**Assignment 4 - Course Project**

Sebastian Scholl 11827056

Clemens Berger 11821616

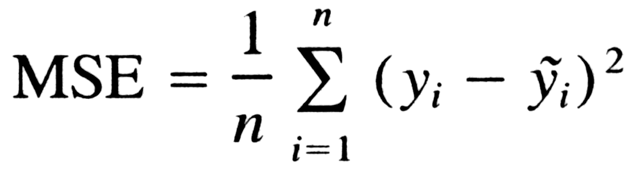
**TODO:** insert other name and matricle numbers

**1) Introduction**

Our group chose topic 1, Correcting Images with Autoencoders. The task is to train an autoencoder that is capable of correcting and re-generating clean images from their distorted versions. As data set we chose Kuzushiji-MNIST. To build a perturbed data set we created our own functions, which modify the data with gaussian noise, rotate or flip the images, add black patches and change the brightness of the original images.

**2) Methods**

The network we build is an autoencoder, therefore we use mean squared error as loss function:

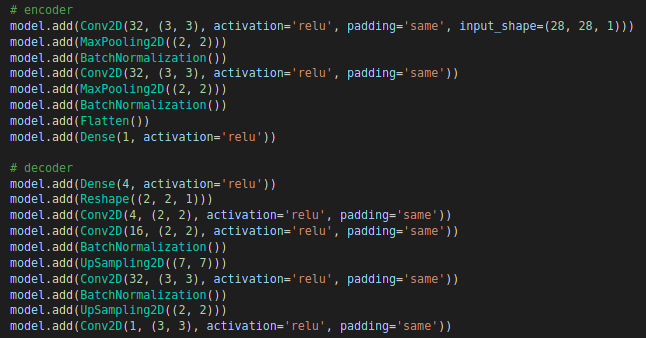


Autoencoder are built as an encoder-decoder architecture, the encoder receives an image, reduces the dimensions and compresses the input data. The decoder then reconstructs the original image from this compressed data.

**3) Results**

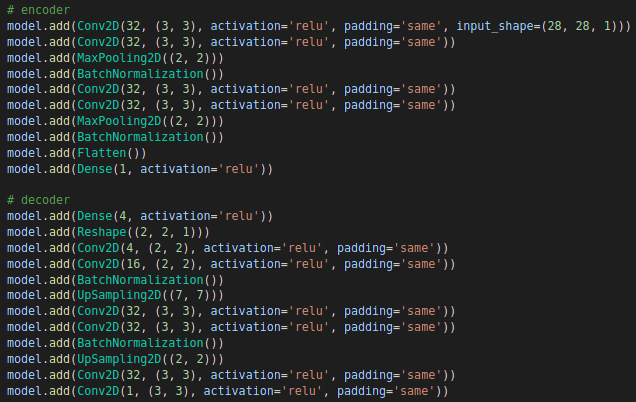
Model 0:

The initial model is a not very deep standard encoder decoder architecture with two convolutions in the encoder and batch normalizations.



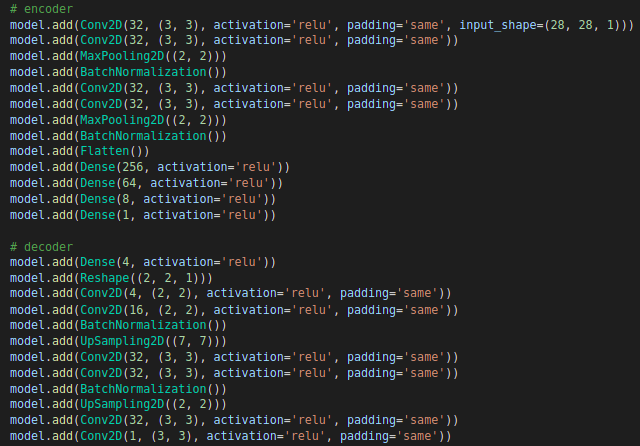
Model 1:

Model 0 with more convolutional layers.



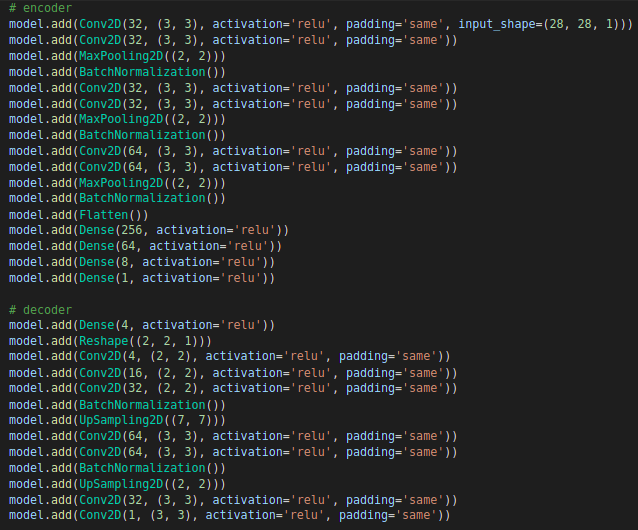
Model 2:

Added more dense layers to model 1.



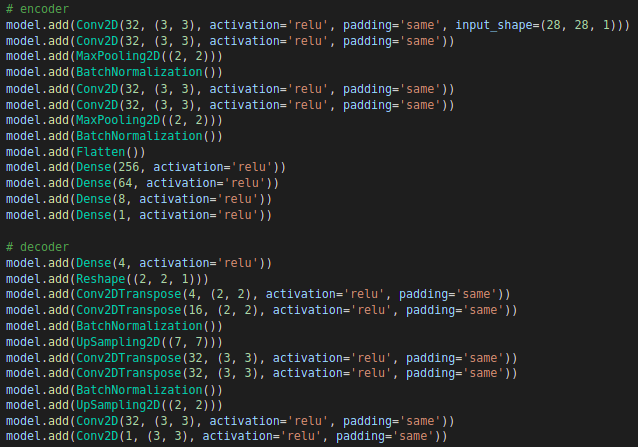
Model 3:

Added another convolutional layer with batch normalization to model 2.

****

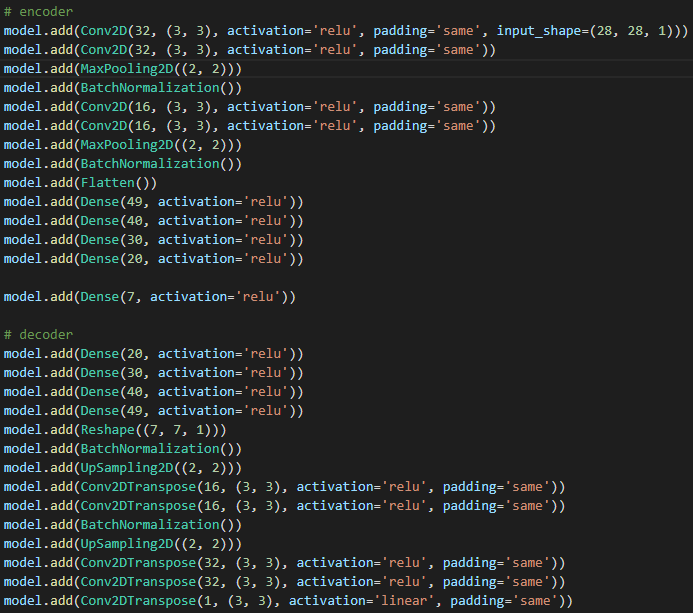
Model 4:

Model 2 with Conv2DTranspose instead of Conv2D layers on the decoder.



Model 5 to 8.1:

With these models we tested out which number of nodes in the bottleneck layer performs the best. For this we fixed the general model structure and just changed the number of nodes in the middle layer. The structure looks as follows:

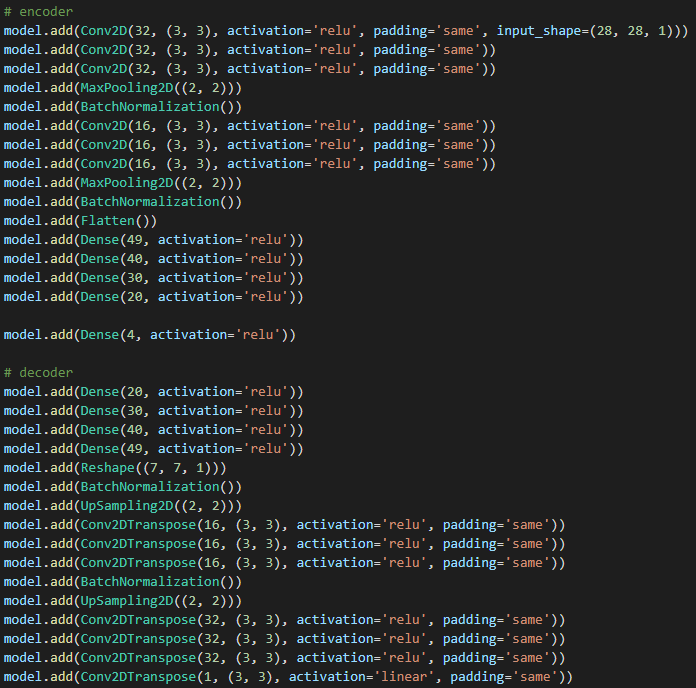


We tested a range of different numbers from 3 to 7 nodes. Model 7 was built with 7 nodes in the bottleneck layer. As one can see, this model performed the worst with a validation loss of 0.1561. Lowering the number of nodes increased the model’s performance significantly. For example, the models 8.1 and 5 have 5 and 4 nodes in the bottleneck layer and showed the best performance. However, if one further decreases the nodes like to 3 in model 6, the losses already increase again.

Since the model 5 with 4 nodes showed the best performance, we fixed this value for the subsequent models.

Model 10:

Here we tried to deepen the model structure by adding some convolutional layers. As one can see, this further increased the model’s performance.

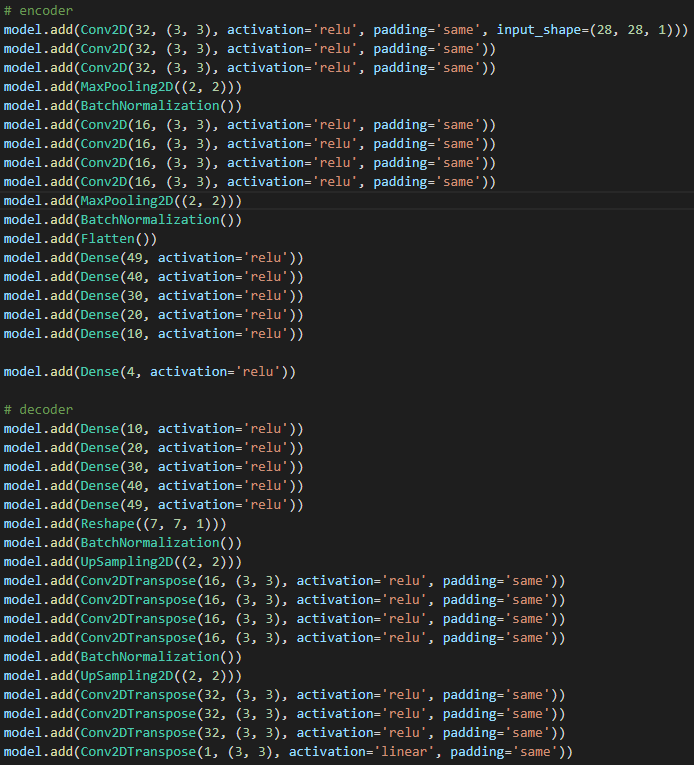


Model 11:

Since a deeper network seems to work better, we added two convolutional layers with 16 feature maps at the encoder and decoder stage. However, the training becomes very time consuming, since one epoch already takes quite some time and the convergence of the model’s loss takes more epochs than before. However, the performance was increased again by deepening the model.

Model 12:

Here we tried to deepen the model at its dense layers. Therefore, we extended model 11 by adding two Dense layers with 10 nodes at the decoder and encoder stage. This led to a way better performance than we had achieved before with a validation loss of 0.0744. The model’s structure looks as follows:



Model 13:

Since the last model already performed quite well we tried to add some regularizers in order to close the gap between the training and validation loss. In this model we just added two L2 regularizers with a factor of 0.001 at the first and last convolutional layer of model 12. The gap between training and validation loss got closer, but the value of the validation loss increased in comparison to the previous model.

Model 14:

For this model we added two Dropout layers with a dropout rate of 0.1 at the beginning and at the end of model 12. This led again to worse results than before without regularizers.

**TODO:** describe other models

Table of all models:

|  |  |  |
| --- | --- | --- |
| Model Nr. | Training loss | Validation loss |
| 0 | 0.1 | 0.1 |
| 1 | 0.09 | 0.09 |
| 2 | 0.09 | 0.091 |
| 3 | 0.109 | 0.109 |
| 4 | 0.111 | 0.112 |
| 5 | 0.0856 | 0.0865 |
| 6 | 0.1158 | 0.1506 |
| 7 | 0.1113 | 0.1561 |
| 8 | 0.1088 | 0.1089 |
| 8.1 | 0.0880 | 0.0886 |
| 9 | 0.1091 | 0.1224 |
| 10 | 0.0820 | 0.0837 |
| 11 | 0.0818 | 0.0830 |
| 12 | 0.0717 | 0.0744 |
| 13 | 0.0771 | 0.0788 |
| 14 | 0.0730 | 0.0755 |
| 15 |  |  |
| 16 |  |  |
| 17 |  |  |
| 18 |  |  |
| 19 |  |  |

**TODO:** insert values in table

**3) Discussion**

**TODO:** briefly describing your main observations. Include a clear description of your final deep learning architecture (e.g., regularization approaches, convolutional layer specifications, activations, latent dimensions, etc.). Report your model training details (e.g., loss function and optimization), and the amount of parameters in your networks. Provide tables that depicts your hyper-parameter search, as well as plots of training, validation and test set losses across training iterations.