Name:

Amir Hossein

Surname:

Mohammad Amri

Student ID:

2045463

Introduction

This homework aims to implement a convolutional autoencoder (CAE) to solve unsupervised problems. The aim of an autoencoder is to learn a lower-dimensional representation (encoding) for higher-dimensional data, typically for dimensionality reduction, by training the network to capture the most important parts of the input image. Other applications of Autoencoders are *denoising* and *anomaly detection*.

In this homework first, we define a Convolutional Autoencoder and train it on some optimizers to compare their performance on the fashion-mnist data set.

Then, using the decoder part of the Autoencoder we calculate the latent space for the training set and visualize it in two-dimensional space. Similarly, we define a Variational Autoencoder (VAE), train it, then visualize the latent space for 2D and 3D space, and compare it with ordinary CAE.

Next, we explore the impact of principal component analysis (PCA) and t-NSE on autoencoder with higher latent space.

Finally, we opt to build a denoising AE (DAE) by feeding the network with noisy inputs and explore what is the performance of DAEs on noise.

Model

Before doing anything, by using a for loop the mean and variance of the fashion-mnist data set are calculated. These two numbers are used for normalizing the dataset which results in faster convergence compared to the unnormalized dataset.

We have defined two different classes for our autoencoder. In both classes, the encoder and decoder parts are capsulated in the same class. The first class (AE3class) has 3 convolutional layers and the second one is of four convolutional layers. Therefore, we can train both networks and compare the results for both networks. Since the input image size is small using filters with sizes three or four will not make much difference so the size of filters is constant (3) but the *number of filters* in each layer is chosen for hyperparameter. Another hyperparameter that is important is the *learning rate*. Also, the number of neurons for the fully connected layer is chosen as a hyperparameter and is tuned during optimization. The latent space size is fixed to two. It is obvious that the higher latent space is the lower the reconstruction loss we will achieve.

For tuning the hyperparameters we use the *Optuna* library. For sake of using Optuna, we should define a function and then pass it to the optuna to optimize it. This function in the code is named *obj*. In this function number of epochs on which the network is trained is set to 15. After training the network on the training set the error of the network on the validation set is computed and returned as the output of the *obj* function. The best way of returning an error for the network is using a k-fold cross-validation error and optimizing the network parameters using k-fold but because using this error takes too much time I simply used the validation error and after tuning the hyperparameters I calculated the k-fold cross-validation to report the error of the network.

For code simplicity, the training process has been divided into some parts and for each part, a function has been defined.

train_epoch() receives the model and training data loader, optimizer, and loss function and performs training the network, and then returns the training error of every batch in a list object.

test_epoch() receives model, validation data set, and loss function and then returns the validation error of every batch in a list object.

PlotReconstructed() simply plots the original and reconstructed image. Note that the plot function can be disabled by setting *plotEachEpoch=True* in the *trainingLoop()* function.

trainingLoop() by leveraging *train_epoch()* trains the network for a number of epochs and calculates validation loss for every epoch using *test_epoch()* and finally calculates the trained network validation error.

In the k-fold cross validation section by using the sklearn built-in function we divide the training sets into 5 folds and use four folds for training the network and one fold for calculating validation error and repeat this process for every five folds and finally report the mean of all five validation losses as the final loss.

Latent space

The AE class returns two outputs when is fed with input. The first one is the reconstructed image and the second one is the output on the encoder part of AE. To visualize the latent space in a for loop we fed the network with all test samples, discard the reconstructed image and just save the encoder's output and its corresponding label in a Dataframe. Then using the library plotly the latent space has been plotted.

Fine-tunning of Autoencoder

After training the Autoencoder we can use it for classification. The encoder part in order to encode the input into low dimensional output has already extracted the features of images so if we use the weights of the trained encoder part of Autoencoder as initial weights during training the network for the classification task it might be faster than initializing the network weights with random values. For this task first, the optimized autoencoder is loaded and a new class named *finetune* is defined. In this class, the encoder part of the Autoencoder and linear layer part of the loaded model is used, and also another linear layer with 10 neurons is added to them. The new linear layer with 10 neurons is needed to do the classification task.

Variational autoencoder (VAE)

In the Variational autoencoder, the input is mapped to a latent space with Gaussian distribution with mean zero and variance of one. For achieving this goal the loss function that is used alongside reconstruction loss, has KL divergence between the encoder's and decoder's distribution. For implementing VAE instead of one linear layer we have two linear layers one represents the mean (μ) and another one represents the variance (σ) of the latent space distribution. And we sample from this distribution using the following formula:

$$z = \mu + \sigma \cdot \varepsilon$$

Where ε is a random sample from normal Gaussian distribution: $\varepsilon \sim N(0,1)$

In the code, VAE has been defined in a separate Jupyter notebook. The *distribution* method in the *VAE* class computes z. the function named *KL_Loss()* calculates the KL divergence loss.

Results

Comparing optimizers

First, we are going to compare the training and validation errors for different optimizers. Adam, Adamx, AdamW, and Adagrad are the optimizers that we compared. To compare we defined a class with three convolution layers and latent space with two neurons. The architecture that we used is shown in Figure 1. The *learning rate* for this part was set to 0.005 and for each optimizer, the network was trained for 50 epochs. The running time for optimizers is shown in Table 1 and the training and validation errors are shown in Figure 2 and Figure 3. For better visualization, we also plotted the running average of errors. As it can be seen the performance of Adam and AdamW are the same and slightly better than Adamx and Adagrad. From table one we see that there is no big difference in running time for optimizers.

Table 1. Running time for different optimizers using Autoencoder with 3 convolutional layers and latent space size 2

Optimizer	Adam	Adamx	AdamW	Adagrad
Running time	20.87	18.30	19.53	19.34

Optimizing hyperparameters

As previously said, the learning rate, number of filters for each convolutional layer, two different optimizers, number of neurons in a single fully connected layer before the latent layer, and network with three or four Conv layers are the parameters that we tuned in this task. The range of each hyperparameter that we chose for tuning is shown in Table 2. The number of epochs that each network is trained is set to 20. Total tries that optuna opts for finding the best hyperparameters is set to 100.

Table 2-a. Hyperparameters for the network with 3 Conv layers

Hyperparameter	First layer filters	Second layer filters	Third layer filters	Learning rate	Latent layer neurons	Optimizers
Range	[8,16]	[16,32]	[32,64]	[0.0001,0.008]	[64,128]	[“Adam”,”RAdam”]

Table 2-b. Hyperparameters for the network with 4 Conv layers

Hyperparameter	First layer filters	Second layer filters	Third layer filters	Fourth layer filters	Learning rate	Latent layer neurons	Optimizers
Range	[8,20]	[16,32]	[32,64]	[32.64]	[0.0001,0.008]	[64,128]	[“Adam”,”RAdam”]

The optimization process is done on both network architectures and some of the optuna results can be shown in Figures 4 and 5. The optimized values achieved by optuna are as follows:

Table 3. Parameters found by optuna

Hyperparameter	First layer filters	Second layer filters	Third layer filters	Learning rate	Latent layer neurons	Optimizers
Range	12	24	48	0.004748	128	Adam

Then the **k-fold cross-validation** of the network is calculated and it is **0.0345**. Next the optimized network is trained for 30 epochs.

Final *test error* of this trained network is **0.026.**

Classification fashion-Mnist using fine-tuned model

In this part, a new model consisting of the encoder part trained Autoencoder is defined and trained on the mnist to classify the data. After training the network only for two epochs we reached the *validation accuracy* of **0.897** and the training time was **1.55** minutes (using GPU) which is faster than using a network with random weights.

Latent space

After training the Autoencoder we want to see how the decoder part of the Autoencoder maps the input in latent space. Since the size of the output is two we can plot the output and see how the inputs are separated in the latent space. The latent space can be achieved by plotting the decoder's output while the input is a training set or test set. The latent space achieved by feeding the decoder by the test set is shown in Figure 6.

By using the generator part of the AE we can generate new samples. For this purpose, the class *generator* has been defined. As it can be seen from the Figure 8 coordinate (-3.07,-2.1) represents a bag. So by choosing a coordinate near this coordinate we can generate an image of a bag that is new and is not in the training dataset. In Figure 7 the output of the generator for the coordinate of (-3,-2) is shown.

PCA and t-SNE

PCA is a technique for reducing the dimensionality of datasets, increasing interpretability while minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance.

t-SNE is a statistical method for visualizing high-dimensional data by giving each data point a location in a two or three-dimensional map. It is based on Stochastic Neighbor Embedding.

In this part, we train an AE with latent space of size 10 and then decrease the dimension with these two methods to output, but before that just perform the output of these methods on two-dimensional latent space shown in Figure 6. This can be seen in Figure 8 and Figure 9.

The Figure shows a slightly better performance applying PCA on the other hand t-SNE shows a much better performance.

In the next step, the network with the best parameters achieved by optuna and with latent space 10 has been trained for 10 epochs. After training we reached a test loss of **0.015** which is a great improvement compared to AE with latent space of size two. One sample of an original image and its reconstruction is shown in Figure 10.

The training error curve and validation error curve are shown in Figure 11. In Figure 12 the output of PCA on the 10D latent space is shown. As can be seen, compared to the output of PCA on the 2D latent it divided data better but still, many points overlap each other. On the other hand, the output of t_SNE on the 10D latent space, shown in Figure 13, shows much better performance in clustering the data.

Variational AE (VAE)

In this part, we defined the VAE with parameters found by Optuna during optimization of normal AE and then trained it for 30 epochs. The training error and validation error curves are shown in Figure 14 and Figure 15. Using the encoder part of VAE the latent space has been plotted in Figure 16. In VAE we aim to train the network in such a way that the latent space does not have sudden space and all the samples are encoded in an approximately fixed range (a normal distribution with mean zero and variance one), which have been achieved by using VAE and we can see from the latent space that all samples are centered around zero.

The same process has been done for a VAE with a latent space of size 3 and the latent space is shown in Figure 17.

Figure 18 shows a new sample generated by VAE.

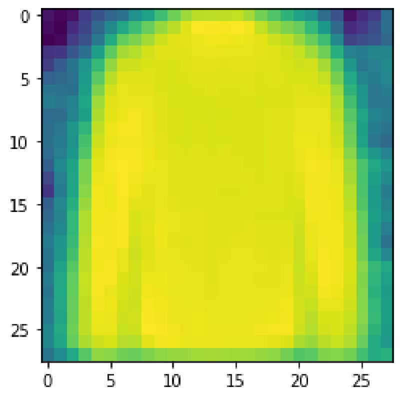


Figure 18. A sample of the reconstructed image of VAE with latent space of 3

Denoising AE (DAE)

In DAEs, during training the network instead of feeding the network with the original dataset, we add Gaussian noise to the original input data and compute the loss by comparing the output with the original data. In this part, we added noise with N(0,0.3) distribution. The network was defined with latent space of size three and trained on the training set and after 30 epochs it reached a test error of 0.0213 and the performance of the trained DAE on test data is shown for ten noisy inputs. The first row is the original image of the test set that the network had not seen them during training, the second row is the noisy image fed to the network, and the third row is the network's output. As it can see from Figure 19 the output is quite satisfying and robust to the noise.

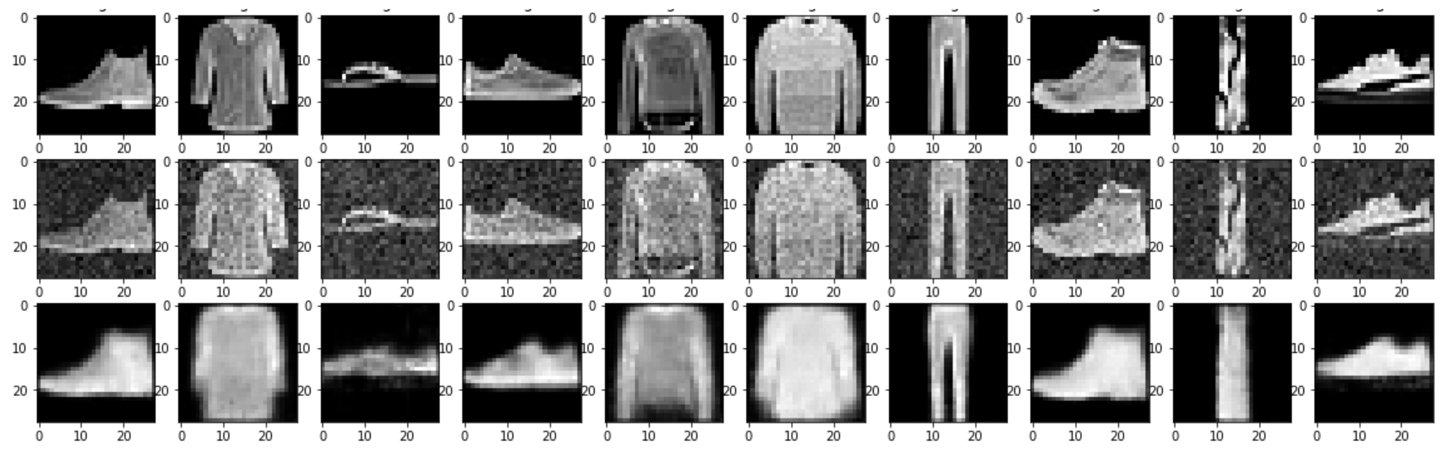
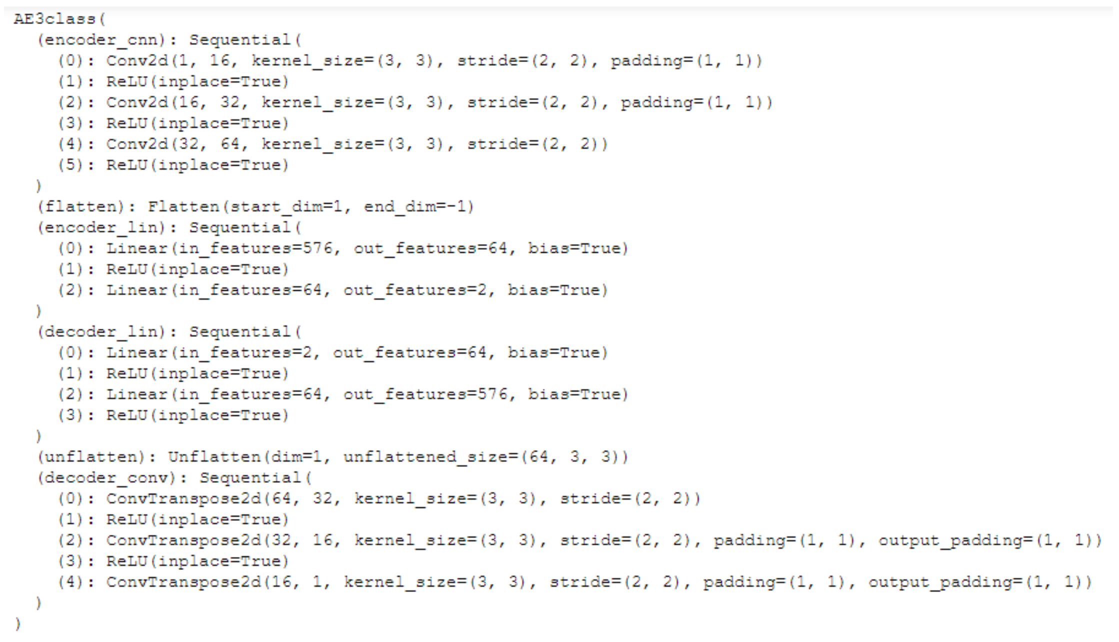


Figure 19. The output of DAE on noisy input of test dataset

Appendix



```
AE3class(
  (encoder_cnn): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2))
    (5): ReLU(inplace=True)
  )
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (encoder_lin): Sequential(
    (0): Linear(in_features=576, out_features=64, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=64, out_features=2, bias=True)
  )
  (decoder_lin): Sequential(
    (0): Linear(in_features=2, out_features=64, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=64, out_features=576, bias=True)
    (3): ReLU(inplace=True)
  )
  (unflatten): Unflatten(dim=1, unflattened_size=(64, 3, 3))
  (decoder_conv): Sequential(
    (0): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2))
    (1): ReLU(inplace=True)
    (2): ConvTranspose2d(32, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): ConvTranspose2d(16, 1, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
  )
)
```

Figure 1. The architecture that is optimized

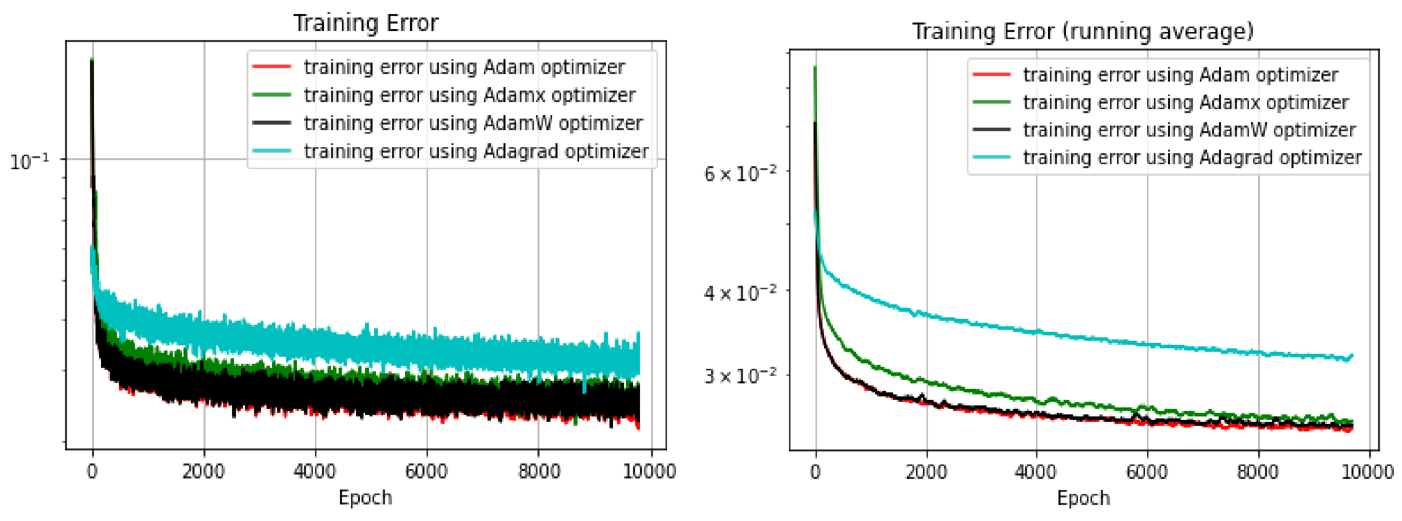


Figure 2. Training error for different optimizers

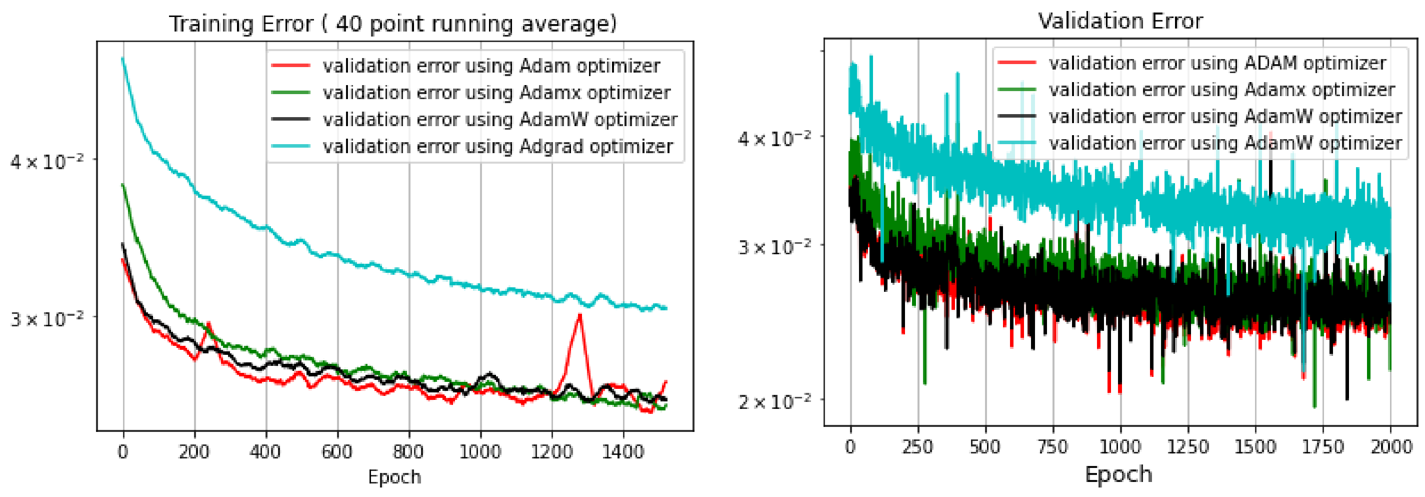
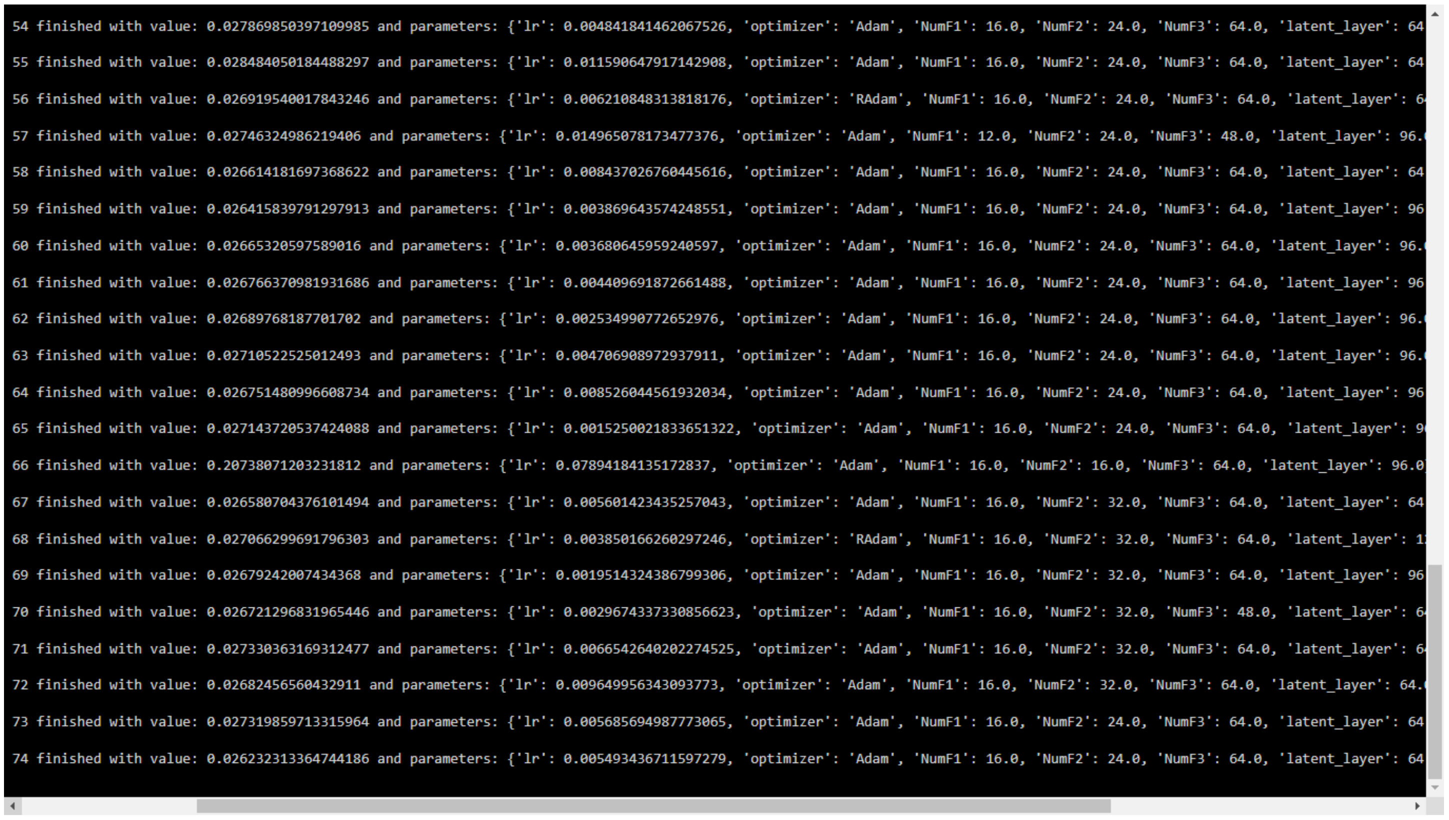
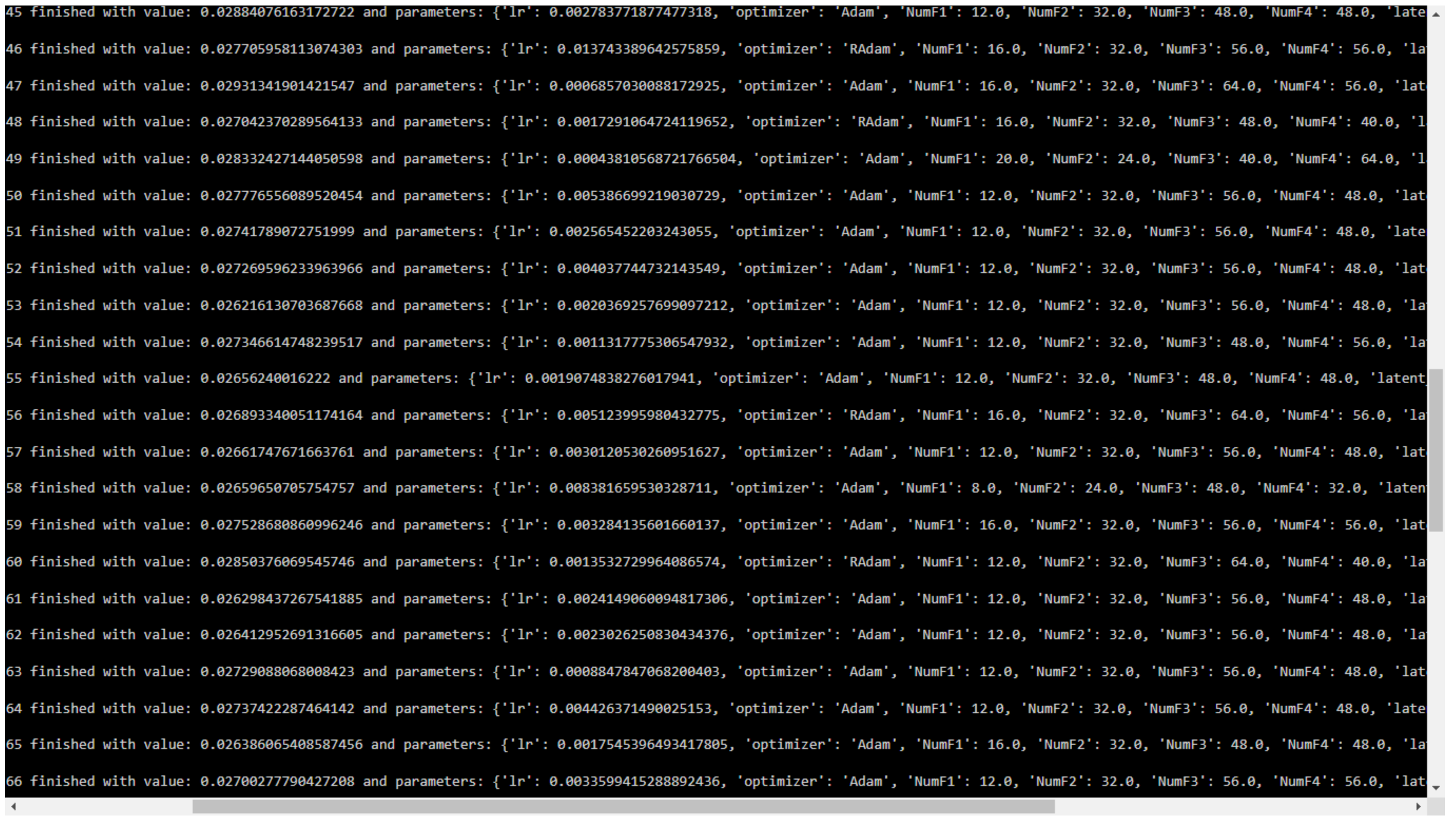


Figure 3. Validation error for different optimizers



```
54 finished with value: 0.027869850397109985 and parameters: {'lr': 0.004841841462067526, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 64

55 finished with value: 0.028484050184488297 and parameters: {'lr': 0.011590647917142908, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 64

56 finished with value: 0.026919540017843246 and parameters: {'lr': 0.006210848313818176, 'optimizer': 'RAdam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 6

57 finished with value: 0.02746324986219406 and parameters: {'lr': 0.014965078173477376, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 24.0, 'NumF3': 48.0, 'latent_layer': 96.

58 finished with value: 0.026614181697368622 and parameters: {'lr': 0.008437026760445616, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 64

59 finished with value: 0.02641583979129791 3 and parameters: {'lr': 0.003869643574248551, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 96

60 finished with value: 0.0266532059758901 6 and parameters: {'lr': 0.003680645959240597, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 96.

61 finished with value: 0.026766370981931686 and parameters: {'lr': 0.004409691872661488, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 96

62 finished with value: 0.02689768187701702 and parameters: {'lr': 0.002534990772652976, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 96.

63 finished with value: 0.02710522525012493 and parameters: {'lr': 0.004706908972937911, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 96.

64 finished with value: 0.026751480996608734 and parameters: {'lr': 0.008526044561932034, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 96

65 finished with value: 0.027143720537424088 and parameters: {'lr': 0.0015250021833651322, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 9

66 finished with value: 0.20738071203231812 and parameters: {'lr': 0.07894184135172837, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 16.0, 'NumF3': 64.0, 'latent_layer': 96.0

67 finished with value: 0.026580704376101494 and parameters: {'lr': 0.005601423435257043, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 64.0, 'latent_layer': 64

68 finished with value: 0.027066299691796303 and parameters: {'lr': 0.003850166260297246, 'optimizer': 'RAdam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 64.0, 'latent_layer': 1

69 finished with value: 0.02679242007434368 and parameters: {'lr': 0.0019514324386799306, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 64.0, 'latent_layer': 96

70 finished with value: 0.026721296831965446 and parameters: {'lr': 0.0029674337330856623, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 48.0, 'latent_layer': 6

71 finished with value: 0.027330363169312477 and parameters: {'lr': 0.0066542640202274525, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 64.0, 'latent_layer': 6

72 finished with value: 0.02682456560432911 and parameters: {'lr': 0.009649956343093773, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 64.0, 'latent_layer': 64.

73 finished with value: 0.027319859713315964 and parameters: {'lr': 0.005685694987773065, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 64

74 finished with value: 0.026232313364744186 and parameters: {'lr': 0.005493436711597279, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 24.0, 'NumF3': 64.0, 'latent_layer': 64
```

Figure 4. Sample of the output of optuna for AE with three Conv layers



```
45 finished with value: 0.02884076163172722 and parameters: {'lr': 0.002783771877477318, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 48.0, 'NumF4': 48.0, 'late
46 finished with value: 0.027705958113074303 and parameters: {'lr': 0.013743389642575859, 'optimizer': 'RAdam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 56.0, 'la
47 finished with value: 0.02931341901421547 and parameters: {'lr': 0.0006857030088172925, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 64.0, 'NumF4': 56.0, 'lat
48 finished with value: 0.027042370289564133 and parameters: {'lr': 0.0017291064724119652, 'optimizer': 'RAdam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 48.0, 'NumF4': 40.0, 'l
49 finished with value: 0.028332427144050598 and parameters: {'lr': 0.00043810568721766504, 'optimizer': 'Adam', 'NumF1': 20.0, 'NumF2': 24.0, 'NumF3': 40.0, 'NumF4': 64.0, 'l
50 finished with value: 0.027776556089520454 and parameters: {'lr': 0.005386699219030729, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 48.0, 'lat
51 finished with value: 0.02741789072751999 and parameters: {'lr': 0.002565452203243055, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 48.0, 'late
52 finished with value: 0.027269596233963966 and parameters: {'lr': 0.004037744732143549, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 48.0, 'lat
53 finished with value: 0.026216130703687668 and parameters: {'lr': 0.0020369257699097212, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 48.0, 'la
54 finished with value: 0.027346614748239517 and parameters: {'lr': 0.0011317775306547932, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 48.0, 'NumF4': 56.0, 'la
55 finished with value: 0.02656240016222 and parameters: {'lr': 0.0019074838276017941, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 48.0, 'NumF4': 48.0, 'latent
56 finished with value: 0.026893340051174164 and parameters: {'lr': 0.005123995980432775, 'optimizer': 'RAdam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 64.0, 'NumF4': 56.0, 'la
57 finished with value: 0.02661747671663761 and parameters: {'lr': 0.0030120530260951627, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 48.0, 'lat
58 finished with value: 0.02659650705754757 and parameters: {'lr': 0.008381659530328711, 'optimizer': 'Adam', 'NumF1': 8.0, 'NumF2': 24.0, 'NumF3': 48.0, 'NumF4': 32.0, 'laten
59 finished with value: 0.027528680860996246 and parameters: {'lr': 0.003284135601660137, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 56.0, 'lat
60 finished with value: 0.02850376069545746 and parameters: {'lr': 0.0013532729964086574, 'optimizer': 'RAdam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 64.0, 'NumF4': 40.0, 'la
61 finished with value: 0.026298437267541885 and parameters: {'lr': 0.0024149060094817306, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 48.0, 'la
62 finished with value: 0.026412952691316605 and parameters: {'lr': 0.0023026250830434376, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 48.0, 'la
63 finished with value: 0.02729088068008423 and parameters: {'lr': 0.0008847847068200403, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 48.0, 'lat
64 finished with value: 0.02737422287464142 and parameters: {'lr': 0.004426371490025153, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 48.0, 'late
65 finished with value: 0.026386065408587456 and parameters: {'lr': 0.0017545396493417805, 'optimizer': 'Adam', 'NumF1': 16.0, 'NumF2': 32.0, 'NumF3': 48.0, 'NumF4': 48.0, 'la
66 finished with value: 0.02700277790427208 and parameters: {'lr': 0.0033599415288892436, 'optimizer': 'Adam', 'NumF1': 12.0, 'NumF2': 32.0, 'NumF3': 56.0, 'NumF4': 56.0, 'lat
```

Figure 5. Sample of the output of optuna for AE with Four Conv layers

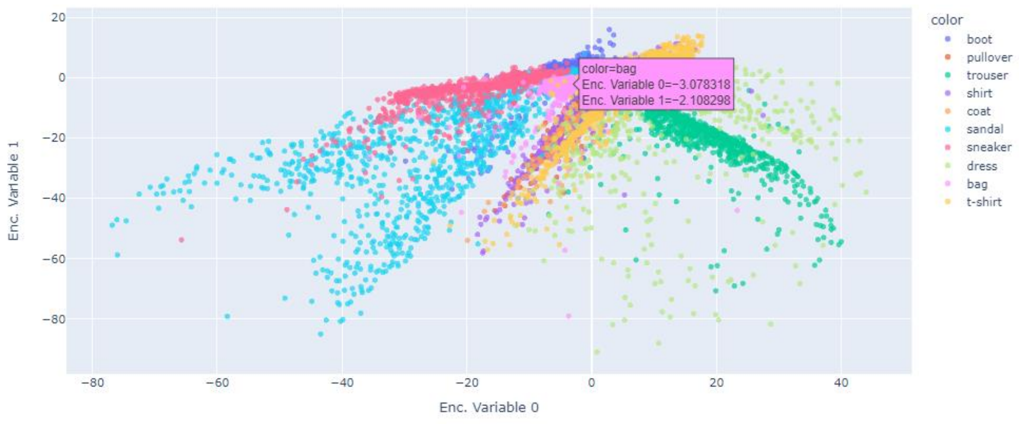


Figure 6. Latent space of Autoencoder

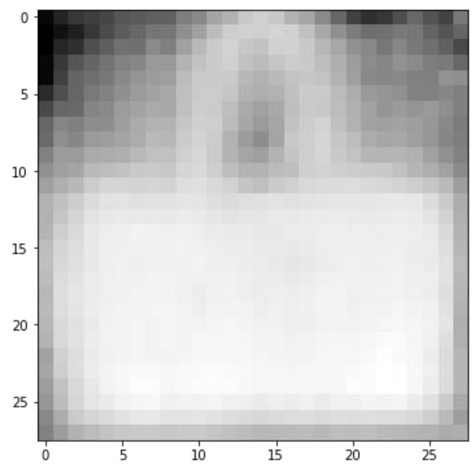


Figure 7. The output of generator for coordinate of (-3, 2)

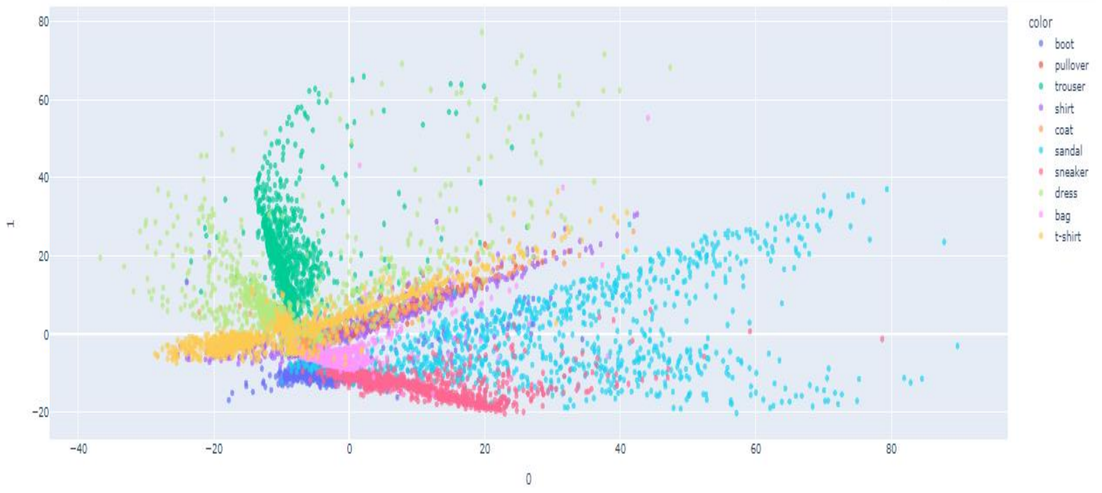


Figure 8. The output of PCA for 2D latent space

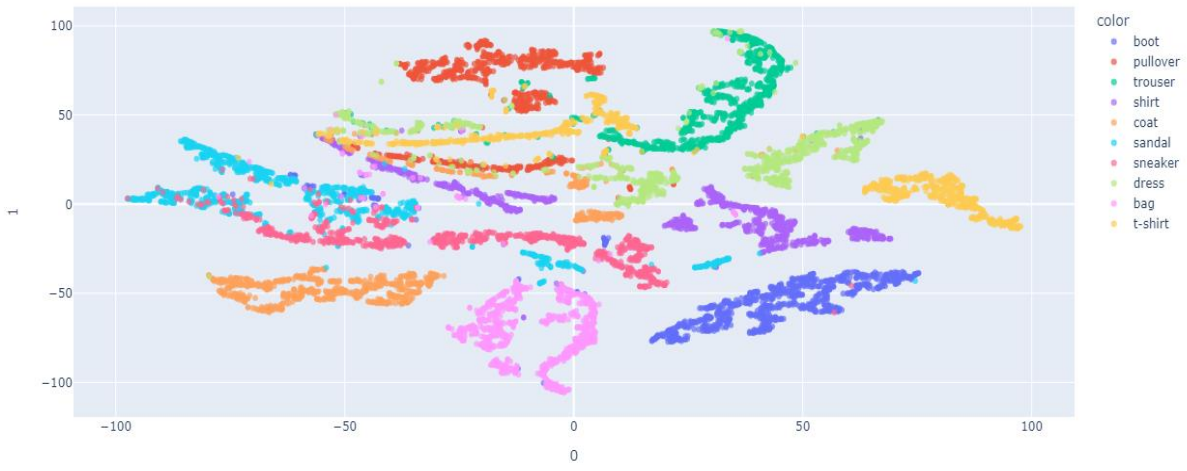


Figure 9. The output of t-SNE for 2D latent space

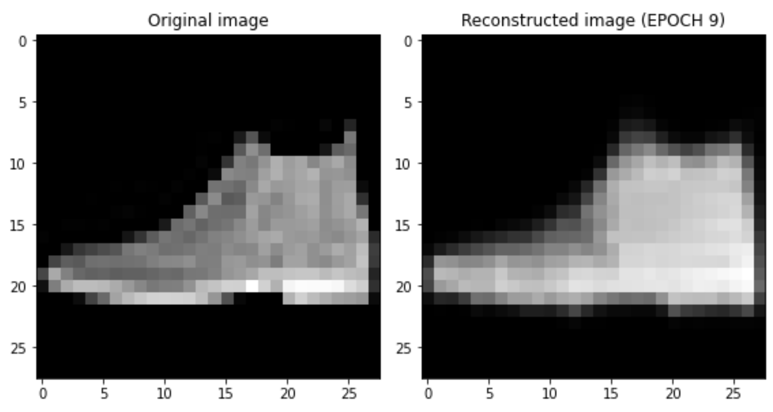


Figure 10. The input image and its reconstruction for VAE with latent space of size 10

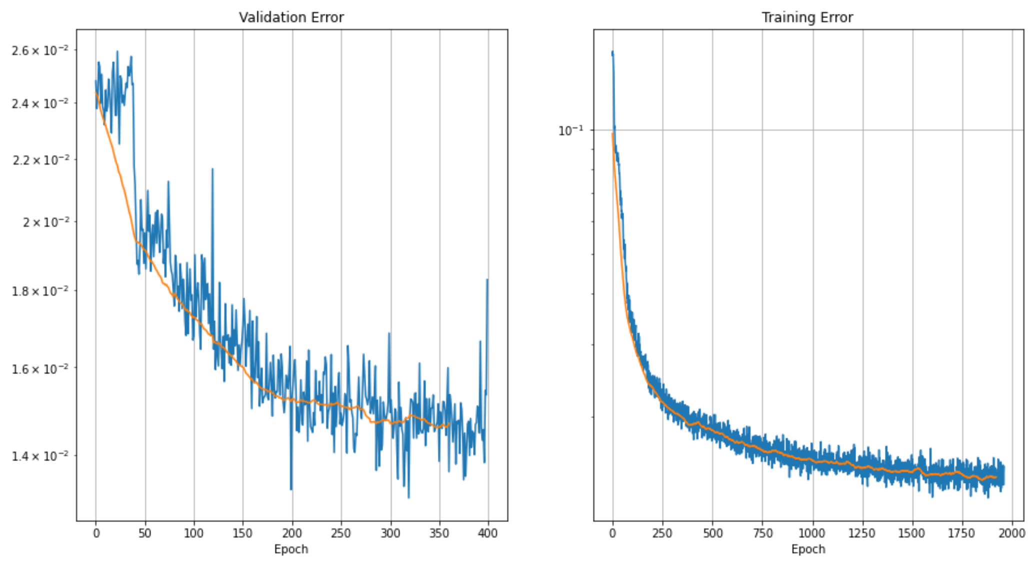


Figure 11. Training error and validation error of AE with latent space of size 10

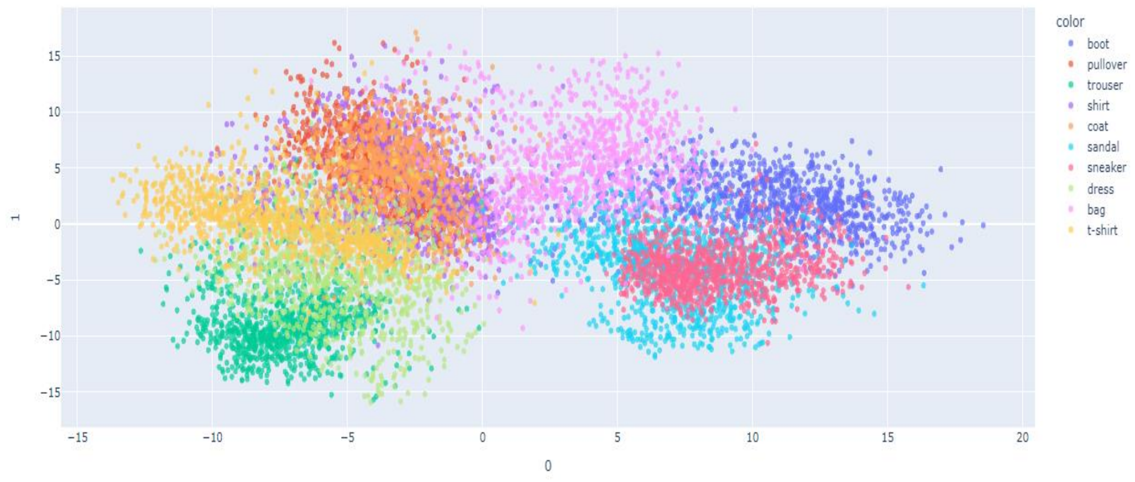


Figure 12. The output of PCA on latent space of size 10

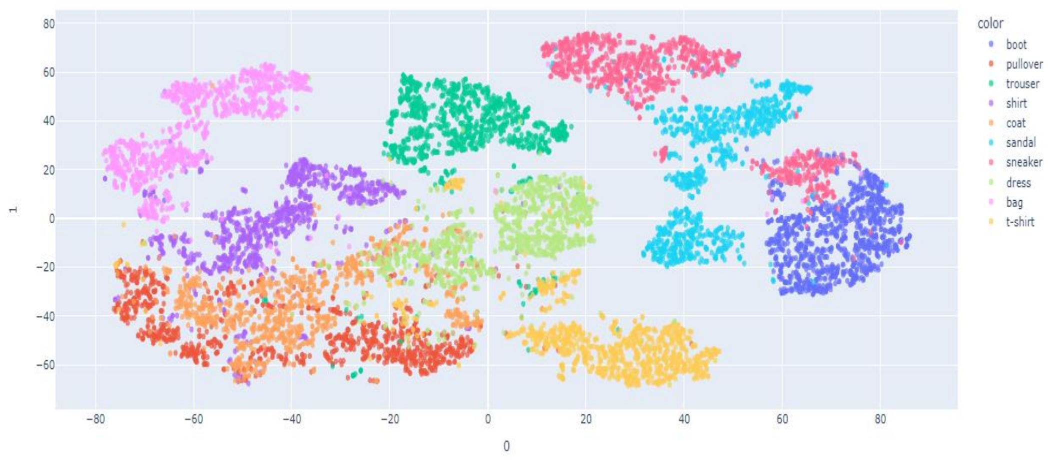


Figure 13. The output of t-SNE on latent space of size 10

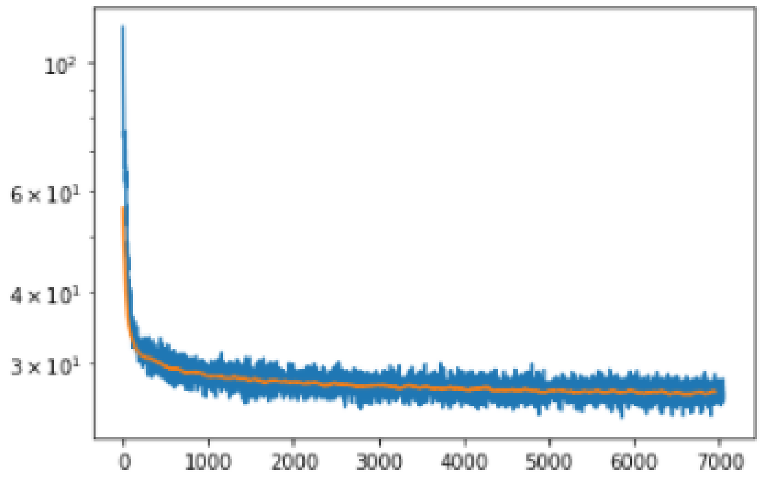


Figure 14. Training error of VAE with latent space of size 2

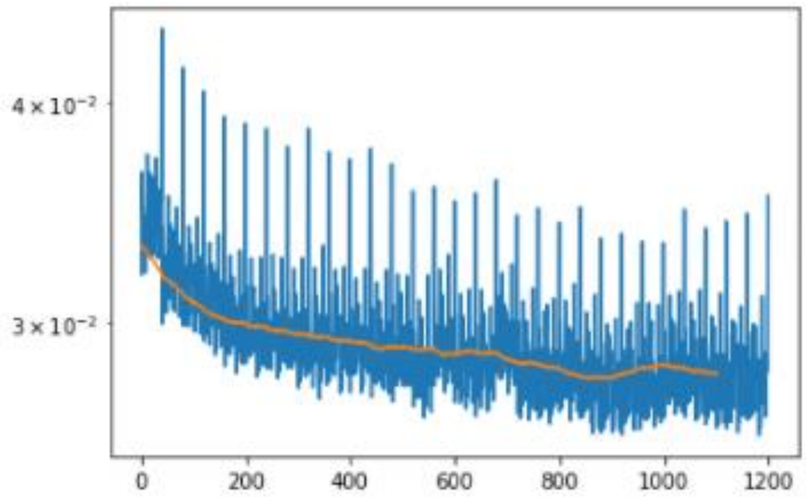


Figure 15. Test error of VAE with latent space of size 2

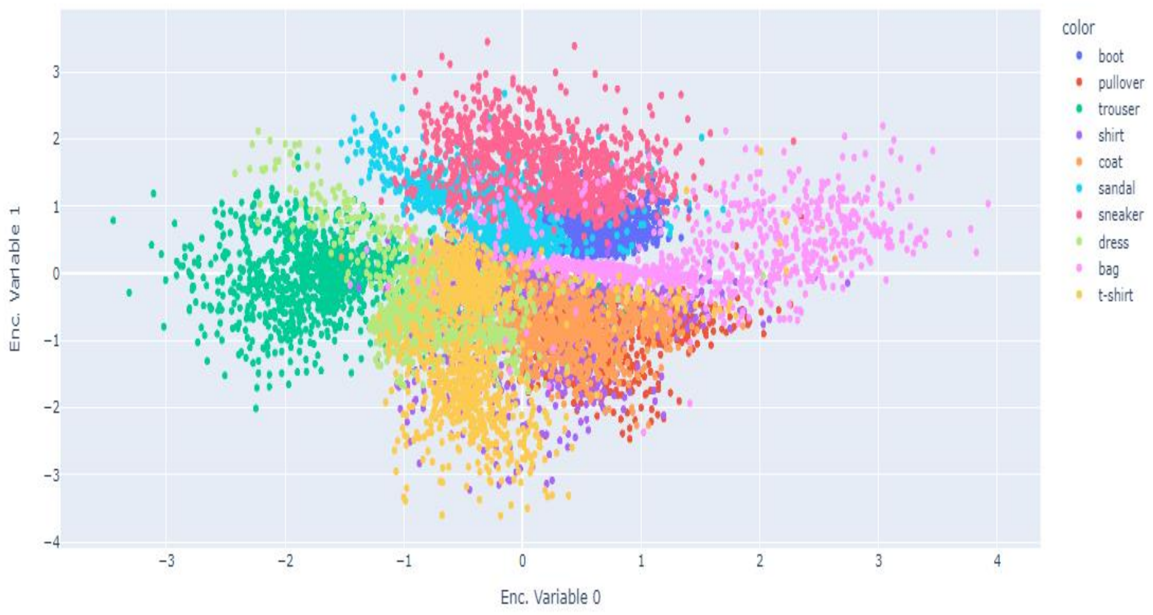


Figure 16. VAE latent space of size two

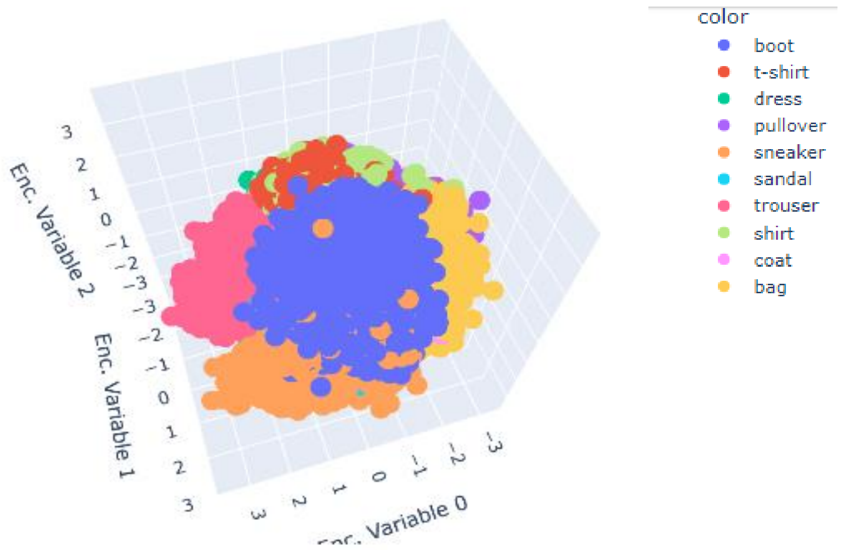


Figure 17. Latent space of a VAE with latent space of size 3