Mini-project 1: Classification task on MNIST comparison

Tobia Albergoni, Matteo Yann Feo, Quentin Bouvet

EE-559 Deep Learning - Spring 2019

Introduction

The aim of this project is to compare the impact of two deep learning techniques, namely *weight sharing* and the usage of an *auxiliary task*, on three different neural network architectures. The three models under scrutiny are a *shallow fully-connected*, a *deep fully-connected* and a *convolutional* network. The two techniques are tested both combined and in isolation for each architecture, resulting in a total of 12 models (3 architectures x 4 technique combinations). The evaluation framework is a simple image classification task that consists in comparing two (14x14) images from the MNIST dataset and predicting which one represents the larger digit. The goal of this report is to clarify the architectures, techniques and the implementation choices adopted over the course of this small project, and to present the results. As a disclaimer, the work does not consider other widespread techniques such as *dropout* or *batch normalization* and no particular focus is put on fine tuning the architectures or parameters.

Overview

Base network architecture

All 12 models share the same high-level architecture, illustrated in Figure 1. It's composed by two distinct parts: two identical *pipes*, followed by a *main classifier*. Each *pipe* accepts a single (14x14x1) image as input and outputs a (10x1) vector. The *classifier* receives the concatenated outputs of the two *pipes* and produces the final classification prediction. A base model hence takes the (14x14x2) task inputs, splits the two images and feeds them to the respective *pipes*. The *cross-entropy loss* is then computed on the (2x1) classification output. This approach eases the implementation of the two techniques to be evaluated.

Weight sharing

In order to implement *weight sharing*, the two identical *pipes* are collapsed into a single structure, and the two input images are sequentially processed by the resulting unique *pipe*. Consequently, the number of parameters of the first part of the network is halved, reducing the overall complexity of the model. Additionally, the *pipe* effectively receives double the training data, as both images train the same set of parameters. This technique should in principle allow the first part of the network to train in a more stable manner and with less risk of over-fitting.

Auxiliary loss

The task at hand and the availability of the digit-class labels of each input image allow to improve the training procedure by adding the evaluation of an *auxiliary loss function*. By interpreting the *pipes* (or the single pipe when *weight sharing* is enabled) as classifiers for the digit-class of each input image, it's possible to compute the *cross-entropy loss* on the (10x1) intermediate representation that each *pipe* outputs, which is considered as a probability vector. This *auxiliary loss* is then combined with the loss on the main classification task through a weighted average, and the backward propagation is started from this aggregated loss. Concretely, the influence of this *auxiliary task* is regulated through the *aux_loss_weight* parameter, which can take values between 0 and 1. The *auxiliary loss* is disabled when this variable is set exactly to 0, and is set to 0.5 to enable it during the evaluation. This technique forces the first part of the network to learn a predefined task, that we know is extremely useful for the main task. In fact, once the *pipes* learn to classify single images, all the remaining part has to do is to learn to compare the intermediate outcomes.

Architectures

Shallow Fully Connected

In this architecture, the two *pipes* are designed as *shallow fully connected networks* with a single hidden layer of 500 units. This value has been validated as a good compromise between performances and number of parameters. The *pipe*'s input size corresponds to a single flattened image (i.e. (196x1)). The *classifier* is also designed as a small *shallow fully connected*

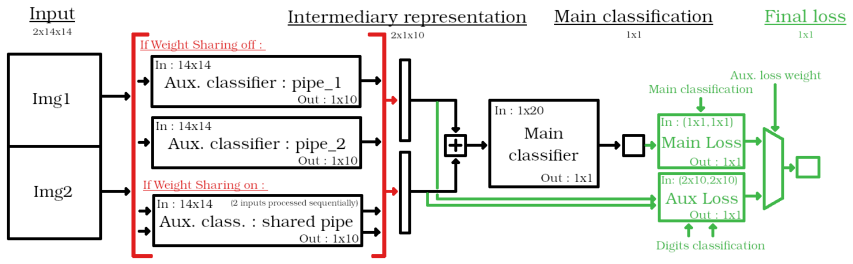

Figure 1: Network structure overview

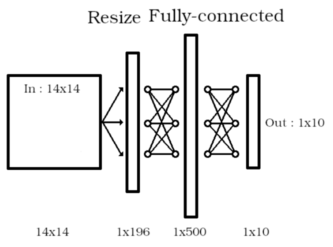

Figure 2: Shallow F.C. net : Pipe

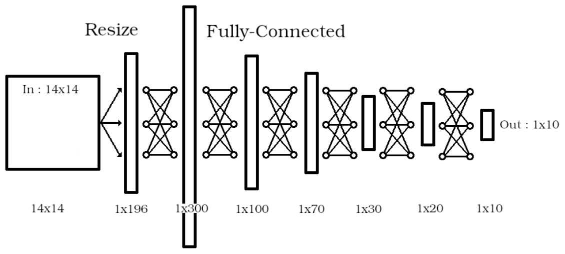

Figure 4: Deep F.C. net : Pipe

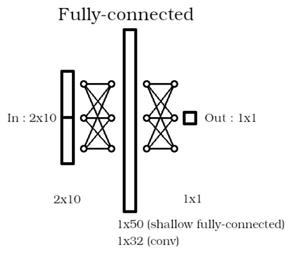

Figure 3: Shallow F.C. and Conv nets : Classifier

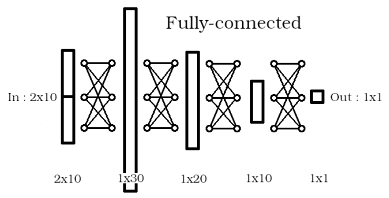

Figure 5: Deep F.C. net : Classifier

network with a 20 neurons input layer (concatenation of intermediate representations) and an hidden layer of 50 units. The resulting structure can be inspected in Figures 2 and 3.

Deep Fully Connected

In this architecture, the *pipes* trade width (the widest layer goes from 500 units in the shallow model to 300) for depth (from 1 to 5 hidden layers of decreasing width). The *classifier* from the shallow network is deepened as well. Note that the number of parameters is kept similar for both fully connected networks (around 200'000). The structure's schema is presented in Figures 4 and 5.

Convolutional

The *convolutional* network that was devised for this project is largely inspired by staple *ConvNets* such as *LeNet5* and *VGG16*. The *convolutional* and *MaxPool* layers substitute the fully connected parts in the two *pipes*, while the final classifier is still fully connected and mimics the *shallow* architecture, albeit with a smaller amount of hidden units (32). The complete architecture can be inspected in the Figures 6 and 3.

Model evaluation procedure

The evaluation procedure employed for each model is now briefly presented. As requested, each combination is trained 10 times. Considering that the provided functions allow to extract many different 1000-samples datasets from MNIST, the validation procedure is not a classical k-fold cross-validation performed with a single dataset, but new train and test data is rather loaded for each run. Additionally, model parameters are randomly initialized at the beginning of each training run according to the default layer initialization of PyTorch. During training, the complete training/test loss and accuracy histories are recorded. When applicable, the same quantities are

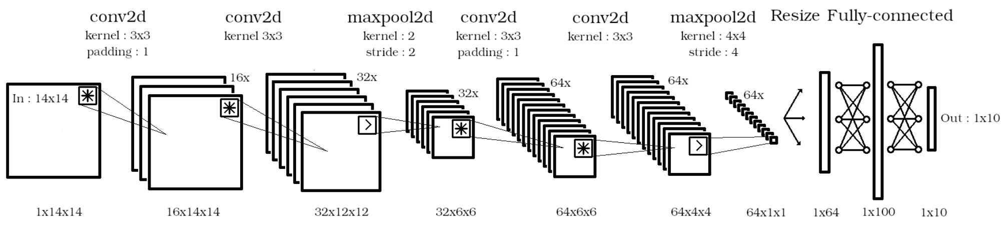

Figure 6: Convolutional net : Pipe

	WS: ✗ AL: ✗	**WS: ✓ AL: ✗**	**WS: ✗ AL: ✓**	**WS: ✓ AL: ✓**
Shallow FC	0.8205 ± 0.0149	0.8259 ± 0.0122	0.8312 ± 0.0055	0.8386 ± 0.0182
Deep FC	0.8118 ± 0.0169	0.8433 ± 0.0132	0.8879 ± 0.0122	0.8999 ± 0.0124
ConvNet	0.8506 ± 0.0115	0.8630 ± 0.0132	0.9026 ± 0.0137	0.9031 ± 0.0167

Table 1: Mean test accuracy (± std) of the architecture/techniques combinations over 10 runs

recorded also for the auxiliary task. The models are trained with 0.001 learning rate and for 50 epochs to guarantee convergence, but the best accuracy results achieved are reported (a sort of *a posteriori early stopping*). The reported figures for each model consist in the mean test accuracy over 10 runs and the respective standard deviation.

Results and conclusions

All the test accuracy results obtained with the above evaluation procedure are summarized in Table 1. Note that it would have been interesting to also report training times and qualitatively analyze training histories, but the presentation of those is left to the code (*test.py* and companion notebook) due to the limited size of this report.

Architectures

In general, the results highlight that the *convolutional* structure of the pipes is the best architecture among the three presented above. This should not be a surprise, since *ConvNets* are regarded as the go-to approach for computer vision tasks. It outperforms the *fully connected* networks for all combinations of techniques, especially when no technique is used, where the architecture alone provides 85% accuracy, a whole 3% more than the second best architecture. Surprisingly, the base *shallow net* sightly outperforms the *deep* counterpart with no techniques, but the latter receives the best benefits from the introduction of *weight sharing* and the *auxiliary task*, challenging the *ConvNet* at almost 90% accuracy when both are introduced and proving quite clearly to be the second best architecture for the task. This is because the *shallow model* barely improves when implementing the two techniques (a maximal improvement of 1.81%, compared to 8.81% and 5.25% for the other two networks).

Techniques

Table 1 also clearly showcases that the introduction of both *weight sharing* and the *auxiliary loss* empower the models, no matter the architecture or the combination. Nevertheless, the two techniques have different impacts and also behave differently on the three networks. *Weight sharing* seems to be less beneficial, providing improvements of (0.54/3.15/1.24%) respectively on the three nets when employed alone, but it is worth remembering that it brings the additional benefit of reducing the complexity and the number of parameters in the model. Adding the *auxiliary loss* in isolation provides instead a better (1.07/7.61/5.20%) boost in accuracy. Note that the *convolutional pipes* easily achieve more than 90% accuracy on the *auxiliary task.* For all architectures, introducing both techniques at the same times provides the best accuracy improvement from the base model (1.81/8.81/5.25%), but we should note that the boosts are not additive and we have diminishing returns when using both. In fact, the *auxiliary loss* alone provides almost the same benefits of the two combined.