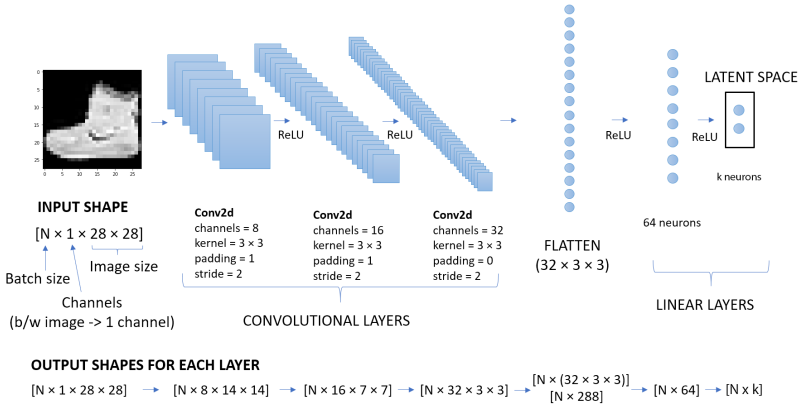
Neural Networks and Deep Learning
Homework 2

Amedeo Giuliani - 2005797

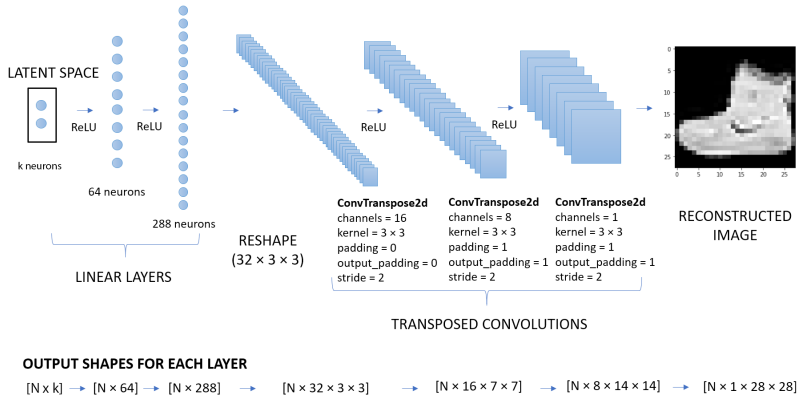
January 11, 2022

1 Network model

A convolutional autoencoder is made of two distinct neural networks: a *Convolutional Neural Network* (CNN) which compresses input images into the latent space, called encoder, and another CNN which tries to reconstruct images from latent codes. In Fig. 1 it is reported the overall architecture. The number of neurons k of the latent space has been set to k = 5, initially.



(a) Encoder.



(b) Decoder.

Figure 1: Autoencoder architecture.

In Fig. 2 are reported the training and validation loss trends in function of the number of epochs elapsed. The validation loss is computed over a small piece of the training set. In fact, a validation set has been created from the training set, reserving the first 1000 samples of the latter. After training, we can see how the network reconstructs the images. We take the first three images of the test set, we feed them to the network, and observe the output of the decoder. The results are shown in Fig. 3.

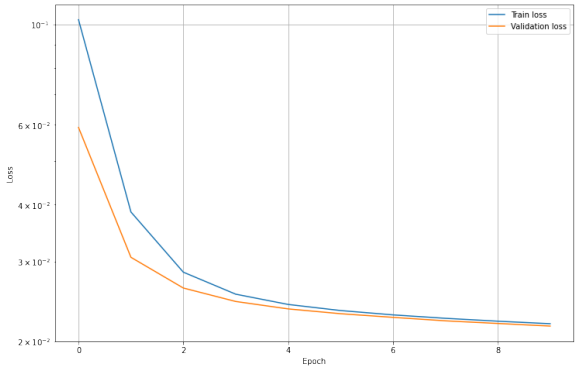


Figure 2: Training vs validation loss trends.

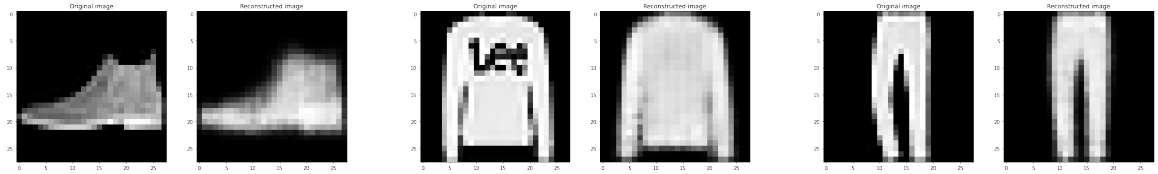


Figure 3: Some examples of image reconstruction.

2 Hyperparameter optimization

We implement grid search with the parameter grid listed in Table 1

Table 1: Hyperparameter dictionary

Hyperparameter	**Values**
Latent space dimension	$[5, 10]$
Weight decay	$[10^{-5}, 10^{-4}, 0]$
Dropout	$[0, 0.25, 0.5]$
Batch size	$[300, 600]$
Number of epochs	$[10, 20]$

With this approach, the returned optimal set of parameters is: [Batch size: 300, Number of epochs: 20, Latent space dimension: 5]

3 Fine-tuning

Now we want to train a layer for a supervised multi-class classification task on top of the encoder. To do so, the convolutional layers of our encoder have been "freezed", that is, the neurons inside them cannot learn anymore, and we swapped the last linear layer representing the latent space with a linear layer of 10 neurons, as much as the classes of our dataset are.
After doing this, we can proceed with the training session. We use the cross entropy as the loss function, and for a fair comparison, we train the network for the same amount of epochs in the homework 1, i.e., 50 epochs. By comparing the test set with the output of the new model, the accuracy is $\sim 85\%$. Furthermore, now the training takes considerably less time with respect to the CNN in homework 1.

4 Latent space exploration

Suppose we want to visualize our high-dimensional data (10 dimensions since we have ten classes). We can thus use *Principal Component Analysis* (PCA) or *t-distributed Stochastic Neighbor Embedding* (t-SNE) to reduce the dimensions to only 2 and plot the data with a scatter plot.

4.1 PCA

With PCA the classes are too packed together. The only two clusters of points easily recognizable are the ones representing the class 0, 8, and 9, which correspond to the trouser, the bag, and the ankle boot, respectively. In fact, these are the easiest recognizable objects in the data set.

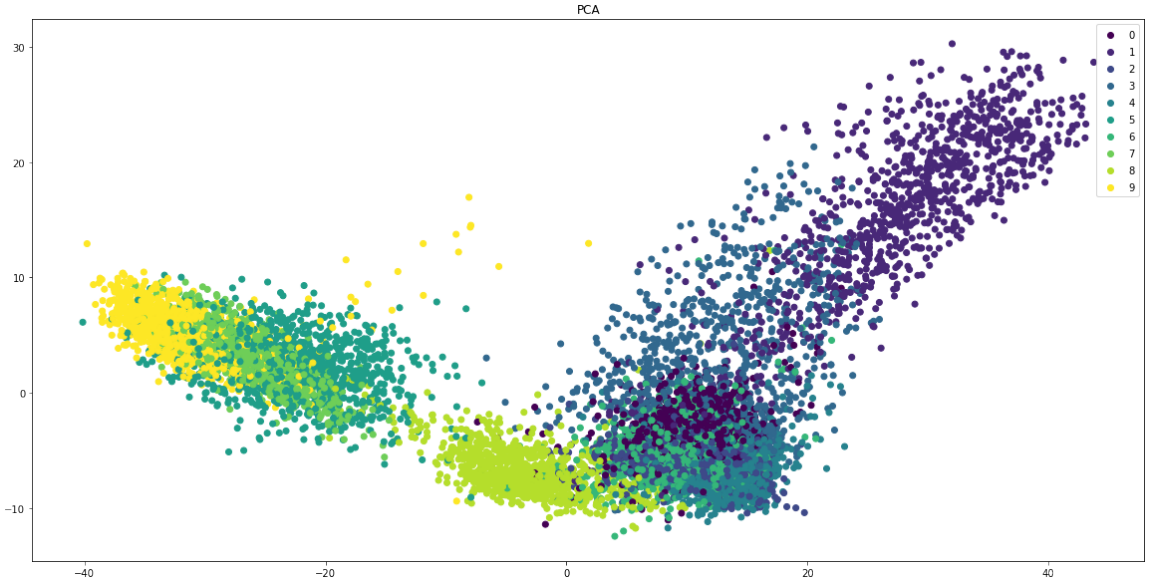


Figure 4

4.2 t-SNE

With t-SNE the situation is better since this approach involves non-linearities. However, there are still some clusters tightly packet together, namely, the points representing class 3, 4, and 6, which correspond to the dress, the coat, and the shirt. These are all pretty similar objects, indeed.

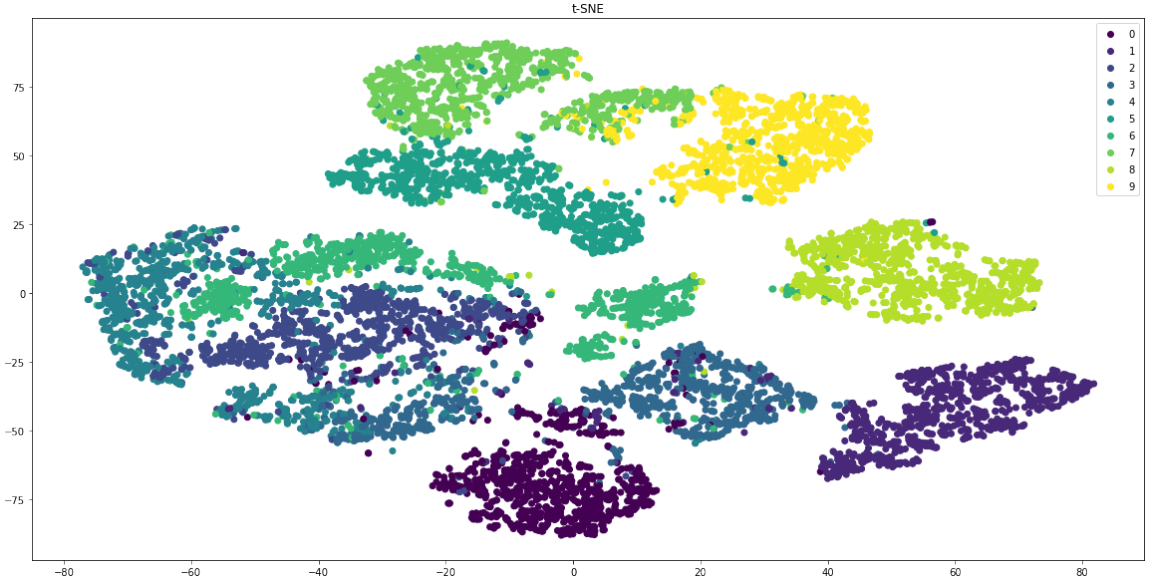


Figure 5

4.3 Sampling from latent space

We generate random latent codes and feed them to the trained decoder to see how it does. As we can see from Fig. 6, the generated images are quite bad: we can distinguish a bag, a dress, a t-shirt but they are all very blurry. This is because the latent space of an autoencoder is discrete. In fact, this architecture is useful for, e.g., de-noising images, but it is not a generative model.

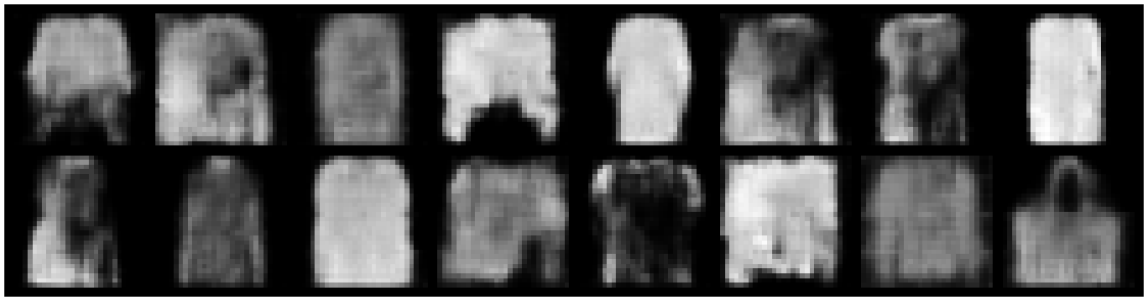


Figure 6: Generated images from random latent codes.

5 Simple DCGAN

We try to implement a simple *Deep Convolutional Generative Adversarial Network* (DCGAN) and train it over the Kuzushiji-Kanji data set. The architecture of such network is depicted in Fig. 7: the generator takes in input a sample from the latent space, and generates an RGB image; the discriminator instead should be able to take that image as input and tell if it is real or fake (that is, it belongs to the data set or not).

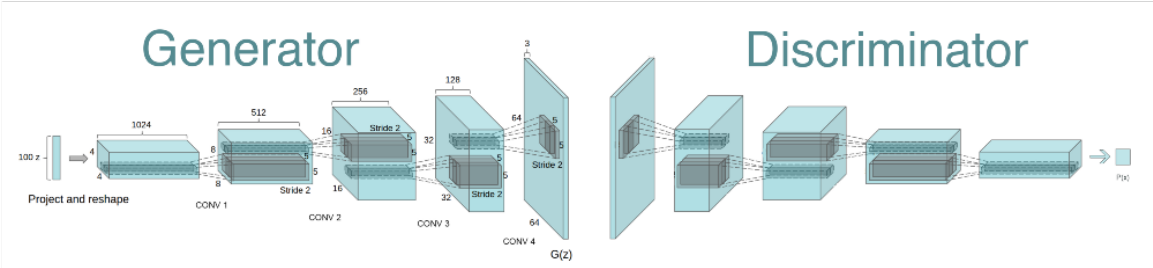
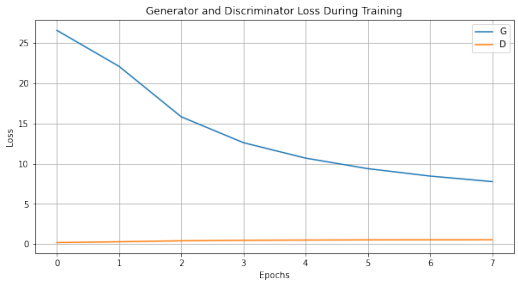
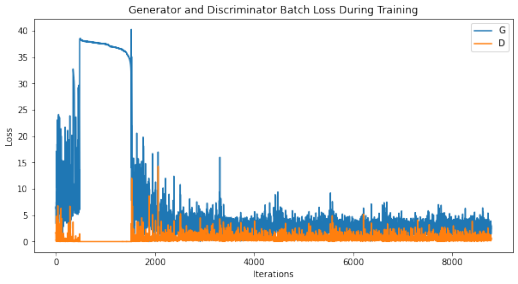


Figure 7: DCGAN architecture.

Since the training of this model is quite computationally demanding, we just trained it for 50 epochs. However, even with these few iterations, the results are pretty good. In Fig. 8a are plotted the global loss curves of the discriminator and the generator, while in Fig. 8b are plotted the losses at "batch level".



(a) Discriminator and generator loss history.



(b) Discriminator and generator batch loss history.

Figure 8

Judging by the above plots, the discriminator is doing quite well since it manages to keep the "complete" loss low while the generator struggles to maximize the probability that the discriminator tells that the fake images are real. But how good are the fake images are? If we plot the images generated by the generator in the last epoch along with the first 64 real images from the data set, as in Fig. 9, we immediately recognize that fake images are quite good.

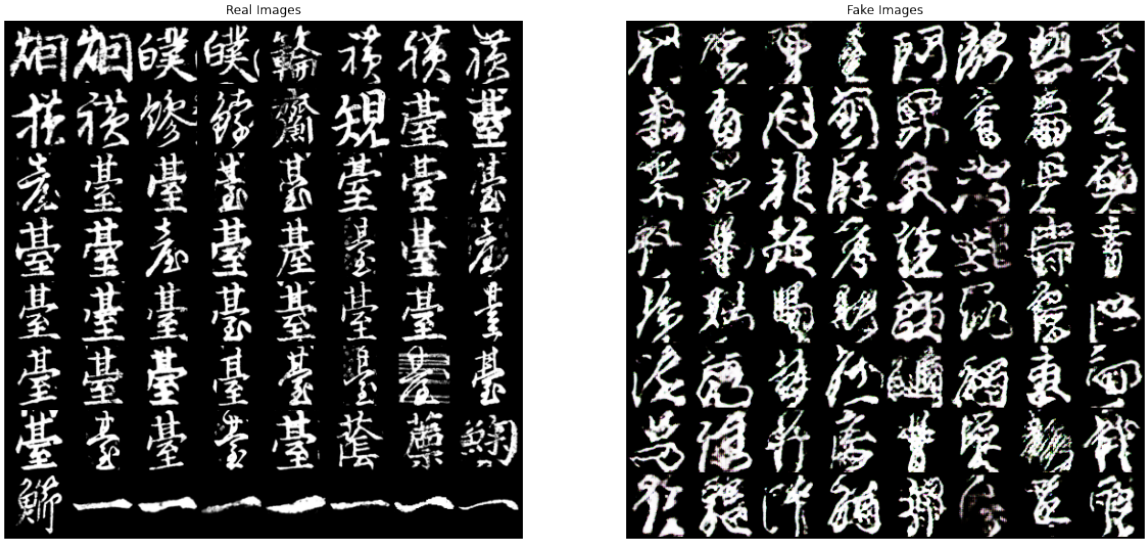


Figure 9: Some examples of real and fake images.

After training has ended, we can continue generating fake images by taking the generator and feed it with with random noise, just like we were doing in the training. Some examples are depicted in Fig. 10.

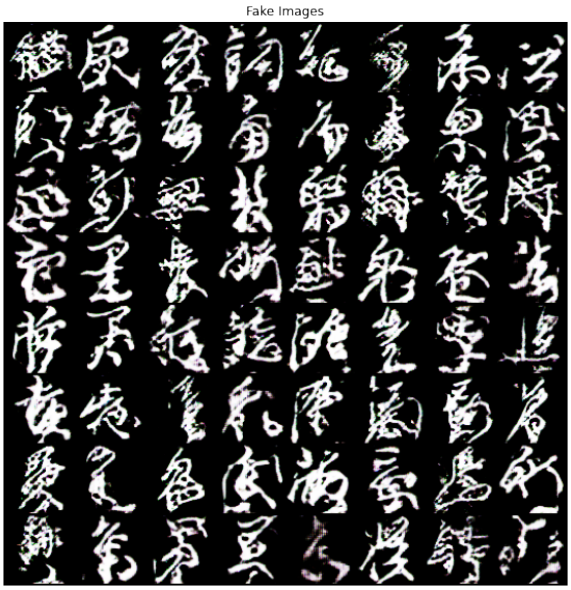


Figure 10: Some examples generated images after training.