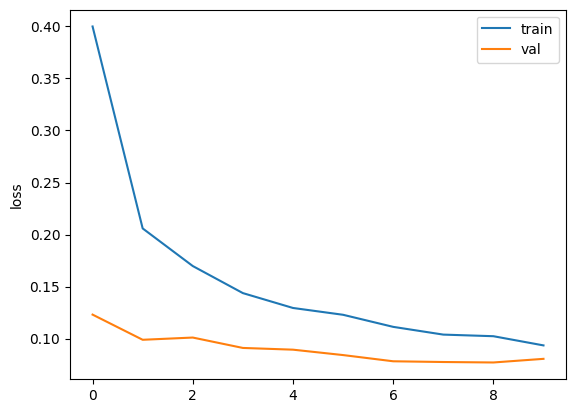
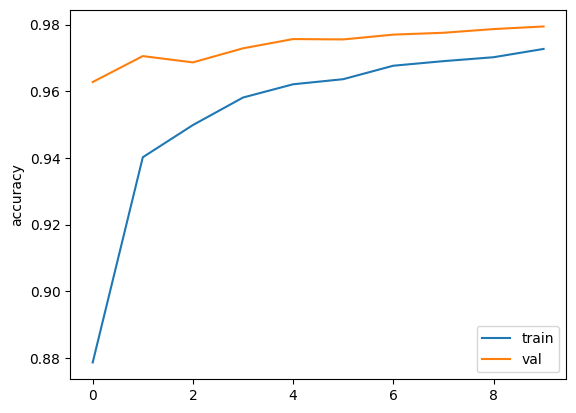
# Deep Learning Project

The task of this deep learning project was to create and train a neural network for number recognition of the MNIST dataset, which contains 70000 hand-written numbers from 0 to 10. The main priority is performance followed by explainability, while data analysis comes last.

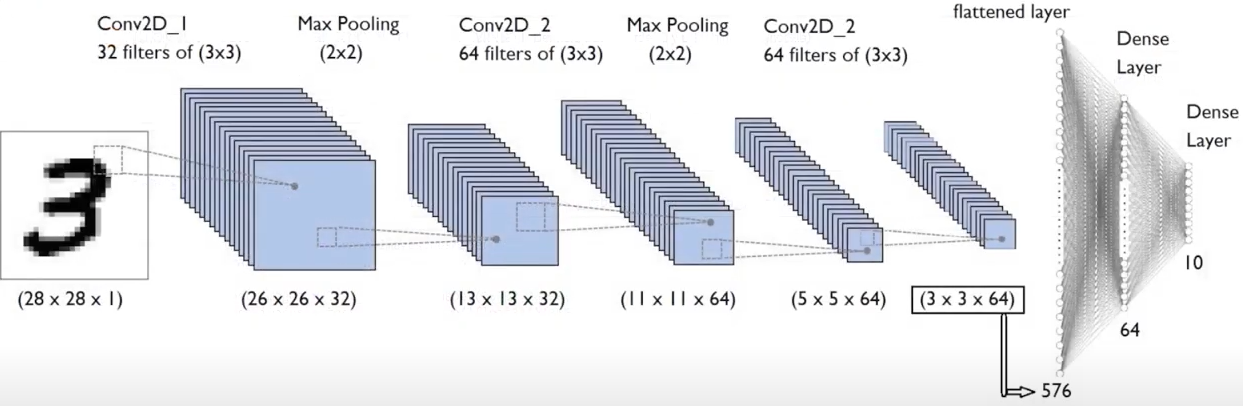
As a first step the given ways of tutorial 1 were tried and rated by their performance. Version 0 is without regularization, where different validation splits (15% and 20%) and early stopping have been tried. Version 1 is with L2 regularizer and version 2 with dropout function.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| version | 1 | 2 | 0 | 0 | 0 with early stopping |
| values | loss: 0.3935 acc.: 0.9508 | loss: 0.0979 acc.: 0.9751 | loss: 0.1154 acc.: 0.9744 | loss: 0.0865 acc.: 0.9817 | loss: 0.0977 acc.: 0.9772 |
| comment | val\_split=0.15 | val\_split=0.15 | val\_split=0.2 | val\_split=0.15 | val\_split=0.15 patience=2 |

Even though version 0 had the best performance, I decided to work on with dropout because of problems with noise of loss and overfitting. The model is a sequential model with 5 layers. The first layer is a flatten layer which flattens the input data from a 28x28 matrix to a 784 element vector. The next two layers are dense layers with 512 and 128 neurons respectively, and both use the ReLU activation function. These layers are followed by two dropout layers with a dropout rate of 0.5, which helps to prevent overfitting by randomly setting a fraction of the input units to 0 during training. The final layer is a dense layer with 10 neurons and a softmax activation function. It is compiled with the Adam optimizer, a sparse categorical crossentropy loss function and accuracy as a metric. The model is then trained for 10 epochs with a batch size of 32 and a validation split of 0.15, very similar to the test splitting of MNIST (training set: 60000, test set: 10000 -> 14%). With this model it is able to reach an accuracy of 98% and a loss of 0.07.



To improve the performance, I tried a model with covolutional layers. The main concept of the code and the concept of convolutional layers is displayed in the picture below.

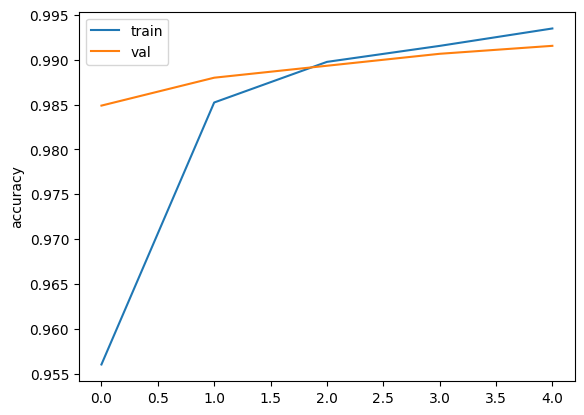
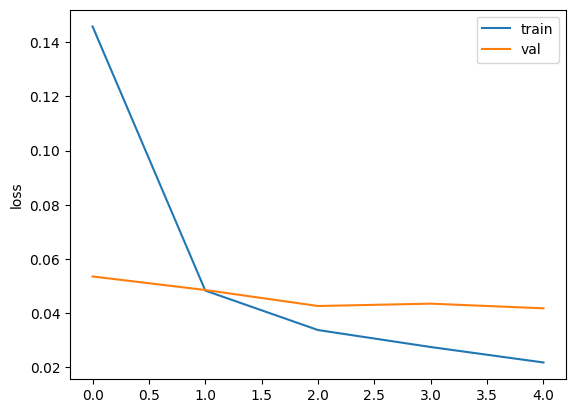


Source: <https://www.youtube.com/watch?v=9cPMFTwBdM4>

The convolutional neural network consists of two convolutional layers, each followed by a max pooling layer, and then a fully connected dense layer.

The to\_categorial method labels in the y\_train and y\_test arrays into one-hot encoded labels. One-hot encoding is a process where a categorical label is represented as an array of binary values, where only one element is 1 and the rest are 0. This is often used in classification problems where the categorical labels are not ordinal and there is no inherent ordering between them.

It is then compiled with the rmsprop optimizer, the categorical cross-entropy loss function and the accuracy metric. The training is performed for 5 epochs with a batch size of 32 and shuffling of the training data. 15% of the training data is used for validation. With this model it was able to reach an accuracy of 99% and a loss of 0.03.





# Summary of the binders

In Binder 1 we created functions with the math of back propagation. Later we tried different optimizations and regularization of the tensorflow libraries.

In Binder 2 we trained on the dogs and cats dataset. The VGG16 model was used to freeze the layers and new layers to train were added.

In Binder 3 we used the U-net for segmentation. We implemented a dice loss and prepared the data to write a generator for training.