HOMEWORK 2 NNDL

Zambelli Francesco (2029014)

Academic Year 2021-2022

1 AUTOENCODER

The first task of the homework consists in implementing, testing and exploring the latent space of an Autoencoder. This kind of architecture has the goal to compress an input image in a vector of dimension *lantent_ dim* (**encoder**), and from this to reconstruct an image as similar as possible to the initial one (**decoder**).

The first step is the **Dataset** importation and transformation (FashionMNIST), the we proceed with the definition of the **Autoencoder** architecture, firstly with generic latent dimension in order to optimize the reconstruction task. Then the architecture is trained and the results are displayed. After this with some scatterplots, the **latent space** is visualized, both with TSNE and PCA. Then a random **hyperparameters optimization** section is implemented in order to maximize the reconstruction abilities. At this point, by sampling some points from the latent space, **new samples** are generated.

After this, the same architecture with **dimension of latent space** equal to **2** is explored in order to have a clearer view of the latent space, and being able to generate new samples with an higher precision.

1.1 Dataset

The dataset used in this homework is the **FashionMNIST**. It consists in a training and a test set, the first one composed by 60000 elements, the second one by 10000. Each element consists in a couple of a piece of cloth image 28×28 with only one color channel, and the corresponding label. Both the datasets are loaded with the *tochvision.dataset.FashionMNIST* command.

Some **transformations** are applied to the samples. In particular both the images and the labels are converted into torch tensors, and the values of the images bits are multiplied by 255 in order to switch from the range [0,1] given by default by the command by torch to import the dataset, to the range [0,255]. This choice generally leads to better results. No transformation to modify the image shape are applied, as random rotation, flip or crop, in order to simplify the task and obtaining a latent structure easier to explore.

The training test is then divided in training and validation set, with a ratio of 0.9, in order to evaluate the performance of the algorithms to be implemented on a bunch of samples which not contributes to the training.

The the training and validation dataloader are finally defined.

1.2 Autoencoder

The architecture of an Autoencoder consists in a neural network composed by two essential blocks. An **Encoder**, which reduce the dimensionality of the input into a few latent variables, and a **Decoder**, which takes this latent variable and tries to reconstruct the input.

The **loss** for this architecture is given by a difference between the input and the output of the net, usually performed by the MSE function, because the goal is to produce an output as similar as possible to the original input.

1.3 Network

Given the fact that the dataset used in this task is composed by images, the most natural type of network to choose in a convolutional one. In particular the encoder is composed by **Conv2d** layers and **BatchNorm2d** for the normalization of the weights during the training, but, differently from a network for a classification task, no dropout layers are used, because, for the image reconstruction task we want to perform, the focus is to be as precise as possible, and there is no interest in increasing the generality by not relying too much on a single neuron.

The Decoder is build in order to be as symmetric as possible with respect to the encoder. This choice is not mandatory but quite reasonable for the structure of the problem. In order to do so the **ConvTranspose2d** layers are used. These acts in the reverse way of a convolutional layer, with the final result to enlarge the image give the input.

The Autoencoder class is composed by two main attributes, the encoder and the decoder. In this way it is possible to separately generate the latent variables and to decode them by reconstruction an image.

The network gets then trained with a function for manual training, and the loss trends and the results are shown in fig. 1a). As it's possible to see, the results are quite good, as the reconstructed images, despite being a little noisy, are very similar to the original ones.

1.4 Hyperparamter Search

In order to choose a good combination of the parameters for the reconstruction task, a hyperparameter research is implemented using *Optuna*. In particular a **Random Search** is performed, choosing the *RandomSampler* function is used as sampler in the *create_study* function, instead of the default *TPESampler*. In this way there is no danger to fall in local minima and focus the research over them, while combination with totally different parameter's values are ignored. The results are shown in tab. 1. As expected, the higher the latent dimension, the better will be the performances of the autoencoder, because it's possible to store a grater quantity of information. But at the mean time it's important to keep the latent dimension small enough, both to perform a good compression task, and also to make the latent space less complex.

1.5 Latent space

Starting from the test samples, vectors representing each input in the latent space are generated. Given their high dimensionality, in order to visualize the distribution over the space, it's necessary to perform some kin of **dimensionality reduction**. This is done in 2 ways: the first one is to use **TSNE**, a function that has the goal to converts similarities between data points to joint probabilities and to minimize the KullbackLeibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data (fig. 2b)). This method has the good propriety to encapsulate the cluster distribution of the high dimensional points and to preserve them in the lower dimensional representation. A throwback is that it's computationally quite demanding. The other method is the **PCA**, which performs a spectral decomposition of the matrix encapsulating the data points and keeps only the dimension characterized by the greatest eigenvalues. It's computationally very efficient, but many details about the grouping of the points are lost (fig. 2a)).

In both cases it's possible to recognize some clusters. In particular the areas occupied by points corresponding to similar pieces of cloth usually overlap.

1.6 New samples generation

In order to generate original samples, we start from a random image of the test dataset, then its corresponding latent vector is computed and finally 2 of the 20 dimension of the latent space are chosen, and the corresponding elements of the initial latent vector are made vary within the range given by the maximum and the minimum value that that specific component of the test dataset latent vectors takes. At this point a problem arises. Given the high dimensionality of the latent space, we encounter the problem of the volume of high dimensional object, which is almost all contained in it's

peel, so it's very hard to explore it uniformly. In addition to this, by having a visual representation of the latent space only via PCA or TSNE, it's impossible to pick a point in the latent space in a region that generates a specific kind of samples. Despite this, the result of the chosen procedure are shown in fig. 3.

1.7 2D latent space

We try now to explore the results of the same architecture of before but fixing the latent dimension to 2. In this way we won't need PCA or TSNE in order to visualize the latent space and the new sample generation task will be much easier.

The Autoencoder is trained with the manual function and after this, the latent vectors are generated and then shown in a scatterplot (fig. 4a)). Some clusters can be seen and in particular the regions occupied by points of the same type seem to vary continuously over the space. So, in order to explore such a space some points are uniformly sampled from a square of dimension close to the boundaries of the region occupied by the test dataset latent vectors, and the corresponding decoded images are generated and reported in fig 4b) . Here we can see that in this way it's possible to go from a certain piece of cloth to another completely different almost continuously by choosing the right path over the latent space.

2 FINE-TUNED AUTOENCODER FOR CLASSIFICATION

We try now to fine-tune the architecture of the previous point in order to implement a **classifier**. The idea is to check if the latent vector representing each image, encodes enough information to retrieve the correct label only by looking at it. In order to do so we consider only the encoder block and attach to it some extra layers which output will be evaluated as in a classification task. The performance of such an algorithm will be then compared with the one reached in the previous homework.

2.1 Method

The state of the Autoencoder trained before are imported in a new module. Then all the parameter's gradients are locked by setting the *.requires_grad* attribute equal to False for each layer. At this point a new network is created by stacking together the encoder of the Autoencoder with locked weights and some extra linear layers (depth=2) with activation (ReLU) and regularizers (BatchNormalization). The final output has dimension 10, equal to the possible kind of image's labels. By printing the *.requires_grad* attribute of each layer it's possible to see that the only part that will be updated during a training is the final extra chunk of linear layers.

2.2 Results

Then it's possible to proceed with the training. The torchbearer library is chosen, in fact it allows to compute very easily the accuracy, both for the training and validation set, alongside the loss, a quite useful tool in the case of a classification task. The loss and accuracy are plotted in a graph, and the results are summarized in the heatmap representing the confusion matrix of the prediction versus the true labels (fig. 5). The accuracy reached is quite high (0.8557), definitely comparable to the ones obtained in the previous work (0.89 in the best case).

3 VARIATIONAL AUTOENCODER (VAE)

In this section a **Variational Autoencoder** is implemented. The goal is the same of the autoencoder task, i.e. being able to compress and reconstruct images, but with some more functionalities which makes the VAE a more suitable tool for the data samples generation.

The first and main step is to implement the **VAE** class, in which is defined the network itself, the function to apply the reparametrization trick, the loss function, composed by the sum of the reconstruction loss and the KL divergence, and the functions for encoding an input sample into the

corresponding latent vector and decode it in order to reconstruct an image. Then the model is trained in a manual way and the results are shown as in the case before. After this the **latent space** is visualized thanks to PCA and TSNE and some **new samples** are generated from it. Finally the same model with latent dimension equal to 2 is implemented and and new samples are again generated from it. Finally the the impact of the variation of the β **parameter** are explored by investigating how this impacts over the way in which new samples are generated.

3.1 Network

As said before, a VAE class was defined in order to contain all the useful functions for the implementation of a variational autoencoder. The architecture of the network is very similar to the one of the Autoencoder described before, with the only difference of the last layer of the encoder. In the VAE case the compressed information about the images are encapsulated in 2 Linear layers, one representing the **mean** μ and the other one the **logarithm of the variance** $\log(\sigma^2)$, both with output dimension equal to the latent dimension. The information from this two layers is then combined in order to create the latent vector z corresponding to the image. The peculiarity of the VAE is that this vector is sampled randomically by a **multivariate Gaussian** with mean μ and covariance matrix $diag(\sigma^2)$. Then the latent vector is given in input to the decoder and the reconstructed image is generated. In order to prevent problems in the back-propagation procedure due to the stochastic process of the generation of the latent vector, the **reparametrization trick** is used, i.e. a stochastic variable ϵ is sampled from a standard multivariate Gaussian and each element of z is given by $z = \mu + \epsilon\sigma$. In this way the source of stochasticity is external from the architecture, so there is no problem in the training procedure. The function that performs the reparametrization trick is defined within the VAE class.

Other useful functions are defined inside the class, such as the one produce a corresponding latent vector z from an input image (*latent_space_sample*), the function that receiving z reconstruct an image (*generate_new_sample*), and finally the one that return the loss of the model. This is given by the sum of two contributions, the **reconstruction error**, given by the MSE loss between the original and reconstructed image, and the **KL divergence**, which tries to minimize the "distance" between the multivariate Gaussians from which the latent vectors are samples, and a standard multivariate Gaussian of the same dimension. This last term has the function of a regularizer for the structure of the latent space.

3.2 Training and Results

The network is then trained manually and the final state is saved in order to be loaded if we desire to further improve the performances. Some images form the test set are then encoded and reconstructed, and the results are shown in fig. 1b) . Despite a little noise, the reconstruction is pretty good, and the shape of the pieces of cloth is well distinguishable.

The latent space is then explored using the TSNE and PCA techniques (fig. 7). In addition to what was done in the case of the Autoencoder, in this case also the distribution of the means μ and logarithm of the variance $\log(\sigma^2)$ are provided alongside to the one of z.

In order to understand how the network encodes the inputs and cluster together the latent vectors in the latent space, it's sufficient to focus on the scatterplot of the μ, because they represent the centers around which the z vectors are generated.

3.3 Generation of original samples

The procedure used to generate new samples starting from latent vectors is analogous to the one used in the case of the Autoencoder. The particularity of the VAE is that it should be able to encode a specific feature of the dataset space in some particular direction of the latent space. Nevertheless, this is quite hard to find, especially if the latent dimension is pretty high (fig. 6b)).

3.4 2D latent space

Again, the network is retrained by choosing a latent dimension equal to 2. In this way it's easier to explore the latent space, as can be seen in fig. 8. A possible difference with the case of the Autoencoder (fig. 2), is that the μ and z vectors that encode the input images, seem to be distributed in a more compact way. In fact in the Autoencoder training there is no term that encourage a certain structure of the latent space, as the KL divergence in the Variational Autoencoder case.

3.5 β VAE

Finally the influence of the β parameter in the VAE loss function is explored. In order to do this it's necessary to use a high dimensional latent space, in order to have a large number of possible directions over which a specific feature of the images can change. As in the first architecture we use 20.

Some models are trained by keeping all the parameters constant and only varying β, then the same sample is encoded and then reconstructed by varying the value of a single variable of the latent space in a fixed range. The results are shown in fig. 9. What should emerge is that the VAEs with higher values of the β parameter are able to make the input sample's features in a more disentangled way with respect to each direction of the latent space. The dataset we used for this task is not really suitable for this kind of task, in fact many different objects are represented, and it's hard to go from one piece of of cloth to a completely different one by only varying some specific feature of the first one (i.e. it's pretty difficult to start form a pullover and ending up with a shoe only by modifying some characteristic features of a pullover). From the results we can see that for small values of β, the variation in position of the the latent space usually leads to images composed by a superpositions of different kind of clothes, while for higher values of β, the general structure of the image remains the same and only some features change. This is especially relevant in the case of $\beta = 100$ in the figure, in which it's possible to see that by changing the value of the variable, the shades over the pullover changes direction and dimension. Despite this fact, for higher values of β it's quite hard to retrieve such explicit results, and the changing behaviour of the generated images is less clear.

4 Generative Adversarial Network (GAN)

As last task, a **GAN** is implemented. The main structure was taken from the following Pytorch tutorial, and was adapted to the FashionMNIST dataset used in the previous sections. The goal of a GAN is to train a generator to create brand new images from random noise input that are able to fool a discriminator which tries to continuously improve his ability to recognize original and synthetic images.

4.1 Network

The main structure of the network is composed by two essential blocks. A **Generator**, which tries to generate from a random noise vector a figure resembling the ones from the compose the dataset, and a **Discriminator**, which tries to discriminate between synthetic and original images. The first one is composed by 3 Convolutional Transpose layers, each one followed by Batch Normalization layer and activated by the ReLU function, while the second has depth equal to 4, Convolutional and Batch Normalization layers and ReLU as activation. Each of the two networks has to be trained separately, so 2 optimizers are required. For both the generator and the discriminator, the Adam algorithm is chosen, and as loss function, in both cases the **BCEloss**, i.e. the Binary Cross Entropy, is used. Then we proceed with the manual training. First of all, the discriminator is trained on the original samples, each one with assigned label equal to 1 (which denotes the "original samples"), and the first term of the discriminator loss is computed. Then the synthetic samples are produced by the generator starting from random noise, and each of them is assigned the label 0(which denotes the "synthetic samples"). The loss obtained in this passage, summed with the one previously obtained, gives the total loss w.r.t. the discriminator's optimizer is updated.

Then the synthetic images are used in order to train the generator. All of them gets assigned with label equal to 1, indicating that now they "pretend to be original samples", and then the discriminator is applied to the synthetic image producing an output, which will be used to compute the BCEloss alongside to the assigned label. Then the generator optimizer is updated w.r.t this loss value. The aim of this procedure is to maximize the ability of the generator to create images that will be labelled as original samples from the discriminator. The more of them are classified as original samples (label equal to 1), the better is the generator performance.

At the end of each epoch, some images are generated from some fixed vectors of random noise defined at the beginning of the training in order to keep track of the performances of the network.

The network was trained for 300 epochs, and the loss trend is reported in fig. 10. As it's possible to see, it doesn't represent the typical behaviour of a GAN loss, which should have at the beginning a decrease in the discriminator error, while the generator one's increase, and after a while n inversion of the trends should occur until a stability is reached. This doesn't happen in our case and the both the losses seem quite flat. Despite this, the results, as we will see, are pretty good.

4.2 Results

At the end of the training, the images generated at the end of each epoch constitute a rich source of information about the progress of the GAN. In particular it's possible to see how the generator abilities to create images similar to the original one, increase with each epoch. The results are shown in fig. 11 and also in the GIF provided in the folder. A noticeable fact is that the same vector of fixed noise can produce images that resembles different pieces of cloth in the first phases of the training, and at a certain point it reaches a stable state in which only one cloth is represented, and the changes in the image are small and tend to fine-tune it.

In addition to this in fig. 12 are reported some images generated by the generator at the last epoch. They are really similar to the original one, almost indistinguishable from the original ones by a human discriminator.

Latent dim.	Optimizer	Learning rate	Weight decay	Weights init.
28	Adam	$5 \cdot 10^{-3}$	$2 \cdot 10^{-4}$	None

Table 1: Result of the hyperparameters optimization of the Autoencoder

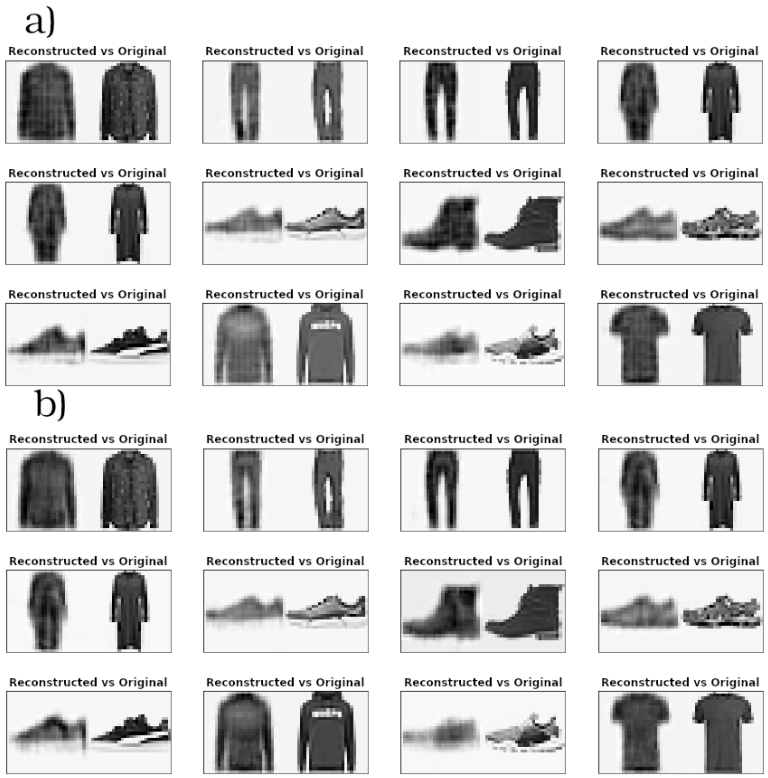

Figure 1: a) Original vs reconstructed samples for Autoencoder. b) Original vs reconstructed samples for VAE

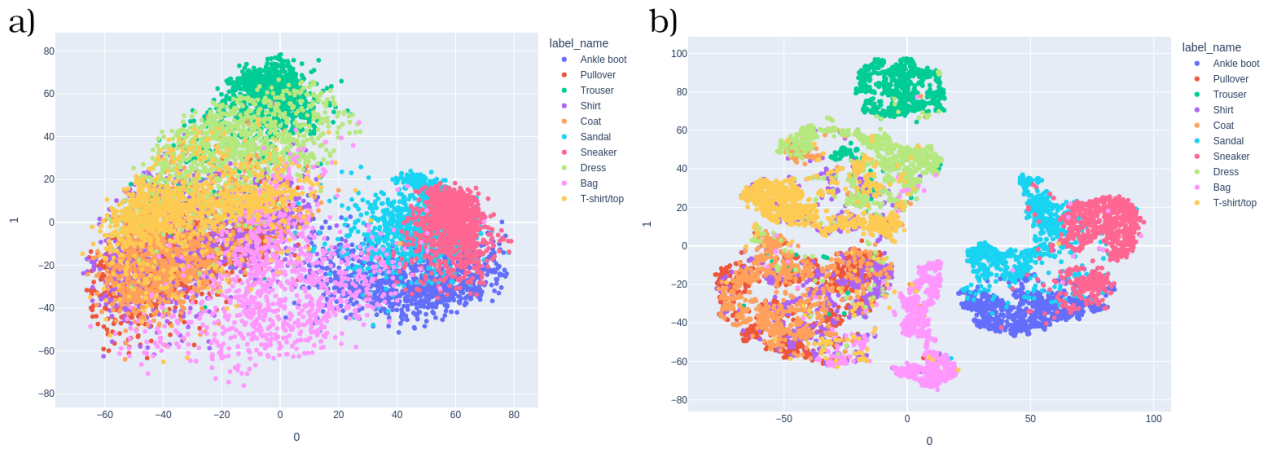

Figure 2: a) Autoencoder's 2 dimensional visualization of the latent space with PCA b) Autoencoder's 2 dimensional visualization of the latent space with TSNE

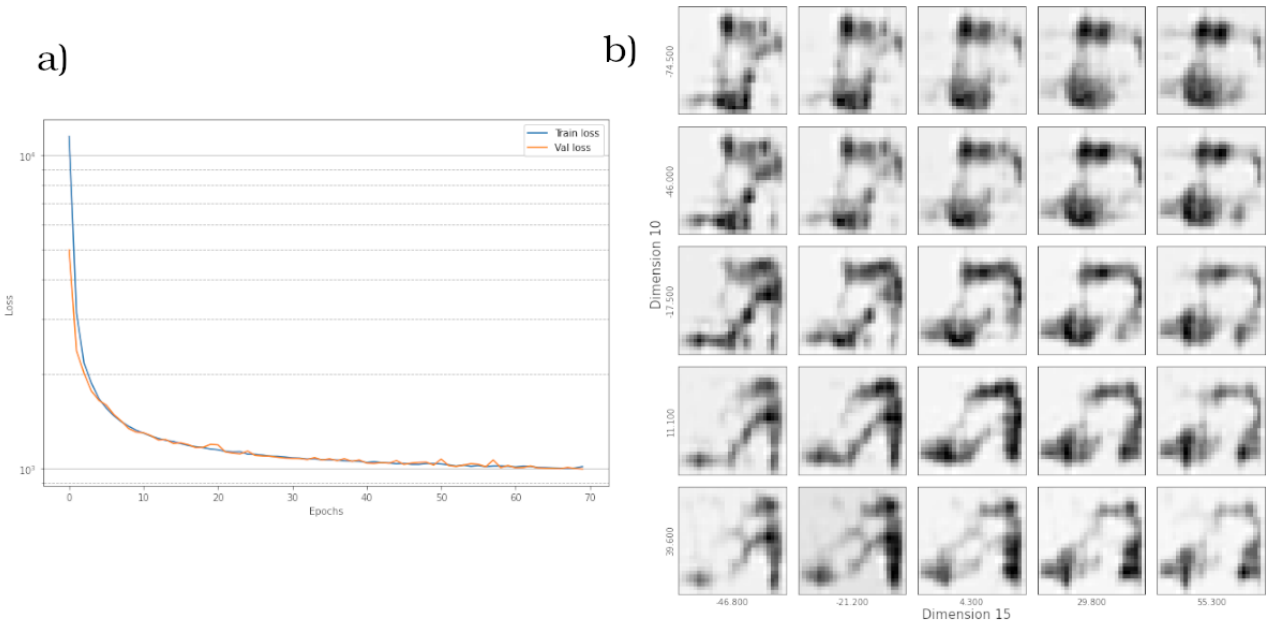

Figure 3: a) Loss trends for Autoencoder. In blue the training loss, in orange the validation one. b) Some examples of images reconstructed from latent vectors, making 2 variables vary in a given range.

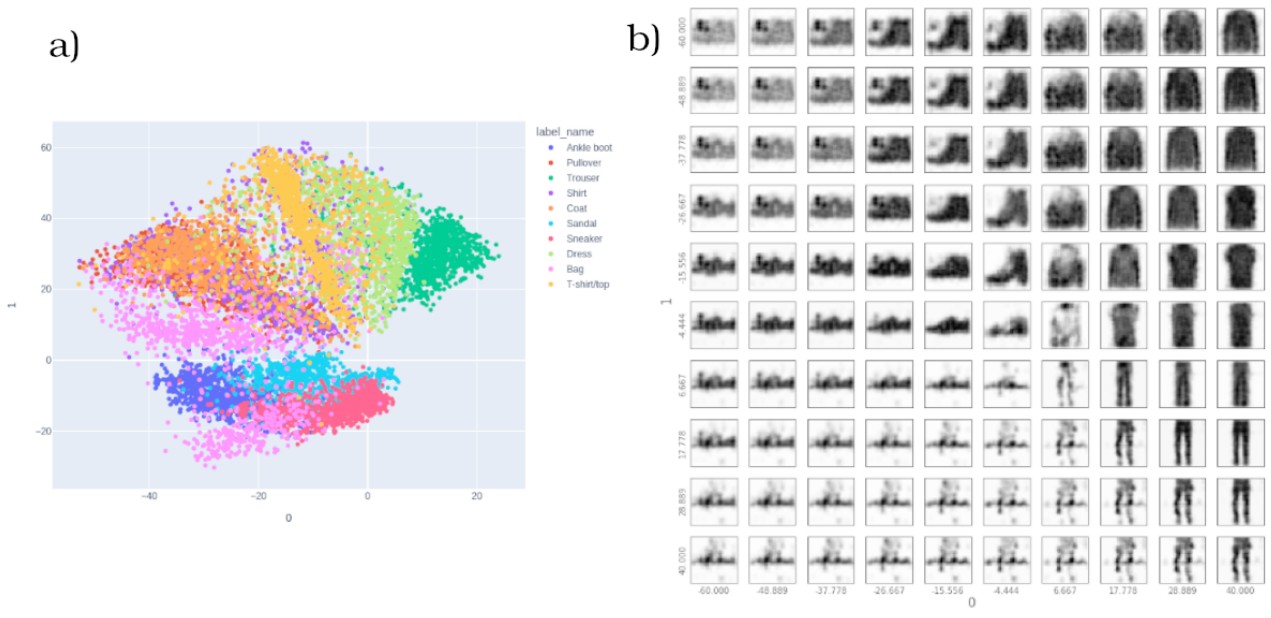

Figure 4: a) Scatterplot of the latent vectors for each sample of the test set in the case of the 2-dim Autoencoder b) Images generated by the 2-dim Autoencoder form latent vectors uniformly generated within a square of center (-10,-10) and side equal to 100.

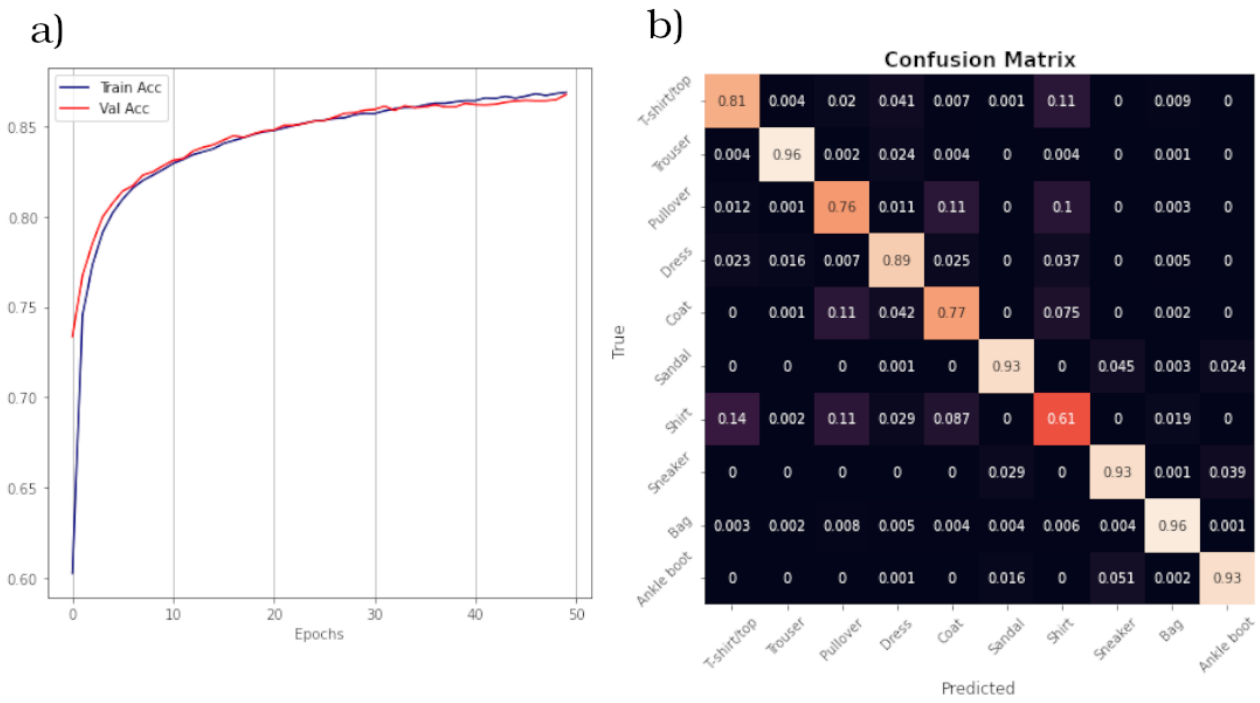

Figure 5: a) Classification accuracy trend with respect to each epoch for fine-tuned autoencoder b) Confusion matrix for predicted vs true labels. The final accuracy is equal to 0.8557

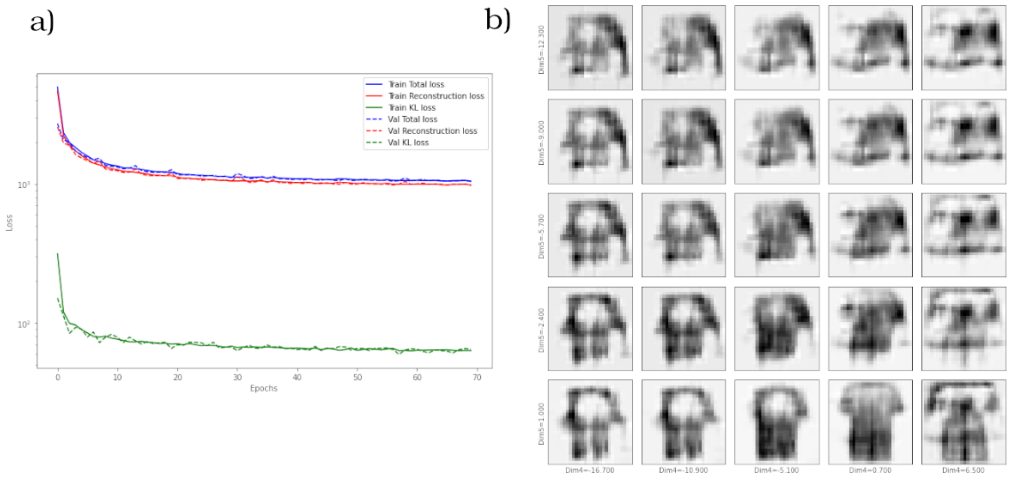

Figure 6: a) Loss trends for VAE. the continuous lines represent the train losses, the dashed ones the validation losses. The KL divergence is represented in green, the reconstruction loss in red and the total one in blue. b) Some examples of images reconstructed from latent vectors, making 2 variables vary in a given range.

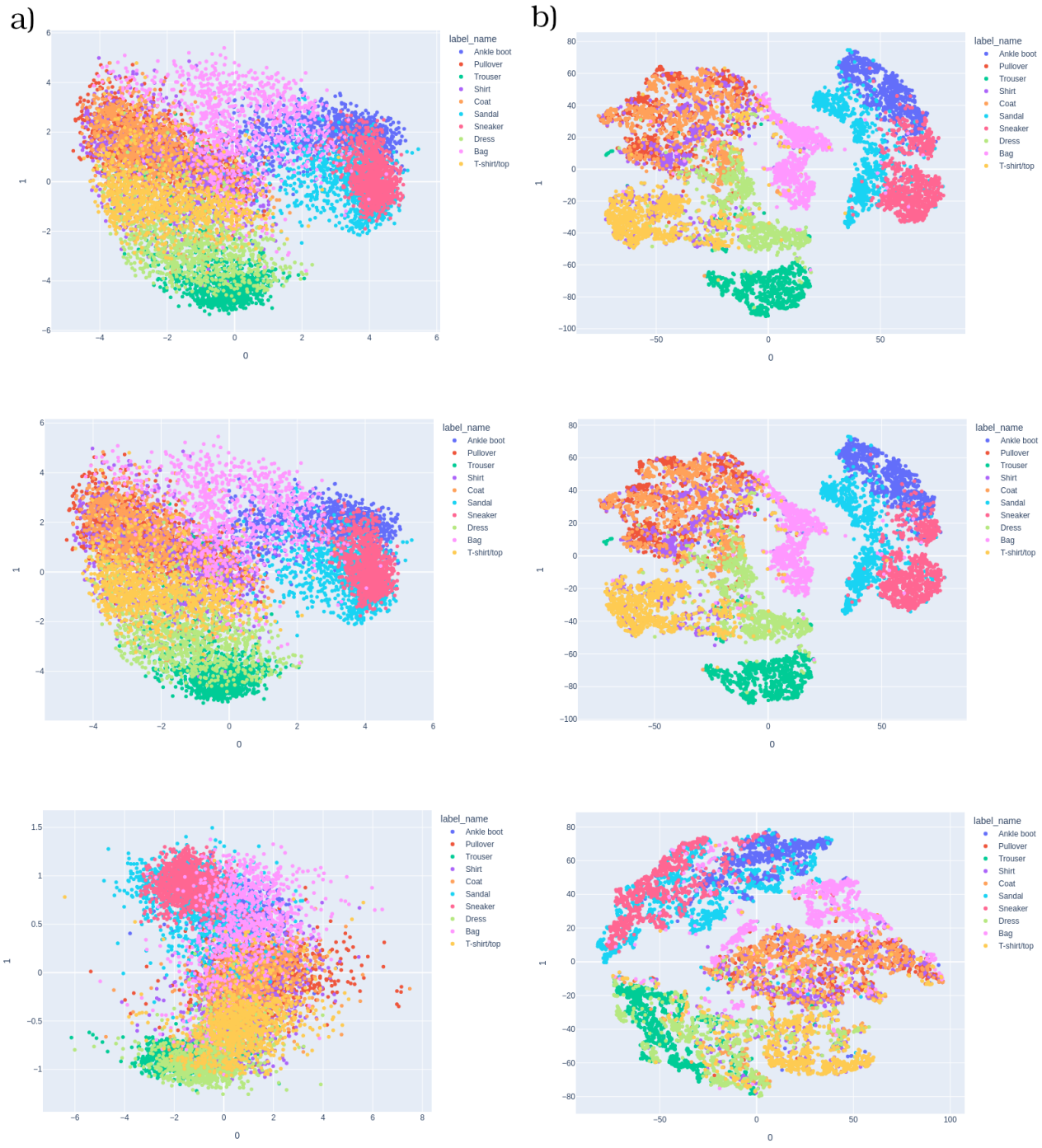

Figure 7: a) VAE's 2 dimensional visualization of the latent space with PCA. From top to bottom we have the means μ, $\log(\sigma^2)$ and the latent vectors z b) VAE's 2 dimensional visualization of the latent space with TSNE. From top to bottom we have the means μ, $\log(\sigma^2)$ and the latent vectors z

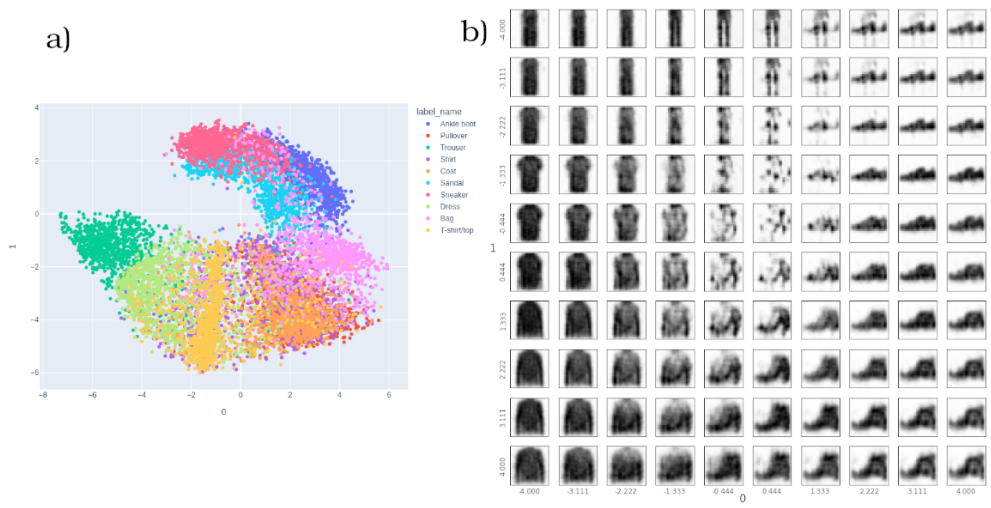

Figure 8: a) Scatterplot of the latent vectors for each sample of the test set in the case of the 2-dim VAE b) Images generated by the 2-dim VAE from latent vectors uniformly generated within a square of center (0,0) and side equal to 4.

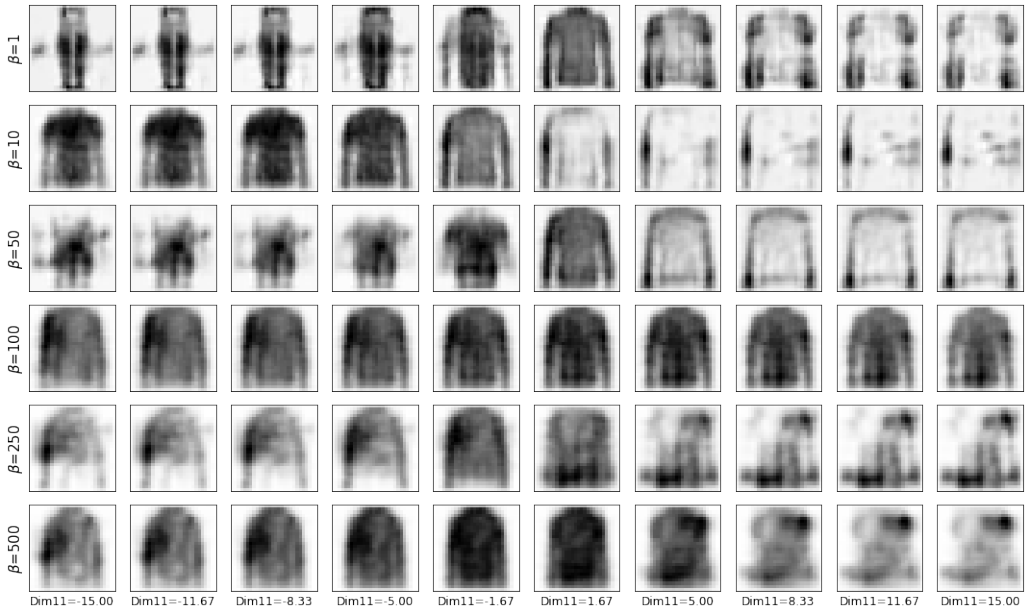

Figure 9: Images reconstructed from the latent vector generated by a test set image making 1 variable vary within a given range for different values of β

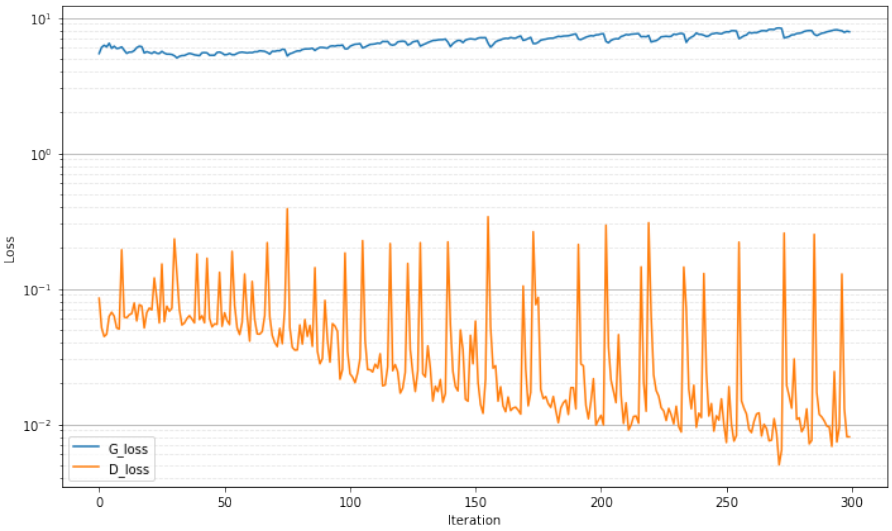

Figure 10: Loss trend for a GAN. The blue line is for the generator, the orange one for the discriminator

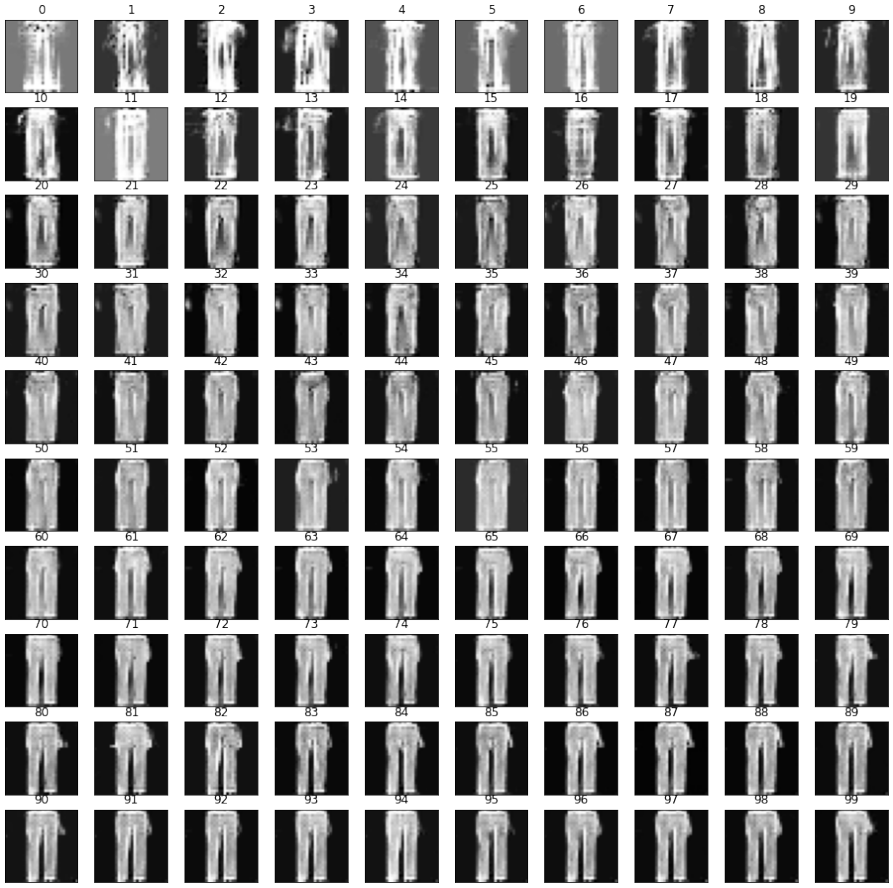

Figure 11: Images produced by the generator from the same fixed noise for 100 iterations.

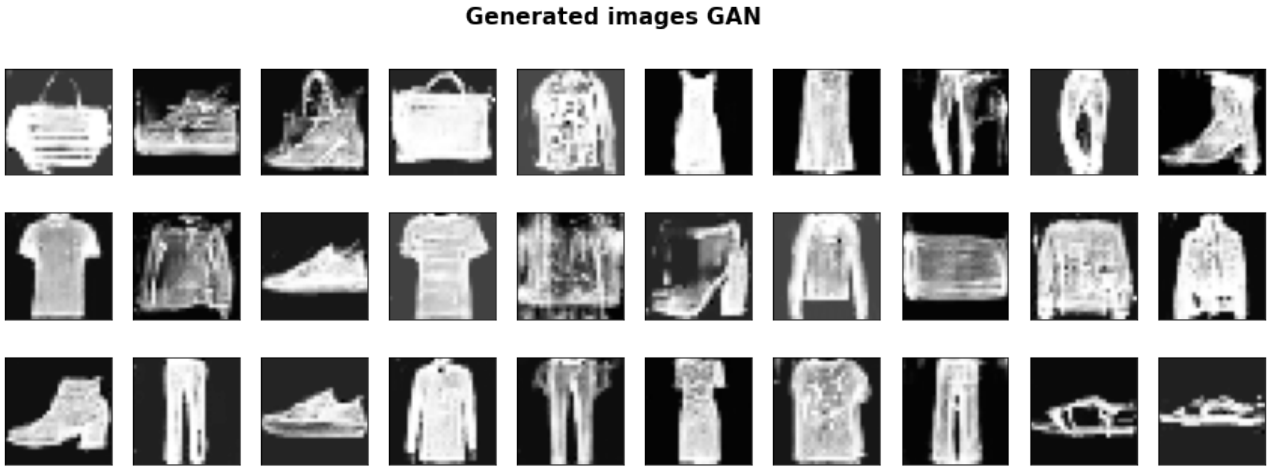

Figure 12: Images produced by the generator at the end of the training from different random noise vectors.