University of Constance
Department of Computer Science & Information Science
Jonas Probst, Simon Giebenhain, Gabriel Scheibler, Clemens Gutknecht

Solution of 4. Assignment

Deadline: 28.1.18

Points: (maximum 110 Points)

Exercise 1	Exercise 2	Exercise 3	Exercise 4	**Total**
20	60	10	20	110

Exercise 1: Transposed Convolutions

a)

b)

c)

[ ] **of 20 Points**

Exercise 2: Encoder-Decoder

a)
The method **normalize_mean()** and **normalize_stddev()** normalize the mean and standard deviation of the input data respectively.

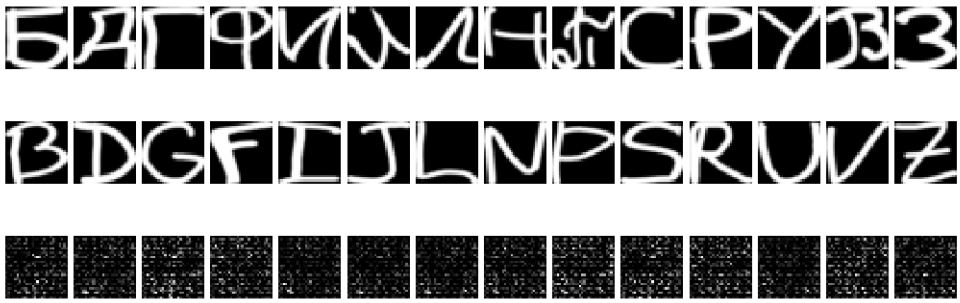
b)
The architecture of our encoder-decoder is described in the following:
Encoder:
The inputs are images of size $[28, 28, 1]$.
We apply a strided convolution with kernel dimensions $[3, 3, 1, 32]$ and strides $[1, 2, 2, 1]$ followed by basically the same convolution with kernel dimensions $[3, 3, 32, 64]$ and strides $[1, 2, 2, 1]$. This yields an output of the following shape $[7, 7, 64]$.
The last step of the encoder is a fully connected layer, which reduces the data to a vector of size 512.
A dropout layer connects encoder and decoder.

Decoder:
First the data of shape $[\text{batch\_size}, 512]$ is transformed to shape $[\text{batch\_size}, 7, 7, 64]$ with a fully connected layer and reshaping.
Afterwards we apply transposed convolutions mirroring the convolutions of the encoder, such that we come form shape $[7, 7, 64]$ to $[28, 28, 1]$ per image. An additional fully connected layer, which maintains shape, follows the convolutions.

Further Details:
We used *parametric ReLU* as activation function and applied *batch normalization* before every activation.
As mentioned above we applied a *droupout-layer* inbetween encoder and decoder.

c)
We struggled to come up with a sensible training method. At first we believed, that classifying(pretrained on the latin training dataset) the output of the encoder-decoder and apllying cross entropy to the result, would be the best idea. We followed this approach in **encoder_decoder_class.py**.

We achieved quit solid results, with a classification accuracy of around 97%. However, the outputs of the encoder-decoder were not readable for humans. This is due to the fact, that the classifier classifies every input he gets, and cannot distinguish between a nice letter and some extremly noisy letter. We tried to fix this, by introducing a noise class, with randomly generated training examples labeled as noise, but it did not help.
Here is a visualization of the results:



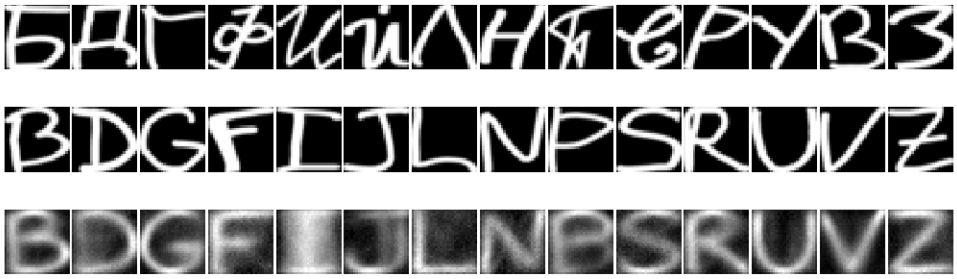
Another approach for the loss function, is to take the L2-loss of the output of the encoder-decoder with random examples of latin letters of the same class. This approach seemed to work quite well. However the outputs are always very similar within each class.

VIS

One could also choose a representative for each class of the latin latters and always compute the L2-loss of the difference of the output of the encoder-decoder and the respective representative. This basically achieves, that for every cyrillic letter of class k the encoder-decoder outputs the representativ of class k. (The same could be achieved with simple classification and return the representative of that class)
Here is a visualization:

VIS

Also one could always compare with the mean of the respective class, which does not really make sense, because the network is then trained to basically classify the image and return the mean of the resulting class. Here are the results:



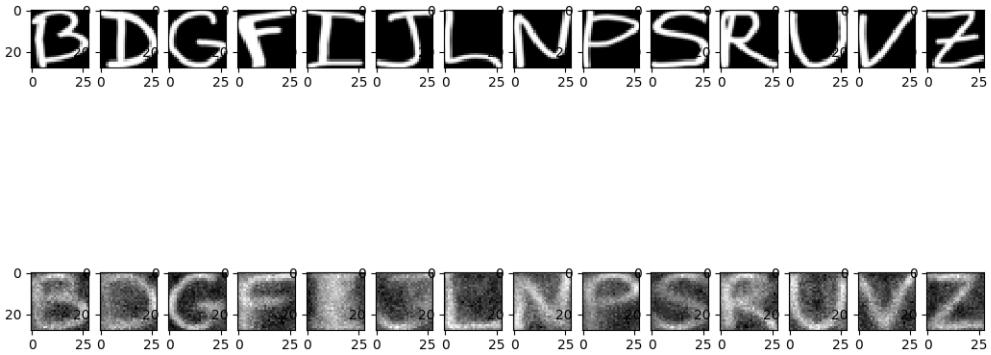
Tricks: Mentioned above in (b)

d)

See (c).

e)

For this exercise we used the training strategy and weights of **encoder_decoder_random.py**. The script **latent_space.py** loades the weights saved in **encoder_decoder_random.train_for_latent_space()**. The encoder part of the network is obmitted, it begins with the 512-tupel, which was in the middle before. All variables are markes as untrainable and only the 512-tupel is trainable. We optimezed for every class, the loss for a single class is the MSE of the output of the encoder and all training examples of that class. The results are shown below:



[ ] **of 60 Points**

Exercise 3: Backpropagation trough a ConvLayer

a)

b)

[ ] **of 10 Points**

Exercise 4: PCA vs. Auto-Encoder

a)

b)

c)

[ ] **of 20 Points**