# Homework 2 – Unsupervised Learning

## 1. Introduction

The goal of this homework is to implement and optimize a neural network that can reconstruct images from the MNIST digit dataset. First, I implement a convolutional autoencoder and optimize the hyperparameters with random search. Using a dimensionality reduction technique, it is possible to explore the latent space dimension of the autoencoder and to generate new samples. The next goal is to implement a second autoencoder that can reconstruct the images while also removing noise. For this purpose, I chose to add gaussian noise to the images before feeding them to the autoencoder.

The simple autoencoder is then adapted to classify the images correctly depending only on the latent space layer outputs by adding a new fully connected layer after the last linear layer in the encoder.

Finally, I chose to implement a variational convolutional autoencoder that can generate new images by sampling from a learned distribution.

## 2. Method

In this section I describe the model architectures and the hyperparameters that can be tuned for the convolutional autoencoder, the denoising autoencoder, the classifier and the variational autoencoder each.

### 2.1 Convolutional Autoencoder

The convolutional autoencoder consists of an encoder and a decoder component. The encoder has 3 convolutional layers with 8, 16 and 32 kernels respectively. The kernel size for all layers is 3 and the activation function used is a rectified linear unit. Next comes a flattening layer and then 2 fully connected linear layers also with ReLU activations. The first layer has 64 hidden neurons, the number of neurons for the second layer is a hyperparameter to be optimized.

The structure of the decoder resembles the structure of the encoder in reverse.

Further hyperparameters to be optimized are the dropout probability for dropout layers added during optimization, the weight decay factor and learning rate for the optimizer and the batch size.

I optimize these hyperparameters by using random search for 5 iterations, each with 5-fold cross-validation. The low number of iterations is due to the training being rather time consuming. Grid search would have taken to long to execute on this model. Further, I fixed the weight decay factor since the network is really susceptible to changes here and often did not learn at all.

## 2.2 Denoising Autoencoder

The model architecture of the denoising autoencoder resembles that of the convolutional autoencoder. The only difference is that inputs are corrupted with zero-mean gaussian noise with a standard deviation of 1 before entering the network. The loss used is mean squared error loss comparing the uncorrupted images to the network outputs.

I use 5-fold cross-validation to ensure a low validation loss and train each model for 40 epochs.

## 2.3 Classifier

The architecture for the classifier looks very similar to that of the encoder in the convolutional autoencoder since the pretrained encoder is adapted to learn how to classify the MNIST dataset correctly. It solely replaces the second linear layer of the encoder with a different fully connected layer that has 10 neurons, one for each digit.

The hyperparameters to be chosen for fine-tuning are again the weight decay factor and learning rate of the optimizer, the batch size and the number of training epochs. However, I fixed these hyperparameters according to the results of the optimization on the regular convolutional autoencoder, since most of the network is pretrained and needs no optimization.

I train this model only once for 40 epochs.

## 2.4 Variational Autoencoder

Also the variational autoencoder makes use of the encoder and decoder architectures already defined for the convolutional autoencoder. However, the latent space dimension hyperparameter of the encoder must now be 2 * dimension of the distribution the encoder is trying to learn. The decoder takes as an input a combination of the encoder output and a random variable sampled from a normal gaussian distribution of the same distribution dimension. I implemented the loss function myself given the loss definition for variational autoencoders on the lecture slides using the following formula to calculate the Kullback-Leibler-Divergence between a gaussian distribution and a normal gaussian distribution: $D_{KL}(p||q) = 0.5 * [\mu_p^T \mu_p + tr\{\Sigma_p\} - k - \log|\Sigma_p|]$

p is the gaussian distribution learned by the encoder and q is the normal gaussian distribution, k represents the dimension of the distributions. I found this formula with the following link: https://mr-easy.github.io/2020-04-16-kl-divergence-between-2-gaussian-distributions/

To ensure that the encoder cannot return invalid values for the covariance matrix, I return a loss of infinity whenever the network provides a negative and therefore invalid value in the latent space units that represent the covariance.

The hyperparameters here are the distribution dimension, the weight decay factor and learning rate of the optimizer, the parameter c defining the likelihood variance for the loss function and the batch size.

This model is also trained once over 40 epochs with fixed hyperparameters taken from the convolutional encoder.

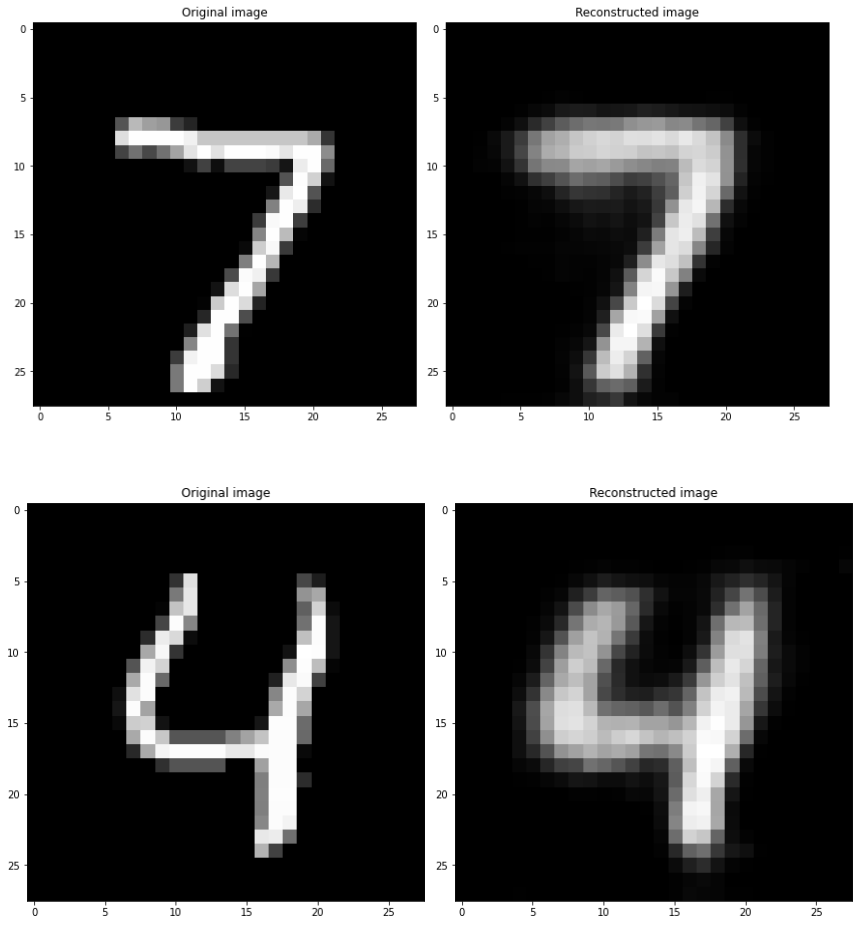
# 3. Results

In this section I present and explain the results obtained when running all of the above described architectures.

## 3.1 Convolutional Autoencoder

First, I ran the convolutional autoencoder without any optimizations as a baseline model. To do this I fixed the hyperparameters as follows: a latent space dimension of 2, a weight decay factor of 0.00005, a dropout probability of 0 and a batch size of 256 samples. The optimizer used is Adam with a learning rate of 0.001.

After 40 epochs the training loss has dropped to about  0.0382 and the validation loss is 0.0381. Since validation and training loss are very similar, it is safe to say that the model does not overfit in this case. The validation loss keeps falling every epoch and might thus decrease further if run for more epochs. The following figure shows some input images next to the reconstructed images by the autoencoder.

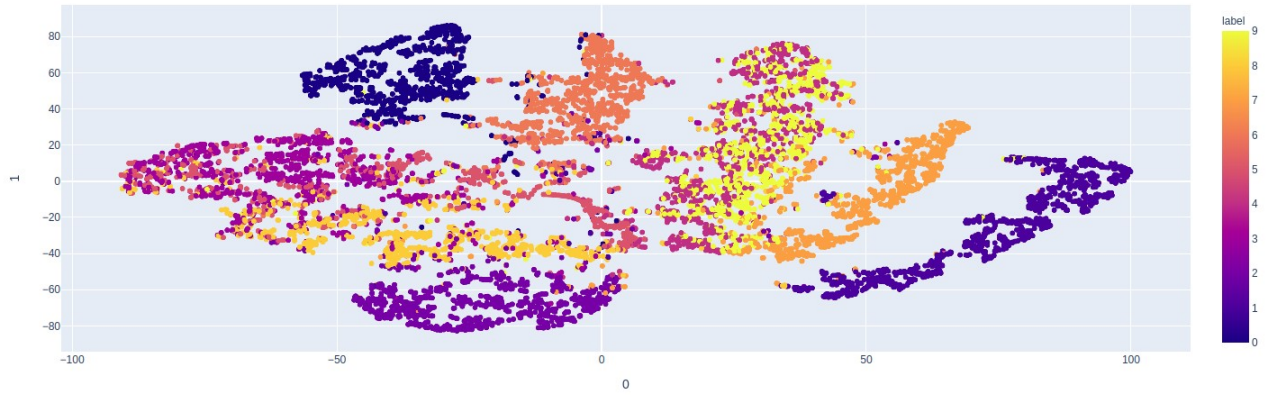



Next, I ran the autoencoder with the Adadelta (lr=1.0) and the RMSProp (lr=0.01) optimizers respectively. The conclusion was that the best performing optimizer for this task is the Adam optimizer, since the validation loss was the lowest.

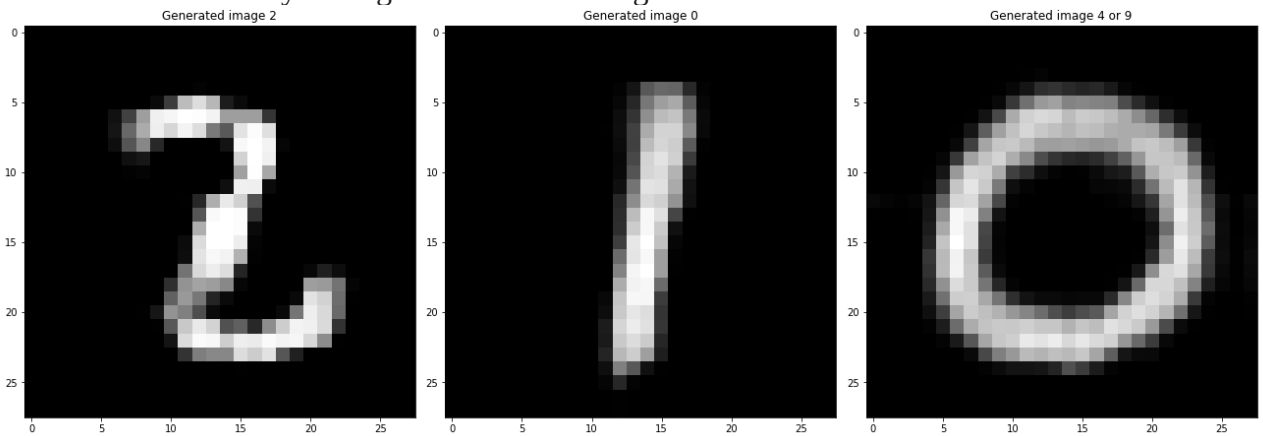
Using cross-validation and random search the performance can be improved considerably. I perform random search for 5 iterations rather than grid search since the training with cross-validation over only 5 folds proved to be highly time consuming already. The hyperparameters to be optimized with

random search are the batch size, the learning rate, the latent space dimension and the dropout probability. I do not optimize the weight decay factor since changing this caused the network to not learn at all in many cases. With this method, the best average validation loss drops to 0.0187 with equal test loss. The best hyperparameters found are a batch size of 128, a learning rate of 0.006 a latent space dimension of 48 and a dropout probability of 0.0775.

To explore the latent space structure and generate some new images, I use an autoencoder trained on optimized parameters but a latent space dimension of 2. The dimensionality reduction methods I tried are t-SNE and PCA. The figure below shows a plot of the reduced features using t-SNE.




The plot suggests that that the different digits can mostly be clearly distinguished in the reduced feature space. Only 4 and 9 overlap strongly. However, when using this plot to generate some new digit examples, the results are not as expected. Other digits appear then the t-SNE plot would suggest as can be seen in the figure below. This may be due to rather high losses close to 0.4 because of the low latent space dimension. Or maybe the second to last linear layer has not enough information to correctly distinguish between all digits.

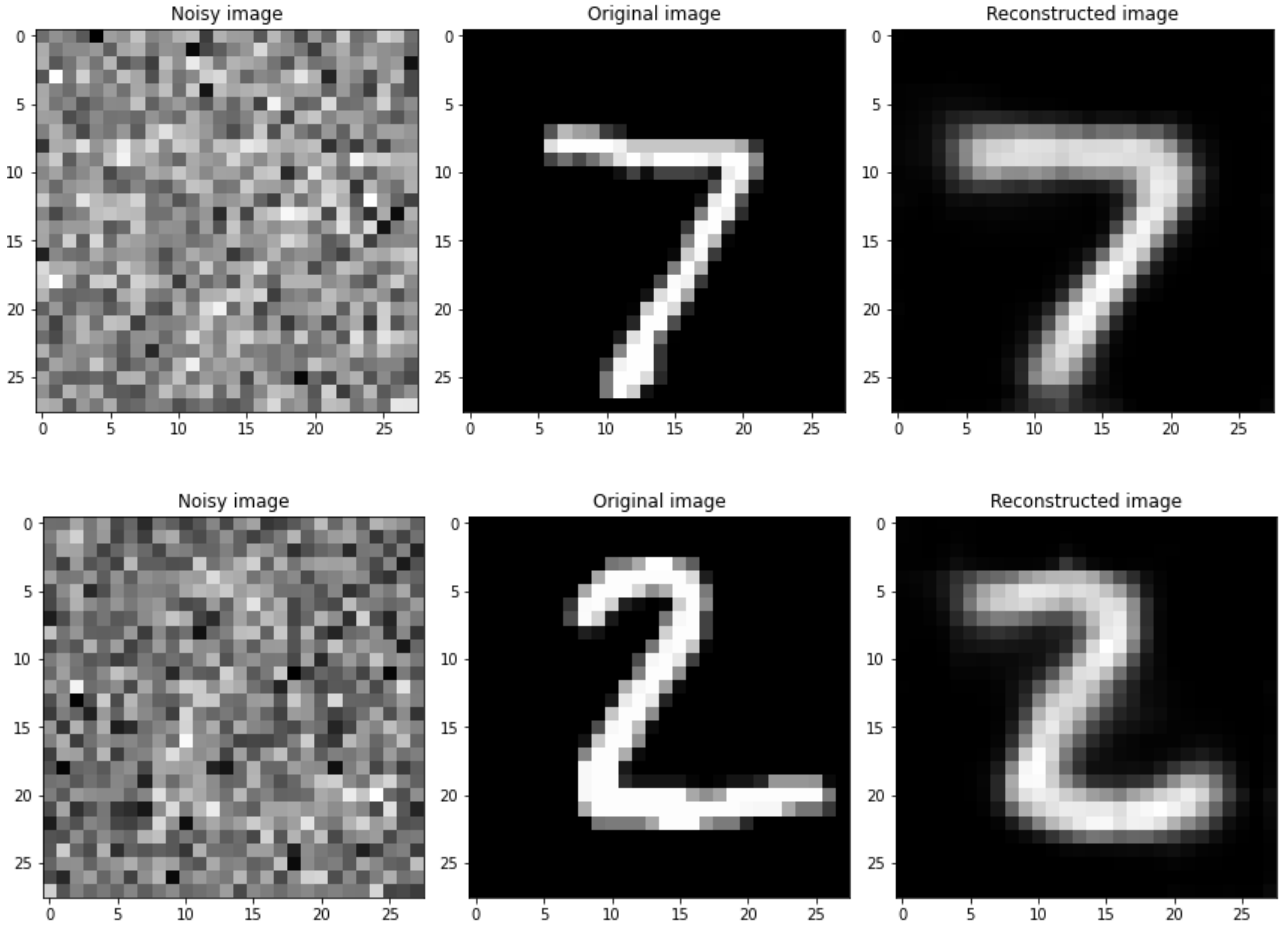



When using PCA instead of t-SNE, this does not improve.

## 3.2 Denoising Autoencoder

The denoising autoencoder adds zero-mean gaussian noise with a standard deviation of 1 to the input images. To obtain good results, I used the optimized hyperparameters obtained from random search on the convolutional autoencoder. The networks are similar enough to assume that these hyperparameters will also be a solid choice for the denoising autoencoder.

The result is a validation loss of 0.0307 and a test loss of 0.0303, which means that it performs slightly worse than the conventional autoencoder but still achieves good results. The following figure shows some results of the denoising autoencoder. On the left is always the noisy image, in the center the original image and on the right the reconstructed image.




## 3.3 Classifier

The classifier model I evaluated has a batch size of 128 and a learning rate of 0.006 taken from the optimized hyperparameters for the convolutional autoencoder. The number of epochs is only 20, since only the last layer needs to be trained.

I evaluated this model for a latent space dimension of 2 to show that this suffices for accurate classifications and of 10 to see how this improves the performance.

For a latent space dimension of 2 the average validation loss is 0.129, slightly higher than the training loss of 0.113, but not enough to suggest overfitting. The test loss is at 0.115 and the classification accuracy is quite high at 96.46%

For a latent space dimension of 10 the results are noticeably better with a validation loss of 0.086 on average, a test loss of 0.064 and a test accuracy of 98.04%.

So even though training was considerably faster than for the CNN in homework 1 the classification accuracy is comparable. For the best choice of hyperparameters and optimizer the CNN achieved

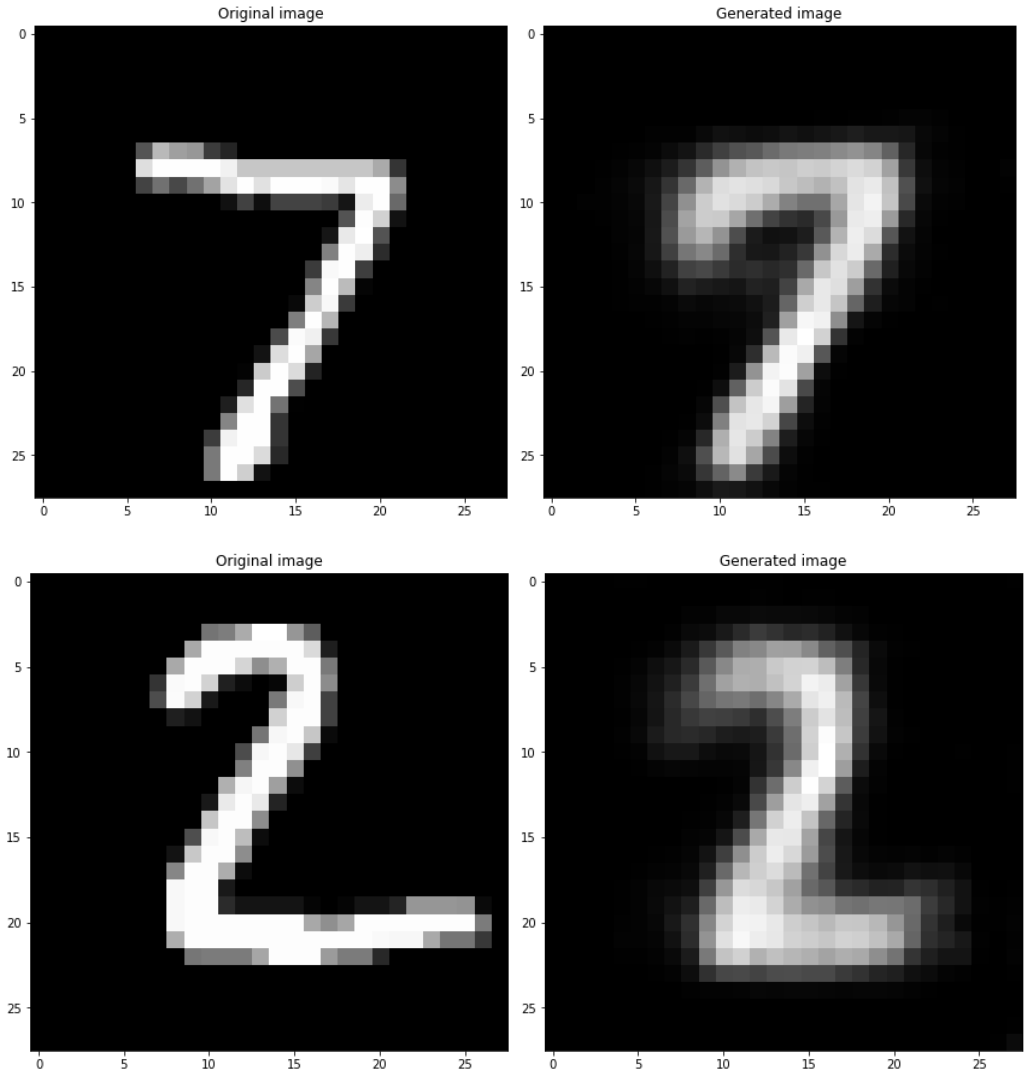
accuracies up to 99.04%, which is only slightly higher than the result for the fine-tuned autoencoder with a latent space dimension of 10. The results with a latent space dimensions of 2 are more comparable to those of the unoptimized CNN architecture.

## 3.4 Variational Autoencoder

To evaluate the variational autoencoder I fixed the hyperparameters as follows: 40 epochs of training, an Adam optimizer with learning rate 0.001, a batch size of 128 for each training and validation and a distribution dimension of 3 which means 6 output units for the encoder and 3 input units for the decoder. I split the training dataset into a training and a validation part.

Training the network with these parameters for 40 epochs I achieve a training loss of 19.383 and a validation loss of 19.473 close to the training loss. Thus also this network is not overfitting. The test loss is 19.521 but could likely be improved using random or grid search to optimize the hyperparameters specifically for this task.

Below are some examples of images generated by passing test data through the variational autoencoder.



# APPENDIX

Fig. 1 – PCA latent space representation for the optimized convolutional autoencoder