Unsupervised deep learning: autoencoders

Lorenzo Mancini 2019098

January 18, 2022

Abstract

The goal of this homework is to implement and test an autoencoder for solving an unsupervised deep learning task. Indeed, autoencoders are particular kind of neural networks that are able extract useful information from the inputs. We're going to investigate the ability of those models to reconstruct input images taken from the Fashion Mnist dataset. Furthermore, we show that such models can be exploited for denoising operations ad classification tasks (transfer learning). Finally we try to build and test a Variational autoencoder.

1 Introduction

The main idea underlying an autoencoder is the following: we want the algorithm to be able to extract and learn some useful information about inputs and store them in the so called latent space. The structure is divided into two parts:

- the encoder, which is able to extract information about inputs and store them in the latent space;
- the decoder, which tries to reconstruct the inputs using the information contained in the latent space;

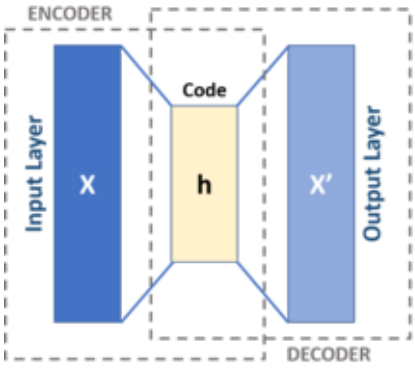

Figure 1: Structure of an autoencoder.

Clearly, the error of the autoencoder can be computed by simply taking the difference between the input and the reconstucted version: the more different they are, the higher the error will be, and we want this difference to be as small as possible.

1.1 Dataset

As said, the dataset used for this task is the FashionMNIST one, which containt 60000 items for the training and 10000 items for the testing part.

2 Standard Convolutional Autoencoder

2.1 Methods

As some standard architectures, the decoder is the specular version of the encoder. The latter consits of three convolutional layers followed by two fully connected linear layers:

- First convolutional layer with 8 filters of size 3, stride 2 and padding 1;
- Second convolutional layer with 16 filters of size 3, stride 2 and padding 1;
- Third convolutional layer with 32 filters of size 3, stride 2 and padding 1;
- First lineal layer with 64 neurons;
- Output layers with dimension equal to the encoded space dimension

As said, the decoder exactly mirrors the encoder. Hyperparameters are choosen following the random search technique: we run 30 training loops with a random combination of hyperparameters at each iteration and we look at the performance for 7 epochs. As loss function we consider the MSE loss.

2.2 Results

From the random search it results that the best parameters can be found around the following ones:

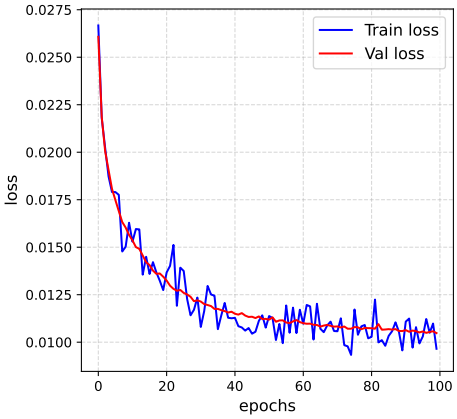

Figure 2: Train and validation loss for the standard convolutional autoencode.

- encoded space dimension equal to 20;
- batch size of 128;
- learning rate of 0.002;
- Adam optimizer;
- no regularization;

Thus we repeat the training for 100 epochs with the previous hyperparameters. In the following figures we can see two examples of reconstructed images

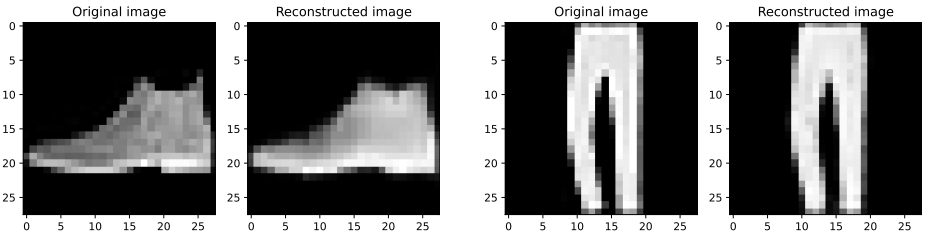

Figure 3: Example of reconstruced images.

We can also try to generate new samples from the latent space:

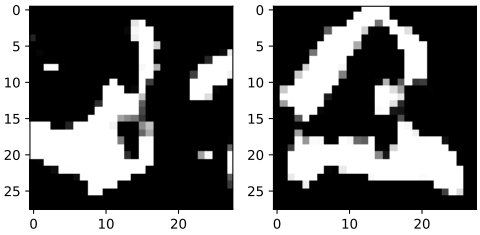

Figure 4: Example of new generated images.

2.3 Latent space visualization

We can try to visualize the latent space exploiting algorithms to reduce dimensions: PCA and t-SNE in our case. Clearly, we want to obtain clusters that can be recognized easily: indeed, as one can see from the figures, in the t-SNE results, clusters are more pushed away from each other than in the PCA.

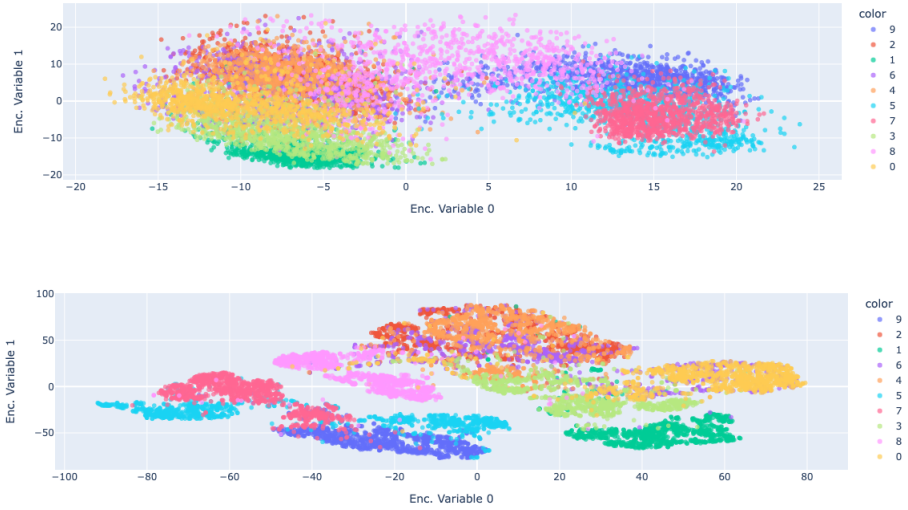

Figure 5: Latent space visualization with PCA (above) and t-SNE (below) algorithms.

3 Denoising autoencoder

Now, we feed the network with the same images as before but with some noise added: we would like to get as output the original image without noise. This operation is called denoising. The sctucture of the network is the same as the one used before except for the first linear layer which is now composed by 128 neurons. Moreover, we present the results obtained with the application of a batch normalization since the performance are slightly better than without batchnorm.

3.1 Methods

In order to do this we exploit the `torch.randn_like` function: we add to the original pixels some random numbers taken from a normal distributions withe the mean 0 and variance 1. Clearly, we can also use a noise factor in order increase (decrease) the noise. For our goal we're going to show the results obtained with a noise factor of 0.3. Again, we consider the MSE loss.

3.2 Results

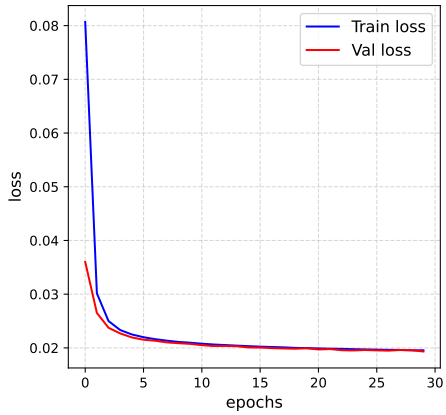

Figure 6: Train and validation loss for the denoising autoencoder

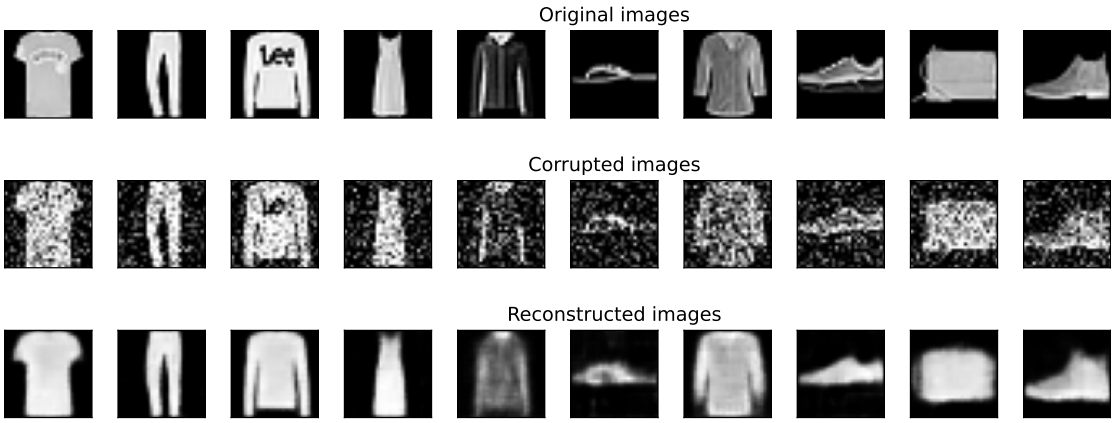

Figure 7: Some examples of original (first line), corrupted (middle line) and denoised (third line) images.

4 Variational Autoencoders

Variational Autoencoders behave in a quite different way with respect to the standard convolutional autoencoders. Indeed, here the encoders give as output the parameters of a probability distribution from which the latent vector will be sampled. Clearly, the aim is to get the simplest probability distribution as possible. Thus, the loss function measures how far we are from the standard normal distribution:

$$\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 + \mathrm{KL}\left[\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}\left(\mathbf{0}, 1_d\right)\right] \tag{1}$$

Since sampling is a discrete process, here we cannot use backpropagation. To this end, we can use reparametrization for the sampled latent representation as follows:

$$z = \Sigma\zeta + \mu, \qquad \text{with } \zeta \sim N(\mathbf{0}, 1) \tag{2}$$

4.1 Results

Here we show an example of new generated samples after having trained the network for 20 epochs:

Figure 8: New generated samples by the VAE after 20 epochs of training.

5 Transfer learning

We can exploit a pre-trained encoder in order to perform a classification task: the operation of taking a pre-trained model and adapt it to a new task is called transfer learning.

5.1 Methods

We can use the encoder trained for the standard autoencoder task and add some fully connected linear layers that we want to train. We add two layers, in particular a f. c. layer with 128 neurons and a final output layer with 10 neurons. This time we use the Negative Log-Likelihood as loss function. Furthermore, the activation function used between the two layers is the ReLU whereas we use the softmax for the output.

5.2 Results

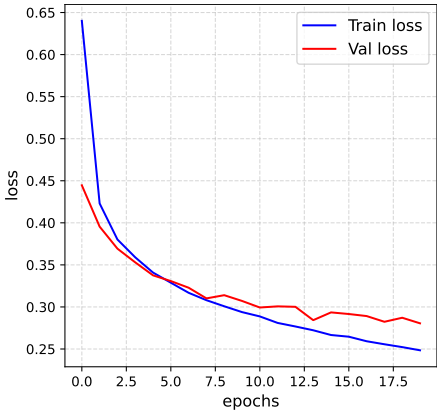

Figure 9: Train and validation loss for the transfer learning class.

The model reaches an accuracy of 88.8% in just 20 epochs.

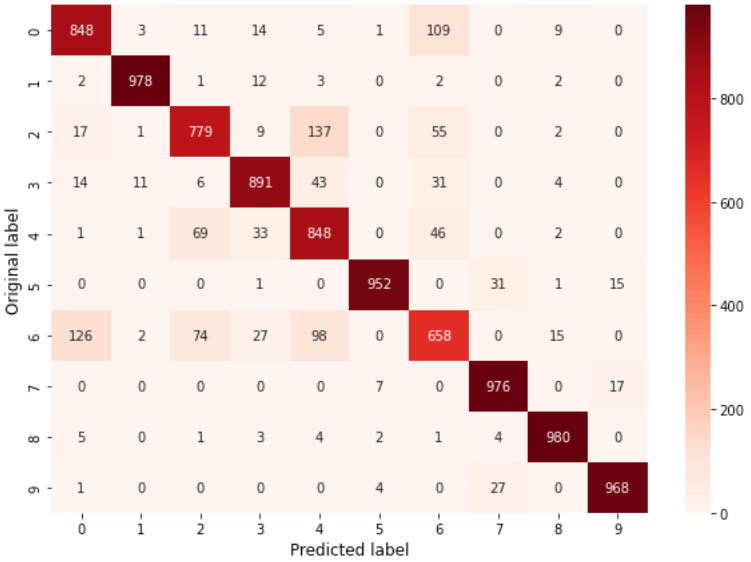

Figure 10: Confusion matrix of the classification task.

Comparing the results just obtained with the ones of the previous homework we can conclude that in this way the classification task is a bit faster: it reaches similar performances in just few epochs.