EE559 Miniproject 1 Report

Olesia Altunina (285467), Mauro Pfister (235440), Joey Zenhaeusern (226652)

Introduction

The goal of this project was to implement a neural network to tackle a specific binary classification problem and assess the performance improvement achieved by employing weight sharing or/and auxiliary losses. The task was, given an ordered pair of MNIST samples, to predict whether the first digit is less or equal to the second.

Networks

To test the performance due to different architectures, we tried three basic network types. The first one, **BasicNet1**, is an ordinary CNN (CNN "feature detector" + MLP "classifier") that takes a *2 x 14 x 14* tensor as an input and makes predictions solely based on the binary targets. The second one, **AuxNet1**, is a CNN with weight sharing and auxiliary losses that make use of the digit classes available. The inputs, two separate *1 x 14 x 14* tensors, are fed into the CNN + MLP to make digit predictions, which are then concatenated into a single vector and fed into the binary classifier (another MLP). Finally, the last architecture that we used, is implemented in the **AuxNet2** class. This constitutes an Inception-like architecture that takes in two separate tensors, runs them through a "feature detector" (CNN), and then branches the outputs into two classifiers, one for digit recognition, the other for binary classification.

AuxNet3 is similar to **AuxNet2** but has 2 convolutional and 3 linear layers (vs 3 conv and 2 linear layers for the previous models). This one was built to check how the distribution of layers might influence the performance.

In the last two architectures, the loss is cumulative of the three losses: the losses of the first and second digit recognition and the binary classification loss.

We used cross-entropy losses due to classification character of the problem, Adam optimiser as the one that converges the fastest with less parameter tuning, and 0.5 dropout on all linear layers for regularisation.

Training and Optimisation

Implementation

Model is the child class of *torch.nn.Module* that allows to perform the training of multiple runs of the same network with different parameters, in this particular case the following:

- **Network parameters** such as the number of channels for convolutional layers and the number of hidden units for linear layers,
- **Learning rate**,
- **Batch size**.

Running *src/test.py* with no arguments reproduces the training of the fastest sufficient model. When adding *--train_best* argument, it trains the best model (takes longer). When adding *--reproduce_all* argument, it reproduces the complete process of training and optimisation.

Optimisation Protocol

To optimise the network parameters, we generated for each net a list of all possible combinations of sizes with values in {16, 32, 64, 128, 256, 512} that yielded the number of trainable parameters

between 65000 and 75000. For each of these sets of parameters, we trained the corresponding nets 5 times. The learning rate was fixed at 0.005 and the batch size at 100.

To optimise the learning rate and the batch size, we picked the best model with the best mean validation accuracy and varied learning rate in {0.0001, 0.0005, 0.001, 0.005, 0.01} with the fixed batch size of 100, and batch sizes in {25, 50, 100, 200, 500, 1000} with the fixed learning rate of 0.005.

Results

The network parameters for the 4 described nets can be found in Fig. 1. The discrepancy in mean validation accuracy is between ~58% and ~94%. It is worth noticing that among the best results there are the models with varying convolutional layers sizes (32-128-64, 128-32-64), and that the lower the accuracy, generally the more the variance of the sample. Among the 4 models, the best is AuxNet2, AuxNet3 being ~5% behind, which suggests that having more convolutional layers, and not linear ones, may be advantageous.

The best learning rate and batch size are shown in Fig. 2.

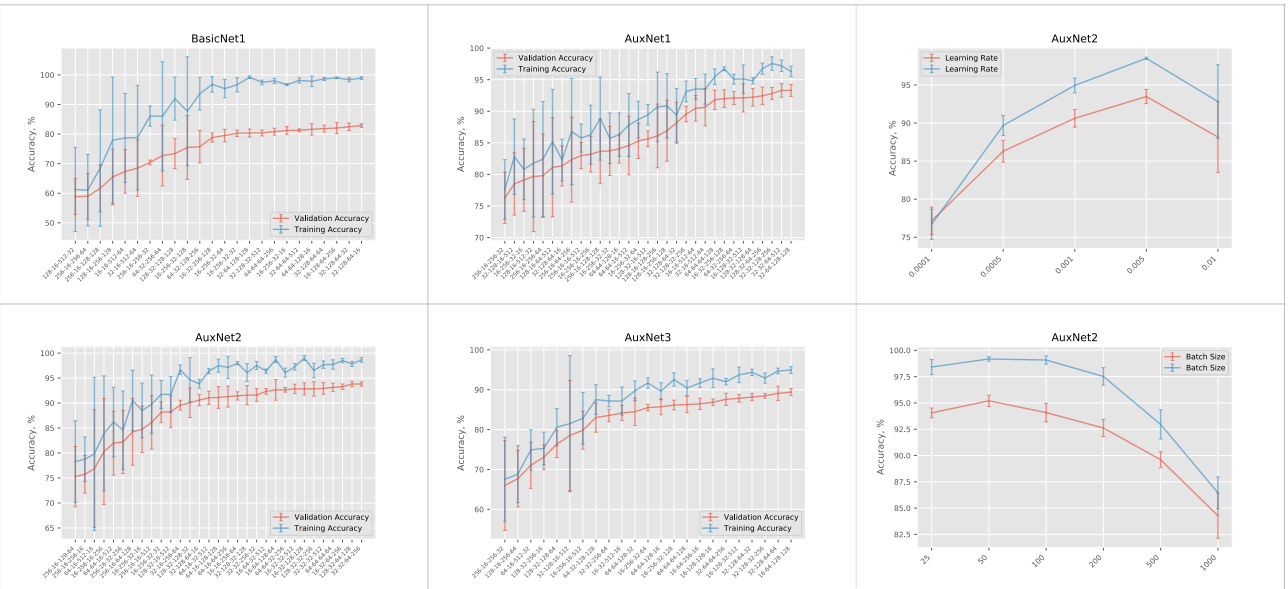

Fig. 1: Best Accuracy and the network parameters.

Fig. 2: Best learning rate and batch size.

Conclusion

Indeed, shared weights and auxiliary losses are helpful for the tasks where the decision is made based on a few separate inputs. But an ordinary CNN with the inputs stacked as channels and with no auxiliary losses also performs better than naive, implying that there are some shared patterns that a CNN can discriminate.