Neural Network and Deep Learning - A.A. 2021/22

Assignment 2

Unsupervised Deep Learning: Autoencoders

Fontana Francesco 2026924

13 May 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 and 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;

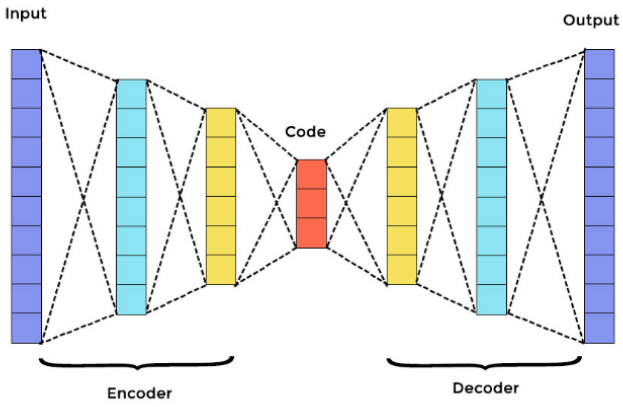

Figure 1: Structure of an autoencoder

Clearly, the error of the autoencoder can be computed by simply taking the difference between the input and the reconstucted version: our goal is to minimize the loss function and this translates in trying to make this difference as small as possible

1.1 Dataset

As already mentioned in the abstract, the dataset used for this task is the *FashionMNIST*, which contains 60000 items for the training part and 10000 items for the testing one.

2 Standard Convolutional Autoencoder

2.1 Methods

The structure of the *encoder* we used, consits of 3 convolutional layers followed by 2 fully connected linear layers; here below in detail:

- 1^{st} convolutional layer with 8 filters of size 3, stride 2 and padding 1;
- 2^{nd} convolutional layer with 16 filters of size 3, stride 2 and padding 1
- 3^{rd} convolutional layer with 32 filters of size 3, stride 2 and padding 1;
- 1^{st} linear F.C. layer with 64 neurons;
- 2^{nd} linear F.C. layers (output) with dimension equal to the encoded space dimension;

As said, the decoder has the same structure of the encoder but mirrored. Hyperparameters are choosen following a random grid-search technique: 30 training loops were performed with a random combination of hyperparameters at each iteration and with a training duration of 10 epochs (see tab.1). As loss function we consider the *MSE* loss.

	Dim. Lat. Space	Optimizer	Learn. rate	Batch size
Random Search	$[2, 20]$	Adam, RMSprop, SGD	$[10^{-4}, 10^{-1}]$ (log-uniform)	$[128, 256, 512]$
Best result (30 Trials)	19	Adam	0.0095	256

Table 1: Hyperparameter space and best result

2.2 Results

From the random search it results that the best parameters can be found around the values indicated on tab.1).

Thus we repeat the training for 100 epochs with these hyperparameters, implementing a k-fold cross-validation procedure (with $k = 3$), obtaining a loss of 1% and in fig.2 we could see the behaviour of the losses during the training procedure. Furthermore, in fig.3 we can see some examples of reconstructed images by the autoencoder.

Moreover, we could also try to generate new samples directly from the latent space, for example, in fig.4 there are 3 images generated from a random input normally distributed.

3 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*.

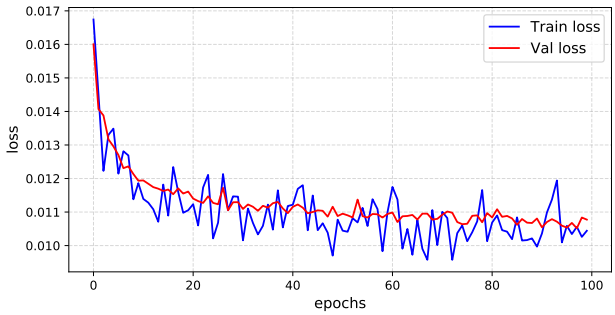

Figure 2: : Train and validation loss for the standard convolutional autoencoder.

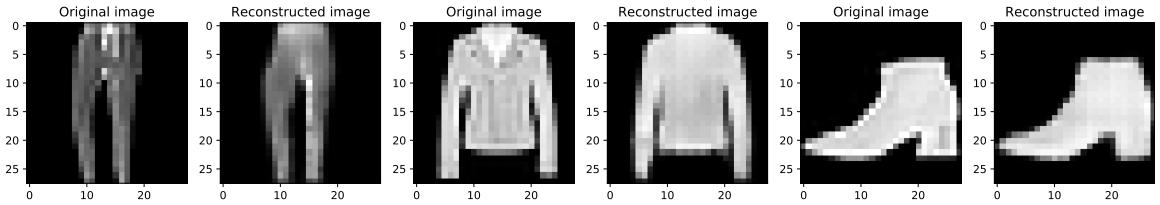

Figure 3: : Example of three reconstruced images from the training dataset

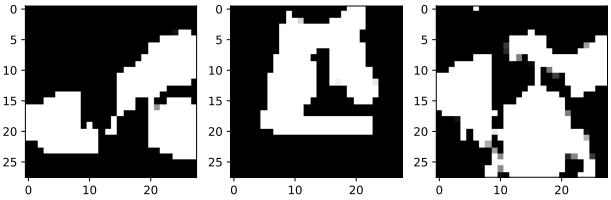

Figure 4: : Example of new generated images from a gaussian random input.

3.1 Methods

We used the encoder trained for the first part *(Section 2)* and add some fully connected linear layers that we want to train. In our case 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, we use different activation function for the layers i.e. *ReLU* between the two layers and *Softmax* for the output.

3.2 Results

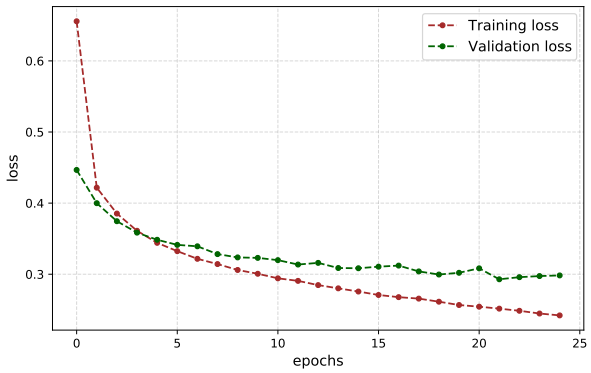

Figure 5: Train and validation loss for the transfer learning .

From fig.5 we could see a little precence of overfitting, nevertheless we achieve a remarkable accuracy score of 88.4% in just 25 epochs. Below there is also the corresponding confusion matrix obtained from the comparison between the network's outputs and the true labels (fig.6).

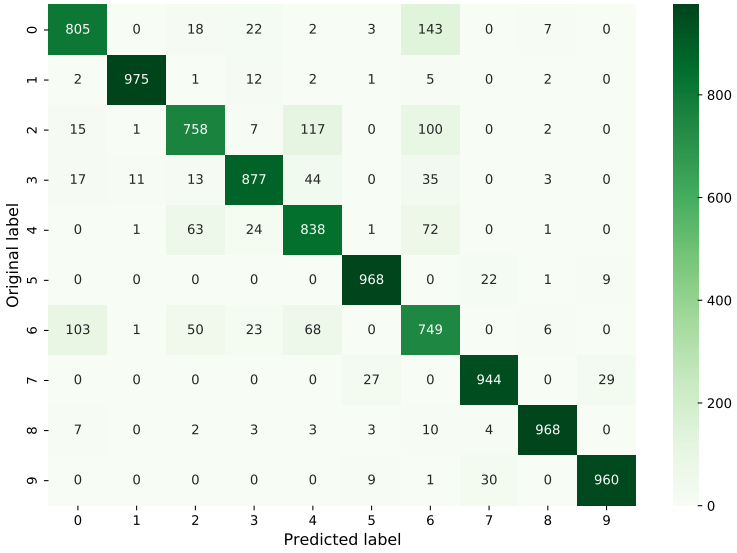

Figure 6: Confusion matrix of the classification task.

Comparing the results just obtained with the ones of the previous homework we can conclude that in this way the classification task has reached approximately the same accuracy but in a quarter of the time (just 25 epochs).

4 Latent space visualization

In order to visualize the latent space, we had exploited algorithms to reduce the dimensionality, such as *PCA* and *t-SNE*. Clearly, we want to obtain clusters that can be easily recognized: indeed, as one can see from the figures below (fig.7), in the t-SNE plot, clusters are farther from each other than in the PCA one.

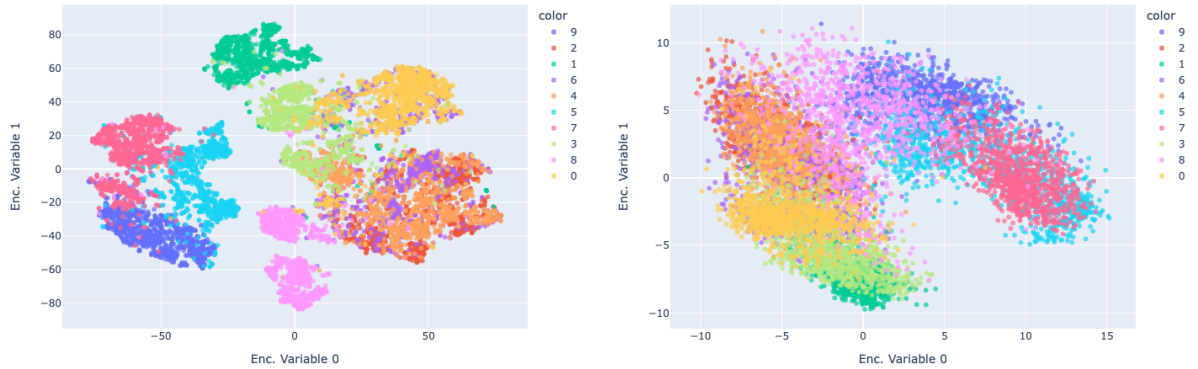

Figure 7: : Latent space visualization with t-SNE (left) and PCA (right) algorithms.

5 Denoising autoencoder

In this part, we try to implement a denoising autoencoder: we feed the network with the same images as before but adding some noise to the input. The scope is to be able to remove the noise, trying to get in output the original image (denoising procedure). The structure of the network is the same as the one used before except for the first linear layer which is now composed by 128 neurons. This procedure has been test both with and without a batch normalization layer in the network and since the results within it are slightly higher, here below are reported those only.

5.1 Methods

In order to do that we exploit the *torch.rand_like* function: we add to the original pixels some random numbers taken from a normal distributions with 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 as before, we consider the MSE as loss function.

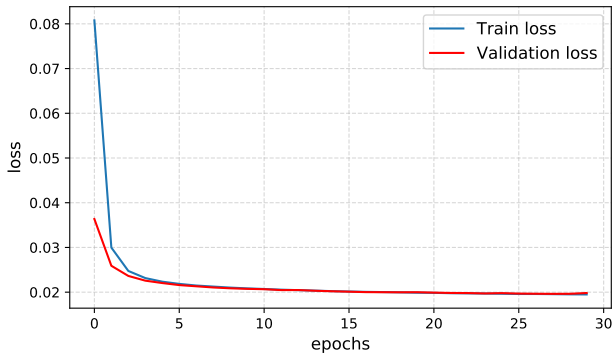

Figure 8: Train and validation loss for the denoising autoencoder.

In fig.8 are shown the curves of the losses during the training task, while in fig.9 there are some example of how the model has been able to remove the noise from the original samples.

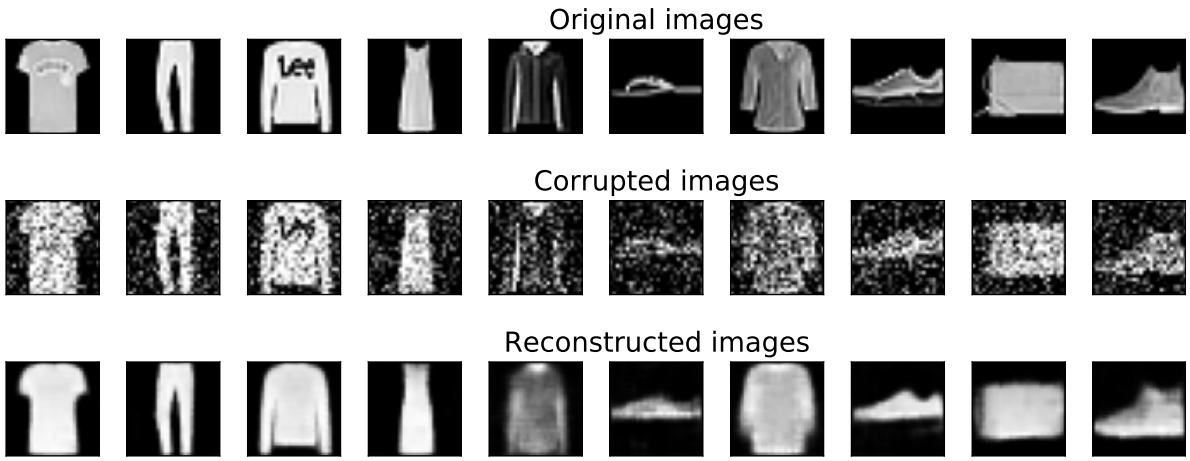

Figure 9: Some examples of original (1^{st} row), corrupted (2^{nd} row) and denoised (3^{rd} row) sample.

6 Variational Autoencoders

Variational Autoencoders behave in a quite different way with respect to the standard convolutional autoencoders. Indeed, here the encoder does not output directly a latent vector, but rather the parameters of a probability distribution from which the latent vector can be sampled. The ability to choose that distribution, and forcing it to be the "simplest" possible through regularization, allows to obtain an unsupervised learner that is more robust to noise, and learns more interpretable representations.
So, we change the architecture of the previous model in order to give as output the parameters of a multivariate Normal distribution $\mathcal{N}(\mu, \Sigma)$ with diagonal covariance matrix $\Sigma = diag(\sigma_1^2, ..., \sigma_d^2)$ chosen for its simplicity and efficiency. In order to keep and use backpropagation we have to do a reparametrization for the sampled latent representation as follows:

$$z = \sigma\zeta + \mu, \quad \text{with } \zeta \sim \mathcal{N}(0, 1)$$

Since the randomness is completely contained in the separate ζ factor, the expected value of z is differentiable with respect to the network's parameters used to compute μ and σ. We make use of MSE as loss function with a regularization term which forces the learned distribution to be as close as possible to a standard Normal distribution:

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

where x denotes an input sample, $\hat{x}$ is the reconstructed sample by the decoder, and KL is the Kullback-Leibler divergence, measuring the distance between two probability distributions.

The model has been trained for 100 epochs, with no dropout, Adam as optimizer with learning rate $\lambda = 0.001$ and a latent space of dimension 19; A few reconstructed samples are shown in fig.10 . The quality is a bit lower than that of the convolutional autoencoder from sec. 2, due to the much more significant regularization. However, the final representations are more robust: a small change in the input is mapped to a small change in the latent space.

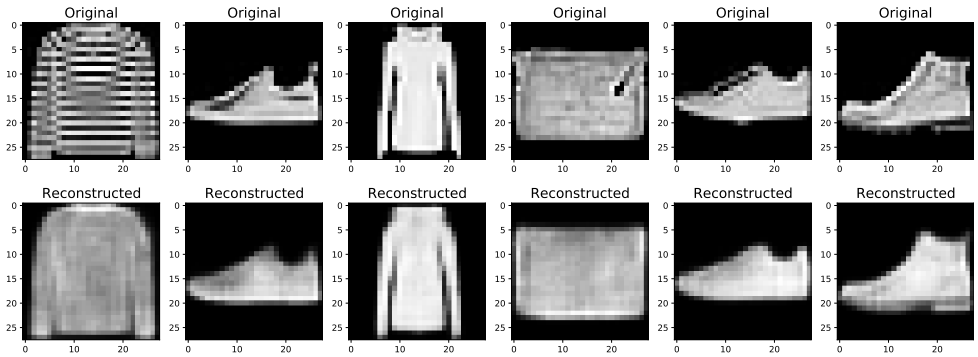

Figure 10: Samples reconstructed by the variational autoencoder.

In the end, in fig.11 in (*Appendix*) we plot an example of new generated images from a random input starting from the latent space (input has the dimension of the latent space).

7 Appendix

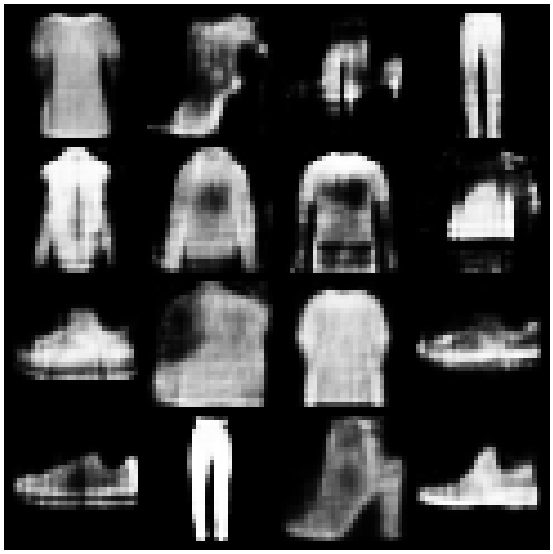

Figure 11: New generated samples by the VAE after 100 epochs of training.