Homework #4

Assignment The aim of this laboratory is to train an *autoencoder* on the MNIST dataset and test its reconstruction capability. In order to do so, the following tasks have been implemented:

- add noises on the dataset: *Gaussian noise* and *obstruction*;
- tuned the network parameters with a *grid-search* and a *k-fold*;

Note: The file *trained_model.py* has been modified in order to deal with a different kind of dataset *"MNIST.mat"* from the one using during the training, so there is an extra transposition in the pre-processing.

1) Preprocessing During the training the dataset samples are used without preprocessing, in Figure 1a an example of a sample. To test the network results I apply two classes of noises on the *test dataset*:

- **GaussNoise:** with a probability of 0.25, this class adds some Gaussian noise to the sample. The mean of the distribution is 0 and I use a standard deviation of 0.05 for the algorithm, but I test the final trained network with different standard deviations. In Figure 1b an example of sample affected with Gaussian noise.
- **Occlusion:** with a probability of 0.25, this class adds in a random position a black rectangle on the sample with borders lengths between 8 and 14 pixels. In Figure 1c an example of sample with occlusion.

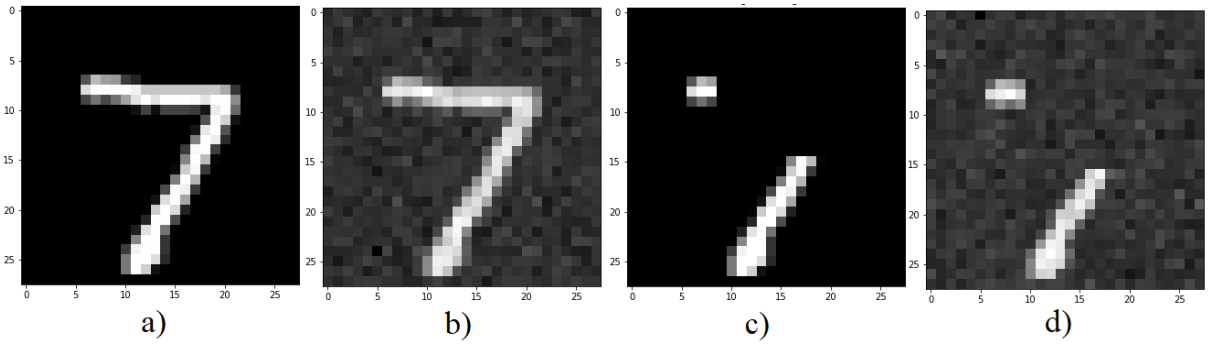

Figure 1: *From the left: a) Original sample; b) sample with Gaussian noise; c) sample with occlusion; d) Sample with Gaussian noise and occlusion.*

2) Model The proposed model is an Autoencoder composed of two main parts. The parameters of the components mirror each other and the main structure is as follows:

- **Encoder**, made of two parts:
  - *encoder_cnn*: a *2D convolutional* layer followed by a *ReLU activation layer*, both repeated for three times (outputs dimensions 8, 16, 32 respectively);
  - *encoder_lin*: a *Linear layer* followed by a *ReLU activation layer* and then another Linear layer (output dimension 64);
- **Decoder**, made of two parts:
  - *decoder_cnn*: a *Linear layer* followed by a *ReLU activation layer* and then another *Linear layer* (output dimensions 32 and 64 respectively);
  - *decoder_lin*: a *transpose 2D* layer followed by a *ReLU activation layer*, both repeated for three times (output dimensions 32, 16, 8 respectively).

The network optimizer is ***Adam*** with a fixed *weight decay* of 10^{-5} and a learning rate to be found during the *grid search*. The loss function is the ***Mean Squared Error*** (MSE). During the grid search and the final training, I use a batch size of **512** for the dataset. I keep these two parameters fixed as I don't see any real improvement changing them, so I focus my attention on tuning other parameters.

3) Training To choose the best learning rate and the encoded space for the Autoencoder, I implement a *grid search* for a learning rate of 0.01 and 0.001 and an encoded space dimension of 2, 4 and 6 (arbitrary choice). The learning rate values have been decided just by observing that the best networks I find, are always with one of the proposed two.
Using a k-fold with 3 folds, I train all the possible networks and keep the parameters providing the lowest mean error on the *validation dataset*. To save time, I block and skip the training of every network having a mean validation above 0.23. This number has been found by observing the means of many networks during my tests before implementing the *grid search*. Each network is trained for 10 epochs. The best network using this method has **0.001** as *learning rate* and **6** as *space dimension*.
To avoid overfitting an *early-stopping* mechanism is implemented: the final network is trained until the loss on the *test set* does not decrease for 5 consecutive epochs, with a lower and upper bound of 50 and 200 respectively. The algorithm stops after 58 epochs, so I keep the network trained for **53** epochs.

Note about the space dimensions For this laboratory I also trained the same network for 53 epochs with a *space dimension* of 2 and 4 to compare the results:

- a higher space dimension provides a faster training: a network with a space dimension of 2 needs almost double the number of epochs of a space dimension 6 to produce the first correct shape of reconstructed input image, and they are trained for the same amount of time. The results are also qualitatively worse using a lower space dimension;
- the final loss on the noiseless dataset are worse than the results I present in the following page: 0.040470 and 0.028216 for space dimension 2 and 4 respectively.

4) Results In Table 1 the final loss of the trained network with the *Test dataset* using different configurations for the noises.

Test Dataset	Test Loss
Without noises	0.022049
With Gaussian noise	0.024555
With occlusion	0.034860
With Gaussian noise and occlusion	0.037118
0.25 probability of Gaussian noise, 0.25 probability of occlusion	0.025046

Table 1: *Mean squared error of the network on the test dataset with different configurations.*

The errors of the *Test dataset* without noises and with only Gaussian noise are very close: the network deals with this kind of noise very well and it's able to correctly reconstruct the image as shown in Figure 3.

In the following figures I report some examples of how the network encodes and decodes the images:

- Figure 2: *the test dataset* is used without preprocessing, the images are correctly reconstructed;
- Figure 3: *the test dataset* is affected by Gaussian noise, but the network is still able to correctly reconstruct the images.;
- Figure 4: *the test dataset* is affected by occlusion, now the network starts to have some problems with the images and sometimes it fails to correctly reconstruct the images;
- Figure 5: *the test dataset* is affected by Gaussian noise and occlusion, the network is still able to reconstruct most of the images, but as before it fails if the occlusion is particularly bad.
- Figure 6: *the test dataset* is affected by Gaussian noise with different standard deviations:
  - Figure 6a: standard deviation of 0.3 on the Gaussian distribution, MSE = 0.119854;
  - Figure 6b: standard deviation of 0.58 on the Gaussian distribution, MSE = 0.379572;
  - Figure 6c: standard deviation of 0.78 on the Gaussian distribution, MSE = 0.659141;
  - Figure 6d: standard deviation of 0.99 on the Gaussian distribution, MSE = 1.036847.

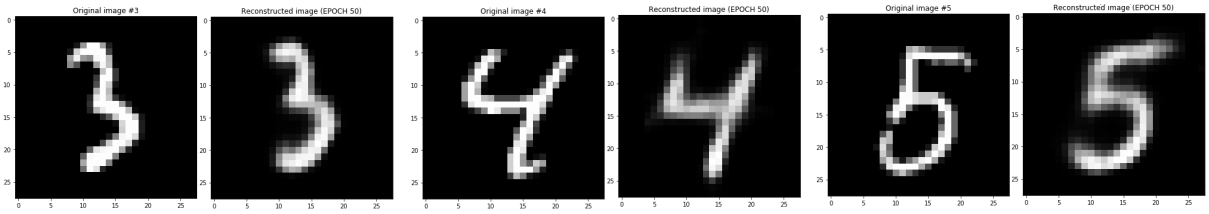

Figure 2: *Examples of input and outputs of the network. From the left, images of a three, a four and a five correctly reconstructed.*

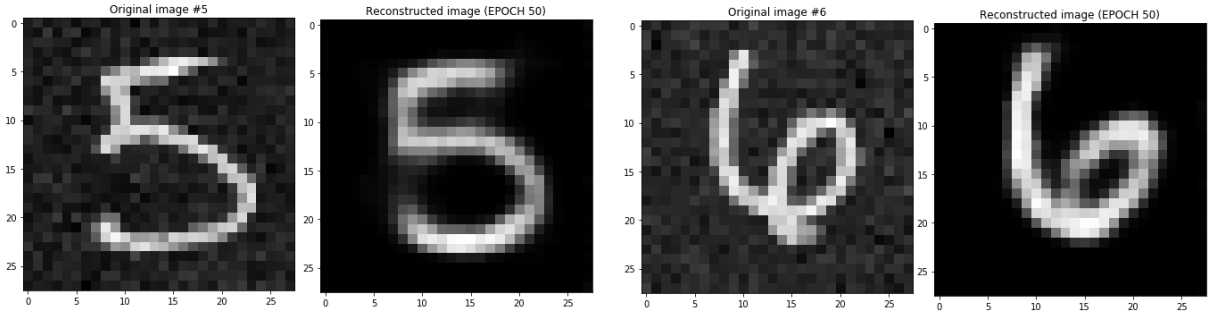

Figure 3: *Example of two images with Gaussian noise and the output of the network.*

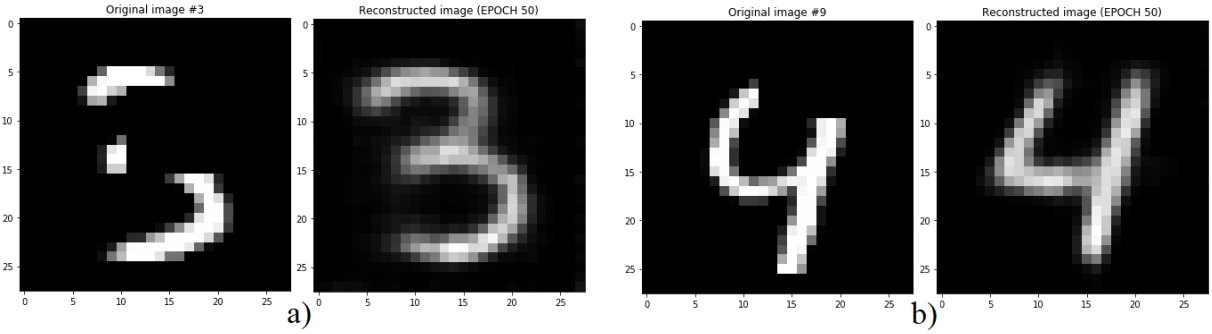

Figure 4: *Example of two images with occlusion and the output of the network. a) An image of a three with occlusion on the left, it is correctly reconstructed on the right; b) An image of a nine with occlusion on the right, its wrong reconstruction of the left.*

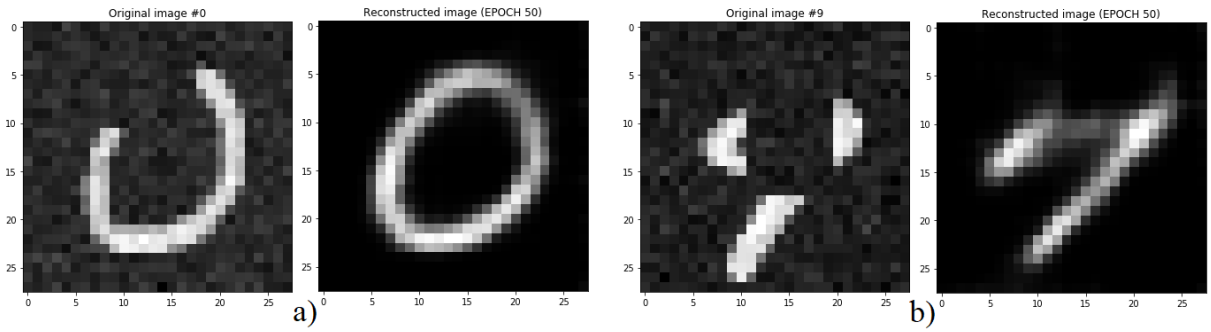

Figure 5: *Example of two images with Gaussian noise plus occlusion and the output of the network. a) An image of a zero on the left, it is correctly reconstructed on the right; b) An image of a nine on the right, its wrong reconstruction of the left.*

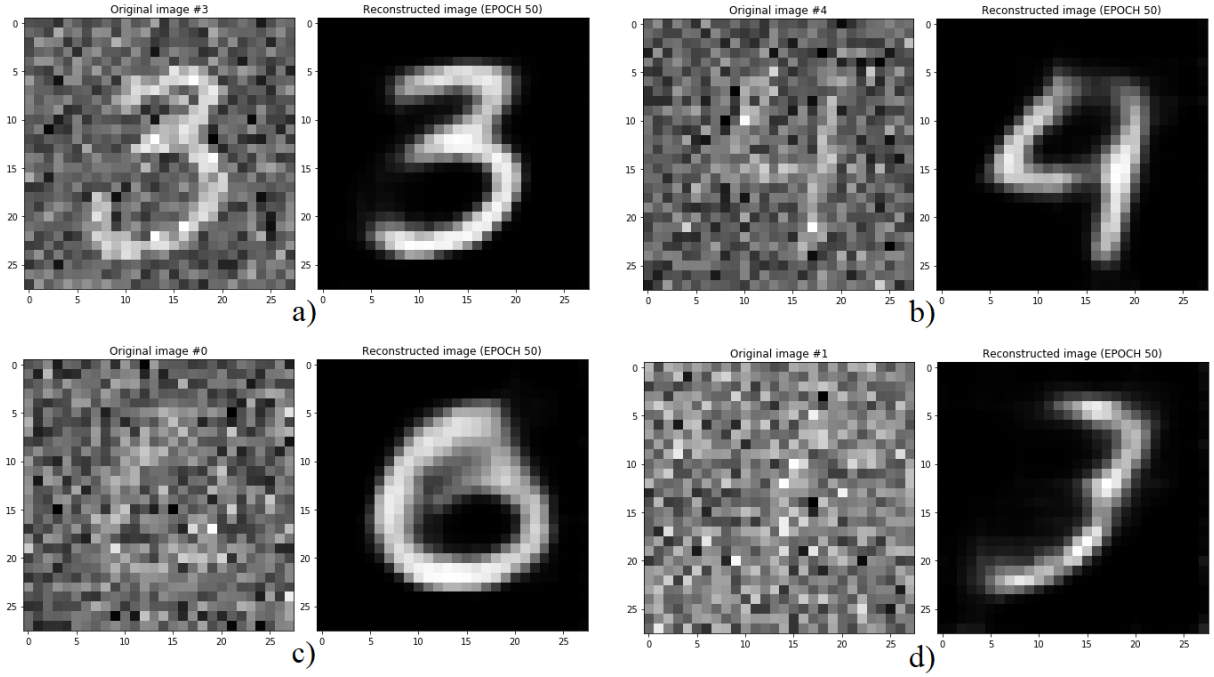

Figure 6: *For every couple of images, on the left the image the network is fed with and on the right the reconstructed output. The used Gaussian noises (zero mean) have different standard deviations (sd): a) a three with 0.3 sd, correctly reconstructed; b) a four with 0.58 sd, correctly reconstructed; c) a zero with 0.78 sd, correctly reconstructed; d) a one with 0.99 sd, wrongly reconstructed.*