Introduction to Deep Learning

Assignment 2

Yven Lommen
s2040964

Emilio Sánchez Olivares
s2985144

Leiden Institute of Advanced Computer Science, Universiteit Leiden

1 Introduction

In this report we describe three experiments that were performed for the second assignment for the course Introduction to Deep Learning. The assignment consists of three tasks. The first task focuses on doing experiments in Keras and Tensorflow in order to gain some practical experience. In the second task we take the setups that achieved the best results in the first task and apply them to permutated data. Meaning that we took the input pixels and then shuffled the values around to see if we get the same results. In the final task we try to implement a tell-the-time convolutional neural network (CNN) which tries to tell the time based on photos of an analog clock. For the first two tasks we used two data sets of both 70 000 images (MNIST and Fashion MNIST). For the third task we used an data set consisting of 18 000 images. [1]
First, in Section 2 we will discuss the experiments to get familiar with Keras and Tensorflow and the results of these experiments. Then, in Section 3 we reuse the setups from the first task and redo experiments using the permutated data and compare the results of Section 2. Further, in Section 4 we explain our setup for the different networks and approaches to tackle the tell-the-time problem and discuss our results. Lastly in Section 5, we make our final remarks and conclude our findings.

2 Experiments with Keras and Tensorflow

To begin with, we set to experiment with the previously mentioned datasets using Keras. For this, we test different setups of MLPs (multi-layer perceptrons) and CNNs (convolutional neural networks. MLPs are trained using 10 epochs, while we reduce the CNN epochs to 5 due to computation. The previously mentioned datasets refer to two collections of images of 28x28 gray-scale pixels, MNIST contains single digit numbers and fashion MNIST clothing items, both resulting in 10 categories. Thus, the networks will be trained for a classification task to identify input images and correctly label them.

We start by splitting the data into train, test and validation sets. Of the 70,000 images, we use 10,000

[1] `https://surfdrive.surf.nl/files/index.php/s/B8emtQRGUeAaqmz`

to test, 6,000 to validate (10% of the remaining images) and 54,000 to train. For time and computation purposes, we limit the number of epochs to 10 and track the loss and accuracy across each epoch.

Then, we proceed to compare the results between both datasets using a simple network[2]
for both MLP and CNN to see how the different datasets affect the networks. Next, using only the MNIST dataset (to avoid redundancies and save computation and time), we set to experiment with the architecture of both reference networks as follows:

- NLP
  - Number of hidden layers: 1, 2, 3 and 5 hidden layers using a ReLU activation function.
  - Weight optimizers: stochastic gradient descent (SGD), Adam, RMSprop and Adamax.
  - Activation function of hidden layers: rectified linear unit (ReLU), sigmoid, softmax and tanh functions.
  - Weight initializers: Random Normal, Random Uniform, Zeros, Ones.
- CNN
  - Number of filters: initial filters 16, 32, 64 and 128 using 3 2D-pooling layers and 2 dense layers.
  - Pooling sizes for 2D-pooling layers: 0, 1, 2 and 3.
  - Hidden layers using 128 initial filters and ReLU activation in the following configurations:
    - 1 2D-pooling layer and 1 dense layer.
    - 3 2D-pooling layers and 2 dense layers.
    - 1 2D-pooling layer and 3 dense layer.
    - 4 2D-pooling layers and 1 dense layer.

.

2.1 Results

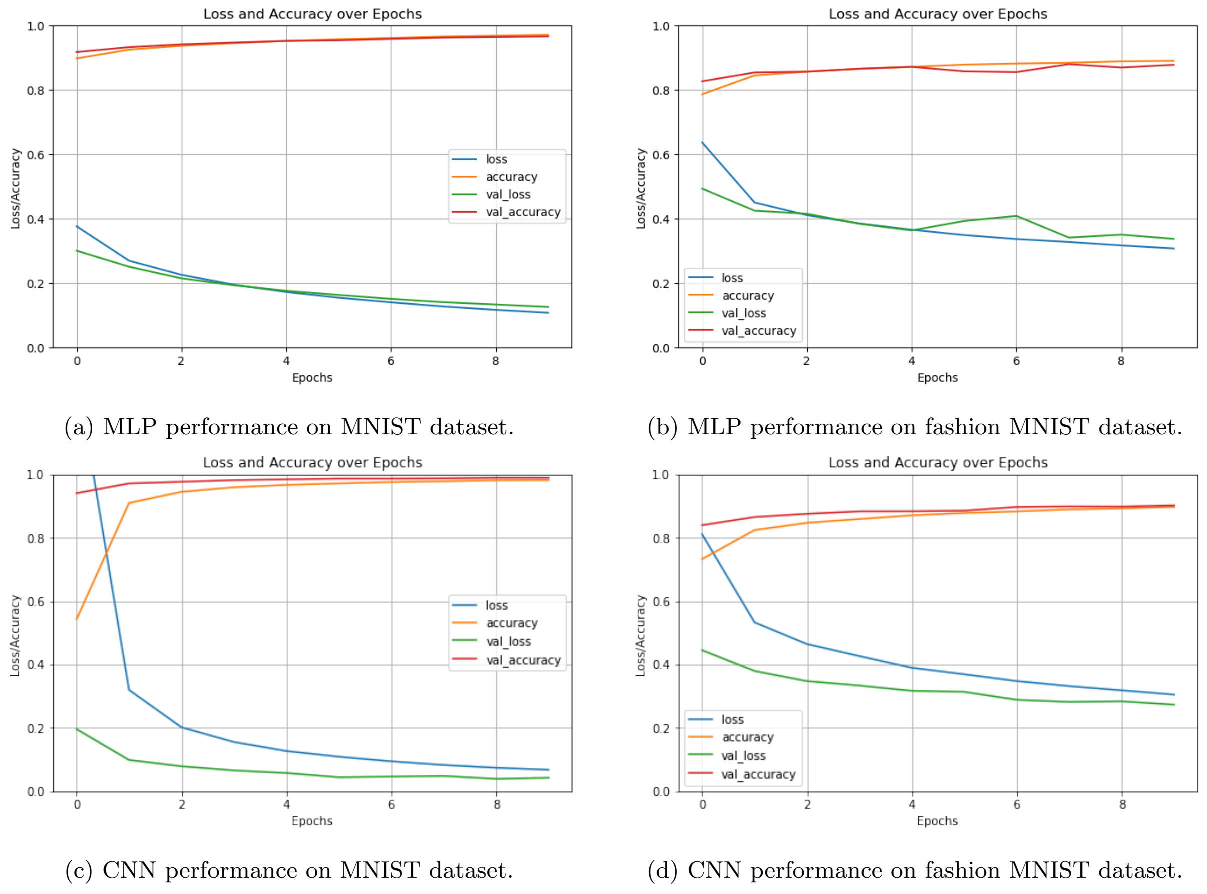
2.1.1 Initial MLP vs CNN

We start by comparing the performance of MLP and CNN networks on both the MNIST and fashion MNIST datasets. Figure 1 shows the history of the accuracy and loss over 10 epochs that were used to train the networks.

At first glance, it is evident that the MLP has a very consistent learning, meaning it starts at a somewhat high performance and improves slightly over the iterations, whereas the CNN starts with a lower performance and takes a few iterations to improve. Additionally, both networks perform better on MNIST than on the fashion MNIST, since the former have accuracies close to 1 and losses under 0.1; while the accuracies for the latter are closer to 0.9 and losses around 0.3.

[2] `MLP with 2 hidden layers: CNN with 2 pooling layers and 3 dense layers.`

Hence, we can say that networks have a better time classifying MNIST than Fashion MNIST. This is likely because the input images from MNIST are sharper than Fashion MNIST due to the small number of pixels that can show a single digit number better than an item of clothing.



(a) MLP performance on MNIST dataset.

(b) MLP performance on fashion MNIST dataset.

(c) CNN performance on MNIST dataset.

(d) CNN performance on fashion MNIST dataset.

Figure 1: Plot of the learning performance of the networks trained on original datasets.

2.1.2 MLP Experiments

The results of the MLP experiments performed on the MNIST dataset are reported in Table 1 and shows the performance (loss and accuracy) of each configuration after the first epoch and after the validation at the end of training.

When it comes to the hidden layers, there is not much difference with the configurations other than more hidden layers tend to have slightly worse performance after the first epoch, but at the end of training the performance is similar among all.

Moreover, the weight optimizers do show different performances. Both SGD and RMSProp have a slower start than current optimizers like Adam and Adamax. Nonetheless, at the end of training they all perform similar except for SGD that shows a higher loss.

As for the activation functions, we took some liberties to observe how the selected functions behave. First of all, it is clear to see that the results are poor when Softmax is used as activation of the hidden layers. Softmax is commonly used as the activation function for the output layer due to it's nature to return class probabilities. The sigmoid function also shows its flaws during the initial iteration, and although it makes up for it, the validation performance is still not on par with one of the most popular activation functions right now, ReLU, which has a consistent performance from start to end. Lastly, tanh appears to start on a similar performance (even a little higher) than ReLU but doesn't improve as much when reaching validation.

Furthermore, weight initializations apparently don't have a great effect in this task. Although random initializations (whether normal of uniform) have better accuracies at first, they also have lower loss. At the validation stage however, the performances are very similar. This is expected since at each iteration all of the weights are updated and at the end they optimized in a similar way.

2.1.3 CNN Experiments

Table 2 shows the performance (loss and accuracy) of the CNN experiments performed on the MNIST dataset after the first epoch and after the validation at the end of training on each configuration.

To begin with, the more initial filters we use, the better the initial accuracy, but loss doesn't seem to be affected since there appears to be no relation to its behavior compared to the number of filters. Nevertheless, during the validation it is clear that more filters perform better both for loss and accuracy. It is worth mentioning that in this experiments, the filters doubled throughout the layers, always in line with the reported initial filter number reported.

Likewise, while the pooling size doesn't seem to have an impact in the validation performance, in the first epoch smaller pool sizes seem to perform better.

Moreover, when it comes to the configuration of hidden layers, it is curious that the simplest network (1 pooling layer and 1 dense layer) has significantly the best initial performance. At the point of validation, all the configurations perform quite similar.

3 Experiments with permutated data

In this section, we will firstly discuss how the data was permutated and later on we discuss the results of the experiments, which were similar to the ones in Section 2 only using the permutated data instead of the original. Hence, we also started by splitting the data in the same way before training the networks and tracking their performance throughout the epochs. However, after an initial run, we saw low performance in training and decided to increase the number of epochs to 30 in order to have a better understanding of how the learning was achieved.

Regarding the network set ups, an MLP was trained with two hidden dense layers using a ReLU activation function; whereas the CNN was trained uses three hidden pooling layers, two dense layers, and two dropout layers, as well as a ReLU activation function.

3.1 Permutation

As a second task, we investigate the effect of permutation on the input data. Both the MNIST and Fashion MNIST data set consist of 28x28 grey scaled images. Each image is represented as an 28x28 matrix. In order to shuffle each image we looped through the data set and for each image we did the following to steps:

- Shuffle the rows of the matrix randomly.
- For each row in the matrix, shuffle the values in the row randomly.

With this permutated data we did redid some of the experiments of Section 2 and the results are discussed in the next section.

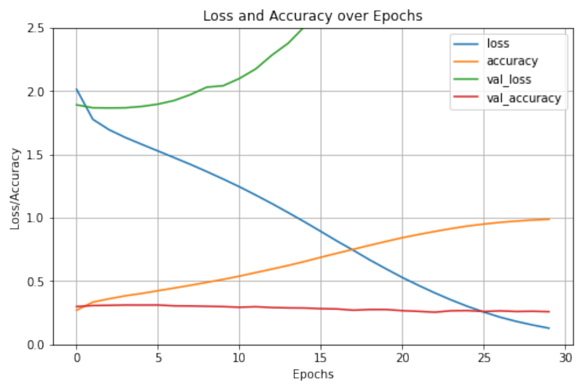
3.2 Results

After training the networks using the permutated datasets, we can definitely see differences in performance in contrast to the original network. As seen in Figure 2, the networks have a hard time initially understanding the data, as evidenced by the low accuracy and high loss registered in the first epochs. And while the MLP is somewhat able to overcome this (although after several more iterations), the CNN is not able to improve as much.

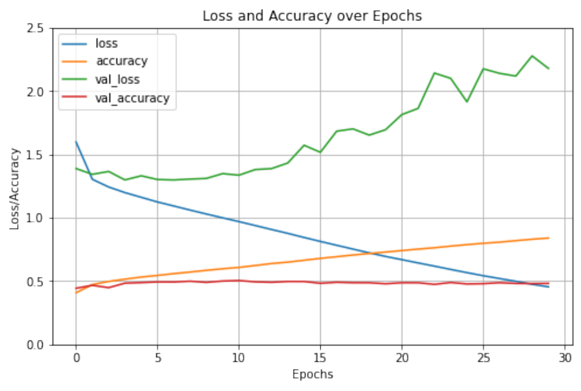
This is consistent with the notion of the networks, since CNNs are developed for image training, and since the pixels in the images have been rearranged, the local searches performed by the network don't provide significant information to match with similar images.

4 Tell the time network

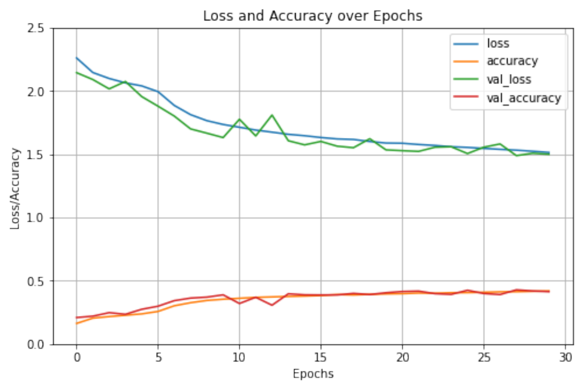
With the tell-the-time problem we refer to the challenge of creating a CNN which can tell the time from a picture of an analog clock. For this problem we focus on minimizing the so-called 'Common Sense Error' (CSE), which is the difference between the predicted time by the network and the actual time on the picture. We chose to express this difference in minutes. For the calculation of the CSE we figured that the maximal difference between the predicted and actual time must be 360 minutes (6 hours) since on an analog clock this is the furthers two times can be from each other. We accounted for this and adjusted our error measure accordingly, since simply subtracting the 'latest time' from the 'earliest time' will cause problems. For instance consider the two times 9:00 and 2:00, when 9:00 - 2:00 = 7:00, resulting in an seven hour time difference, but they are actually five hours apart. We used this CSE mainly in the regression approach described in Section 4.1 and the multi-head approach described in Section 4.3. For the classification approach, no exact time will be given so using the CSE is not convenient.



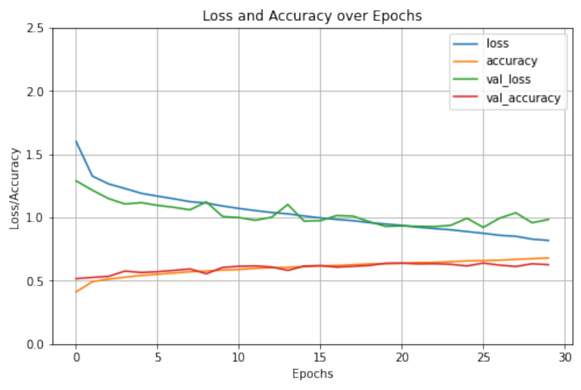
(a) MLP performance on permutated MNIST dataset.



(b) MLP performance on permutated fashion MNIST dataset.



(c) CNN performance on permutated MNIST dataset.



(d) CNN performance on permutated fashion MNIST dataset.

Figure 2: Plot of the learning performance of the networks trained on permutated datasets.

4.1 Regression

With the regression approach, we let the network output a single value. This value, $\hat{y}$, is an rational number thus $\hat{y} \in \mathbb{R}$. Thus for instance the time 03:30 will have the label 3.5. One problem for this representation is that the original data has to be re-labeled, doing so we have to round up some values when recalculating the actual time. This rounding can cause some false values and thus information loss. For this network we modeled it after the ideas of the ideas used in the textbook 'Hands-On Machine Learning with Scikit-Learn and TensorFlow', for the CNN used on the Mnist data set. The architecture is as follows:

- A convolutional layer with 64 filters, a kernel size of 7, and the same input shape as our input (150x150x1). Followed by a max pooling layer with pool size 2.
- Then we repeat the following structure twice: Two convolutional layers with 128 filters followed by a max pooling layer with pool size 2. In the second repetition of this structure, we double the filter size to 256.
- Then we flatten the inputs.

- Further, we repeat the following structure twice: a dense layer with 128 units followed by a dropout layer with a dropout layer of 50%. In the second repetition of this architecture the units in the dense layer are decreased with 50% to 64 units.
- Finally, we have one output layer with one unit.

We chose to use this architecture, since it performed pretty well for the Mnist data set. However, since this is a regression problem, we don't know how it will perform. For the loss function we chose the categorical entropy. We further used a stochastic gradient descent optimizer and the MSE and accuracy as metrics for the training.

4.2 Classification

With the classification approach we let the network output different classes. We used 24 classes, thus putting certain times in 'buckets' with each other. The first class for instance takes every label between 00:00 - 00:29, thus having 30 labels per bucket.

For the architecture of this network we chose the same architecture as for the regression approach discussed in Section 4.1. We again motivate this because the architecture performed well for the Mnist data set. This time it is more similar because it is again a classification problem. For the training we again chose for the same parameters of the regression approach. However, we left out the MSE metric since this is not a regression problem.

4.3 Multi-head model

With the multi-head model we create a network which outputs two values, one value for the hour and one for the minutes. This problem is similar to the regression problem, because both output heads will output a number.

For the architecture of the multi-head approach we chose a different one than for the other two approaches. It is given as follows:

- It starts with an input layer with the same input shape as the images (150x150x1).
- Then a convolution layer with 64 units and a kernel size of 3, followed by a max pooling layer with pool size 2.
- Then a dropout layer with a dropout ratio of 25%.
- Then the inputs are flattened.
- Then a dense layer with 128 units followed by a dropout layer with dropout ratio of 50%.
- Finally, both output heads are connected to the last layer.

For this training, we chose the same settings as previous with both the accuracy and MSE as metric.

4.4 Results

In this section, we shall discuss the results of the experiments conducted with the three approaches for the tell-the-time problem. We shall firstly discuss the experimental setup along with the data preprocessing. And after that we shall discuss the results for the individual networks.

4.4.1 Data preprocessing and experimental setup

The input data consists of 18 000 mages each of 150x150 pixels. However, a color channel was added for the CNN's thus resulting in input in the form of 150x150x1, because they are grey scaled images and having only one value. The input values were in a range of [0,255], we scaled this down to a range of [0,1] for normalization. The data was also split into a train and test set with a 80/20 ratio respectively.

For the experimental setup we chose to let each CNN train for 30 epochs and report the the score of the metrics on the test set. Each CNN was trained in this manner for five times, then the final score for each metric was the average of these five repetitions in order to receive stable results. For the metrics we firstly chose the earlier discussed CSE. Further we chose the Mean Squared Error (MSE) which is calculated as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{1}$$

where Y_i is the true label and $\hat{Y}_i$ is the predicted label. We further used the accuracy as our final metric, which is the amount of correctly classified instances divided by the total amount of classified instances. Note that we do not use each metric for every approach.

4.4.2 Regression

For the regression network the results were not good. The CSE had an average over 182 minutes, meaning that on average the predicted time differed three hours from the actual time. The maximum difference between two times on an analog clock is 6 hours. This implicates that the performance is arbitrary and not better than random guessing. The accuracy was 0,00 and the MSE was measured at 47,06. Needless to say that this network does not perform good by any means. Our best explanation for this is that the input is not consistent enough, since the pictures are taken from different angles and sometimes have glance on the clocks. Another explanation is that there is a bigger input shape than with the Mnist data set, however we tried adding convolutional layers because of the higher pixel density. But this yielded no better results.

4.4.3 Classification

The classification network did not perform better than the regression network, yielding a 0,04 accuracy on average. Seeing that there are 24 classes, this again means it does not perform better than a random classifier. Increasing the amount of layers again did not boost the performance. We did not increase the amount of classes since we did not expect better performance with more classes. We expect the bad performance for the same reason as discussed in Section 4.4.2

4.4.4 Multi-head

The multi-head approach again performed bad, the CSE was again 182. The accuracy and the MSE for the head that predicted the hour were 0,08 and 41,9 respectively. For the head that predicted the minutes the accuracy and MSE scored a 01,02 and 1167,8 respectively. We again think this bad performance has the same reason as in Section 4.4.2.

5 Conclusion

While the set-up experiments were only performed on MNIST, we expect similar (or even slightly worse) performance and results when using fashion MNIST according to what we observed in Section 2.1.1. In addition, in Section 2.1.2 we observed that the most popular state-of-the-art hyperparameters such as Adam and Adamax appear do to yield somewhat better performance of MLPs in this classification task and hence the reason for their popularity. Also, in Section 2.1.3 we found interesting the fact that for CNN in this particular case, a simpler model with fewer hidden layers performs better. While this experiment only consisted on observing behavior of different configurations, we would be interested in replicating the results to see if it was an isolated event, if more hidden layers result in some kind of over-fitting, or what could be the cause of this results.

In the end, it is worth noting that the aim of this task is to observe and describe different architectures of MLPs and CNNs for this given task, and while we can explain some of the behaviors seen, others are not likely to happen when training the networks for different tasks or with other datasets.

Regarding the permutation experiments, we observed that the nature of CNNs to locally analyze images is overwhelmed in comparison to MLPs. This is in line with the intuition that an image whose pixels are scrambled would be more difficult to correctly classify with the naked eye.

For the ’tell-the-time’ problem, none of the approaches unfortunately yielded good results. We applied the idea of transfer learning by reusing the architecture which we used in the first and second task, but the results were not good. We think this might come due to the inconsistent input images, for the MNIST and Fashion MNIST this data was a lot more consistent. Because the clock images were all taken from different angles and also could have glance. We think with more stable pictures the networks could have performed better.

6 Appendix

Experiment	Initial Loss	Initial Accuracy	Validation Loss	Validation Accuracy
Hidden Layers				
1 Hidden Layer	1.0321	0.7391	0.1643	0.9532
2 Hiddel Layers	1.0228	0.7276	0.1094	0.9675
3 Hidden Layers	1.0736	0.7014	0.0930	0.9709
5 Hidden Layers	1.2589	0.6369	0.1663	0.9497
Optimizers				
SGD Optimizer	0.5900	0.8510	0.1121	0.9664
Adam Optimizer	0.0999	0.9714	0.0780	0.9744
RMSprop Optimizer	0.0500	0.9865	0.0663	0.9787
Adamax Optimizer	0.0293	0.9931	0.0685	0.9784
Activation Function				
ReLU	0.989	0.7503	0.1121	0.9662
Sigmoid	2.2465	0.2437	0.3615	0.8983
Softmax	2.3018	0.1103	2.301	0.1135
tanh	0.8507	0.783	0.1545	0.9551
Initializer				
Random Normal	1.1236	0.7101	0.1227	0.9645
Random Uniform	1.1091	0.706	0.1209	0.9649
Zeros	1.1362	0.6987	0.1218	0.9632
Ones	1.1586	0.6912	0.1226	0.9634

Table 1: Results of MLP experiments. We report The loss and accuracy after the first epoch and the validation set.

Experiment	Initial Loss	Initial Accuracy	Validation Loss	Validation Accuracy
Filters				
16 Initial Filters	1.8841	0.3155	0.1113	0.9777
32 Initial Filters	0.4184	0.347	0.0661	0.9821
64 Initial Filters	1.2623	0.5694	0.0407	0.9883
128 Initial Filters	1.0856	0.6347	0.0437	0.9867
Pooling Size				
Max Size 0	0.8343	0.7271	0.0492	0.9858
Max Size 1	0.8401	0.7278	0.0523	0.9854
Max Size 2	1.3000	0.5506	0.0576	0.9843
Max Size 3	1.5284	0.4704	0.0581	0.9834
Hidden Layers				
1 Pooling Layer, 1 Dense Layer	0.6453	0.7976	0.0496	0.9840
3 Pooling Layer, 2 Dense Layer	1.9076	0.3108	0.0345	0.9910
1 Pooling Layer, 3 Dense Layer	1.6884	0.3946	0.0732	0.9818
4 Pooling Layer, 1 Dense Layer	1.0596	0.6537	0.0403	0.9861

Table 2: Results of CNN experiments. We report The loss and accuracy after the first epoch and the validation set.