EE-559 Project I

Gloria Dal Santo, Erick Maraz, Daniel Filipe Nunes Silva

***Abstract*—This project report presents four different deep learning solutions to an image classification problem. The problem consists in, given a pair of MNIST images, classifying whether the first digit is greater or smaller than the other one. A particular attention is given to the use of weight sharing and auxiliary losses as a tool to improve the performance.**

I. Introduction

To carry out the classification problem and to investigate the effects of auxiliary losses and weight sharing, we started by designing two different baselines: a Multilayer Perceptron (MLP) and a simple Convolutional Neural Network (CNN) plus Feed Forward Network (FFN) as final classifier. We then gradually improved the architecture of the latter by adding auxiliary losses and weight sharing to get to the final, best performing model.

A description on the available dataset and the employed data loader is given in Section II. In Section III the implementation of the CNN and MPL based baselines is discussed, followed by the implementation of more complex models employing auxiliary losses (Section IV) and weight sharing (Section V). The evaluation and comparison of the presented architectures is covered in Section VI, which presents the final results.

II. Data and Tuning

A. Pre-processing and Loading

The training and testing datasets consist in 1000 pairs of MNIST 14×14 images, each depicting a gray scale digit. Each sample is a $2 \times 14 \times 14$ tensor, with associated target (class to predict $\in \{0, 1\}$) and corresponding classes (classes of the two digits ($\in \{0, \dots, 9\}$). To ease the loading process, we use the PyTorch's `torch.utils.data.DataLoader` object. This solution is usually justified by problematic large datasets, which is not the case of this project. On the contrary, we decided to use the `DataLoader` to lighten the main code and to ensure that the data is being reshuffled at each epoch, making the trained model more robust. Data initialization is randomized by `mnist_to_pairs` function in `dlc_practical_prologue.py`. All the employed layers are taken from Pytorch's modules `torch.nn` which ensures that the weights initialization is also properly randomized.

We normalize the dataset by subtracting its mean value and dividing by the standard deviation. Before normalization, the value of the pixels ranges between 0 and 255 which may cause the model to learn large weight values. Normalization reduces and centers the pixel values, decreasing considerably the risk of instabilities that large weight values can cause. Moreover, having standardized data may allow us to a larger learning rate, speeding up the training process.

B. Hyper-parameter tuning

In the design of the models many hyper-parameters are involved, among which: the number of hidden units, the value of the learning rate η, batch size, and dropout probability. To ensure that the combination of hyper-parameters is the closest to the optimal, we perform binary search. The new model, created with the newly generated combination of parameters, is trained on 800 samples taken from the train set and it is tested on a validation dataset composed of the remaining 200 samples. Binary search has proven to be a faster alternative to grid search, which was originally implemented, at the expense of a less exhaustive search.

We restricted the decision on the optimization algorithm to Adam and Stochastic Gradient Descent (SGD) algorithms. To investigate which one among Adam and SGD gives better performance on our models, we first ran a binary search on the learning rate and then we compared the effects of the optimizers on the losses and error rates.

III. Baselines - BaseNet

As base models, we designed a simple MLP and a CNN plus a FFN block used as final classifier.

The detailed structure of the networks is depicted in Figures 2-4.

A. Design

The first implementation consists in a simple 3-layers MLP comprised of 2 blocks of Linear-BatchNorm-ReLU, and a final Linear layer. After testing the model on different configurations, we have decided to apply **batch normalization** after each layer as it has always proven to speed up the convergence of the cost function while reducing its variance and mitigating the vanishing gradient. This strategy has been applied also in the following models. The second implementation consists in a feature extractor made of 3 blocks of Conv2d-BatchNorm-ReLU, a MaxPooling operation, and a final classifier comprised of 2 blocks of Linear-BatchNorm-ReLU with a final Linear layer.

We tested both networks including also dropout for regularization. The use of dropout is usually justified by a reduction in the risk of overfitting a model on the training dataset. However, after several tests, we concluded that batch normalization made dropout unnecessary.

ReLU's activation function outperformed Tanh and Sigmoid on both models. We also tested the leaky ReLU activation to avoid the problem having dead neurons but we discarded it because it did not provide an overall superior performance.

For these and the following models we use **Cross Entropy Loss** as cost function to compute the losses. It combines log softmax and the negative log likelihood loss, making it well suited for classification tasks.

B. Performance

In the appendix, Figure 6 and 7 illustrate the evolution of the losses and accuracy over epochs. Both architectures show poor performance. In particular, from the trend of the test losses it can be deduced that the network is overfitting the training set. On the other hand, both networks show fast convergence and low standard deviation. When compared to the MLP, the CNN+FFN network exhibits an incrise in accuracy, from 80.94% to 82.08%

IV. Auxiliary Losses - AuxNet

We based our advanced models on the CNN+FFN network only, as it has shown better performance.

To improve the model, we consider two auxiliary cross-entropy losses computed on the available classes of the two digits ($\in \{0, \dots, 9\}$). The improved architecture can be divided into three main blocks:

- *feature extractor:* 4 blocks of Conv2d-BatchNorm-ReLU with MaxPooling operation before the fourth block.
- *0-9 classifier:* 2 blocks of Linear-BatchNorm-ReLU with a final Linear layer whose output is a 10x1 tensor aimed at classifying the input image according to the digit that it represents.
- *0-1 classifier:* as the 0-9 classifier, it consists of 2 blocks of Linear-BatchNorm-ReLU with a final Linear layer. The output is a 2x1 tensor which classifies the first digit as greater or smaller than the second.

Both feature extractor and 0-9 classifier are duplicated.

The input is split into two 1x14x14 tensors, each representing one of the two digits. Each partition feeds its own feature extractor and 0-9 classifier. The outputs of the 0-9 classifiers are then concatenated to form a 20x1 tensor which is then sent to feed the last classifier.

The auxiliary losses are summed to the loss computed on the final output. To investigate the relative importance of the auxiliary losses, we scaled the losses prior to the sum as follows:

$$\mathcal{L}_{\text{tot}} = \alpha\, \mathcal{L}_{\text{out}} + \beta\, (\mathcal{L}_{\text{aux}_A} + \mathcal{L}_{\text{aux}_B}) \tag{1}$$

The following table summarize the results obtained with some combination of α and β:

	α	β	Test **accuracy**	Test **std**
1	1.00	1.00	0.9560	0.0068
2	1.00	0.75	0.9521	**0.0046**
3	1.00	0.50	0.9465	0.0070
4	0.50	1.00	0.9608	0.0065
5	0.25	1.00	0.9643	0.0072
6	0.10	1.00	**0.9669**	0.0055

TABLE I: Different losses configurations

It turns out that, to obtain higher accuracy, the auxiliary losses should be weighted more with respect to the loss computed on the network's output. This result highlight the importance of the auxiliary losses in improving the network's performance.
In the final implementation we use the sixth configuration ($\alpha = 0.10$, $\beta = 1.00$)

While the baseline network show a faster convergence and slightly improved accuracy when using **SGD** optimizer, for this model we use **Adam** with learning rate $\eta = 0.001$ (and default parameters) as it provides an increase in accuracy of about 5%, 18% lower standard deviation, and faster convergence over the epochs with respect to SGD (prior to testing the model with SGD optimizer we have binary searched for the optimal learning rate).

V. Weight Sharing - SiameseNet

In the previous architecture we can identify two independent duplicated networks and a final classifier. In this model we remove one of the duplicated networks, which is composed of the feature extractor and 0-9 classifier, to assess the effect of weight sharing. The feature extractor and the two classifiers follow the same structure as before, apart from small changes in the number of hidden units. In this model, the input is still split into two tensors each representing a digit. The first portion of the network, made of feature extractor and 0-9 classifier, is feed with the two input tensors. The two digits are passed sequentially so that to obtain two intermediate outputs on which two auxiliary losses are computed. The two 10x1 tensors coming from the 0-9 classifier are then concatenated and sent to the final classifier.

As with the previous model, we tested different combinations of α and β in the computation of the loss in eq.(1). The accuracy shows a similar trend, increasing for smaller values of α. As the accuracy and standard deviation are almost constant for $\alpha \in \{0.1, 0.5\}$ we arbitrarily set $\alpha = 0.1$

VI. Results

The results are computed by taking the average of the losses and accuracy on 15 rounds. In each run we train the network on 30 epochs and we compute the losses and the accuracy on both train and test dataset. From the results, summarized in Table II, we can conclude that SiameseNet performs the best with a mean accuracy of 97.65% and standard deviation of $4 \cdot 10^{-3}$, confirming the initial assumption on the importance of weight sharing and auxiliary losses. AuxNet shows slightly lower results, with a decrease in accuracy of about 1.09% and increase training time of about 11.27%, showing that weight sharing improve the performance while speeding up the training. Both baselines present a test accuracy of about 80% and standard deviation of $12 \cdot 10^{-3}$.

As shown in figure 1, in SiameseNet the train and test accuracy stabilize after 15 epochs and from the trend of the losses we can deduce that we are not overfitting.

	train accuracy		test accuracy		
	mean	**std**	**mean**	**std**	train time (s)
BaseNetMLP	1.000	0.000	0.8094	0.0124	5.0999
BaseNetCNN	1.000	0.000	0.8208	0.0121	7.7502
AuxNet	1.000	0.000	0.9669	0.0055	18.7073
SiameseNet	1.000	0.000	0.9765	0.0044	16.8130

TABLE II: Results computed over 15 rounds

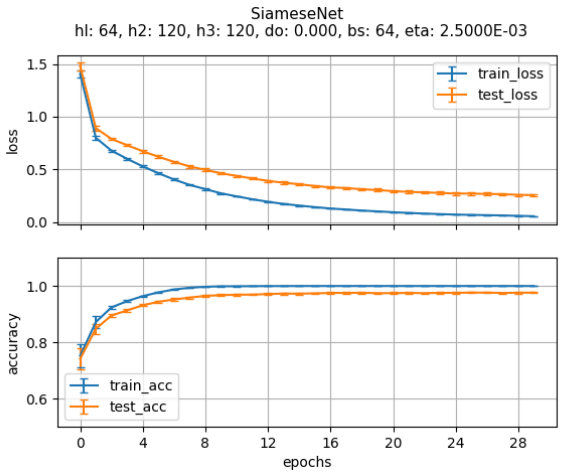

Fig. 1: Performace estimation - SiameseNet

VII. Conclusions

In the presented project we have investigated the performance of four different models for image classification. From the obtained results we can conclude that in image classification, feature extractor based on Convolutional Neural Networks outperform simple MultiLayer Perceptrons. CNNs apply 2D convolutions that preserve the structure of the signal, hence presenting an ideal solution when dealing with 2D images.

When extra information on the input samples is available, our results showed that these can be used to improve considerably the performance of the model by computing auxiliary losses. If moreover the input can be split into two different but equally meaningful input tensors, the model can be further improved applying a siamese-like structure.

APPENDIX A
NETWORK ARCHITECTURES

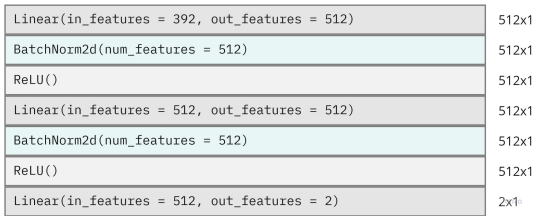

Fig. 2: MultiLayer Perceptron baseline network [111138 parameters]

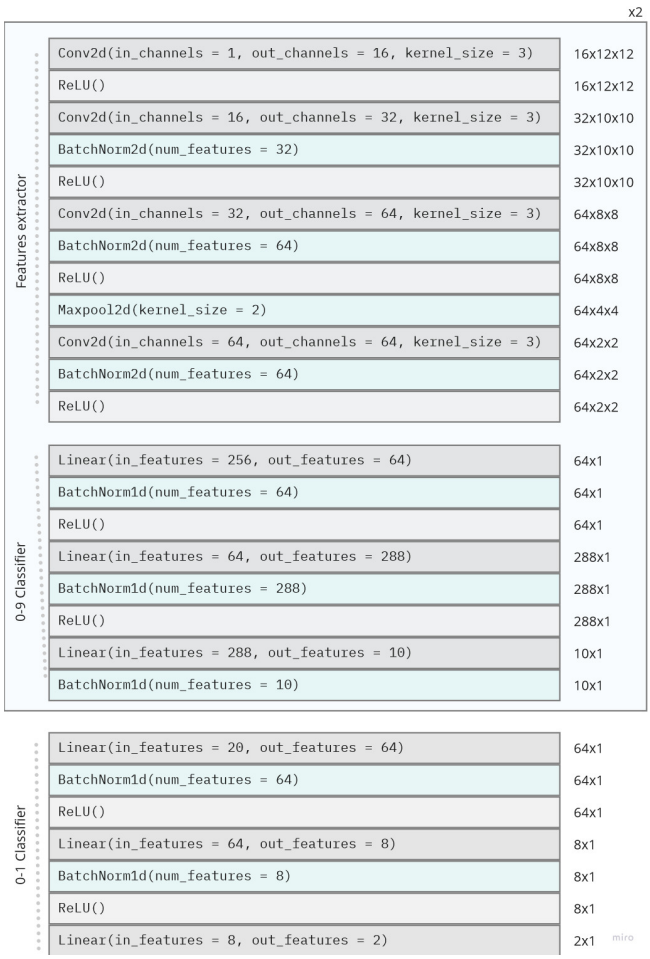

Fig. 3: Convolutional + Feed Forward Network network with auxiliary losses [466946 parameters]

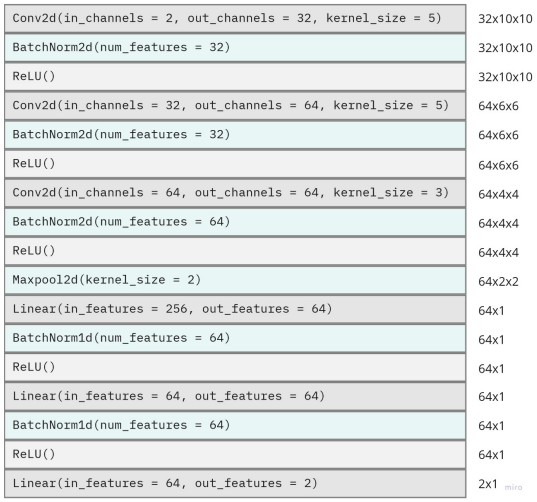

Fig. 4: Convolutional + Feed Forward Network network [200678 parameters]

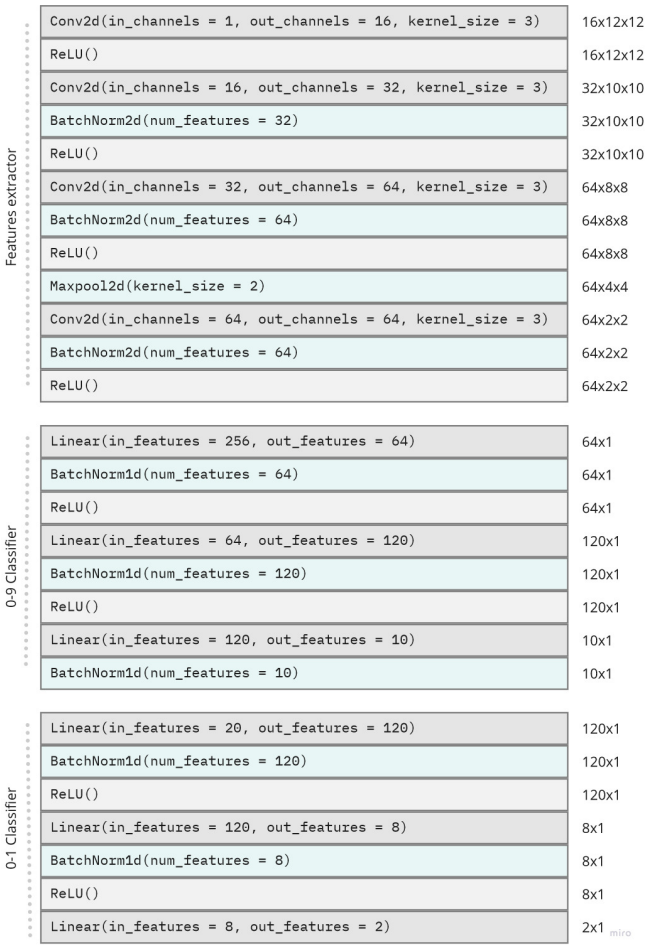

Fig. 5: Siamese network [90400 parameters]

APPENDIX B
PLOTS

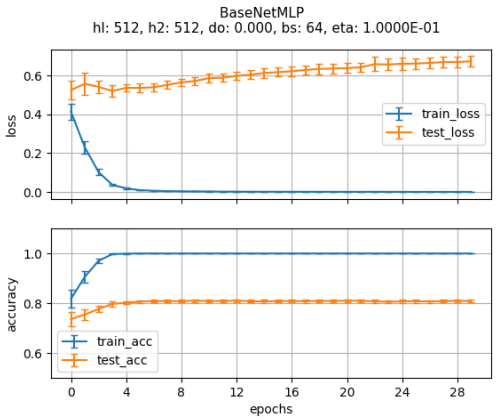

Fig. 6: Losses and accuracy of BaseNetMLP averaged over 15 rounds

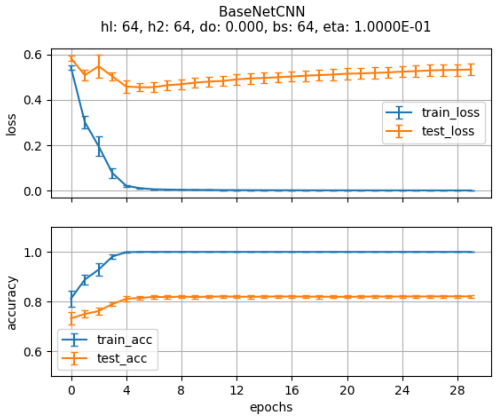

Fig. 7: Losses and accuracy of BaseNetCNN averaged over 15 rounds

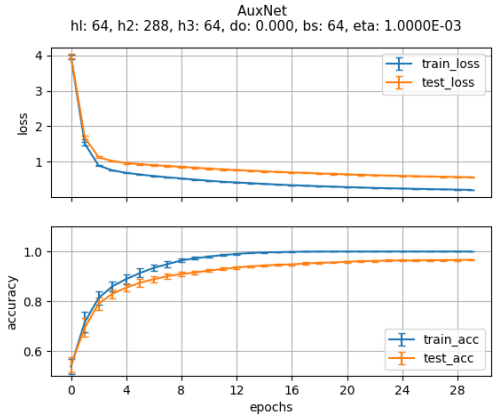

Fig. 8: Losses and accuracy of AuxNet averaged over 15 rounds