Deep Learning Project 1: Report

Raphael Bonatti
Lena Csomor
Marijn van der Meer
Department of Computer Science, EPF Lausanne, Switzerland

I. Introduction

As a part of the course 'Deep Learning', this group implemented different Machine Learning architectures that compare two digits displayed in an image. We compared the performance improvements are achieved by using specific architectures, different optimisers, weight sharing and auxiliary losses. We started with a baseline model, which is a very basic neural network, and evolved to increasingly sophisticated architectures such as CNNs, RNNs and Siamese Networks. While we tested many architectures that often were very similar to each other, in this report we kept only the ones we deemed interesting and that showed progress in our thought and coding process.

II. Methods

A. Data Generation

As input for the models, the famous MNIST dataset is used. The test and training sets consist of 1000 randomly chosen pairs of images, both 14x14 pixels of grayscale handwritten ciphers. So one sample is of 2x14x14 (Figure 1). For each pair, the model should predict whether the first digit is lesser or equal to the second.

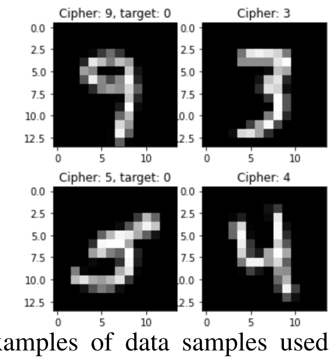


Figure 1: Examples of data samples used as input to the model. Pairs of grayscale cipher images from MNIST of size 14x14. The target variable of each pair is 1 when the cipher on the first image is smaller or equal to the cipher on the second image.

B. Evaluation Metrics

As evaluation metrics we chose to use accuracy and F1-score to assess the performance of our model on the test set. We used both measures in case the data set was not well balanced in terms of target values. We also displayed the average loss, which reflects how well the architecture models the data it is given. All performance assessments are aggregated values from at least 10 runs with randomly chosen training and test data sets and, if not indicated otherwise, 50 epochs of training.

III. Models

A. Baseline Model

The baseline model is a neural network with only one hidden linear layer. We used softmax as an activation function, the loss is calculated with cross-entropy and the optimizer used is SGD.
The network is fully connected, and the two images are flattened into a one-dimensional input tensor. The output consists of 2 logits, where the first describes the probability of the output being zero, while the second describes the output being one.
The performance of the baseline can be found in Table I. For such a simple model, its performance is already surprisingly good with an average accuracy of 77% and F1 of 79%.

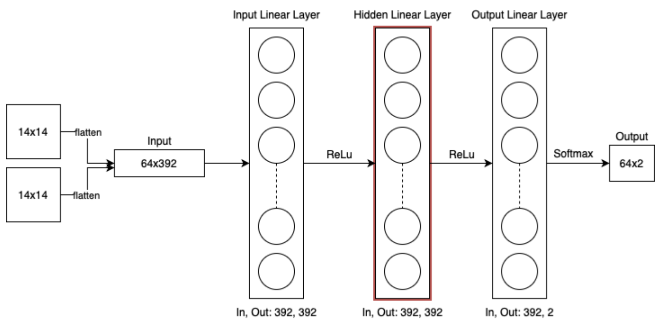


Figure 2: Visualisation of the baseline model, a fully connected network with one hidden linear layer.

B. Siamese Models

A Siamese Network contains two sub-networks that are trained in parallel. Each single network uses one half of the input for their computations, such that they output comparable vectors. The two sub-networks can, but do not have to, share weights. In our case, siamese networks are

advantageous because our datasamples come in pairs. This way, we can first perform computations on both individual sample of a pair, concatenate the results of both networks and add a final part that will assess whether one cipher is smaller or equal to the other.
We will present different siamese models and assess their performance.

1) Linear Siamese Network: At first, we only focused on simple siamese networks with only linear layers. The first model is a fully connected siamese network:

- Weight sharing: no
- Activation function: softmax
- Learning rate: $1e-3$
- Loss: cross-entropy
- Optimizer: SGD

The performance roughly matches the baseline model with approximately 75% accuracy and 77% F1-score (c.f. Linear Siamese Model 1, Table I).
This model was extended by adding shared weights (Figure 3). As can be seen in Figure 3, the two outputs of the siamese parts are concatenated, flattened and entered into a last linear layer before getting the final logits. With a 75% accuracy and 78% F1-score (c.f. Linear Siamese Model 2, Table I), adding weight sharing does not improve the performance of this model significantly nor exceed the baseline, so we concentrated on more sophisticated, deeper models.

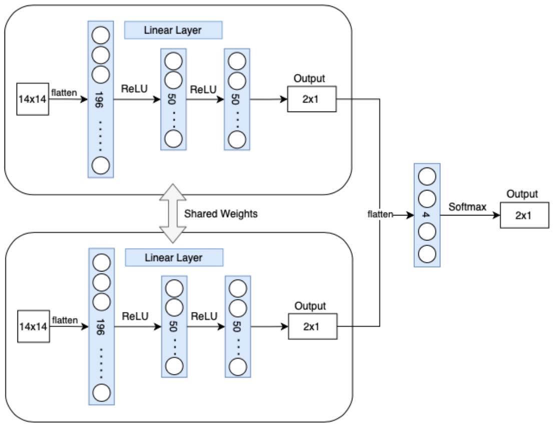


Figure 3: Visualisation of Linear Siamese Model with shared weights.

2) Convolutional Siamese Network: We replaced linear with convolutional layers which can reduce the total number of units in a network, and thus also reduced the number of parameters. Convolutional units also consider the context of their neighbourhood, which is very important in image processing task such as ours.

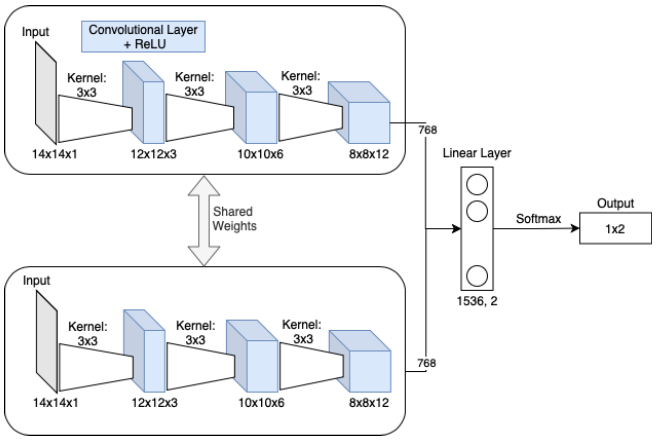


Figure 4: Visualisation of Convolutional Siamese Model 1 with shared weights.

Model 1: We started with a siamese network with 3 convolutional layers that is fully connected at the end:

- Weight sharing: no
- Activation function: softmax
- Learning rate: $1e-3$
- Loss: cross-entropy
- Optimizer: SGD

Architecturally, it is the same as Model 2 presented below, but without weight sharing. Even though this model is more complex than the baseline, the performance still does not exceed the baseline with 75% accuracy and 76% F1-score (c.f. Siamese CNN 1, Table I).

Model 2: In a next step, we added shared weights to the CNN siamese network *Model 1* (Figure 4).

- Weight sharing: yes
- Activation function: softmax
- Loss: cross-entropy
- Learning rate: $1e-3$
- Optimizer: SGD

This improves performance slightly with 77% accuracy and 79% F1-score (c.f. Siamese CNN 2, Table I), with an increase of around one percent compared to the baseline.

Model 3: Not satisfied with the results so far, we built a more complex model and integrated auxiliary losses (Figure 5). This model consists of a siamese network with convolutional layers and

- Shared weights: yes
- Loss: cross-entropy
- Optimiser: Adam
- Learning rate: $1e-3$
- Activation function: softmax
- Other features: max pooling and dropout

This model uses auxiliary losses, meaning each part of the siamese network first predicts the cipher on the image (inside of rectangle in Figure 5). The logits of these sub-parts are then concatenated and flattened and used as input to a linear layer that predicts the final binary logits. The auxiliary loss is used to evaluate the cipher predictions of the sub-parts of the model. For training, the total loss that is evaluated is calculated as the a weighted sum of all losses:

$$\mathrm{l} = w_1 * \text{aux-loss}_1 + w_2 * \text{aux-loss}_2 + w_3 * \text{final-loss} \quad (1)$$

Where aux-loss_1 and aux-loss_2 evaluate the cipher prediction and final-loss the final binary output prediction. *Model 3* uses $w = [0.8, 0.8, 1]$ as weights for the weighted loss. The performance of this model is much better and even higher than the baseline with 81% accuracy and 82% F1-score (c.f. Siamese CNN 3, Table I). We note also that the accuracy of the cipher predictions is already high with 79% accuracy.

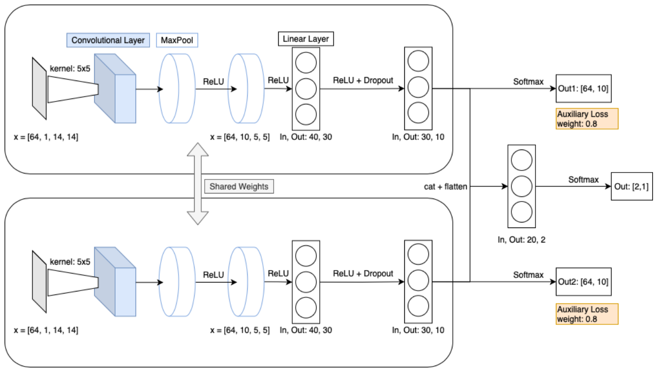


Figure 5: Visualisation of CNN Siamese Model with auxiliary losses.

C. *RNN Siamese Network*

In a next sequence of experiments, we tried siamese networks with a recurrent neural sub-network instead of a CNN. Recurrent neural networks have an internal state memory and are often used for tasks such as handwriting recognition for example, which we thought could match our task well. More specifically, we used Elman RNN's which consists of three layers and a set of context units that form the internal memory. We initialised the hidden weights to 0 and use tanh as activation function.The siamese network consists of two Elman RNN which are fully connected at the end. Their outputs are flattened, concatenated and put through another linear layer to give the final result (Figure 6). The model also relies on auxiliary losses.

- Weight sharing: yes
- Activation function: tanh in RNN and softmax for final logits
- Loss: cross-entropy
- Learning rate: $1e-3$
- Optimiser: Adam
- Weights for total loss: $w = [0.8, 0.8, 1]$ (Eq.1)

The performance is below the baseline at this point with 71% accuracy and 74% F1-score (Siamese RNN 1, able I). We see that compared to the previous CNN, the cipher accuracy is lower with 51%. Unlike to the CNN architectures that are run for 100 epochs, the Siamese RNN were run only for 50 epochs because after 50 it started to highly overfit to the training set.
We also experimented with the same architecture, but with a longer linear model at the end and tried training for a longer period of time. This did not improve the performance in any way with 71% accuracy, 74% F1-score and 58% accuracy on ciphers (Siamese RNN 2, Table I). We suspect that a reason why performance was not improved with a longer linear model is because the final output is mainly determined by how well the network recognises the individual ciphers, which is done in the RNN and not the linear layer.

D. *Best Model*

As our best model we returned to CNN siamese networks. However, this time we decided to go a step further and make it even deeper to improve Siamese CNN 3 (Figure 7). This model consists of 6 convolutional layers, each doubling the number of channels. In the middle, we also added batch normalisation to make the architecture more stable and dropout to reduce overfitting. We also use ReLU between layers and keep using dropout between the last three convolutional layers. This model also uses auxiliary losses.

- Weight sharing: yes
- Activation function: softmax
- Loss: cross-entropy
- Learning rate: $1e-3$
- Optimiser: Adam
- Weights for total loss: $w = [0.8, 0.8, 1]$ (Eq.1)

After training this architecture for 100 epochs, we finally reached an overall accuracy of over 85% and F1-score of 87%, while the accuracy in cipher prediction is 92% (Table I). The evolution of evaluation metrics and losses during over the epochs can be seen in Figure 8.

IV. Summary

In conclusion, the best model we found for this task is a deep siamese network CNN network. Our optimal model uses both auxiliary losses to predict the ciphers on the image pairs and a final loss to produce a binary output. It also uses shared weights between the siamese modules. This model provided the best final accuracy (85%) and cipher prediction (92%). Compared to RNN or linear siamese structures, CNN models took longer to train, but they reached better performances than the other architectures without overfitting. In further work to try to improve the accuracy, we could try going even deeper as this strategy seemed to work best. We should also explore other optimisers and data pre-processing or augmentation techniques.

V. Appendix

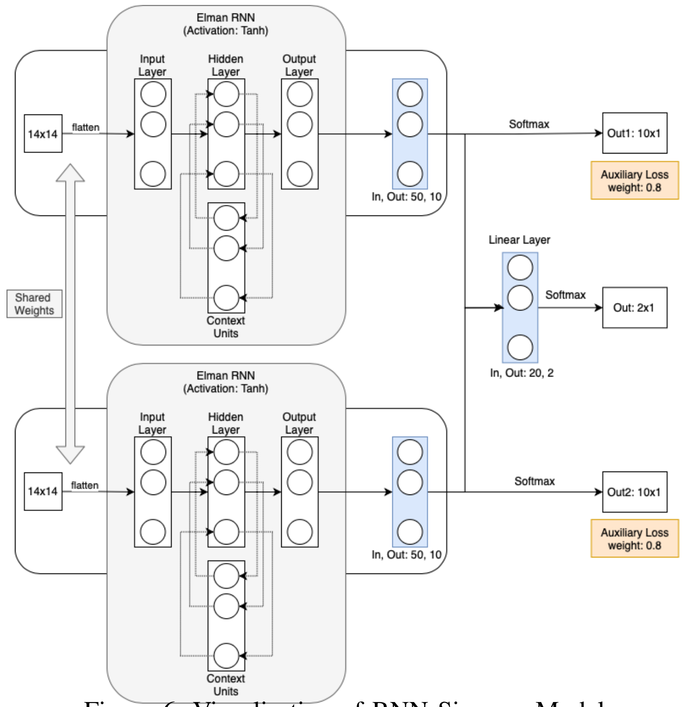


Figure 6: Visualisation of RNN Siamese Model.

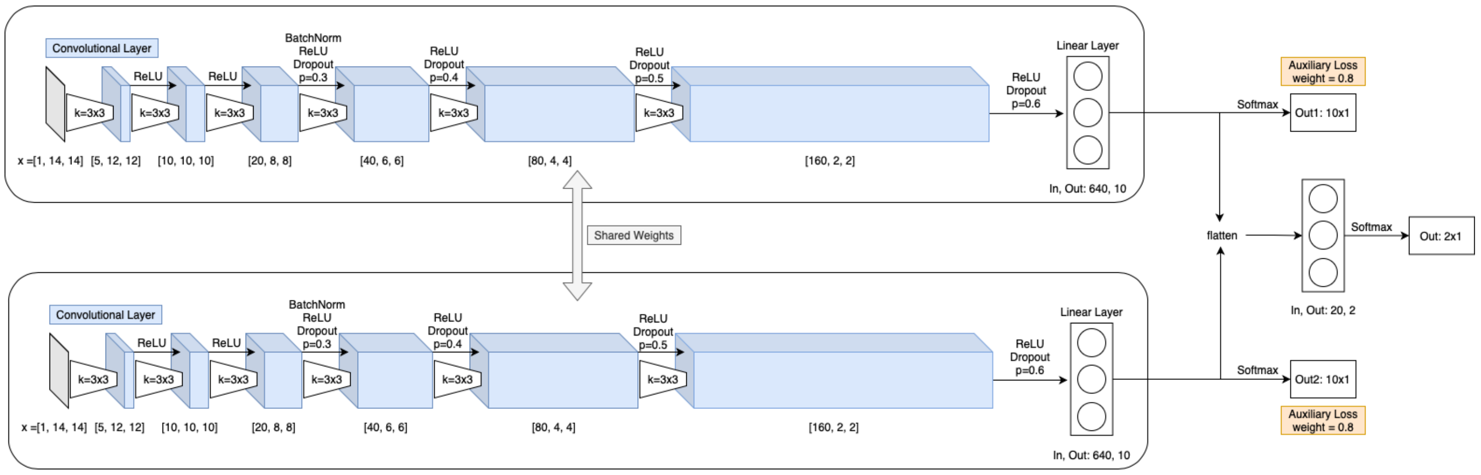
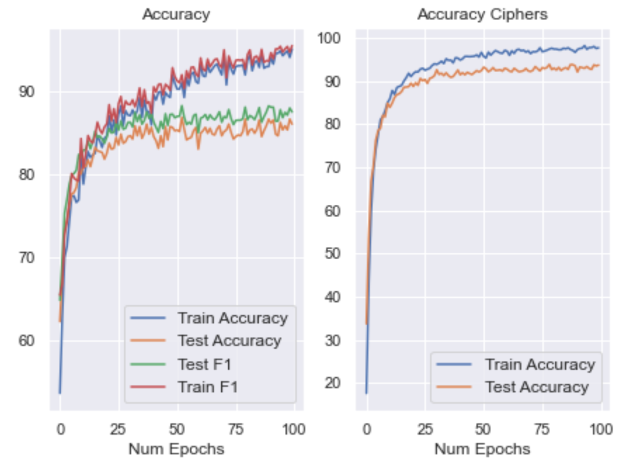
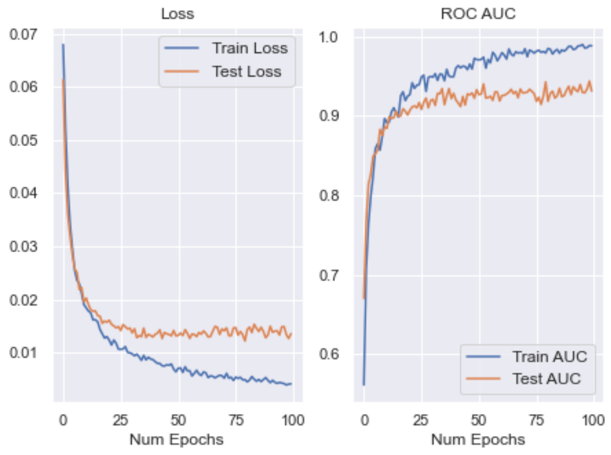


Figure 7: Visualisation of Deep Siamese CNN Model



(a) Evaluation metrics of Deep Siamese CNN over 100 epochs. On the left metrics on the final binary prediction and on the right accuracy on cipher prediction.



(b) On the left, taining (blue) and test loss (orange) for 100 epochs. On the right, AUC metric on training (blue) and test (orange) sets for 100 epochs.

Figure 8: Loss and evaluation metrics for training and testing of Deep Siamese CNN model (Figure 7).

Model	**Measure**	**Accuracy**	**F1**	**Cipher Accuracy**	**Loss**
Baseline	Mean	76.74	78.94	-	0.0110
	Std	1.3055	1.4073	-	0.0009
Linear Siamese Model 1	Mean	75.08	77.67	-	0.0086
	Std	1.5664	1.3844	-	0.0005
Linear Siamese Model 2	Mean	75.58	78.16	-	0.0084
	Std	1.5747	1.4938	-	0.0007
Siamese CNN 1	Mean	75.38	76.75	-	0.0088
	Std	1.6067	2.2532	-	0.0004
Siamese CNN 2	Mean	77.67	79.79	-	0.0083
	Std	1.5441	1.1749	-	0.0007
Siamese CNN 3	Mean	81.03	82.56	79.01	0.0243
	Std	2.9799	3.1201	4.8458	0.0031
Siamese RNN 1	Mean	71.40	74.17	50.85	0.0432
	Std	2.0244	1.2061	0.5553	0.0013
Siamese RNN 2	Mean	71.42	74.23	58.25	0.0501
	Std	2.0670	2.4225	3.5251	0.0020
Best Model	Mean	85.77	86.98	91.92	0.0184
	Std	0.7655	0.5811	0.5618	0.0019

Table I: Performance of all models explored in this report. Performance was assessed after 50 epochs for all except CCN models (Siamese CNN 1,2,3 and best model) which were run for 100 epochs for by averaging results on the test sets over 10 runs with randomly chosen training and test sets.