Homework 4

Daniel Castro (87644) and João Tiago Aparício (97155)

I. INTRODUCTION

In this experiment we aim to optimize a pre-existing architecture that classifies numbers in images on the MNIST data-set. This data-set has 70k examples of handwritten digits, Image size of 28x28, 1 channel and 10 output classes: [0-9]. To solve this classification task we used as a base model the one we were given in lab class 9. For this implementation we used Keras and Tensorflow on a Google colab notebook. This allowed us to not only simultaneously work on a same document but use a GPU as hardware accelerator.

II. MODELS

A. Feed-Forward Neural Network

As a baseline we used this network. One detail on implementation, dictated that we had to flatten the images before passing to the dense layer. And used a sparse categorical crossentropy for loss and a stochastic gradient descent for optimizing. The benchmark values obtained were loss: 0.2955 and accuracy: 0.9176.

B. Convolutional Neural Network

Given that the CNN's are known to be better at recognizing patterns in images, we focused on improving the given CNN instead of the Feed-Froward approach.

[we can show this empirically with the following results]

Convolutional Neural Networks: In the case of the CNN instead of flattening the input, we used a 2D convolution layer in conjunction with a max pulling. This allowed us to experiment with different kernel sizes and number of kernels.

To obtain a network with a high percentage of accuracy (higher than 99%) without being overfitted and a lower loss level. We made 10 experiments focusing on improving in each one a specific parameter. Those 10 experiments were, number of hidden layers, number of convolution-subsambling pairs, number of filters, kernel size, activation of the convolution, hidden layer size, activation of the hidden layer, dropout, optimizer and regularization. In these 10 experiments, we applied a greedy strategy in the sense that we did not test all combinations of parameters nor all the possible values but we experimented with one parameter, with different values (low, medium, high values) and fixed all the other parameters of the network. We changed the 10 parameters that in our opinion were the one that affected the network the most.

We can also dived this 10 experiments in 3 sub categories, the convulutional layer, the hidden layers and the regularization layer but these subsections are not sequential, i.e., not all the experiments of hidden layers are sequential.

Our baseline was the following: an input layer with 784 neurons (28 x 28), a 2D convolution layer with 16 filters,a kernel size of 4, ReLu activation and same padding, then a 2D Max Polling with polling size of 2, dropout as 50%, 1 hidden layer with 32 neurons and an output layer with 10 neurons and a softmax activation. We did not change the input layer because it is 28x28 the image size, and if possible we do not want to lose information in the input. We did not changed also the output layer, since this is a classification problem and we want to know what number the image represents. And softmax activation is good for classification problems such as this one, rather than a linear function for instance. As softmax outputs produce a vector that is non-negative and sums to 1. this is useful when you have mutually exclusive categories (i.e. This is 1 hence it it cannot be a 3). There are a few parameters that we did not change, for example padding on the convulutional 2D, since we concluded that it would not be useful to have circular convolution wrap-around effect because this would negatively alter the desired effect of the convolution.

We started by testing the parameter in the homework 4 statement: different number of hidden layers. The results for this test were:

- 1 Hidden Layer: loss: 0.0335 - accuracy: 0.9892
- 2 Hidden Layer: loss: 0.0357 - accuracy: 0.9889
- 3 Hidden Layer: loss: 0.0321 - accuracy: 0.9894

Following the Occam's razor Principle, we concluded that 1 hidden layer would be enough since more layers increase the complexity but do not increase accuracy or decrease loss enough.

Then we tested the convulutional layer. We tested the number of convolution-subsambling pairs:

- 1 Pair: loss: 0.0372 - accuracy: 0.9868
- 2 Pairs: loss: 0.0220 - accuracy: 0.9930
- 3 Pairs: loss: 0.0210 - accuracy: 0.9930

The number of filters:

- 16 Filters: loss: 0.0236 - accuracy: 0.9930
- 24 Filters: loss: 0.0261 - accuracy: 0.9935
- 32 Filters: loss: 0.0221 - accuracy: 0.9935
- 48 Filters: loss: 0.0211 - accuracy: 0.9939
- 64 Filters: loss: 0.0198 - accuracy: 0.9932

The kernel sizes test:

- 3 Kernel Size: loss: 0.0232 - accuracy: 0.9929
- 4 Kernel Size: loss: 0.0205 - accuracy: 0.9938
- 5 Kernel Size: loss: 0.0197 - accuracy: 0.9935
- 6 Kernel Size: loss: 0.0206 - accuracy: 0.9938

The convolutional activation functions:

- ReLu Activation: loss: 0.0221 - accuracy: 0.9931

- Tanh Activation: loss: 0.0250 - accuracy: 0.9920

The parameters that yielded better results were: 2 pairs of convolution-subsambling with 48 filters, kernel size of 4 in both convulution-subsamblings and the activation parameter chosen was the ReLu activation. We could have tested a different number of filters and kernel size for each convulution-subsambling but the goal of this exercise is not to be an exhaustive training of all combinations.

After the convulutional layer, we changed the hidden layer parameters: the number of neurons:

- 32 Neurons: loss: 0.0201 - accuracy: 0.9935
- 64 Neurons: loss: 0.0211 - accuracy: 0.9932
- 128 Neurons: loss: 0.0205 - accuracy: 0.9935
- 256 Neurons: loss: 0.0229 - accuracy: 0.9927

32 neurons was the best parameter since it was the less complex and with better loss and accuracy.

The activation functions tests:

- ReLu: loss: 0.0211 - accuracy: 0.9929
- Tanh: loss: 0.0219 - accuracy: 0.9935

For the activation we experimented with ReLu and Tanh, however both were similar so we chose the ReLu, as it avoids vanishing gradient problem and it is on average faster than Tanh.

The dropout (a type of regularization) that was initially at 0.5, we tested with 0.25, 0.5 and 0.75.

- 0.25: loss: 0.0649 - accuracy: 0.9908
- 0.5: loss: 0.0652 - accuracy: 0.9928
- 0.75: loss: 0.0807 - accuracy: 0.9929

The 0.5 was the best choice in our case because it had the best accuracy, less loss and smaller number of epochs. In the case of dropout, it clamps some weights to 0, stopping the flow of information through these connections, allowing for a better accuracy.

The optimizer parameter was tested using Adaptive Moment Estimation (adam) and Stochastic gradient descent (sgd). The sgd had a much worse performance than the adam optimizer.

- adam: loss: 0.0203 - accuracy: 0.9938
- sgd: loss: 0.0347 - accuracy: 0.9897

The regularization was an important parameter to test since it was requested by the homework 4 statement, however it did not improve our network.

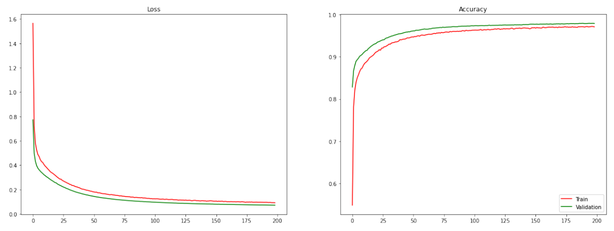
- L1: loss: 0.3977 - accuracy: 0.9871
- L1 and L2: loss: 0.4227 - accuracy: 0.9853
- L2: loss: 0.0737 - accuracy: 0.9906

Neither L1, L2 or the hybrid between L1 and L2 improved our network. The results yielded from those was actually not better than previously. Our theory is that as l1 and l2 regularization prevent over-fitting by shrinking on the weights, it creates more noise on prediction form less relevant weights. Early stopping is also applied, in this case we see a positive impact that avoids over-fitting.

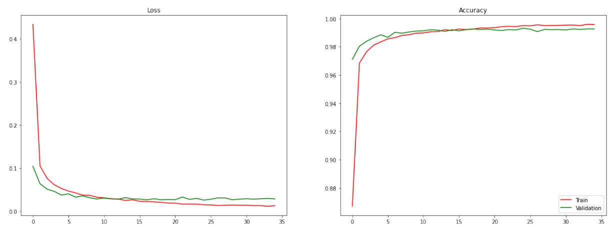
III. Results and Discussion

Our evaluation approach is based on the accuracy and loss both during validation and test. We also made use of validation split for selecting the validation set. In terms of better results, adding extra layers and extra complexity to the model does not necessarily positively impact the classification. And even if a very slight improvement was seen, a simpler model is always preferred, as it states in the Occam's razor Principle.

The following two sets of graphs represent the loss and accuracy of the baseline CNN model given and the CNN model we tweaked.



Initial/baseline CNN loss: 0.0631 - accuracy: 0.9802 training: 200 epochs



Best CNN loss: 0.0215 - accuracy: 0.9930 training: 33 epochs

As we can see by the validation accuracy of the graphs above, the effects of over-fitting are negligible, as we don't see an upwards tendency for loss in validation. Moreover, the accuracy was increased an the loss was reduced somewhat significantly. Also the number of training epochs for convergence is also lower. Yielding a faster training time. One reason for this could be for instance the use of ReLu instead of Tanh.

IV. Conclusions

In this set of experiments we were able to output an improved model to solve the classification problem in the MNIST dataset. Achieving not only better training performance but higher accuracy and lower loss through the use of a simpler network.

V. Source Code

Our source code is available at: https://colab.research.google.com/drive/1drMJBKO6UOzVaKH8Z4IctfEJKcloSLYp?usp=sharing#scrollTo=13tVOFWonpZa

Source code for this experiment was based on practical lab 9 available at: https://colab.research.google.com/drive/1GuOWptT1DEo1qdZgIJ78TLPGDso_JWJb.