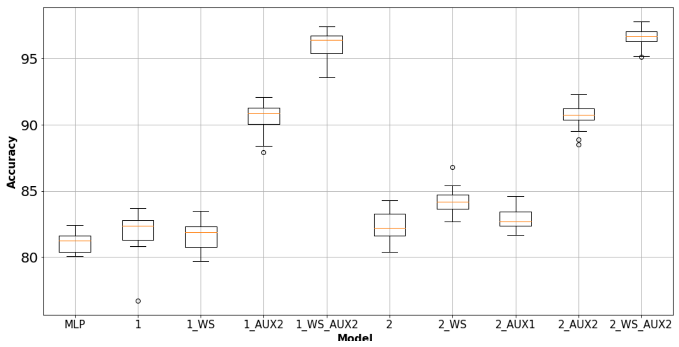

Figure 2. Accuracies distributions per model over 20 independant trainings

relationship between those digits. Interestingly, we found that even though the digit recognition accuracy is around 94% for the best models, their final accuracy is even bigger, around 96%. Intuitively, we can interpret that for our model, when deciding whether left digit is smaller than right digit, even if it is not sure whether the digit is 1 or 2, it will not be impacted by this miss-classification on the final task, as it is very likely left digit is smaller than right one whether it is 1 or 2. We did not have time, but it would have been interesting to explore the activation functions of the last classification layer based on some chosen probability distribution of the two digits.

V. Conclusion

We have seen that changes in model architecture can bring performance improvements, especially the ones that take into account the nature of the data, and use its full spectrum. Although the secondary model with weight sharing and independent pass of the two images makes more sense, the improvements of this method is limited when no additional information regarding the features to extract from the data is provided. The real gain in accuracy comes with the auxiliary losses on the digits. Intuitively, the task can be decomposed in 'recognizing the digit' and 'comparing them', where clearly the second task is easier to learn.

References

[1] Szegedy, Christian, et al. ‘Going Deeper with Convolutions’. ArXiv:1409.4842 [Cs], Sept. 2014. arXiv.org, http://arxiv.org/abs/1409.4842.

[2] Lu, Lu, et al. ‘Dying ReLU and Initialization: Theory and Numerical Examples’. ArXiv:1903.06733 [Cs, Math, Stat], Nov. 2019. arXiv.org, http://arxiv.org/abs/1903.06733.