Homework 4 - Autoencoder

Alberto Zancanaro - ID: 1211199

I. Introduction

This homework was focused around the creation and testing of an Autoencoder and evaluate its results in the copying of image and the clean up of image. We will introduce the work, explain the code implementation and the problem encounter. In the end we will show the results.

Autoencoder are a type of unsupervised machine learning model usually used to learn efficient data coding. They are composed by two element: the encoders that have to codify the data in a lossy manner and the decoder that have to reconstruct the data from the compressed representation.

II. Problem Structure

The autoencoder will first training in the task of copying of the input image. After that we corrupt the original image and try to training the autoencoder to clean the corrupted image. The last tests were focused on the generative property of the autoencoder.

III. Code Implementation

A. Network Model

The Autoencoder is divided in two components: encoder and decoder. The structure of the encoder consist in a convolutional network followed by a standard feedforward neural network. The decoder has the same structure but mirrored. The parameters are equals between decoder and encoder; in table 1 and table 2 you can see the parameter for the encoder. The last layer of the encoder (and the first of the decoder) has n as value because it is the *encoded space dimension*[1] and changes thorough the various tests.

The activation function between every layer is the SELU (scaled exponential linear units):

$$selu(x) = \lambda \begin{cases} X, & \text{if } x > 0 \\ \alpha e^{x} - \alpha, & \text{if } x \leq 0 \end{cases}$$

with λ and α fixed parameters of the functions and set to their defualt value provided by PyTorch[2]. The only exception is the last layer of the decoder that use a sigmoid activation function. This was done because the pixel image have values between 0 and 1 and the output of the sigmoid is a value between 0 and 1. In this way we do not need to rescale the output value.

One of the biggest difficulties was to rewrite the code in order to has no static structure for the network. In this way we can create a net network by simply specifying the parameter in the train script. In this way it is also possible use different structure for encoder and decoder[3].

[1]Sometimes in the report will be abbreviated with esd

[2]The default values are $\lambda = 1.0507009873554804934193349852946$ and $\alpha = 1.6732632423543772848170429916717$

Input Channels	Output Channels	Kernel Size	Stride	Padding
1	6	(3,3)	(2,2)	(1,1)
6	39	(3,3)	(2,2)	(1,1)
39	72	(3,3)	(2,2)	(0,0)

TABLE 1: Parameters Convolutional Part

Input Features	Output Features	Bias
648	145	True
145	98	True
98	52	True
52	n	True

TABLE 2: Parameters Feedforward Part

More precisely the code allows to customize:

- Convolutional part: the kernel, the number of layer and the number of channels.
- Linear Encoder: Number of layer and number of neurons for each layer.
- Linear Decoder: Number of layer and number of neurons for each layer.
- Activation Function: customizable but equals for all layers (both convolutional and normal).
- Device: specify the physical device that will be use for the training. If not specify the CPU will be selected bu default.

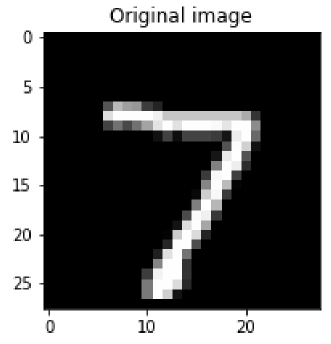
All the layers and activation function were implemented thorough *Pytorch*.

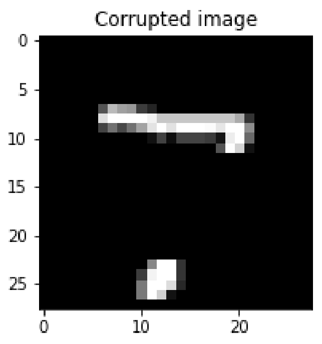
B. Dataset

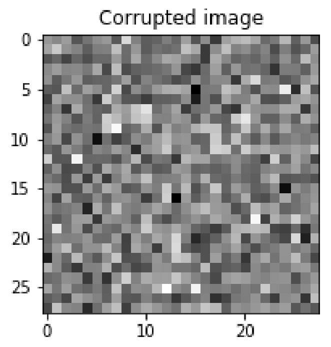
The data used is the MNIST dataset provided by the *PyTorch* framework. It consist in image of hand written digit in grayscale images. The dimension of the image is 28×28 pixels.

In figure 1 it is possible to observe an example of the image used during the training. More precisely in image 1a there is an original version of the image (that the autoencoder learn to copy) and in image 1b, 1c and 1d there is always the same image but corrupted with different type of noise (in order: occlusion, gaussian and gaussian + occlusion)

[3]For simplicity during the test we use the same strucuture for both encoder and decoder. Also we code didn't allow to have the convolutional part different. This was done to simplify the passage from linear to convolutional layer.


(a) Original Image


(b) Occlusion Noise


(c) Gaussian Noise

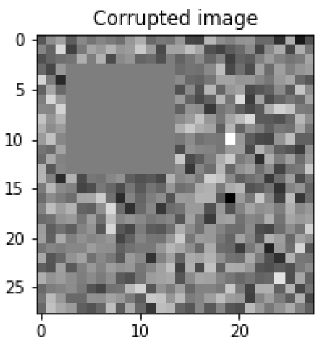

(d) Gaussian + Occlusion Noise

Fig. 1: Example of the image in the dataset

C. Training

We use the *Dataloader* class provided to *PyTorch* to easily load the data during the training. We train each network on 60 epochs with the ADAM optimization function with a learning rate of 0.001, MSE as a loss function and a batch size of 512.

IV. Results - Image Copying

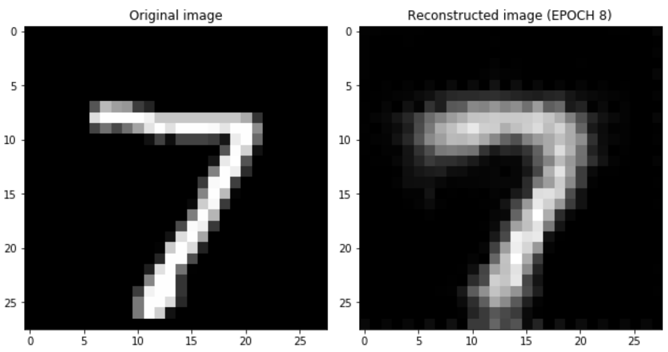
In this part the tests were focused on the ability of the autoencoder to copy an input image and how the copy will be affected by the dimension of the *encoded space dimension*. The results can be seen in figure 2.

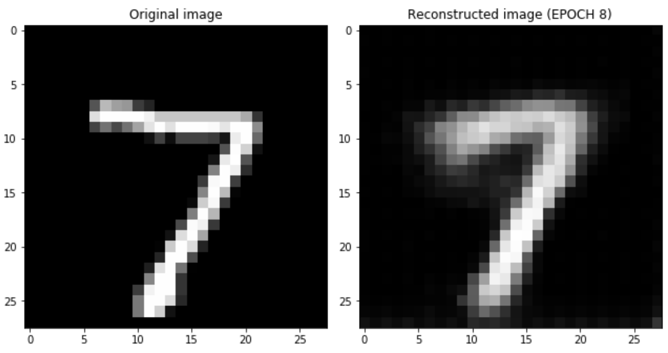
From the analysis of the results we can immediately notice that with this network there is no major difference in the number of epoch needed to reconstruct the shape. In fact both the network with an encoded space dimension of 2 and the network with an encoded space dimension of 6 after 8 epochs are capable of recreate a blurry seven.

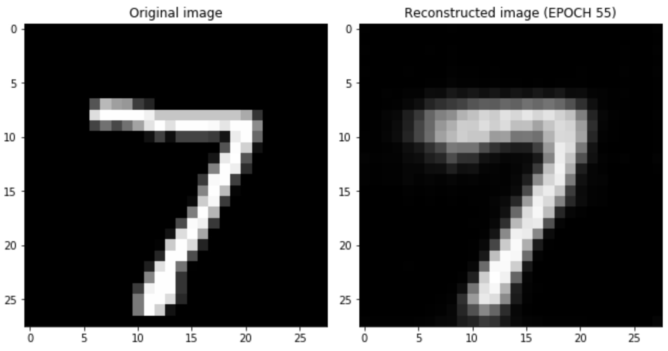
The main difference is noted in the long run. In fact we see that the reconstruction of network with $esd = 6$ are more sharp and precise. Instead the reconstruction of the network with $esd = 2$ remain blurry.

This is coherent with the theory since a lower dimension of the encoded space dimension means that the network has a poorer condensed representation of the input. Consequently the ability of network to copy the input will be reduced and the copy will be less precise.

In image 3 we can see the difference in test error. We can immediately notice that the error for $esd = 2$ remain more or less constant after 15 epochs and also remain higher than the


(a) Reconstruction after 8 epochs (encoded space dimension = 2)


(b) Reconstruction after 8 epochs (encoded space dimension = 6)


(c) Reconstruction after 55 epochs (encoded space dimension = 2)

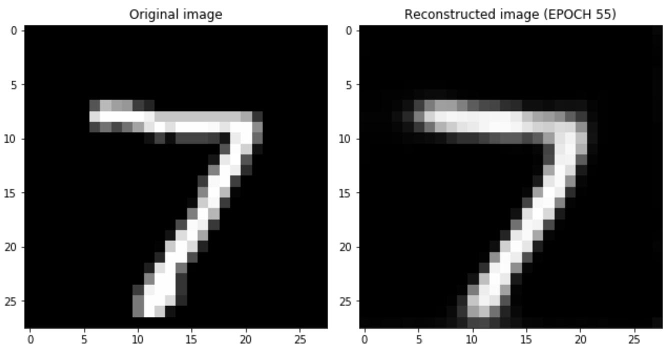

(d) Reconstruction after 55 epochs (encoded space dimension = 6)

Fig. 2: Results of the copying

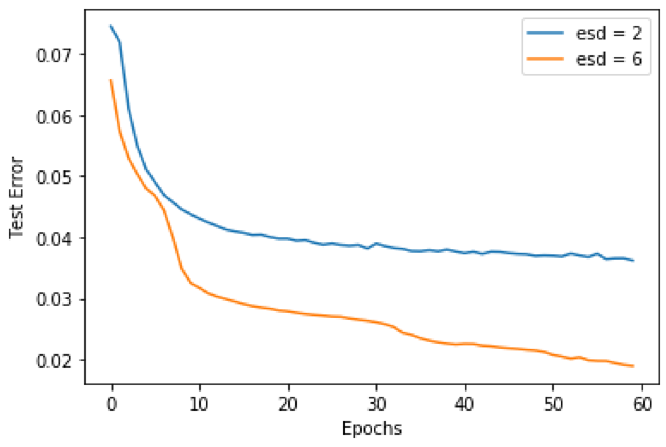


Fig. 3: Test error

error for $esd = 6$. This is another confirm of the theory since the network with $esd = 2$ produce image that are less similar to the original so the difference between the original and the copy will be higher.

V. Results - Clean Image

In this part the tests were focused on the ability of the autoencoder to clean a corruptedinput image and how this process will be affected by the dimension of the *encoded space dimension*. We will discuss only the more complex case (gaussian + occlusion) since the other case are simplification of the latter. Nevertheless we will show some of the results of the simpler case.

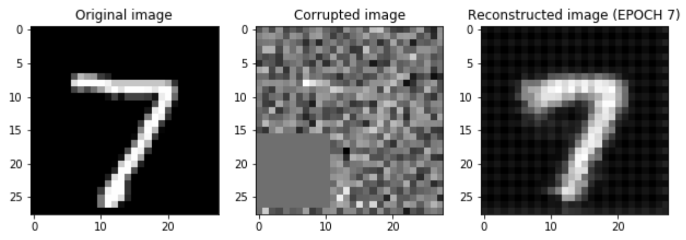
Even in this problem we see that after a few epochs (7) the network more or less manage to produce the correct output in both cases (image 4a and image 4b). LIke in the copying of the image the output with $esd = 6$ is more precised and less blurry.

An interest thing to noticed is the inability of the network with $esd = 2$ to clean the image in the correct way in advance phase of the training. In the epoch 31 in fact the network with $esd = 2$ produce an output that resemble a 9 more than the original 7. This was not the only case in which the network fails. In fact considering the epoch from 10 to 60 the network output a wrong number 12 times; instead the network with $esd = 6$ was wrong only 2 times[4].

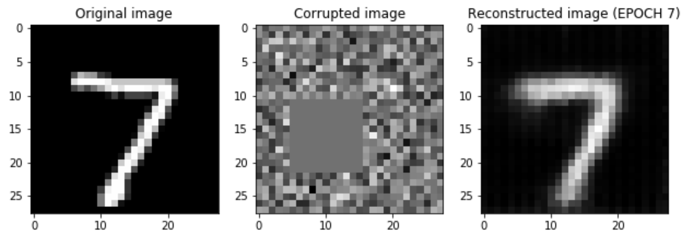
It should also be noted that the error was to confuse the 7 with the 9. This number can be easily confused by the model since have a lot of common traits.

In the cleaning process also the output of the network of with $esd = 6$ are more blurred respect the output of the same network in the copying task. This is quite logic since in this case the input is a corrupted version of the image and a part of the original data can be completely lost affecting the reconstruction skills of the network.

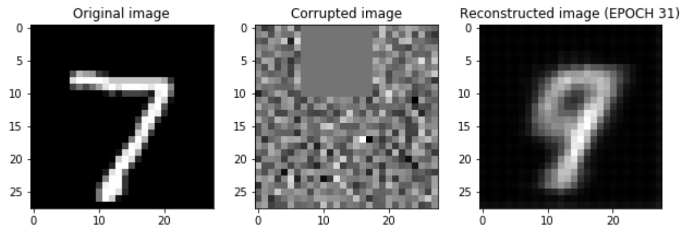
[4]We consider the epoch after the tenth because the first 10 are considered a sort of minimal training.



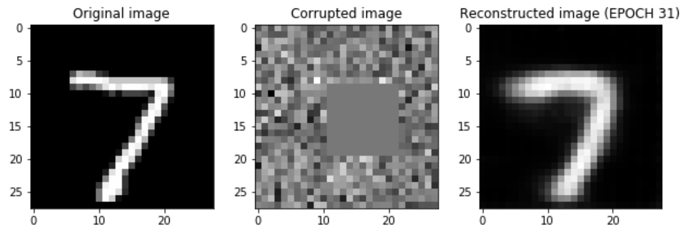
(a) Cleaning after 7 epochs (encoded space dimension = 2)



(b) Cleaning after 7 epochs (encoded space dimension = 6)



(c) Cleaning after 31 epochs (encoded space dimension = 2)



(d) Cleaning after 31 epochs (encoded space dimension = 6)

Fig. 4: Results of the cleaning

In figure 5 it is possible to observe the performance of the network with $esd = 6$ in the cleaning of images corrupted only by occlusion or gaussian noise.

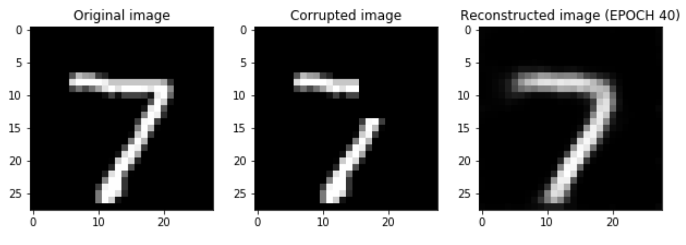
Since all example are with the number 7[5] in figure 6 are shown example of the cleaning with other numbers. We can noticed that the number is generally reconstructed, even if not perfectly and with the shape a little bit different.

VI. Results - Generative Model

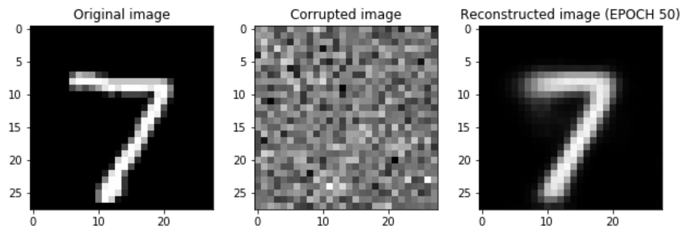
Since the decoder create the output from n value (the dimension of the encoded space) theoretically the decoder can be use as generative model. More precisely if we give in input a random vector of n numbers the decoder will have to output a results that resemble a number.

The degree of similarity depend of the output with a real number depend of course by the input value. In this test we do not try to search the best values for this task but we only

[5]Because is the first image of the dataset and we use that in automatic.

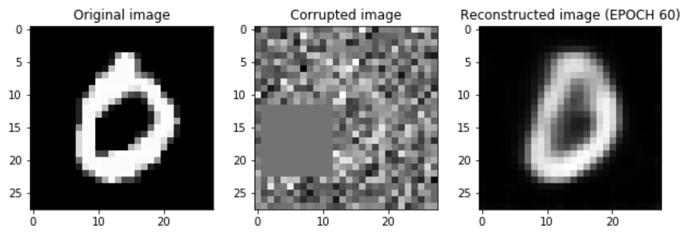


(a) Cleaning with only occlusion noise

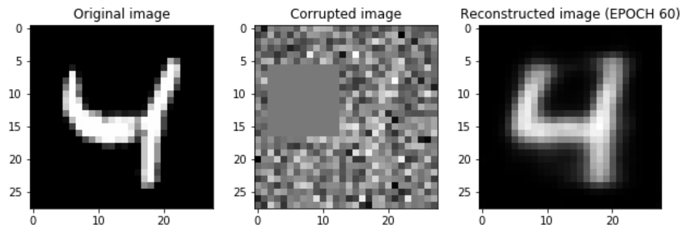


(b) Cleaning with only guassian noise

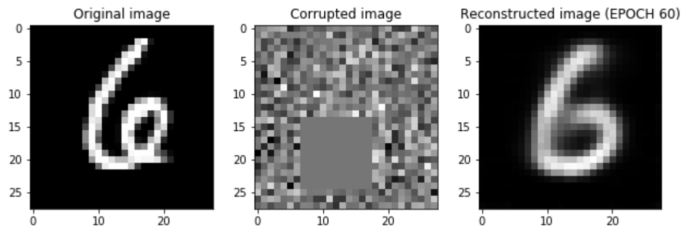
Fig. 5: Example of other noise



(a) Cleaning of a 0



(b) Cleaning of a 4



(c) Cleaning of a 6

Fig. 6: Example of other numbers

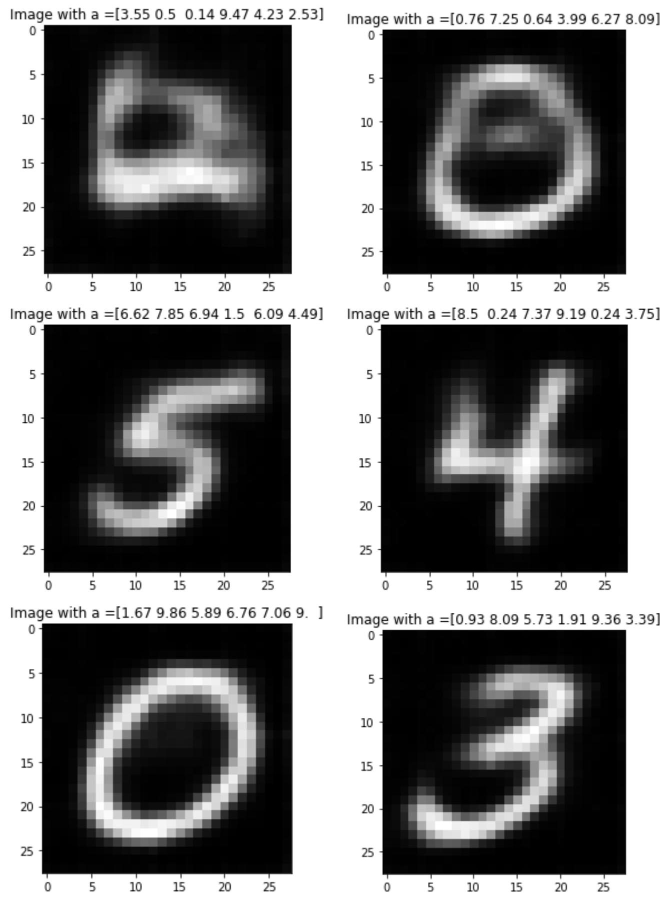


Image with a =[3.55 0.5 0.14 9.47 4.23 2.53]
Image with a =[0.76 7.25 0.64 3.99 6.27 8.09]
Image with a =[6.62 7.85 6.94 1.5 6.09 4.49]
Image with a =[8.5 0.24 7.37 9.19 0.24 3.75]
Image with a =[1.67 9.86 5.89 6.76 7.06 9.]
Image with a =[0.93 8.09 5.73 1.91 9.36 3.39]

Fig. 7: Results of the random generation

want to demonstrate the ability of the network of generate image that are similar to hand written digits.

All the results can be seen in image 7. The vector of random values (called a) used as seed is shown above each image. Excluding some senseless results, others are quite good and seems really written by person instead to be generate from a computer.