NNDL Homework 2
Unsupervised Deep Learning

Enrico Convento – ID: 2023572

February 10, 2022

1 Introduction

This homework deals with unsupervised deep learning, as far as the dataset is concerned, the FashionMINST is still used, three different structures have been implemented namely a basic autoencoder (AE), a variational autoencoder (VAE) and finally a GAN (Generative Adversarial Network). The report is organized in two main sections, the section Methods which introduces the approaches used and the section Results which describe what have been obtained.

2 Methods

2.1 Autoencoder

The basic autoencoder is implemented as follows:

- The *Encoder* class is used to build the encoder which is a convolutional neural network which maps the input into the so called 'encoded space'
- The *Decoder* class permits to create the decoder, its function is basically the opposite of the encoder, i.e. taking data from the 'encoded space' and decode it
- The *Training* class build an object that handle all the training process of the overall system

The encoder that composes the basic AE is build by three convolutional layers which use kernel of dimension 3x3, strides of 2 and padding of 1, followed by two linear layers and the output layer associated to the latent space, in this case the encoded space has dimension 8. The associated decoder is the mirror image of the encoder. After performing the training, a section of the notebook is dedicated to the visualization of the encoded space by means of PCA and t-SNE. The regularization methods that have been applied are scheduled in the following:

- Early stopping
- Pruning
- Dropout

Following the lineup of the notebook, after analyzing these methods, I trained several autoencoders to see how the representation of latent space changes according to the dimensions used. After that using the AE trained before with a latent space of dimension 2 some samples have been generated from the encoded space. As a last step, an automatic tuning was performed with Optuna to try to obtain an optimal network, which will be tested later, the results obtained will be described in the Sec.3 and also manual tuning was performed in order to try to get better results.

2.2 Fine Tuning

In the first notebook also a section in which the encoder, the same that compose the basic autoencoder of the section above, is taken and fine-tuned in order to get a classifier and solve a classification task, in particular the same of the previous homework. Two classifiers have been implemented, regarding the first one three linear layers are added in the tail of the encoder, so that the latent space is mapped into 64 features, than 288 and in the end 10, the weights associated are saved into **class1**. Regarding the second structure, only one output layer is being added, i.e. a layer composed of 10 neurons, in this case the associated weights file is **class2**. Of course, during the fine-tuning procedure the encoder weights have been frozen. Everything is then trained, and the results are analyzed by comparing those with what had been obtained in the previous homework.

2.3 Variational autoencoder

As introduced, the second part of the homework deals with Variational Autoencoders. In a nutshell, a VAE is a variation of a standard AE in which the latent space is encoded by a distribution and it is regularized during training in order to ensure that it has good generative properties. The VAE structure is composed by an encoder, a decoder and a particular latent space structure. The class *VariationalAutoencoder* handle the assembly of the whole system, the encoder and the decoder exploited are really simple, both are composed by two convolutional layers respectively of 8 and 16 channels which uses kernels of size 4, stride of 2 and padding of 1. There are two main differences with an AE:

1. The **latent space**, the encoder does not directly map points into latent variables, but those inputs are mapped into a multivariate normal distribution. The idea that leads to this came from the fact for a standard AE the latent space is sparse and difficult to handle, exploiting what has been introduced above the latent space is forced to be smooth. This kind of space is not only useful in terms of generative properties, but it is a way to introduce regularization in the training. In practice, this regularization is done by enforcing distributions to be close to a standard normal distribution in such a way, the mean should be close to 0, preventing encoded distributions to be too far apart from each others and the covariance matrices will be close to the identity, preventing that the encoder returns distributions with very small variance.

2. The **loss function**, in this case the loss is composed by two terms, the reconstruction term and a regularization terms expressed as KL divergence, which aims to regularize the organization of the latent space by making the distributions returned by the encoder close to a standard normal distribution.

The variational AE with a latent space of dimension 5 in been trained for 50 epochs, the results can be seen in Sec.3.3. After this, the VAE is being trained with a latent space of dimension 2 in order to provide an interesting proof of the generative properties of such structure.

2.4 GAN

The third method deal with generative adversarial networks i.e. GAN, in particular for the homework a DCGAN is being implemented. In a nutshell, a GAN is composed of two main blocks, the first is the generator whose purpose is to generate 'examples', and the discriminator whose purpose is to classify both real and generator-supplied examples. The two models are trained together, 'playing' against each other until a convergence is reached. In the case in question, a DCGAN was chosen to better process the images, it is composed of a generator which is made up of a linear layer that maps the latent space to an high dimensional space and three transposed convolutional layers, used for upscaling the image, as output activation is used *tanh* which map the output in $[-1, 1]$. Batch normalization layers have been used to stabilize the training. The discriminator block is composed of two convolutional layers followed by a linear layer, for the output is used the *sigmoid* in this case. The overall system is manage by the class *Training*, inside such class the two sub-models that compose the GAN are build and than other functions were implemented for handling the training. It is composed by the following methods:

- **generate_samples**, it generates samples from the generator, by feeding it with a multidimensional noise and return a value in $[-1, 1]$
- **train_step_generator**, it updates the generator and return the loss associated
- **train_step_discriminator**, similar to the previous function
- **train_epoch**, it exploits the two previous function to perform one-epoch training of the GAN
- **train**, it runs the training

3 Results

3.1 Autoencoder

After training the AE for 50 epochs the results are shown in Fig.1, it can be observed that only after 50 epochs the AE is able to reconstruct the image in a coherent way, the results are interesting, but it is possible to do it better. In Sec.A of the Appendix, other images reconstructions are shown.

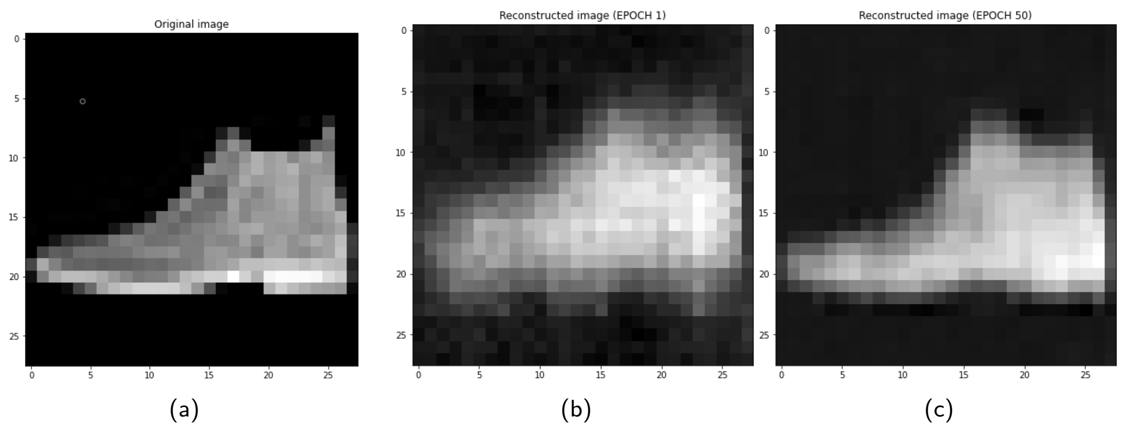


Figure 1: Original image and reconstructions

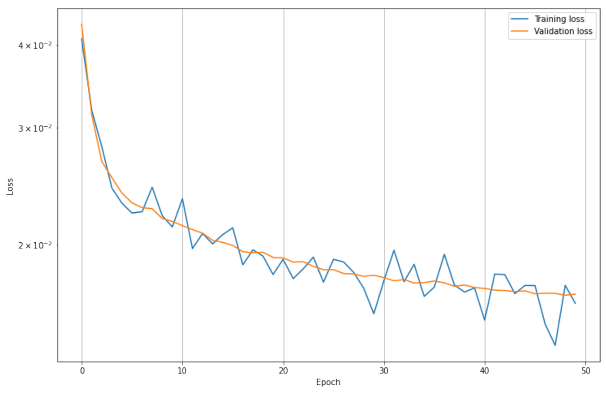


Figure 2: Reconstruction loss

Regularization Given the results obtained with the basic solution, regularization mechanisms introduced above are used to improve the results. The early stopping is introduced to avoid overfitting. After observing the results it appears that the loss continue to change also by very small amount, so a constraint should be introduced to check if an 'improvement' should be considered valid; the use of this technique does not lead to big turns. The second method used is pruning, the idea behind it is that often if the network has many parameters they can be redundant, so they do not contribute strongly to the output except in terms of robustness. What you do is prune after some epochs a fraction of the weights of the nets that have the lowest L1 norm value. Also in this case the effect is very scarce, observing the trend of the loss we have something very similar to the basic case but with more computational effort. As last regularization technique, the dropout is used and also in this case the effect is scarce since the loss value in the basic case reported above is in any case lower when comparing on the 50 epochs used for the training.

Encoded space Before proceeding with the analysis of various latent spaces with different sizes the encoded space of the basic AE used before is explored by means of PCA and t-SNE, the results can be observed in App in Fig.A, from that can be observed that the representation obtained using t-SNE is more clear, while using PCA we are not able to tell if the AE is performing well or where we have to improve. This section analyzes also how the representation provided by latent space changes as the chosen dimension changes. The representation of the various spaces is carried out using t-SNE and the images regarding this are reported in Sec.A of the Appendix. The size is made to vary from 2 to 14 with steps of two, from what is observed the more the size increases the more the points associated with the various elements are spread in space. Obviously the points have areas in which the elements, typically elements that have similarities, meet.

Sample generation In order to provide some nice proof of the generative properties of the AE we have sampled with a uniform distribution the latent space and observed how the decoder reconstructs inputs from arbitrary latent vectors, this is shown in Fig.3.

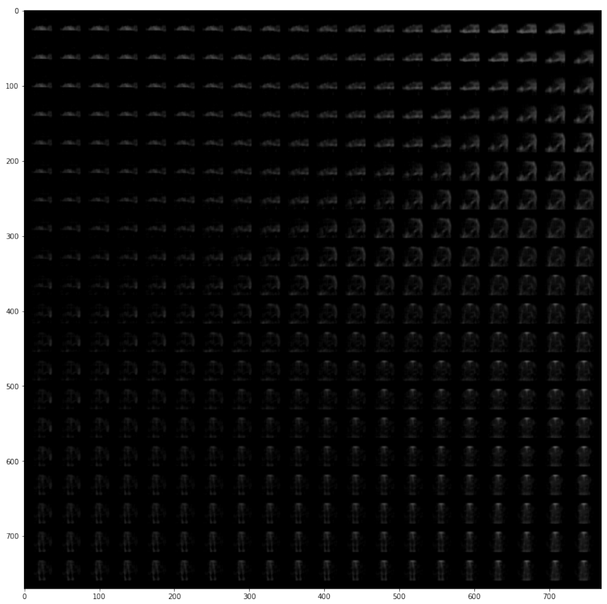


Figure 3: Generated samples form latent space (AE)

Optuna and manual optimization The automatic tuning is performed through Optuna, for a total of 10 trials, the tested parameters are shown in the following table:

Number of convolutional layers	2,3 or 4
Number of linear layers	1,2 or 3
Number of channels	4 to $4 * 2^j$ (j : convolutional layer number)
Kernel size/ stride / padding	2 to 6 / 1 to 4 / 0 to half kernel size
Number of nodes for linear layer	8 to $512 * 2^{-j}$ (j : convolutional layer number)
Encoded space dimension	1 to 16
Learning rate/ weight decay	10^{-5} to 10^{-1}/ 10^{-7} to 10^{-1}
Training batch size	64 to 512
Patience	1 to 10

However, even in this case the optimal network that was then tested does not produce good results; the optimal loss values are very far from the ones obtained using the AE realized above. After that also a manual tuning was performed, but also in this the results are not improved.

3.2 Fine Tuning

This section analyzes the results obtained by performing fine-tuning by assembling a classifier. The results are good, almost comparable with the results obtained in the previous homework in fact the first network reaches an accuracy of approximately 80% while the second which is simpler reaches approximately an accuracy of the 70%, it is taken into account that in the supervised case the best accuracies are around 90%. Also in terms of time the results are very similar, analyzing the same time window we obtain similar loss values, however it must be said that the time for which the AE has been trained is not taken into account. In Sec.B of the App. in Fig.8 is shown the trend of the loss during training and validation for the first net in order to confirm what said above, while in Fig.7 are shown the confusion matrices associated to the two classifiers.

3.3 Variational Autoencoder

In Sec.C of the Appendix is possible to observe reconstructed images from the VAE that exploit a latent space of dimension 5. The quality of those images is close the ones generate with the AE. In order to get some ideas about the generation properties of a VAE we can also sample uniformly the latent space and see how the decoder reconstructs inputs from arbitrary latent vectors, the results. As said above this done for a VAE that uses a latent space of dimension 2 which is easier to handle, the results can be observed in Fig.4.

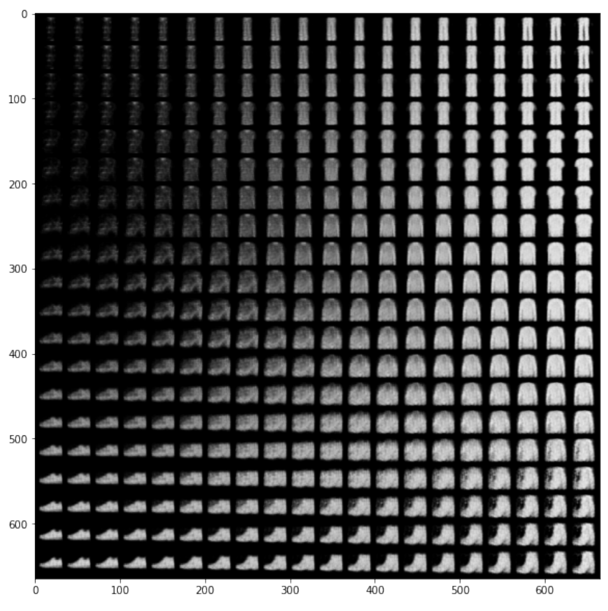


Figure 4: Generated samples form latent space(VAE)

VAE produces a latent space that is more compact and smooth due to the way it is build. In the following two implementations details will be described, those regards the implementations of the 'special' loss function used and the other one regards the so called 'reparametrisation trick'.

Loss function It is interesting to get some attention about how the loss, which is shown in the equation below, is implemented in practice:

$$\mathcal{L} = \mathbb{E}\left[\log p(\mathbf{x} \mid \mathbf{z})\right] - \beta D_{KL}\left(q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})\right)$$

Where $p(\mathbf{z}) \sim \mathcal{N}(0,1)$. The terms that compose such loss have already been introduced in the methods part. The first term is implemented in terms of L2 norm between original image and reconstructed one, this is not exactly what described in the equation above, but it can be done since we are treating Gaussian distribution. The second term associated to the KL divergence is implemented as follows:

$$D_{KL}\left((\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})\right) = \frac{1}{2}\left[-\sum_i \left(\log \sigma_i^2 + 1\right) + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right]$$

and this is due to the fact that we are dealing with Gaussians. So in the end, minimizing the loss reported above is equivalent to minimize the loss function below reported:

$$\mathcal{L} = \|x - \hat{x}\|^2 + D_{KL}\left(q(\mathbf{z} \mid \mathbf{x}) \| \mathcal{N}(0,1)\right)$$

Reparameterization trick Training such kind of architecture could lead to unwanted errors, in particular associated to the backpropagation. The sampling process need to be done in a way that allows the error to be backpropagated through the network. The trick called reparametrisation trick is a way to make the gradient descent possible taking into account the random sampling. Basically we sample from $z = \mu + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0,1)$. This allows the mean and log-variance vectors to remain learnable parameters of the network while still maintaining the stochasticity of the entire system, thanks to the term 'eps' in the function *latent_sample*.

3.4 GAN

In this section the results associated with the implemented GAN will be commented. First it is necessary to say that this structure is more complex than the other two and training it requires a greater computational effort and some practical details. The major problems associated with this model are two of which the main one is the non-convergence, then there is also the mode-collapse problem, i.e. the generator produces limited examples and then slow training occurs. A lot of tricks are available to improve these deficient points, however there is no real methodological approach to implement such a system. In Fig.5 it can be observed the trend of the losses associated to the GAN, notice that after an initial transient the losses reach both an equilibrium, this saddle point is a maximum w.r.t. to a player and a minimum w.r.t. the other. In Sec.D of App. it can be observed some image generated after the training, it must be said that the images built with such architecture are the best ones if compared with the others, the main drawback of this is the fact that the GAN requires a lot of time to be trained.

Loss function In this section, practical details about how the GAN is implemented will be explained. The main part associated to this equilibrium game is associated to the loss, in particular the loss function that used is :

$$\mathcal{L} = \quad E_x[\log(D(x))] \quad + \quad E_z[\log(1 - D(G(z)))]$$

and as introduced before the generator tries to minimize this function while the discriminator tries to maximize it. The loss reported is implemented by means of BCE and the link between the two could be tricky. The idea come from the fact that discriminator is trained to correctly classify real

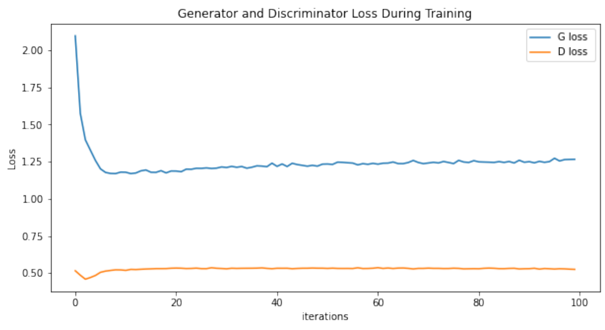
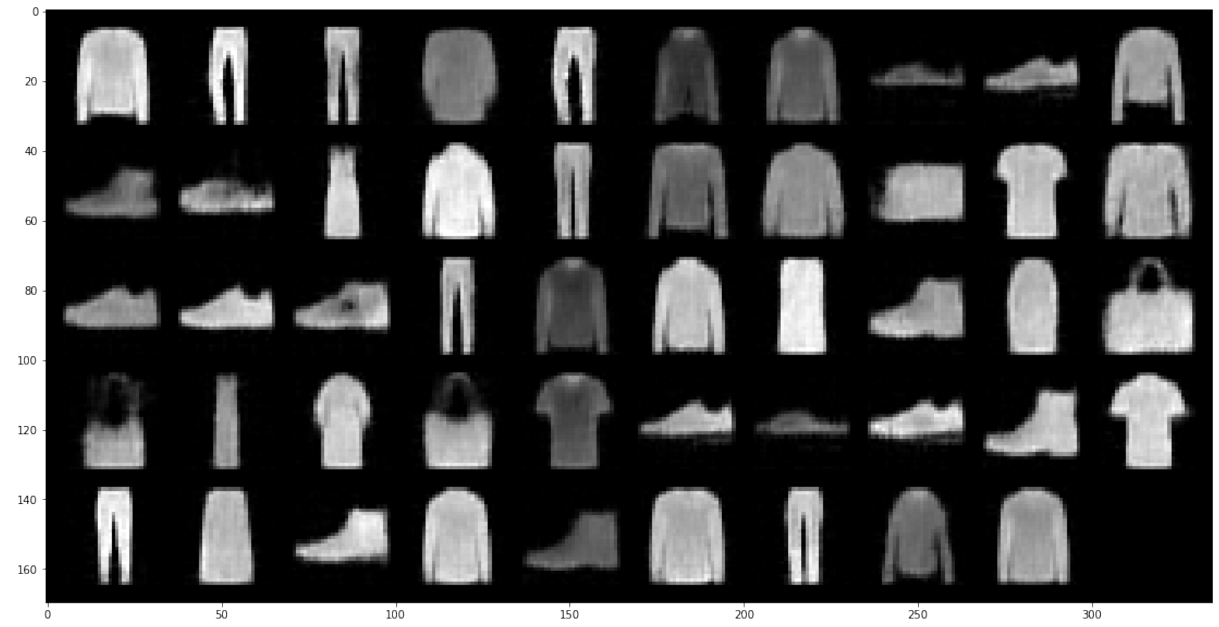


Figure 5: GAN losses

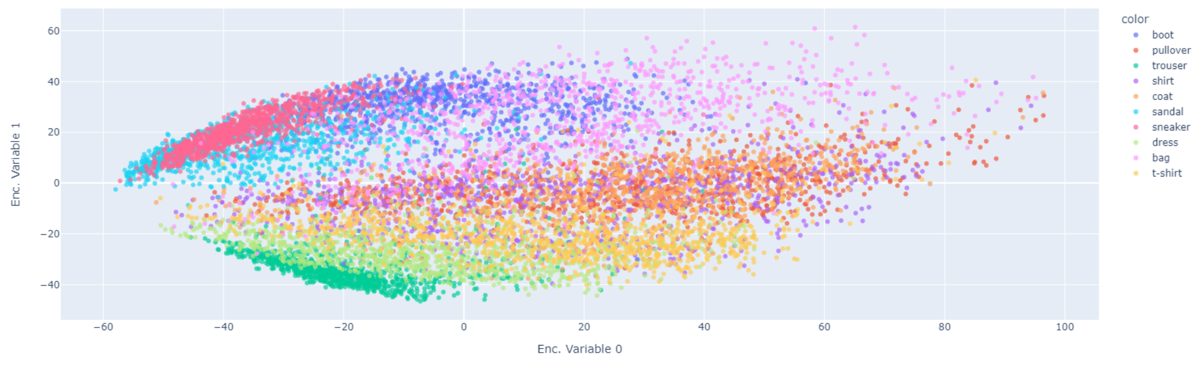
and fake images, this leads to maximize the log of predicted probability of real images and the log of the inverted probability of fake images. This can be seen in terms of BCE loss, the training is close to what it is done in a binary classification problem training on two mini-batches of data; one coming from the dataset, where the label is 1 for all examples, and one coming from the generator, where the label is 0 for all examples. So maximize the loss reported above coincide with minimizing the BCE loss, and this can be easily done by backpropagation. The generator is more tricky. The generator aims to minimize $log(1 - D(G(z)))$. In this practice, this is not used because it leads to the saturation of the generator loss. So rather than training it to minimize $log(1 - D(G(z)))$, the generator is trained to maximize $logD(G(z))$. In the previous case, the generator sought to minimize the probability of images being predicted as fake. Here, the generator seeks to maximize the probability of images being predicted as real. In practice, this is also implemented as a binary classification problem, like for the discriminator. Instead of maximizing the loss, the labels are flipped and the BCE is minimized.

A Autoencoder

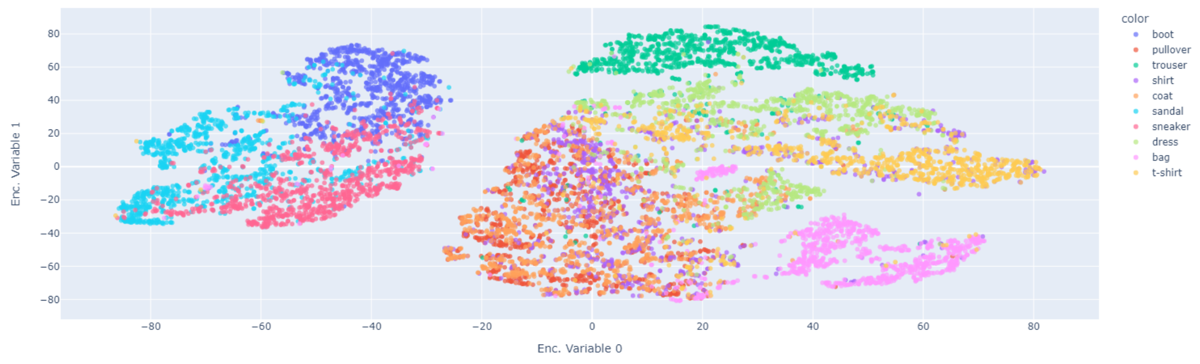
(a) *Original images*



(b) *AE reconstructions*

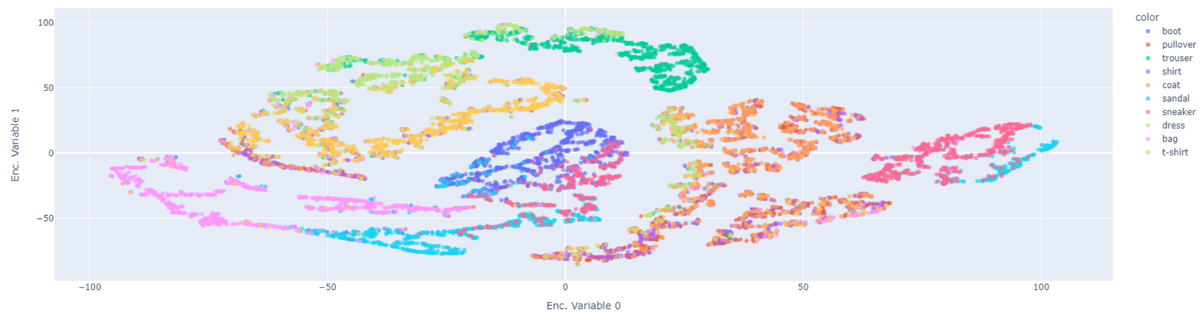


(a) *PCA*

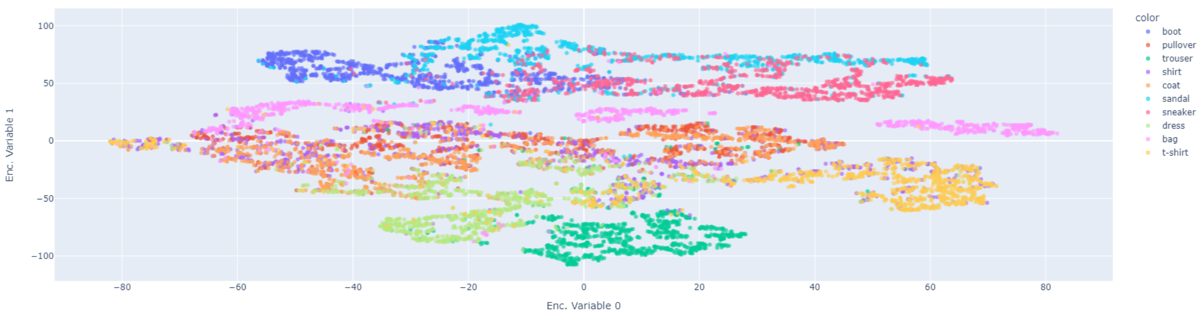


(b) *t-SNE*

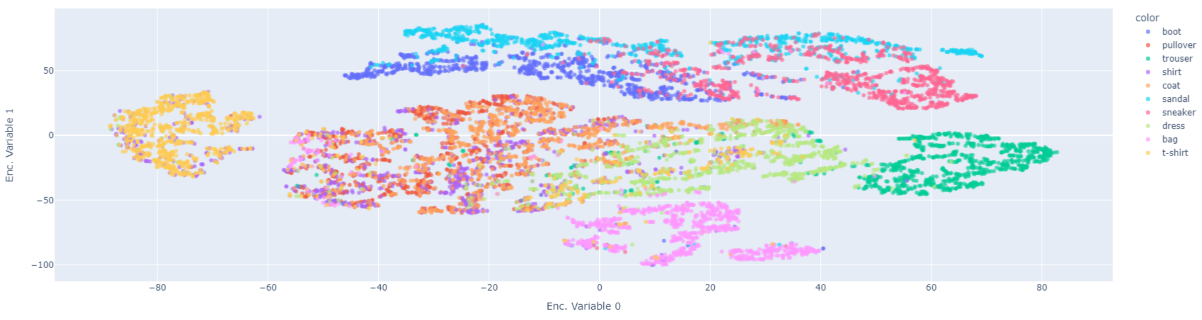
Encoded space



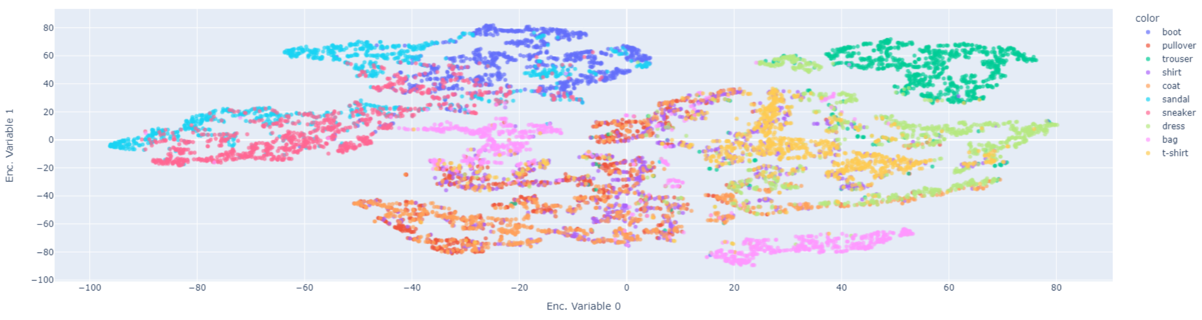
(a) *Latent space dimesion: 2*



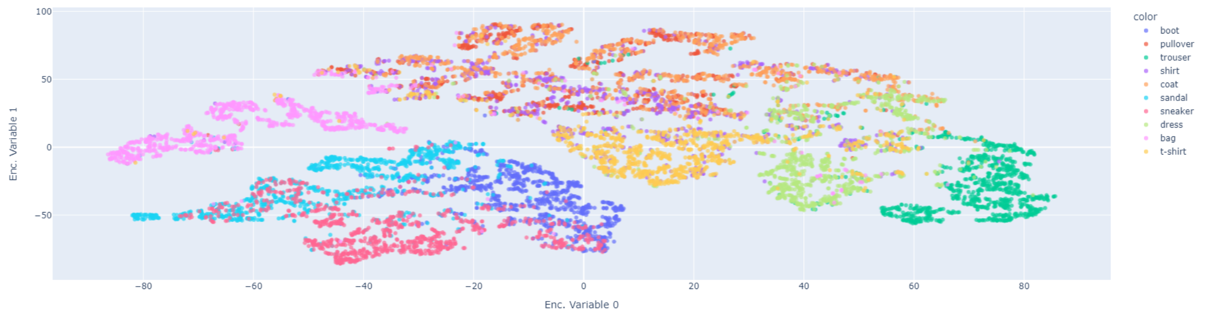
(b) *Latent space dimesion: 4*



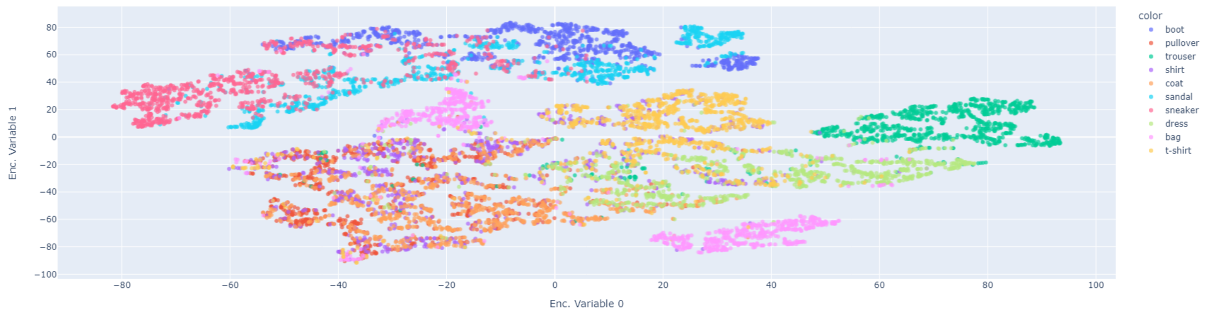
(c) *Latent space dimesion: 6*



(d) *Latent space dimesion: 10*



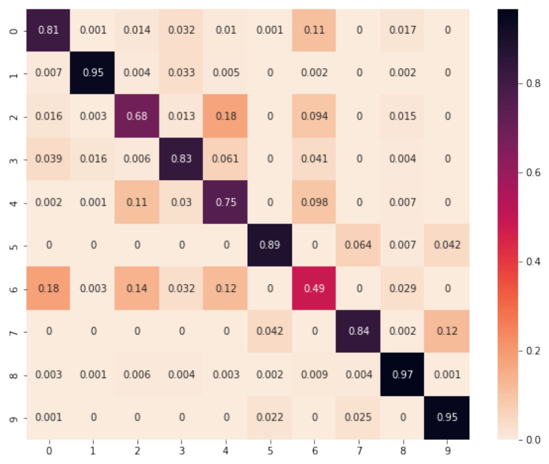
(a) *Latent space dimesion: 12*



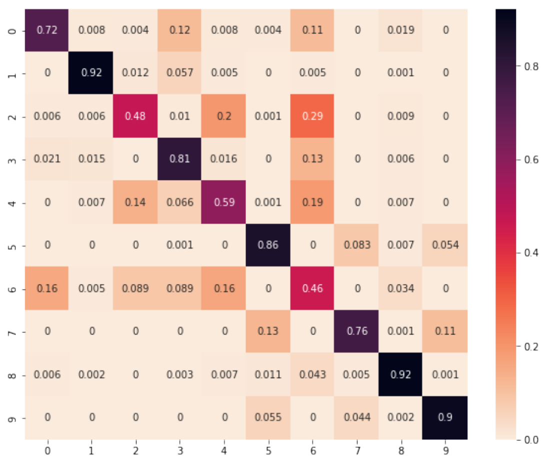
(b) *Latent space dimesion: 14*

Figure 6:

B Fine-tuning results



(a) *Network 1*



(b) *Network 2*

Figure 7:

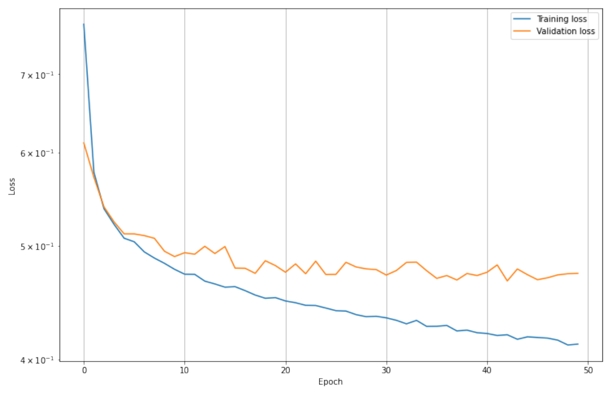


Figure 8: Loss trend of the first classifier implemented

C Variational autoencoder

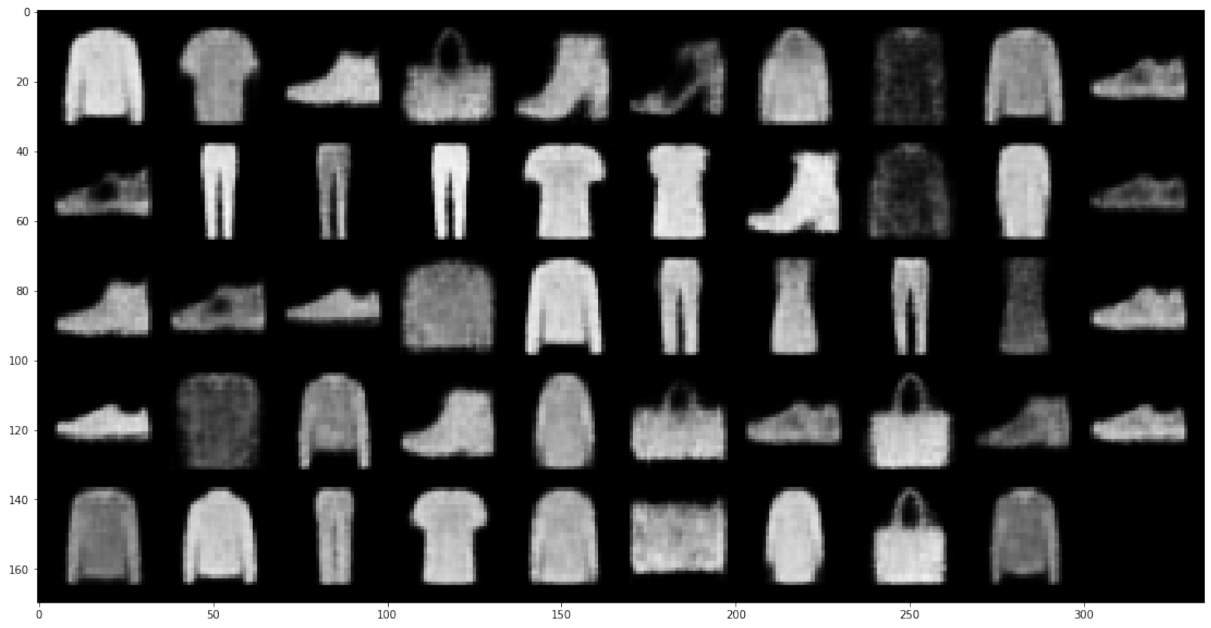


Figure 9: VAE reconstructions

D GAN

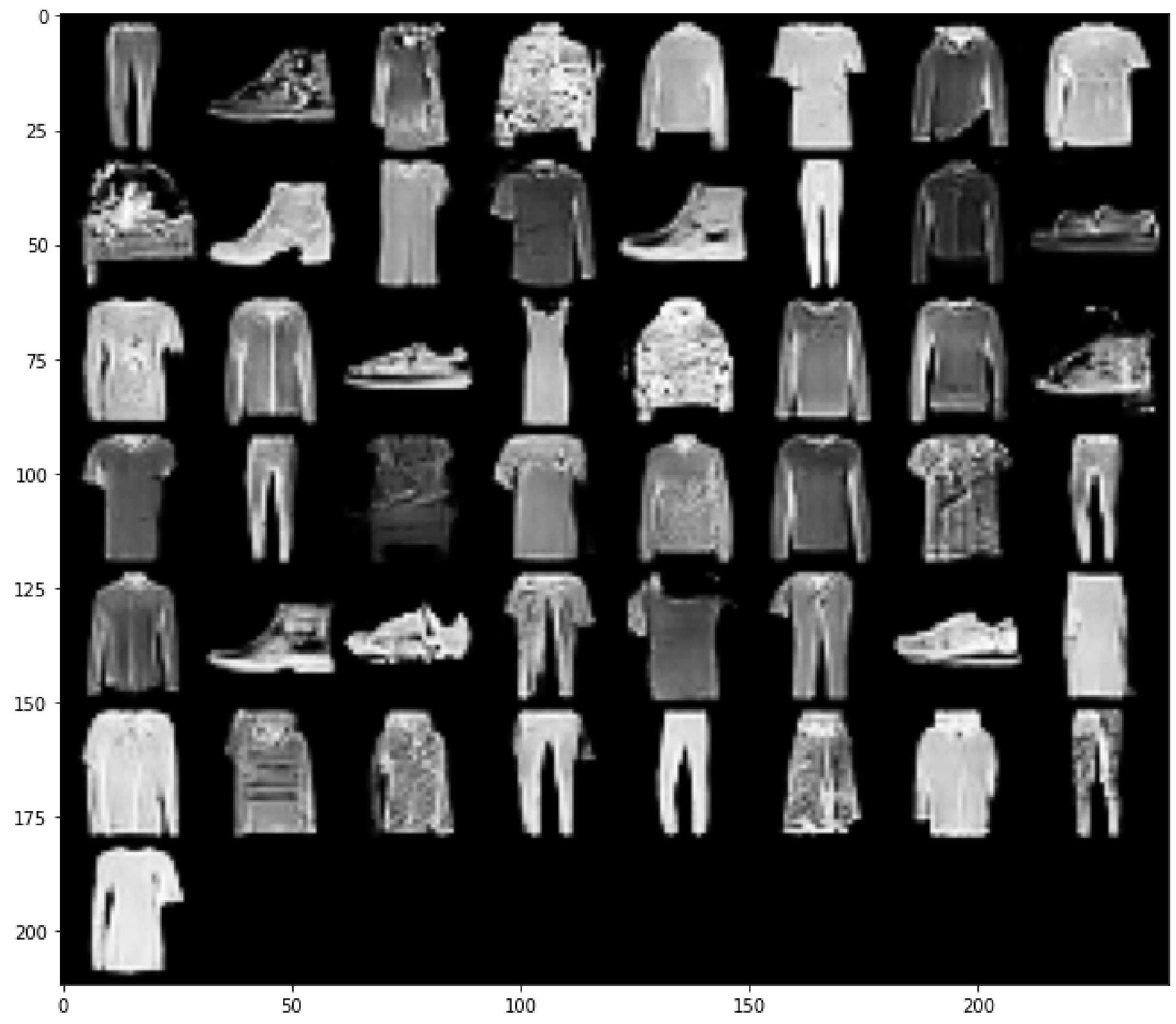


Figure 10: GAN reconstructions