Neural Networks and Deep Learning 2021/2022

# Unsupervised Deep Learning

**Capettini, Hilario (2013031)**

September 10, 2022

## Contents

## Abstract

In this work I explored different unsupervised learning techniques based on deep learning. First a convolutional autoencoder was trained and tested on the fashion MNIST dataset, its learning parameters were optimized using OPTUNA. Then the best convolutional autoencoder was used in the context of transfer learning to perform a supervised classification task. Finally a variational convolutional autoencoder and a Generative Adversarial Network were trained and tested with the purpose of generating new samples.

**Keywords**
Unsupervised Learning, PyTorch, Autoencoder, Fine Tuning, Neural Networks, Variational Autoencoder, GAN

## 1 Introduction

Unsupervised learning is the paradigm of trying to obtain information from a distribution that is not labelled [1], in this case we use the Fashion MINST dataset [2] containing 60000 greyscale images of fashion articles.Although this dataset contains labels for the images we are not going to use them.

## 2 Method

### 2.1 Convolutional Autoencoders (CAE)

An autoencoder is a neural network that is trained to attempt to copy its input to its output [1]. It is composed by two major elements the encoder that tries to reduce the dimensionality of the data by encoding the important information and the decoder that tries to re-build the original image based on low dimensionality information saved in the latent space. In this implementation the decoder is a reflected version of the encoder, which is customary but not mandatory.

The chosen architecture of the encoder, which can be seen in image 1, was the following:

- **First convolutional layer:** The depth is one because the input images have only one channel (greyscale), the number of filters $nf$ is a tunable hyperparameter of the network, the receptive field size (kernel) was chosen to be $3 \times 3$, stride 2 and padding 1 help me to reduce the final dimension avoiding the usage of pooling layers. As activation function I used the Rectified Linear Unit (ReLU).
- **Second convolutional layer:** The depth is $nf$, the number of filters is $nf \times 2$ and the other parameters are kept as in the first convolutional layer.
- **Third convolutional layer:** The depth is $2 \times nf$, the number of filters is $nf \times 4$, padding equal to 0 and the other parameters are kept as in the first convolutional layer.

  At this point the $[4nf]$ images are flattened and sent to a fully connected linear layer
- **First linear layer:** This layer has $4nf \times 3 \times 3$ input units and 64 output units. The activation function is ReLU.
- **Second linear layer:** This is the final layer where we move from 64 neuron inputs to the final latent

| Hyperparameter | Interval |
|---|---|
| $nf$ | $[4, 8, 16]$ |
| $n_{latent}$ | $[2, 10]$ |
| Optimizer | [SGD,ADAM] |
| $lr$ | $[1e-5, 1e-1]$ |
| $wn$ | $[1e-5, 1e-1]$ |

Table 1: Search space for Optuna

space encoding the information contained in each image, $n_{latent}$.

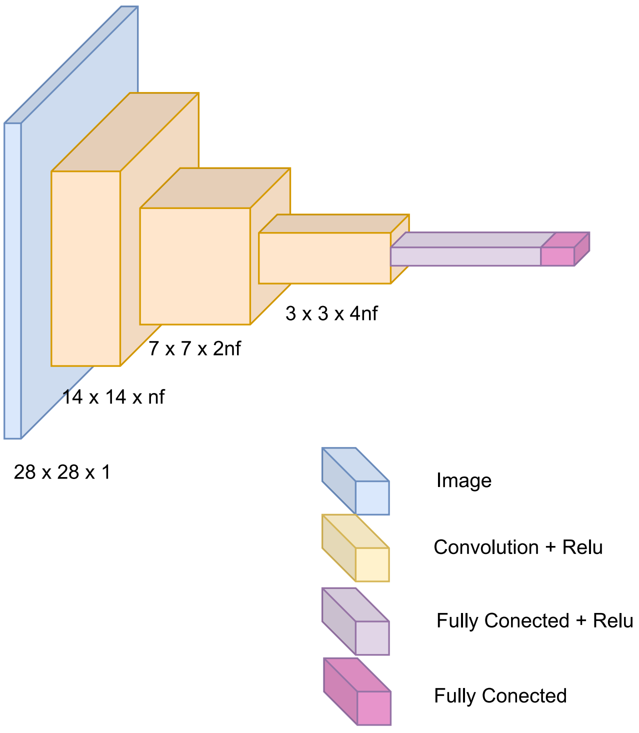


Figure 1: Architecture of the encoder, the decoder is a reflected version of the encoder.

For the CAE training I explored two optimizers, the classical *SGD* and the more advanced *ADAM*, for both of them I set the learning rate $lr$ and weight decay $wn$ as hyperparameters to tune. The chosen loss for the optimisation was the mean squared error (MSE) between each image in the input and the produced image by the decoder.

The chosen architecture and setup allow me to optimise 4 hyperparameters and the network optimiser. The optimisation of the hyperparameters can be costly for this architectures, that is why I decided to avoid the brute force gridsearch and went for the Bayesian optimisation approach which gave me the chance to get better results faster. To do so I used Optuna [3], which is a hyperparameter tuning framework that implements this Bayesian optimisation. The searchspace for Optuna can be seen in table 1.

## 2.2 Latent space

Now we want to obtain some visual information about the latent space where the main features of our images are encoded. As the best results are usually retrieved when using a encoding space with a dimensionality bigger than 2 we are forced to use some dimensionality reduction technique to move the main characteristics from $n_{latent}$ dimensions into 2 so we can plot it. For that purpose we used two main techniques:

- In **Principal Component Analysis (PCA)** both the compression and the recovery are performed by linear transformations and the method finds the linear transformations for which the differences between the recovered vectors and the original vectors are minimal in the least squared sense [4].
- **t-distributed stochastic neighbour embedding (t-SNE):** is a dimensionality reduction technique presented by Hinton [5] which is based on non-linear transformations. In general this technique allow to obtain better defined clusters than PCA.

## 2.3 Fine tuning

At this point of the work we have a convolutional autoencoder trained in such a way that is able to compress images into a feature space of dimension $n_{latent}$ and reconstruct them based on the information there stored. Now our task is to use this network and the learnt weights to tackle a completely different task, the classification of the images under the supervised learning paradigm. The idea that we can use some knowledge obtained in one task and use it so perform another one is known as transfer learning.

In practical terms what I did was to connect the encoder of our CAE to a simple linear layer with input dimension $n_{latent}$ and output dimension equal to the number of categories to classify, 10. Then we disabled the weight update of the convolutional layers of the encoder, in this way we keep the knowledge obtained about the identification of the images (the filters we have learnt). And we allowed the update of the weights of the linear layers.

In this case no optimisation of the hyperparameters was performed. We used ADAM as the optimizer with a small learning rate in order to modify the weights of the encoder linear layers only in a small portion. The loss used for training was the CrossEntropyLoss between the predicted label and the real label of the images.

## 2.4 Variational autoencoder (VAE)

A variational autoencoder can be seen as an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has "good properties" that enable generative process. The change in approach to obtain such a latent space is that we are no longer directly generating the latent space but we are learning a distribution that characterises this space. So the network is learning the parameters of a normal distribution, the *latent distribution*, from which we will be sampling the latent space representation of the images. In figure 2 it can be observed the general training for a VAE.

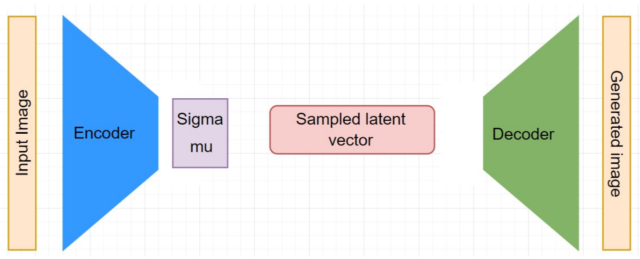


Figure 2: Variational Autoencoder pipeline.

The architecture of the VAEs encoder and decoder is the same as the architecture for the CAE encoder and decoder. In particular the initial number of filters was set to $nf = 8$ and the dimension of the latent space to $n_{latent} = 2$ so I was able to directly generate images from this space. The chosen optimiser was Adam with a learning rate equal to $1e-3$ and no weight decay.

The loss function for the VAE has two terms, one that is responsible of the correct reconstruction (MSE in this case) and a regularization term making the latent space more regular. For the later purpose the Kullback-Leibler divergence is used. In this implementation the last term importance is weighted using a parameter $\beta$ which was set to $\beta = 0.8$.

### 2.5 Generative Adversarial Networks (GAN)

This deep learning model was introduced in 2014 by Goodfellow in [6], this model has two main component; a generator that take as input some noise and return an image that should resemble the ones we are using for training, and a discriminator (a binary classifier) that should identify if the image is real or created. This two components compete, the generator trying to fool the discriminator and the discriminator trying to catch the fake images. During this game the generator should improve the quality of the images so that at the end we can generate almost real images using it. The training scheme is presented in figure 3. As the training is a time consuming I designed the discriminator and generator as fully connected networks. The generator takes as input a noise vector normally distributed (size 100) and process it along four linear layers whose activation's functions are ReLU (by exeption of the last activation where I used the hyperbolic tangent to keep the pixel values between $-1$ and 1). At the end of the

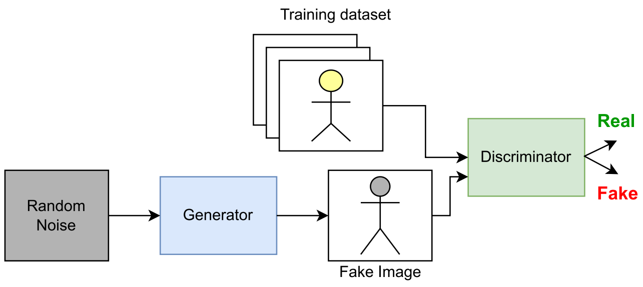


Figure 3: Training pipeline for the Generative Adversarial Network.

processing the results are stacked returning a $28 \times 28$ greyscale image.

The discriminator takes as input a vector with all the image pixels ($28 \times 28 = 784$), process it along 4 linear layers whose activation's functions are ReLU functions, these layers also have dropout capabilities which help the model to regularise. I set the dropout probability to 0.3.

The training of the network was done by optimizing the *Binary Cross Entropy* function for both networks at the same time. To do it the loss for the discriminator was estimated on its predictions and the real labels, while for the generator the loss was estimated on the discriminator predictions on generated data and the real labels (we measure if the generator fools the discriminator). In both cases the optimiser was ADAM with a learning rate of $lr = 1e-4$.

## 3 Results

### 3.1 Convolutional Autoencoders (CAE)

After going through 100 trials, the best hyperparameters according to Optuna were:

- Initial number of filters $nf = 16$
- Number of dimensions for the latent space $n_{latent} = 10$
- Optimiser *Adam*
- Learning rate $lr = 4 \times 10^{-3}$
- Weight decay (L2 reg) $wn = 1 \times 10^{-5}$.

The importance of the different hyperparameters can be observed in figure 4, surprisingly the regularization is pointed by optuna as the most important parameter followed by the optimizer method and in third place the initial number of filters. It is important to mention that this result changed between different studies.

In figure 5 can be observed the evolution of the training and test losses as a function of the epochs. In figure 6 we can observe the evolution in the generated images comparing two different epochs in the training process, there we see how the definition of the objects improve.

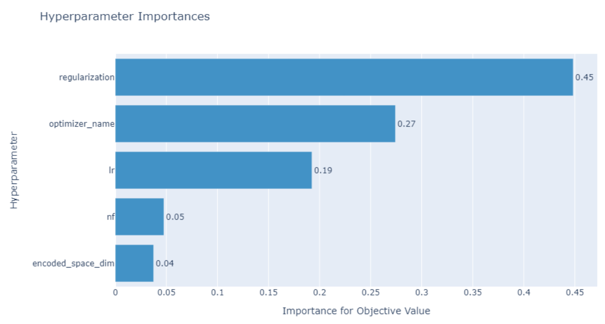


Figure 4: Relative importance of the different hyper-parameters.

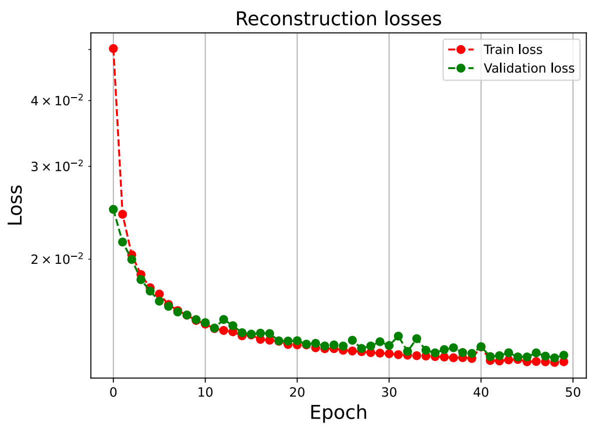
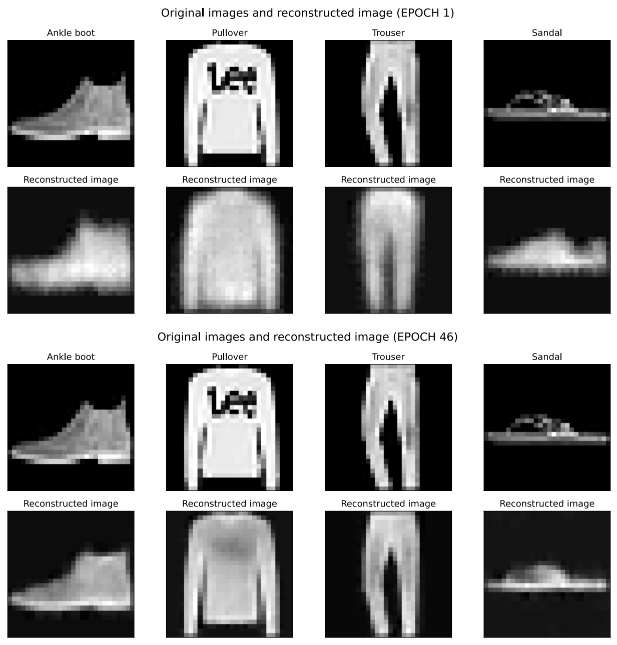


Figure 5: Evolution of the training and test losses along the epochs for the best performing model.



Original images and reconstructed image (EPOCH 1)
Ankle boot
Pullover
Trouser
Sandal
Reconstructed image
Original images and reconstructed image (EPOCH 46)

Figure 6: Evolution in the generated images quality.

## 3.2 Latent space

For the interpretation of the CAE latent space I applied two dimensionality reduction techniques, the results can be observed in figure 7. In both images we can observe the formation of clusters for the different fashion items. However the separation between clusters become more clear for the T-SNE technique while for the PCA there is some overlapping between the different categories.

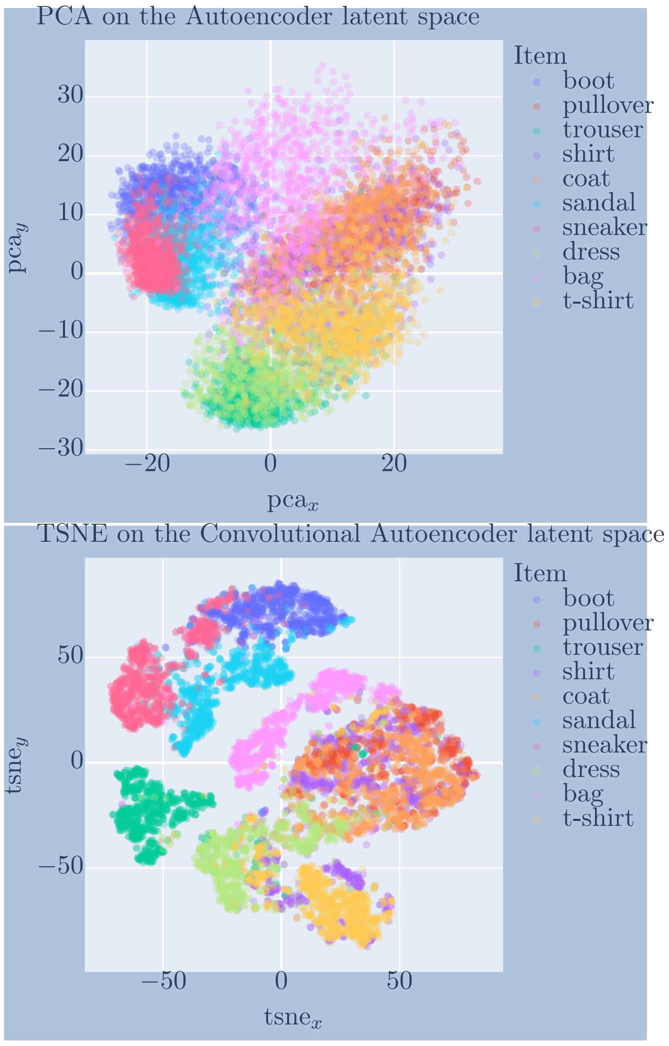


Figure 7: Dimensionality reduction from latent space (10 dimensions) to 2D for visualisation.

Finally I trained the same model but having a latent space with two dimensions in order to directly generate samples from it. Figure 12 in the appendix, shows its latent space structure for the test dataset and figure 13 shows some images generated from this latent space. There we observe that the change from one image to the other is not always smooth, that is something expected from the classical autoencoders where no regularization of the latent space is required and we have this lack of smooth change between one region and the other.

### 3.3 Fine tuning

For this task we obtained a final accuracy on the test set of %87 which is slightly below the %91, 56 we obtained for Homework 1. In figure 14 we observe the confussion matrix showing the accuracy for the different items. Figure 8 shows the evolution on the training and validation losses per epoch. There it can be observed that after 8 epochs it starts to overfit and it is no longer learning much, which is very similar to the results I obtained for homework 1. The big difference was in the computer time that was demanded for training, the fine tunning approach took approximately 4 times less time to get the final results. As here we are not training the convolutional layers, it was considerably less time demanding.

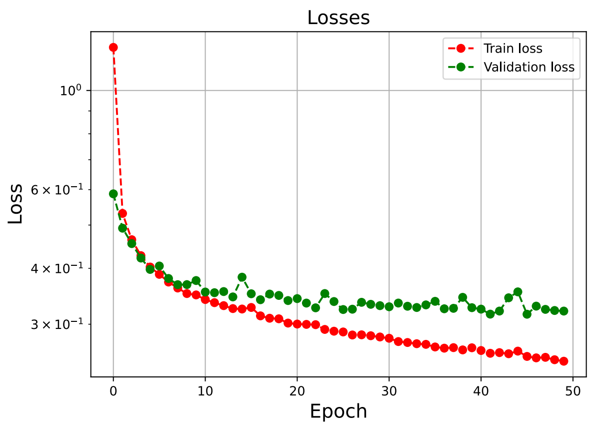


Figure 8: Evolution of the training and validation losses for the supervised classification.

### 3.4 Variational autoencoder (VAE)

Figure 9 shows the position of the test images in the VAE latent space, here some overlapping between categories can be observed but the most important feature is the smooth change from one cluster to the other, and also the lack of gaps between clusters. This is a very important feature of VAE latent space. This property allow us to use the VAE to generate new samples which will resemble the original data, it can be seen in figure 15, where we observe that the change from one object to the other is smooth, a big difference with images generated using CAEs decoder (figure 13).

### 3.5 Generative Adversarial Networks (GAN)

Figure 10 shows the evolution of the training losses for the generator and for the discriminator. There we can observe that they are in synchrony, proof of a good training. We also observe that in the first phase the generator loss decrease, an indication that the generator is fooling the discriminator.

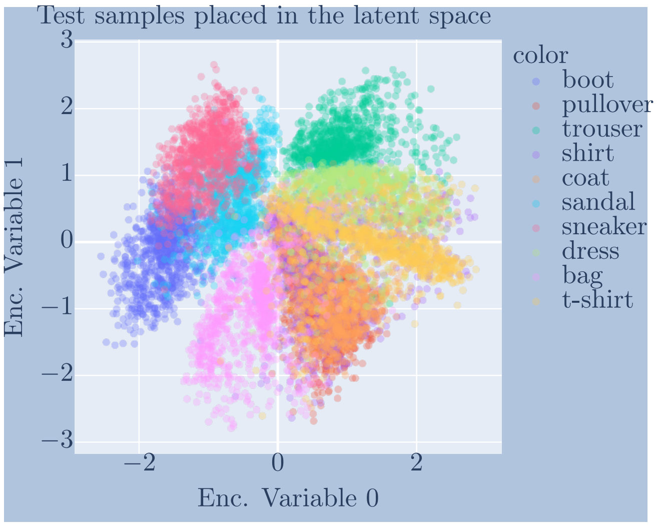


Figure 9: Variational Autoencoder latent space.

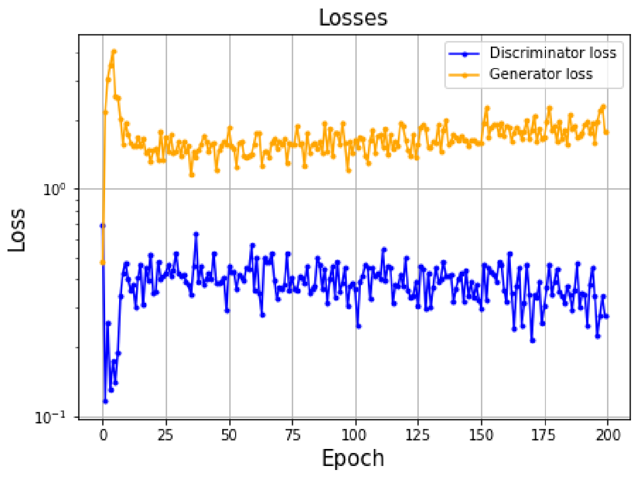


Figure 10: Evolution of the training losses for the generator and discriminator.

Figure 11 shows the evolution in the quality of the generated images after 5 and after 100 epochs of training. As expected initially the generator is performing poorly and we only obtain some shadows of the items and a lot of salt-pepper noise. However as we keep training we obtain images of products that we can confidently classify as what they are.

## 4 Conclusion

To conclude, in this work I managed to implement a Convolutional Autoencoder whose hyperparameters were successfully optimised using a Bayesian optimisation technique. This lead to a smooth latent space characterising the different images in the MNIST dataset. Furthermore the trained encoder was successfully used for a supervised classification task proving to be a useful technique when the computational resources are limited.

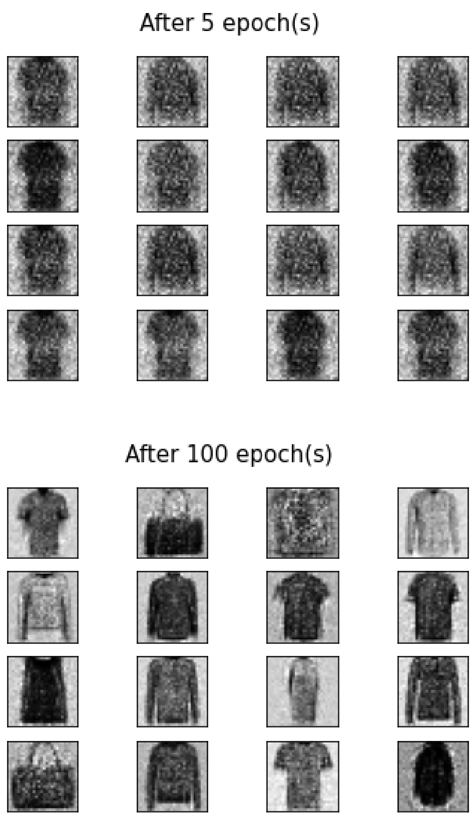


Figure 11: Generation of images using GAN for different training epochs.

Finally in this work I also implemented a Variational Autoencoder and a Generative adversarial Network. Both of them proved to be efficient method to generate synthetic data resembling the original dataset. Which could be useful for augmenting a small dataset for example.

**Code**

All the code used for this notebook can be found at https://github.com/hcapettini2/NNDL.

## References

[1] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[2] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747, 2017.

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[4] Shai Shalev-Shwartz and Shai Ben-David. Understanding Machine Learning - From Theory to Algorithms. Cambridge University Press, 2014.

[5] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. Advances in neural information processing systems, 15, 2002.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. Advances in neural information processing systems, 27, 2014.

## A Appendix

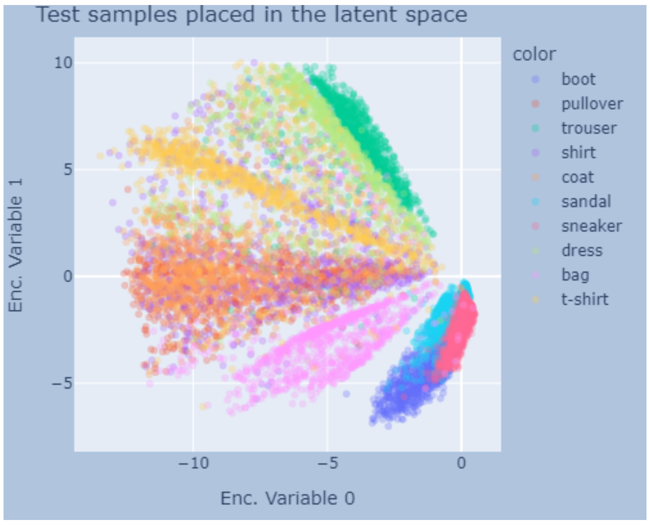


Figure 12: GAN latent space structure for the test dataset

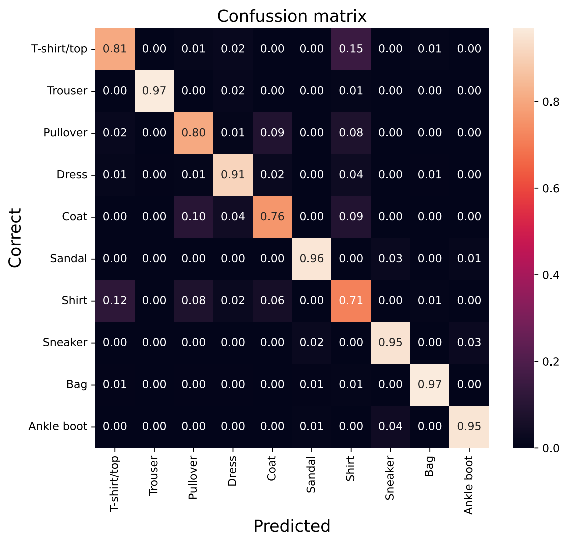


Figure 14: Confusion matrix for the supervised classification task.

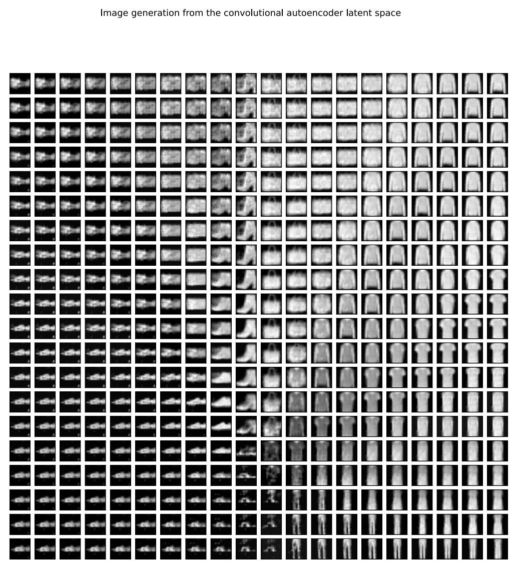


Image generation from the convolutional autoencoder latent space

Figure 13: Images generated using the CAE decoder.

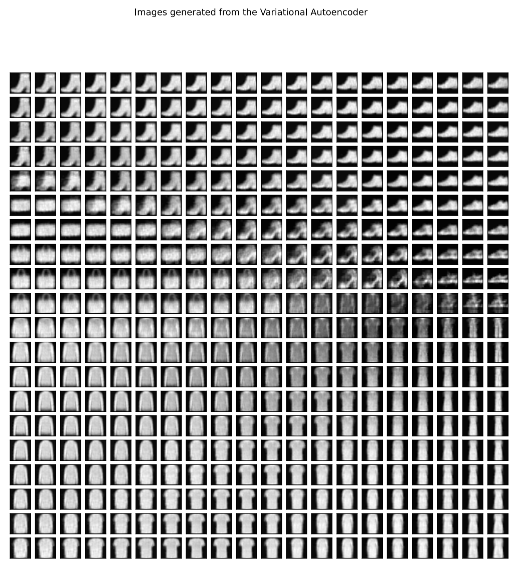


Figure 15: Images generated using the VAE decoder.