Assignment: CNN and MNIST

David Bertoldi – 735213
email: d.bertoldi@campus.unimib.it

Department of Informatics, Systems and Communication

University of Milano-Bicocca

1 Inspecting the data

The MNIST dataset contains 70 000 images of handwritten digits (0 to 9) that have been size-normalized and centered in a square grid of pixels. Each image is a 28×28 array of floating-point numbers representing grayscale intensities ranging from 0 (black) to 255 (white).

The labels consist of a vector of values, corresponding to the digit classification categories 0 through 9.

The dataset is already divided into training and test sets, respectively with 60 000 and 10 000 samples.

Figure 1 shows an 10 entries of the training dataset.

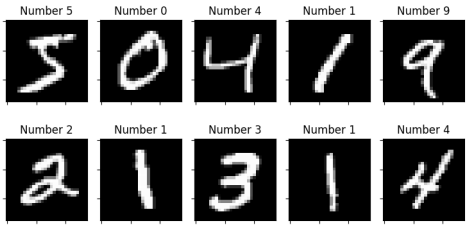

Figure 1: The first 10 samples of the train dataset

The training population presents a distribution with mean $\mu = 6\,000$ and standard deviation $\sigma \simeq 340$ and thus we didn't notice any important unbalance in the data. For this reason we assumed the data followed a distribution $X \sim U(\mu, \sigma)$ and no data augmentation on less populated classes was taken into account. Figure 2 shows the data distribution for both training and test datasets.

2 Preparing the data

Before training a CNN using this images, encoded in 28×28 matrices with values from 0 to 255, we rescaled each value in the continuous interval $[0, 1]$ and expanded the dimensionality of the matrix to $1 \times 28 \times 28$. This encoding will be used in every section of this work: small values increase the efficiency in the calculations and `Conv2D` requires an additional dimension that describes the channel (in this case just one for the grayscale).

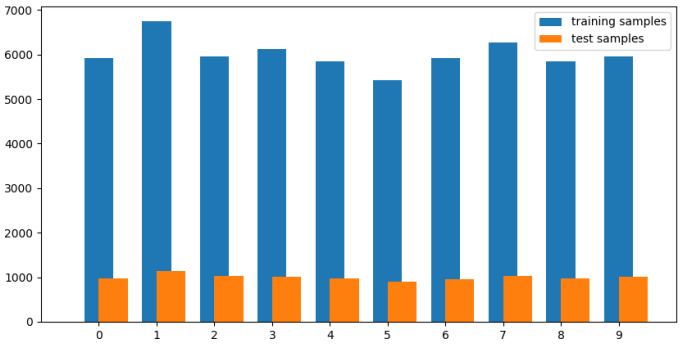

Figure 2: Histogram of the frequency of samples in the dataset

2.1 Data split

As noted in section 1, the dataset is divided into training and test samples. A set of items to be used for validation is missing and thus is retrieved from the training set: 15% of the images are randomly used for validation (along with their labels) for a total of 9 000 samples.

The labels were encoded in one-hot vectors so that the 1s were set in the index representing the numerical class.

3 Building the network and training

The aim of this section is to describe a CNN with less than 10 000 parameters that is able to classify with the highest possible level of test categorical accuracy the numbers from the dataset in 10 epochs and batches of 128 samples.

3.1 The network

The CNN presents a typical architecure formed by convolutional layers followed by pooling layers and ending with dense layers.

In particular there are 2 convulutional layers covering the whole 28×28 matrix, formed by 8 3×3 filters, for a total dimension of $28 \times 28 \times 8$. These two layers are followed by a *max pooling* layer that halves the the widht and height of the outcoming activation map. For this problem we tested both *max pooling* and *average pooling*; the first one performed slighty better (+0.2% in test categorical accuracy): usually *average pooling* smooths out the image and has harder times

to identify sharp features while *max pooling* chooses only the brightest pixels of the image that are in this case the ones defining the handwritten digit. Although we noticed a slighty improvement using *max pooling*, the images are too small to actually benefit from the methods' differences.

The structure continues with another one convolutional layer aligned with the 2D spatiality of the last pooling layer but doubled in the depth, that is $7 \times 7 \times 16$. The convolutional layer is reduced in spatiality by another *max pooling layer* $7 \times 7 \times 16$ and flattened in a 1D array of 784. The input flows to the output layer activated by *Softmax* function. Figure 3 summarizes the entire architecture and Table 1 highlights the number of parameters in each layer.

Layer	Size	Parameters
input	$28 \times 28 \times 1$	0
Conv2D-1	$28 \times 28 \times 8$	$(3 \cdot 3 \cdot 1 + 1) \cdot 8 = 80$
Conv2D-2	$28 \times 28 \times 8$	$(3 \cdot 3 \cdot 8 + 1) \cdot 8 = 584$
MaxPool-1	$14 \times 14 \times 8$	0
Conv2D-3	$14 \times 14 \times 16$	$(3 \cdot 3 \cdot 8 + 1) \cdot 16 = 1\,168$
MaxPool-2	$7 \times 7 \times 16$	0
Flatten	$1 \times 1 \times 784$	0
Dense	$1 \times 1 \times 10$	$(784 + 1) \cdot 10 = 7\,850$
Total		9 682

Table 1: Summary of the layers' dimensions and count of the parameters for each layer

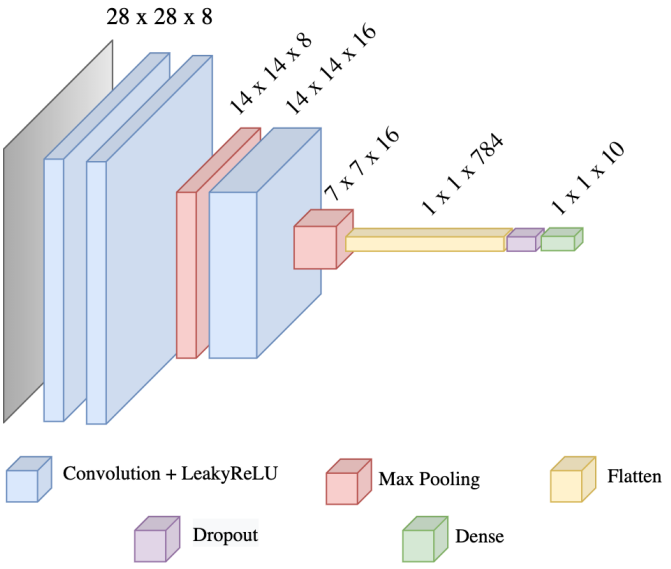

Figure 3: Architecture of the CNN

On the structure, we tried less deeper convolitional layers (starting from $28 \times 28 \times 4$ and then scaling to $14 \times 14 \times 8$) but the overall categorical accuracy dropped by 0.8%.

About the convulutional layers, Tensorflow allows to specify the padding: no padding and padding with zeroes. The first one would had reduced the spatiality of each layer by 1, the second one preserves the dimensions by putting evenly 0s on the margins. Adding 0s with this particular dataset is not an issue, because most of the images (if not all of them) do not contain information along the margins. We benchmarked the performances of both methods and we noticed that the zero padding performed had an higher validation categorical accuracy of +0.3%.

3.2 Training

The choice of the optimizer was among *RMSProp* and *Adam*. None of them have shown signs of getting stuck in local minimum regions but *Adam* performed better overall, with +0.9% on test accuracy.

The only regularization used is the *dropout* technique applied to the flattened layer with a rate of 10%. This allowed the model to not overifit too much and generalize better the problem. The choice of the probability to drop out a neuron is based on the chart in Figure 4 where we moved the rate from 0% to 90% and logged the results for training, validation and test categorical accuracy.

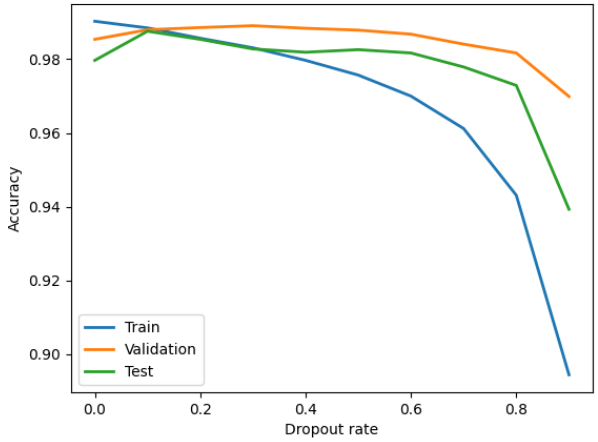

Figure 4: Loss

We chose a drop rate of 10% because it maximized the test categorical accuracy, raised the validation categorical accuracy and introduced a noticeable amount of underfitting, which decreased the training categorical accuracy. [1]

The behaviour of the model during the training phase is described in the plots of the loss and categorical accuracy in Figure 5 and Figure 6.

The validation loss was always lower than the training loss. This is effect is caused by the *dropout* because it penalizes model variance by randomly removing neurons from the flatten layer only during the training; the models underfitted and since *dropout* is disabled during the validation we had lower validation loss. The same for the categorical accuracy, being higher during validation.

The best training and validation categorical accuracy reached in this configuration were 98.57% and 98.86% respectively, using *Adam* with learning rate of 10^{-3}.

3.3 Evaluation

We tested the quality of the model's predictions using the provided test dataset, applying very same transormations on

[1] This is the best configuration with 10 epochs and batch size of 128. We noticed better results with higher drop rate (40%) with more epochs, limited by *early stopping* techniques, and 256 batch size: this configuration produced a 98.92% in test accuracy.

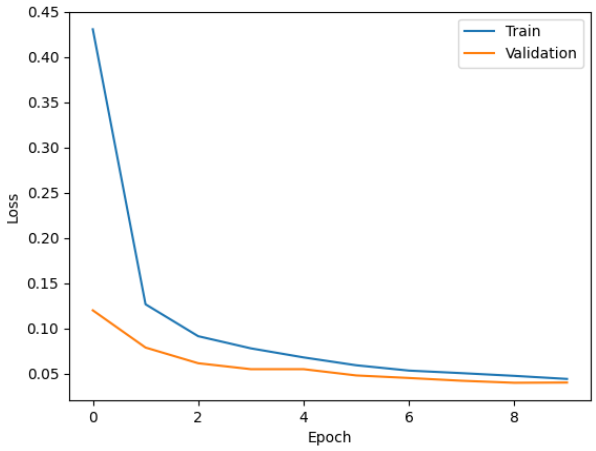

Figure 5: Loss

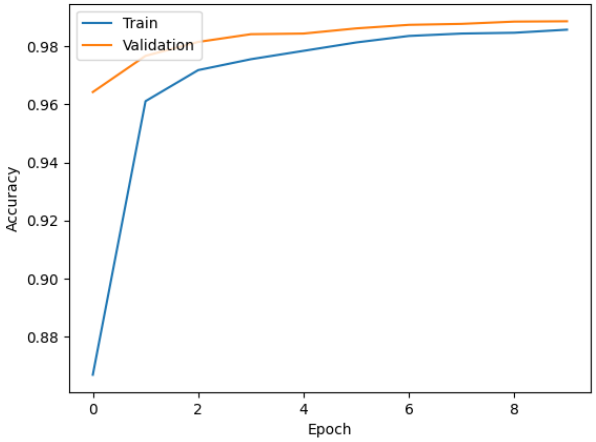

Figure 6: Categorical accuracy

the images used for the training. The categorical accuracy over the test set reached 98.65% and can be analyzed with the help of the confusion matrix (Figure 7).

We noticed that the model confused the 0.9% of the 4s with a 9: this can be explained by the fact that the two digits have similar forms, in particular when the 4 is written quicky and in its *"closed form"*. Apart from these cases, that have a realitively low impact, the model gave really good results.

4 Implementations

The training of the model can be started with `trainer.py`. It generates the needed images for the project in `./images/` a model checkpoint (hdf5 file) in the working directory. The validation can be performed with `eval.py` and requires a valid hdf5 file produced by the trainer.

5 Conclusions

In this work we trained a CNN with less than 10 000 parameters, achieving the best test categorical accuracy with 10 epochs and 128 elements per batch. We tested an architecure with two sections containing convolutional and pooling

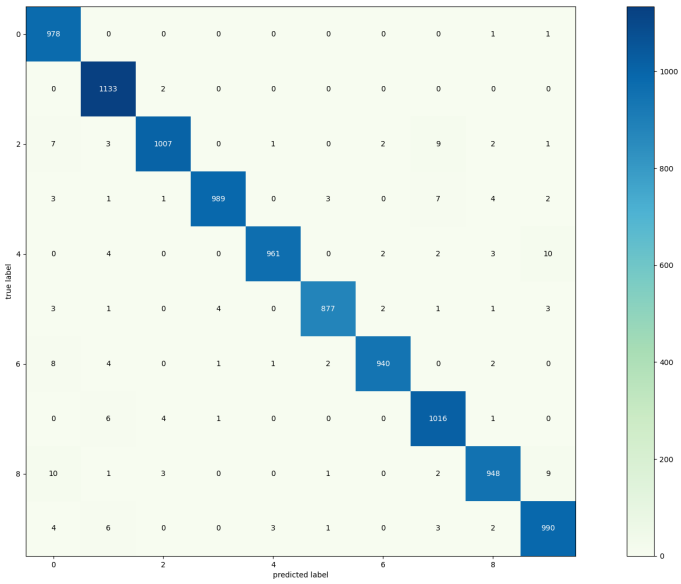

Figure 7: Confusion matrix over the test data

layers and ending with a dense ouput, benchmarking different dimensions and applying different rate of drop out. The CNN requires a significative lower number of parameters compared to a dense FNN, achieving better results. The application of local filters (or kernels) helped the model to better recognize sharp features in the images.