Ivancich Stefano 1227846

Neural Network and deep learning course 2020/21
Homework 2

---

1. Introduction

In this homework we have to implement and test neural network models for solving unsupervised problems. The homework is based on images of handwritten digits (MNIST).
The **basic tasks** for the homework require to test and analyze the **convolutional autoencoder** implemented during the Lab practice, explore the use of advanced optimizers and regularization methods. Learning hyperparameters should be tuned using appropriate search procedures, and final accuracy should be evaluated using a cross-validation setup.
Reporting the **trend of reconstruction loss** and some examples of image reconstruction.
Implement the **denoising autoencoder**.
Use the convolutional encoder as base model for the **classification task** and compare the classification accuracy with Homework 1.
Explore the **latent space structure** (e.g., PCA, t-SNE) and generate new samples from latent codes.
Implement **variational (convolutional) autoencoder**.

2. Convolutional Autoencoder

The encoder architecture is composed by 3 convolutional layers followed by 2 linear layers all with Relu and Dropout. The decoder architecture is symmetric.
We didn't use **cross validation** in this specific case, since we have 60.000 images, we did not think that we can overfit on a such simple task.
A **grid search** of the following hyperparameters was ran:

- **encoded_space_dim:** 2,4,10,128
- **Conv1:** 8,16,32, 64 filters of size 3x3
- **Conv2:** 8,16,32, 64 filters of size 3x3
- **Conv3:** 8,16,32, 64 filters of size 3x3
- **FC1 number of neurons:** 32,64,128
- **FC2 number of neurons:** = **encoded_space_dim**
- **Layers activation:** ReLu for all except the last layer of the encoder that has no activation.
- **Dropout:** 0, 0.25, 0.5, 0.75
- **Optimizer:** Adam
- **Learning rate:** 0.1, 0.01, 0.02, 0.001
- **Regularization:** "L2" with values 1e-3, 1e-4, 1e-5 and 0 (no regularization)
- **Max epochs:** 3000 (we did not choose to tune this value, because the early stopping will take care of it)
- **Early stopping:** max 10 epochs without improvement

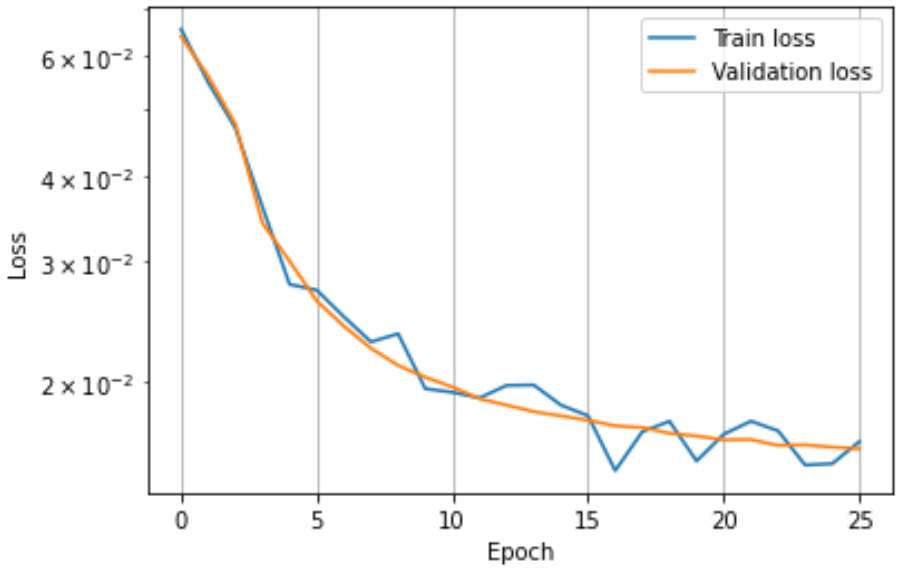
The best hyperparameters turned out to be:

- **encoded_space_dim: 10**
- **Conv1:** 8 filters of size 3x3
- **Conv2:** 16 filters of size 3x3
- **Conv3:** 32 filters of size 3x3

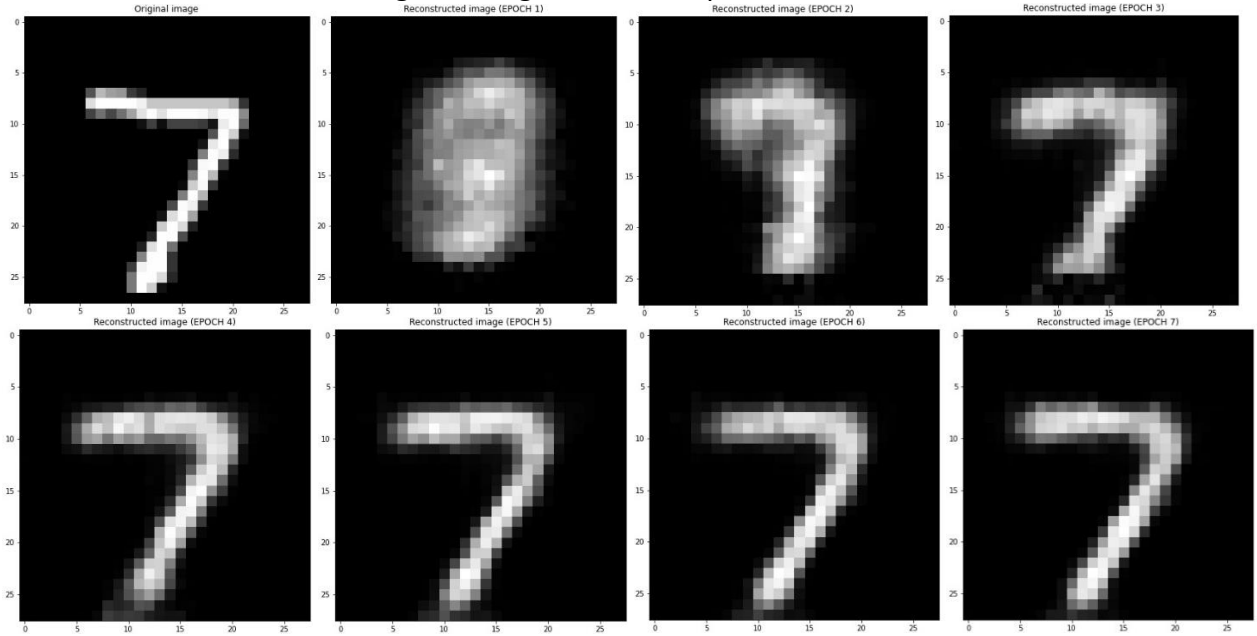
- **FC1 number of neurons:** 64
- **Learning rate:** 0.001
- **Regularization:** 1e-5 (L2)
- **Dropout:** 0. Strangely the dropout was slowing down a lot the learning and also creating bad loss and bad reconstruction.

Reconstruction loss Results:

- **Train Loss:** 0.016
- **Val Loss:** 0.016
- **Test Loss:** 0.016

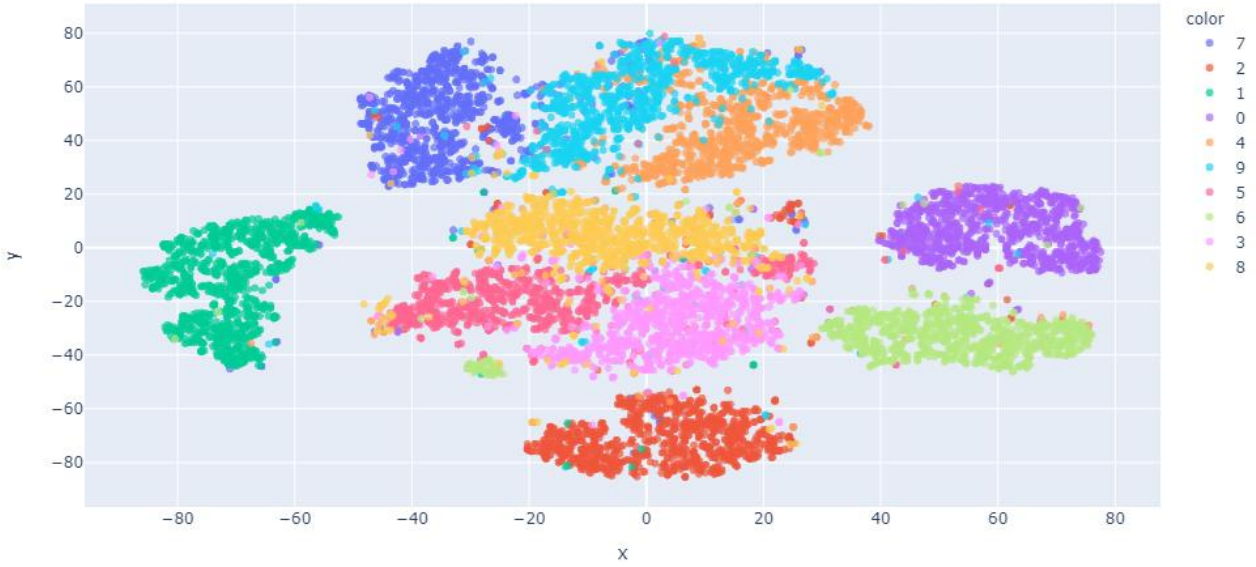


Here some Reconstructed images through the first 7 epochs:



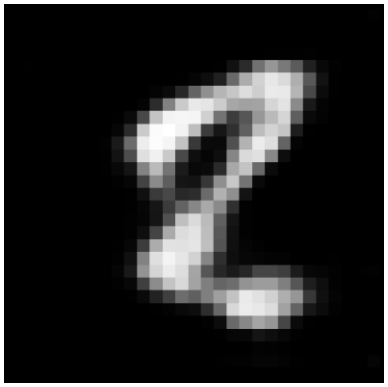
3. Latent space structure and Sample generation

Since the code is a 10-dimension vector we can't visualize the space. So, we used the T-SNE with PCA to reduce the codes generated from the images of the Test Set in 2-dimension vectors.
As you can see in the image below, the encoder did a quite decent clustering.

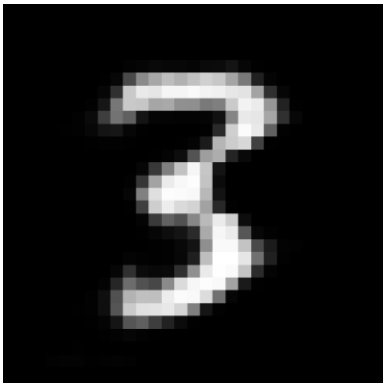


The **generation of new samples** for a 10-dimension vector is not easy, random values generate often bad images, because each feature control one "shape" of the digit, using random values we mostly generate strange numbers. That is not the case of the Variational Autoencoder as you will see in the next pages. Here some examples:

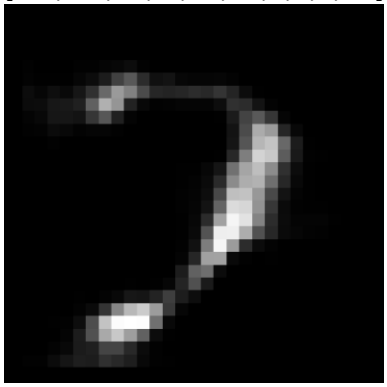
[-12,-25,10,-5,31,-5,7,8,9,10]



[-12,-25,10,-5,-31,60,7,8,9,10]



[-12,-25,10,-5,31,60,7,8,9,10]



4. Convolutional Denoising Autoencoder

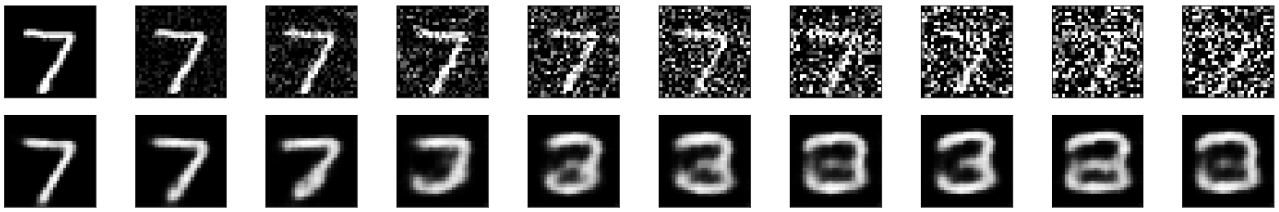
The architecture and hyperparameters search are the same used for the Convolutional Autoencoder, we just input a noised image and used the original image as comparison in the loss function during training and testing.

The gaussian noise is generated using `torch.randn` that returns random values from a normal distribution with mean 0 and variance 1 (= Gaussian noise). And just applied this to each pixel.

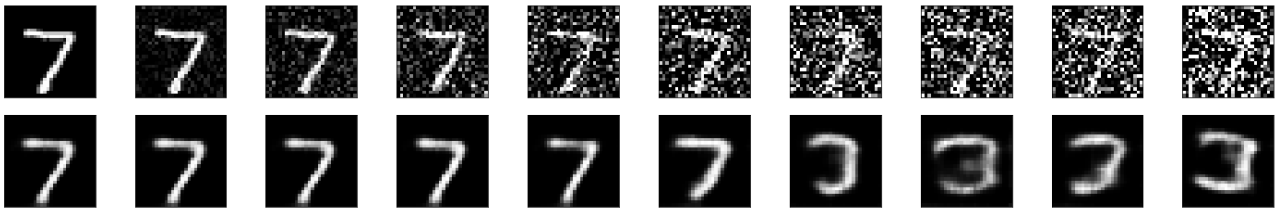
```
noisy_img = image + noise_factor * torch.randn(image.shape)
```

The `noise_factor` variable is a number from 0 (no noise) to 1 (complete noise).

This is an example of the **normal autoencoder** reconstruction for increasing values of noise:



This is an example of the **Denoising Autoencoder** reconstruction for increasing values of noise:



As you can see the denoise autoencoder can reconstruct much better the original image.

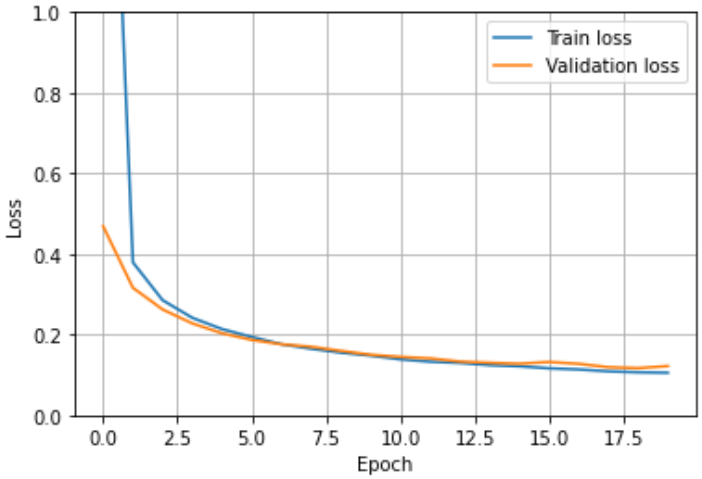
5. Supervised Classification task

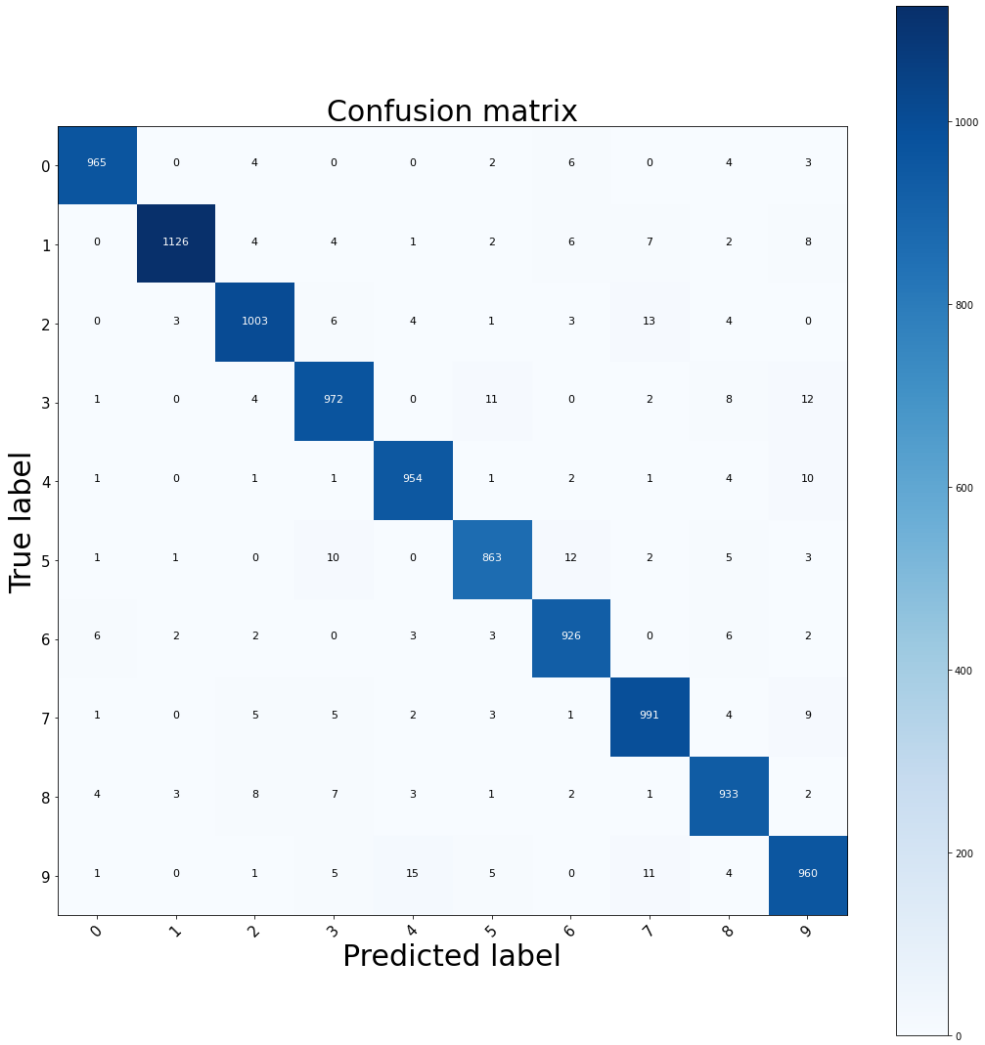
We added a fully connected layer composed of 10 neurons (one per class 0...9) on the encoder, then we freezed all layers except the last 2.

We obtained very good result without any fine tuning since the first training, 96.7% test accuracy against the 98.5% of the previous homework with an even simpler network. That means that unsupervised pre training is a powerful tool.

After some brief fine tuning the **best results** achieved are:

- Train Loss: 0.109
- Val Loss: 0.121
- Test Loss: 0.099
- Train Accuracy: 0.986
- Val Accuracy: 0.986
- **Test Accuracy:** 0.989





6. Variational autoencoder

We tried a simpler network for this part. The encoder architecture is composed of 3 fully connected layer with Relu activation, the decoder is symmetric.

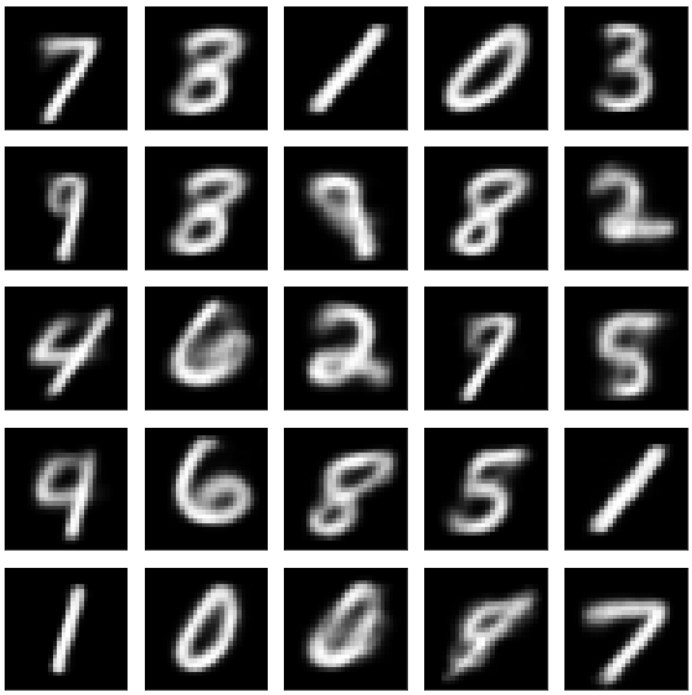
- **Encoder Input:** 784 values
- **Encoder FC1 number of neurons:** 512
- **Encoder FC2 number of neurons:** 256
- **Encoder Mean coding FC number of neurons:** 2
- **Encoder Variance coding FC number of neurons:** 2
- **Decoder Input:** 2 values (sampling from the output of the encoder)
- **FC1 number of neurons:** 256
- **FC2 number of neurons:** 512
- **Decoder Output:** 784 values

Since the sampling operation made at the beginning of the decoder is a discrete process, we cannot use backpropagation the normal way. So we use a different loss function made of two terms:

- **Reconstruction loss**: This loss compares the model output with the model input. We used **binary cross entropy**.
- **Latent loss**: This loss compares the latent vector with a zero mean, unit variance Gaussian distribution. The loss we use here will be the **KL divergence loss**. This loss term penalizes the VAE if it starts to produce latent vectors that are not from the desired distribution.

$$L_{KL} = -\frac{1}{2}\sum_{i}(1 + \log(\sigma_i^2) - \sigma_i^2 - \mu_i^2)$$

The **generation of images** in this case it's just made by sampling from the distribution (we give as input a random vector to the decoder).
Here some **generated images**:



As you can see the generation of images it is much better with a VAE then the previous ConvAE, now almost all images are recognizable numbers made with a much smaller network and fastest training. While giving random vectors in input at the big ConvAE, often results in strange images.