Homework II - unsupervised learning

Neural Networks and Deep Learning
Winter term 2021/2022
Prof. Alberto Testolin

Monday 4th July, 2022

Nathanael Schmidt-Ott URSNATHANAEL.SCHMIDTOTT@STUDENTI.UNIPD.IT
Home institution: Göttingen University
Guest Student at the University of Padova

1. Introduction

This report presents solutions for the second homework of the *Neural Networks and Deep Learning* lecture at the *University of Padova*. The goal of this work is to implement unsupervised learning techniques to analyse images from the widely used library MNIST Fashion. In particular, I first train a convolutional AutoEncoder (AE) in an unsupervised fashion. I then fine tune the AE in a supervised learning task, combining it with a simple Multi Layer Perceptron (MLP) to generate predictions for the class labels. Finally, I fit a convolutional Variational Autoencoder (VAE) and compare it to the AE. In particular, I explore the latent space of both approaches and generate new samples using both models. In all tasks, I tune a selection of hyper-parameters using the automatic tuning tool *Optuna*. The remainder of this report is structured as follows, section 2 introduces the methods, while section 3 presents results and section 4 concludes.

2. Methodology

2.1 The Convolutional Autoencoder

Autoencoders in general consist of two building blocks: the *encoder* and the *decoder*. The encoder f projects an input image x into some *latent space* of arbitrary dimension d: $f : x \rightarrow \mathbb{R}^d$. The *decoder* takes the latent representation $h = f(x)$ of the original image as input and aims at reconstructing the original image: $g : h \rightarrow \mathbb{R}^{p\times p}$, where p is the dimension of the input space. Thus I define the reconstructed image as $r = g(h) = g(f(x))$.

It is convenient define the encoder and the *decoder* separately. Since the input data are images, I use convolutional layers in the *encoder* as well as in the *decoder*. In particular, I include three convolutional layers followed by two linear layers in the *encoder*. Though not a hard constraint, it is common practice to define the *decoder* symmetric to the *encoder*. I follow this approach in this work. Tables 1 and 2 summarise the architecture of the *encoder* and the *decoder*. n is the batch size, which I set to $n = 256$ throughout this analysis. In contrast, I learn the optimal dimension of the latent space d and the dropout rate dr during training. I discuss the exact procedure in the following section. In the last convolutional layer of the *decoder* I include a Sigmoid activation function in order to ensure that all output pixels are bound between 0 and 1. In this first model, I train the autoencoder for a total of 42 epochs and provide results obtained using Adam and stochastic gradient descent (SGD) as optimisers. Moreover, I set the learning rate to $lr = 0.0005$, the penalty for weight decay $\alpha = 0.0001$, and the dropout rate in the linear layers $r = 0.2$ after some trial and error. To calculate the loss, I use the mean squared error (MSE) between the reconstructed image r and the original image x: $L(x_i, g(f(x_i))) = \frac{1}{N}\sum_i(x_i - g(f(x_i))^2$. Finally, I initialise weights using kaiming weight initialisation.

Table 1: The ***encoder***

Layer	Input	Output	Kernel	Stride	Padding	Activation	Dropout
Conv layer 1	n x 1 x 28 x 28	n x 8 x 14 x 14	4 x 4	2	1	ReLU	
Conv layer 2	n x 8 x 14 x 14	n x16 x 7 x 7	4 x 4	2	1	ReLU	
Conv layer 3	n x16 x 7 x 7	n x 32 x 3 x 3	3 x 3	2	0		
Linear layer 1	n x 288	n x 64				ReLU	dr
Linear layer 1	n x 288	n x d					dr

Table 2: The ***decoder***

Layer	Input	Output	Kernel	Stride	Padding	Activation	Dropout
Linear layer 1	n x d	n x 64				ReLU	dr
Linear layer 1	n x 64	n x 288				ReLU	dr
Conv layer 1	n x 32 x 3 x 3	n x16 x 7 x 7	3 x 3	2	0	ReLU	
Conv layer 2	n x16 x 7 x 7	n x 8 x 14 x 14	4 x 4	2	1	ReLU	
Conv layer 3	n x 8 x 14 x 14	n x 1 x 28 x 28	4 x 4	2	1	Sigmoid	

2.2 Tuning hyperparameters

Autoencoders include many hyper-parameters, which ideally should all be tuned empirically using some appropriate technique such as grid-or random-search or automatic tuning tools such as *Optuna*. In this work, I focus on the main hyper-parameters and set the remaining ones - in particular those that concern the architecture of the encoder and the decoder arbitrarily after some trial and error. Specifically, I focus on tuning (i) d - the dimension of the latent space, (ii) dr - the dropout rate in the linear layers, (iii) α - the strength of the penalty of weight decay, (iv) - the optimiser (Adam, SGD) and finally (v) lr - the learning rate. I learn the hyper-parameters using the automatic tuning tool *Optuna* and consider 20 different trials, where different hyper-parameter constellations are explored. *Optuna* allows to efficiently search the hyper-parameter space, pruning unpromising trials. Since the dataset is relatively large while computational capacity is limited, I do not consider a cross validation set up when tuning the hyper-parameters. Rather I set aside 20% of the training data as validation set. Accordingly, the training and validation datasets include 48 000 and 12 000 samples respectively.

2.3 Supervised learning

Unsupervised learning may serve as a first step to pre-train a model. The general idea is that the *encoder* extracts some meaningful and general features from the data which can then be used as input to some downstream and relatively simple supervised learning algorithm. This allows to transfer the pre-trained *encoder* to similar supervised learning tasks and reduces the computational burden when tuning the supervised algorithm as the model only needs to be fine tuned in the supervised setting. In this work, I use the trained *encoder* with optimised parameters, which I first train in an unsupervised fashion to obtain latent representations of the input images. In a second step, I use the latent representations as input to a Multi-Layer Perceptron (MLP) with two hidden layers. Specifically the final classifier looks as follows:

Table 3: The Final Classifier

Layer	Input	Output	Activation	Dropout
Encoder	n x 1 x 28 x 28	n x d		
Linear 1	n x d	n x $No1$	ReLU	mlp_dr
Linear 2	n x $No1$	n x $No2$	ReLU	mlp_dr
Output	n x $No2$	n x 10		

The *encoder* is defined as in section 2.1. mlp_dr, $No1$ and $No2$ are hyper-parameters which I tune using *Optuna*. The former is the dropout rate, while the latter two define the number of hidden units in the linear layers. Furthermore, I use two different optimisers: one for the *encoder* and one for the *MLP*. I treat the optimiser for the *MLP* as hyper-parameter including the learning rate (*mlp_lr*) and *mlp_α*: the strength of weight decay in the MLP. For the encoder, I rely on the hyper-parameters that I tune during unsupervised learning. As loss function I use Cross-Entropy Loss (CEL), which is appropriate in a multi-class classification problem. Again, I use a validation set approach to tune the hyper-parameters and I test the final model on the test set, containing an additional 10 000 samples.

2.4 Exploring the latent space

To explore the latent space, I first extract the latent representations of all samples in the test data. To get an idea of how well the classes separate from each other in the latent space, I reduce the dimension of the latent representations to 2, using Principal Component Analysis (PCA) and t-Stochastic Neighbour Embedding (t-SNE). This allows me to easily plot the latent representations. Moreover, I explore the latent space by creating an image transversal. This implies taking latent representations of 4 different images and altering them step by step to iteratively move across the latent space until reaching the other latent representations. I expect this transversal to change abruptly in one step, since the latent space of autoencoders are generally not smooth but irregular. Finally, by randomly sampling from the latent space, I generate new samples, mimicking the behaviour of generative models.

2.5 Variational autoencoder

Compared to the vanilla implementation of autoencoders, the variational autoencoder introduces a probabilistic mapping in the latent space rather than a single point. Thus, the latent representations of the inputs are no longer deterministic mappings (i.e. points), but probability distributions in the latent space. One the one hand this makes the latent space smoother, on the other hand it enables us to generate new samples. In practice, the target distribution of the encoder is often a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu_x}$ and covariance matrix $\boldsymbol{\Sigma_x}$. As commonly done, I assume the latent dimensions to be independent, making $\boldsymbol{\Sigma_x}$ a diagonal matrix. Thus, the output of the encoder are two vectors of dimension d, where the first vector is $\boldsymbol{\mu_x}$ and the elements of the second vector are the diagonal elements of $\boldsymbol{\Sigma_x}$. Since sampling is a discrete process, which hinders backpropagation, I sample from a standard normal distribution and linearly transform the samples using $\boldsymbol{\mu_x}$ and $\boldsymbol{\Sigma_x}$ to create latent representations: $\boldsymbol{z_x} = \boldsymbol{\Sigma_x}\boldsymbol{\eta} + \boldsymbol{\mu_x}$, $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. The loss function for the VAE includes an additional regularisation term that penalises the divergence between the sampled gaussian and the standard normal gaussian. This term becomes 0 if the sampled distribution is a standard normal distribution. The architecture of the VAE is very similar to the architecture of the presented AE. In fact, I use the same *decoder* as specified in table 2 and only adapt the *encoder* to include the sampling process. Finally, I do not consider dropout in the linear layers.

3. Results

3.1 The Convolutional AutoEncoder

Figure 1 displays the trend of the reconstruction loss of the basic implementation of the convolutional AE.

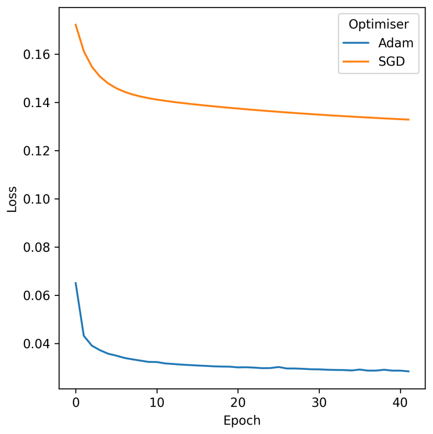


Figure 1: Reconstruction loss

The orange line displays the evolution of the reconstruction loss when using SGD as optimiser while the blue line depicts the loss when using the Adam optimiser. Clearly, the Adam optimiser outperforms vanilla SGD both in terms of speed of convergence as well as in terms of reconstruction loss. Using the Adam optimiser, the reconstruction loss of the model decreases relatively quickly after only a few epochs and then appears to slowly converge. In fact, the quality of the reconstruction of an exemplary image quickly increases in the first epochs and only improves slightly after epoch 20 as depicted in figure 7a in the appendix. In contrast, when using SGD the reconstructed image still looks like a random generation even after all 42 training epochs (figure 7b in the appendix).

3.2 Parameter tuning

Table 4 presents the tuned hyper-parameters for the AE that I obtained using *Optuna*. As expected from the analysis in section 3.1, Adam rather than SGD produces more accurate results.

Table 4: Tuned AE

Parameter	Value
d	185
Optimiser	Adam
Learning rate	0.0058
Dropout rate	0.005
α	$1.038e^{-5}$

Figure 8 in the appendix displays the final validation loss for all 20 hyper-parameter combinations that I explored using the automatic tuning tool *Optuna*. The red dot highlights the trial - or put differently the hyper-parameter combination - which yielded the lowest validation loss. In particular later trials achieve very similar reconstruction error to the best trial. In fact, the hyper-parameters of these trials are relatively similar to the hyper-parameters of the best trial. I refit the AE using the tuned hyper-parameters to reconstruct exemplary images taken from the validation set.

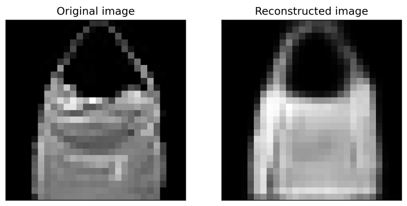


Figure 2: exemplary image reconstruction

Figure 2 as well as figures 10a and 10b in the appendix show such reconstructions for three different kinds of objects. In all three examples, the AE manages to reconstruct the general shape of the input image, but fails to reconstruct smaller details. Taking for instance the bag, displayed in figure 2, we see that the reconstructed image is generally similar to the original image. However, it lacks the nuances of the fabric of the bag itself. Moreover, the edges are not as clearly defined as in the original image. Perhaps training the algorithm for longer, with a more complex architecture could improve upon this. However, given limited computational capacities I leave it there. Nonetheless, comparing the reconstructed image from the model with tuned hyper-parameters (figure 2) to the final reconstruction of the untuned AE (figure 7a in the appendix), the former is more nuanced than the latter.

3.3 Supervised learning

I transfer the trained *encoder* to the classification task aimed at predicting the class labels of the MNIST-Fashion dataset. Table 5 presents the tuned hyper-parameters for the classifier that I stack on top of the pre-trained *encoder*. The combination of the *encoder* and the *MLP* with their respectively tuned hyper-parameters are the full classifier (CL). I fine tune this classifier for 10 epochs on the full training data and use the trained model to make predictions on the test set.

Table 5: Tuned MLP

Parameter	Value
$No1$	12
$No2$	74
Optimiser	Adam
Learning rate	0.0002
Dropout rate	0.0676
α	0.1679

Table 6 presents results for the final classifier as well as for the CNN that I trained in the previous homework on the same dataset. Differently to the CNN, I do not augment the data before training the final model. Nonetheless, the trained classifier achieves a final accuracy of 90.08%, which is considerable given that I trained the model for merely 10 epochs, which took about one minute on a GPU. In contrast, the final test-accuracy of the CNN trained with augmented data, is 93.51% but training the model took considerably more time: in total the final CNN trained for about 28 minutes in 26 epochs on a GPU. In assignment 1, I also trained a CNN without augmented data. However, I did not test the accuracy of that model on the test set. The final validation accuracy for the CNN without augmented data, was 93.2 %, while the final validation accuracy of the present classifier was 90.33 % - only marginally higher than the final test error. Nonetheless, the CNN with augmented data from assignment 1, outperforms the present classifier in all classes when looking at the F1-score in table 6.

Table 6: Test Results

Class	F1 CNN	F1 CL	n
T-shirt/Top	89.21	85.36	1000
Trouser	99.35	98.24	1000
Pullover	90.71	83.02	1000
Dress	93.31	89.97	1000
Coat	90.22	84.04	1000
Sandal	98.89	97.65	1000
Shirt	79.94	72.82	1000
Sneaker	97.07	95.77	1000
Bag	98.32	97.41	1000
Ankle Boot	97.55	96.74	1000
Accuracy	93.51	90.08	
# epochs	26	10	
Training time (s)	1677	61	

3.4 Exploring the latent space

Figure 3 depicts the embeddings obtained using tSNE on the latent representations of all samples in the test set. The colours indicate to which class a particular latent representation belongs to. Clearly, we see that some classes separate relatively well from each other and that samples belonging to the same class cluster together in the latent space. However, there are also some classes, such as class 2 (pullovers) and class 4 (coats) or class 0 (T-shirt/Top) and class 6 (Shirt) which do not separate well from each other. The reason why these do not separate well from each other might be that these objects are conceptually similar. Thus, the latent representations of samples belonging to these classes are close to each other in the latent space. Figure 9 in the appendix presents the results obtained using PCA. However, as expected, the classes separate better when using more sophisticated tSNE. Figure 4 shows decoded samples that I obtained by randomly sampling values from the latent space. None of the images appears to be particularly similar to any of the input images that I used for training.

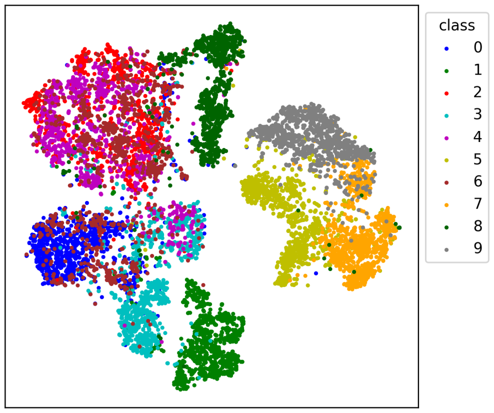


Figure 3: tSNE on the latent space

In fact, randomly sampled samples from the latent space only sometimes represent meaningful images, because the latent space of AE is generally irregular. Thus, sampled latent representations may exhibit strongly different outputs even if they are close to training samples in the latent space. Nonetheless, some newly generated samples are somewhat similar to input images. For instance, sample 8 in figure 4 could be some type of shirt.

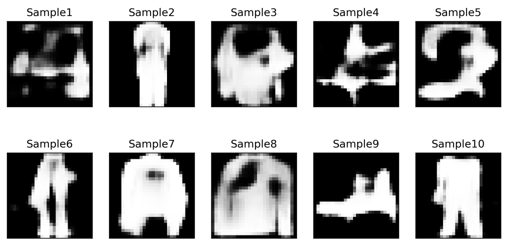
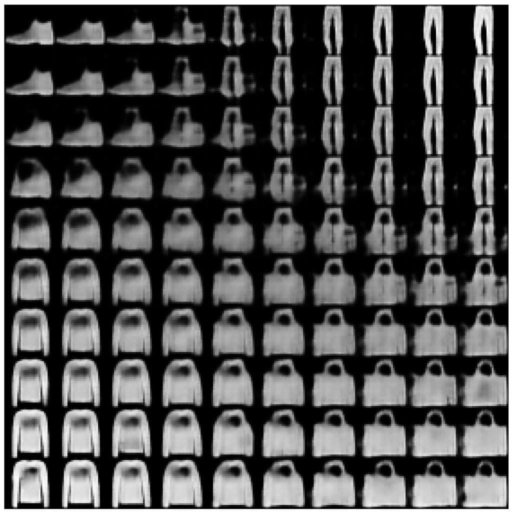


Figure 4: Newly generated samples

Finally, to highlight the irregularity of the latent space, figure 5a shows how from the decoded latent representation of an exemplary image changes when taking equally sized steps towards the latent representation of another exemplary image. The plots on the far left and on the far right depict the decoded latent representations of the original input images and the images. The plots in between depict the decoded latent representations of the steps in between. If the latent space were smooth, I would expect a gradual change from one shape to the next shape. However, in this case the shape appears to change abruptly from one plot to the other.


(a) Latent space transversal AE

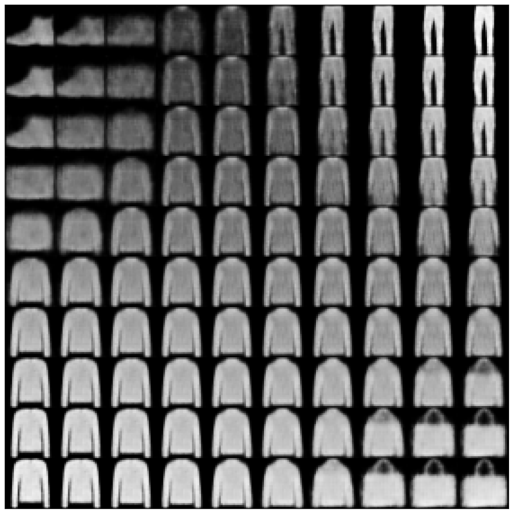

(b) Latent space transversal VAE

Figure 5: The latent space in AE and VAE

3.5 Variational Autoencoder

The latent space produced in the VAE is smoother than the latent space in vanilla AEs. In fact, figure 5b shows the transversal from four latent representations of four different objects to each other. The changes between the different decoded latent representations is gradual, implying indeed that the latent space is smooth. For instance, moving from a shoe in the top left corner to a bag in the bottom right corner, the depicted shape only changes very gradually, producing interesting shapes along the way, which are not meaningless. Table 7 in the appendix shows the tuned hyper-parameters for the VAE, that I obtained using the automatic tuning tool *optuna* and figure 11 in the appendix depicts the loss of the VAE against the epoch. Finally, we can use the VAE to generate new samples from scratch, simply by sampling d values from a standard normal distribution. Figure 6 depicts such new samples. In contrast to the newly created samples using the vanilla implementation of AE (see figure 4), the generated samples from the VAE are all meaningful and could potentially stem from the original dataset.

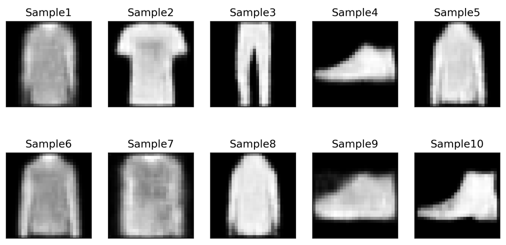
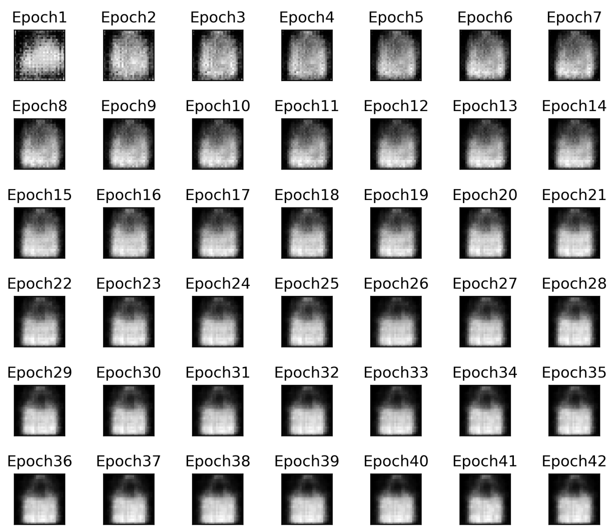


Figure 6: New samples generated using the VAE

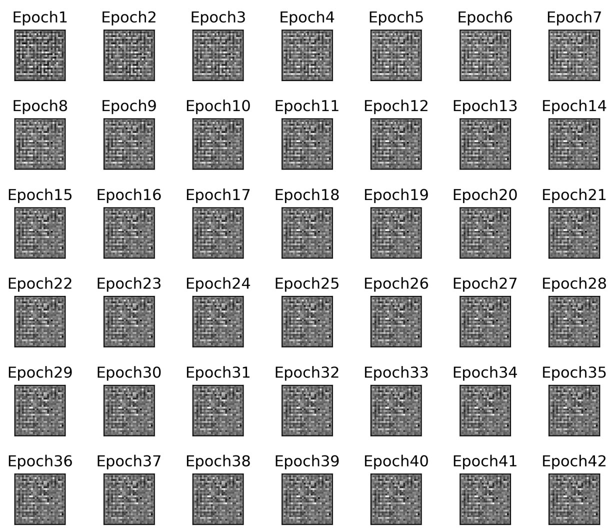
4. Conclusion

This report gives an overview fo the methods and the results for an unsupervised learning task on the MNIST-Fashion dataset. In particular, I explore different model architectures such as convolutional autoencoders and variational autoencoders. I show how the latte improves upon the former generating a smoother latent space and allowing to naturally generate new meaningful samples. While the presented architectures show quite satisfying results, more thorough hyper-parameter tuning and more complex and deeper architectures might enable even better performance of the model. However, this is beyond the scope of this work.

Appendix



(a) Training progress with Adam as optimiser



(b) Training progress with SGD as optimiser

Figure 7: Training progress

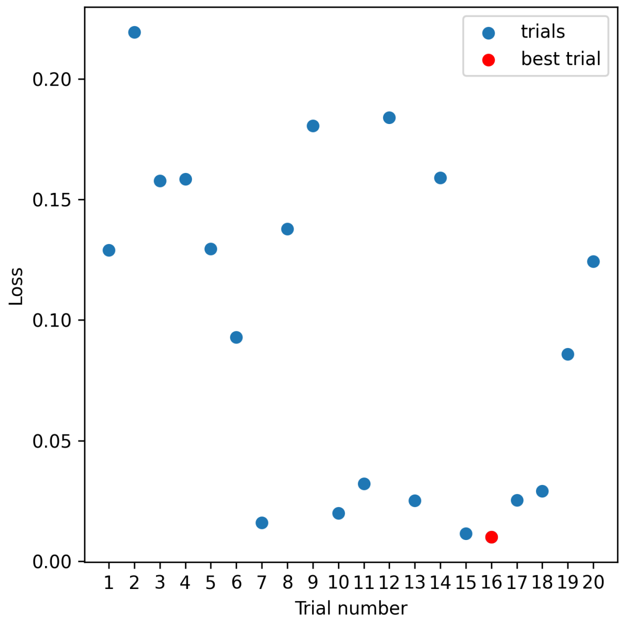


Figure 8: hyper-parameter tuning results

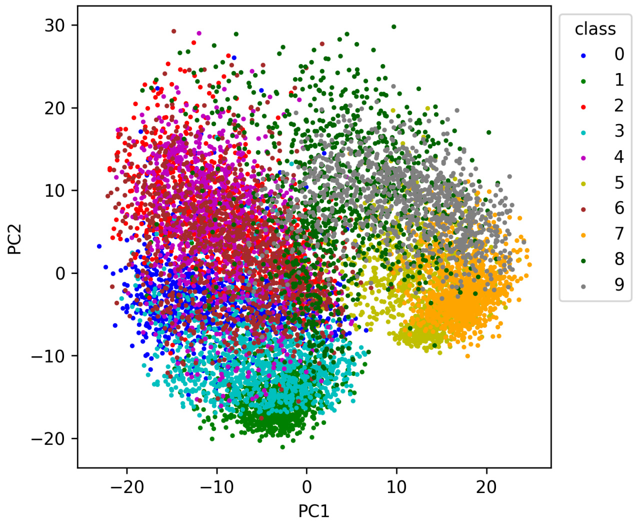
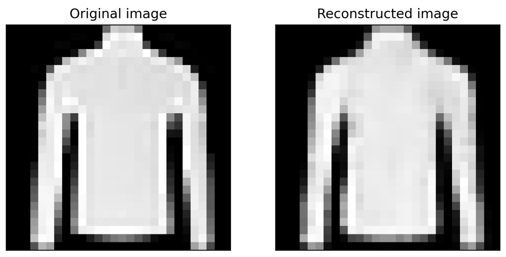
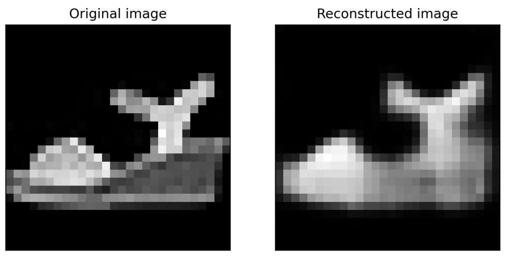


Figure 9: PCA on the latent space



(a) exemplary image reconstruction 2



(b) exemplary image reconstruction 3

Figure 10: Further exemplary image reconstructions

Table 7: VAE hyper-parameters

Parameter	Value
latent space dimension	44
Optimiser	Adam
Learning rate	0.0002
α	0.0018

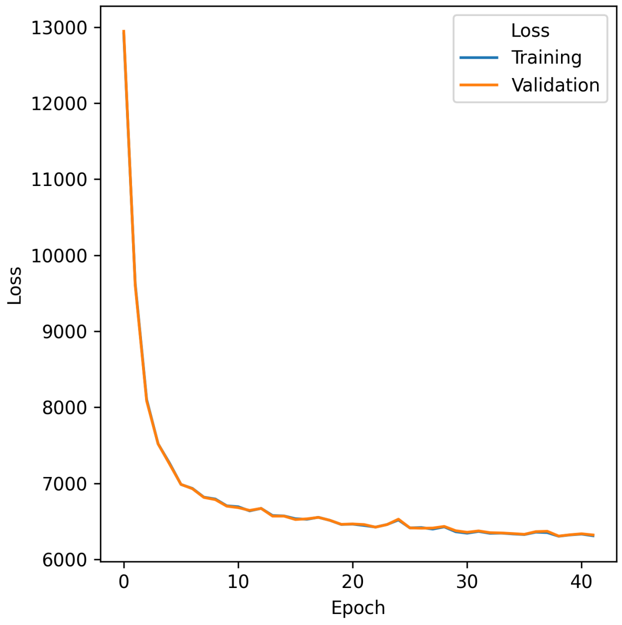


Figure 11: Loss of VAE