UNIVERSITÀ DEGLI STUDI DI PADOVA

Homework #2
Unupervised Deep Learning

Monaco Saverio - 2012264

Neural Networks and Deep Learning - Professor: A. Testolin

***Abstract*—As for the second homework, the main models of Unupervised Deep Learning are investigated on the FashionMNIST dataset.**
Firstly, an Autoencoder has been developed, and its hyperparameters were properly optimized.
Then the best model has been then fine-tuned aiming for a working Classifier.
As for the last model, a Generative Adversarial Network has been implemented for the objective of generating completely new samples.

DATA

FashionMNIST is a dataset of low-resolution (28×28) grayscale images of fashion products from 10 categories (such as "Trousers", "Sandal", "Bag"). In the dataset there are 60,000 images for the training-set and 10,000 for the test-set.

Fig. 1: Samples of FashionMNIST

This set was used throughout all tasks of the second homework and no data augmentation was performed on it.

I. CONVOLUTIONAL AUTO-ENCODERS

The task of an Auto-Encoder is to reconstruct its input image, thus it is not a Classification, nor a Regression, nor a Generative model.
It consist in a Encoder, where the input images gets convoluted in a smaller space (called *latent space*) and a Decoder, where the 'compressed' images get de-convoluted to their original size.
The power of an Autoencoder lies in the 'bottleneck' shape (see Fig. 2). Through learning the Neural Network learns the most efficient weights and biases to compress and retrieve the input images as good as possible performing a dimensionality reduction.

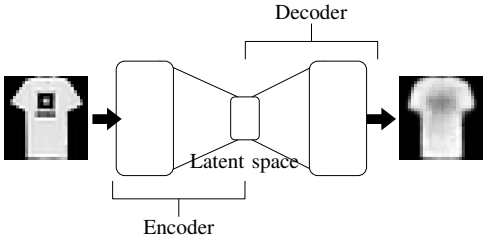

Fig. 2: General overview of a Convolutional Auto-Encoder

Convolutional Auto-encoder are used mainly for *anomaly detection*, and *denoising*.

A. *Methods*

The Autoencoder was implemented through a Pyhton class using *Pytorch* for building the layers. The only free parameter for the initialization is the size of the *latent space* dimension, while the activation was kept fixed to ReLU.

Encoder:

- First Convolutional layer: in_channels=1, out_channels=8, kernel_size=3;
- Second Convolutional layer: in_channels=8, out_channels=16, kernel_size=3;
- Third Convolutional layer: in_channels=16, out_channels=32, kernel_size=3;
- First Linear layer: in_features=3*3*32, out_features=64;
- Second Linear layer: in_features=64, out_features=latent_dim;

Decoder: is a specular Network of the Encoder built with the use of `nn.ConvTranspose2d`

for the de-convolution.
The Mean Square Error Loss was adoperated to evaluate the models, hence the difference squared of each pixel of the real image and its reconstructed was computed.

B. Results

Firstly, a 2-Dimensional latent space Autoencoder was trained for inspecting the points of the testing images in the Latent Space (that can be plotted as a matter of fact as a 2D scatter plot):

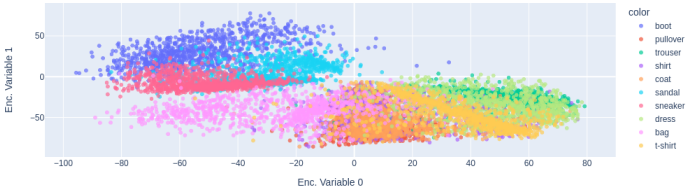

Fig. 3: Latent space of Autoencoder

Each dot represents the coordinate of a single image of the test dataset in the 2-D latent space, and its color represents its label (that was never given to the Network).
Without ever getting the information of the label to the network, it grouped each dot according to them, in addition, dots of different class that are close in the latent space, can be considered semantically close, so it can be defined a metric in the *latent space* on how much clothes look alike. To demonstrate visually the last statement, dots along a line that connects the blue patch and the red patch were decoded (namely from the "Boot" group to the "Sneaker" group):

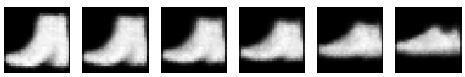

Fig. 4: Gradual transformation of a Boot to a Sneaker

After the inspeciton of the latent space of this simple model, a Grid Search was performed using Optuna Python library for hyperparameters optimization. The dimension of the encoded space this time was kept fixed to 16 (instead of 2), while the initial learning rate, regularization term of the optimizer, and the optimizer itself were tunable:

- Optimizer: Adam, SGD;
- Initial Learning rate: from 1e-4 to 1e-3;
- Regularization term: from 1e-5 to 1e-3;

Best model:

- Optimizer: Adam;
- Learning Rate: 0.00099
- Regularization: 3.5237e-5

Epochs: 60
MSE on Test Set: 0.017237

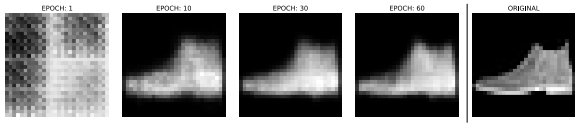

Fig. 5: Performance of Autoencoder during its learning process

The Model seems to properly reconstruct the shape of the input image, however, it does not seems to learn during its training how bright a certain pixel must be.

II. Fine-tuning an Autoencoder

With some modifications to the model, autoencoders can perform the classification task. In order to achieve this, we can place on top of the encoder a fully connected neural network and completely remove the decoder part. If the layers of the encoder are frozen, the task of classificating the images will actually be a task of classificating the points in the encoded space to their labels.

A. Methods

The structure of the fine-tuned autoencoder is:
Encoder: same as in the previous section, weights and biases frozen.
Fully Connected Neural Network:

- First Linear Layer: input_features=latent_dim, output_features=256;
- Second Linear Layer: input_features=256, output_features=256;
- Third Linear Layer: input_features=256, output_features=128;
- Last Linear Layer: input_features=128, output_features=10;

Between each layer ReLU activation function was chosen and as last activation LogSoftmax was applied.

B. *Results*

Fine-Tuned Autoencoder:

- Loss Function: Cross Entropy Loss
- Optimizer: Adam;
- Learning Rate: 2e-4
- Regularization: 1e-5

Epochs: 100
Accuracy on Test Set: 83.49 %

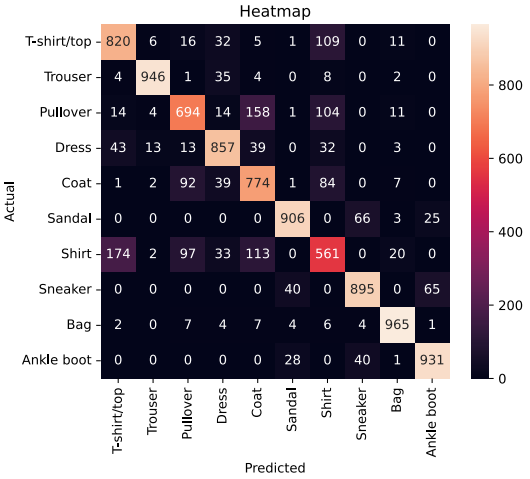

Fig. 6: Confusion matrix of the model on the test dataset

Compared to the dedicated Convolutional Network from Homework 1, the accuracy is slightly lower. However, with the hypothesis of having an already well trained autoencoder, transfer learning results in faster learning times.

III. Generative Adversarial Networks

The last model chosen for the generative task is the Generative Adversarial Network (GAN). A GAN is in reality two Neural Networks, the Generator that tries to imitate the input dataset, and the Discriminator that tries to tell the difference between the fake images from the generator and the real images of the dataset.
At the beginning of learning, the discriminator always does a better job in detecting what images are fake, since the generator will output images close to random noise, but eventually the generator picks up until ideally the discriminator cannot distinguish a fake image from a real one.

A. *Methods*

The GAN was implemented through a single Python class, containing both models for discriminating and generating:
Discriminator:

- First Linear Layer:
  input_features=28×28,
  output_features=hidden_size;
- Second Linear Layer:
  input_features=hidden_size,
  output_features=hidden_size;
- Third Linear Layer:
  input_features=hidden_size,
  output_features=1;

For the Discriminator network, LeakyReLU activation was chosen between eachlayer and Sigmoid for the last.
Generator:

- First Linear Layer:
  input_features=latent_size,
  output_features=hidden_size;
- Second Linear Layer:
  input_features=hidden_size,
  output_features=hidden_size;
- Third Linear Layer:
  input_features=hidden_size,
  output_features=28×28;

For the Generator network, between each layer ReLU was chosen as activation function and Tanh as last activation.

The free parameters for the initialization of the GAN are `hidden_size` that determines the size of the hidden layers on both networks and `latent_size` that is the size of the random input image for the generator.

B. *Results*

Due to the extremely long periods of time for the training of a GAN, no Grid Search was adopted.

Parameters of the GAN:

- latent_size = 100
- hidden_size = 784
- Epochs = 300
- Learning Rate = 1e-4

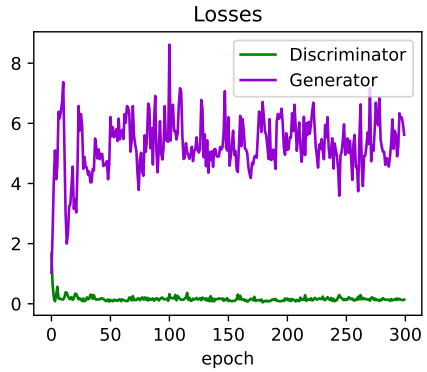

Fig. 7: Losses of Discriminator and Generator Networks

During training, samples of generated images were stored:

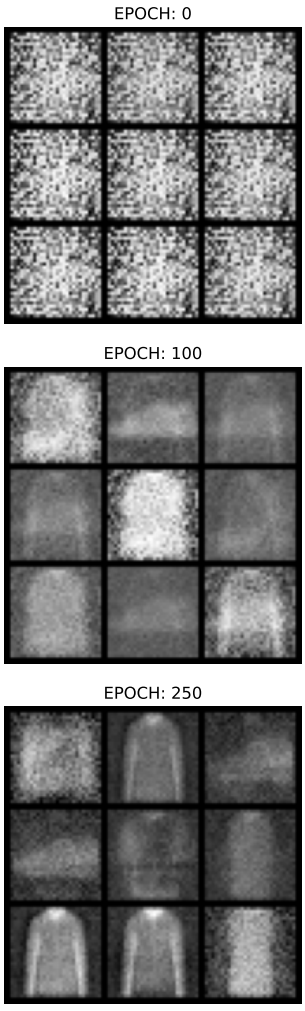

Fig. 8: Fake generated samples

In some images it is possible to see that the generator tries to output multiple clothes at once although it becomes less common with further training.

The generated data appear *noisy*, coincidentally Autoencoders can denoise images, the following images were generated through the GAN and then fowarded through the Autoencoder found using Optuna:

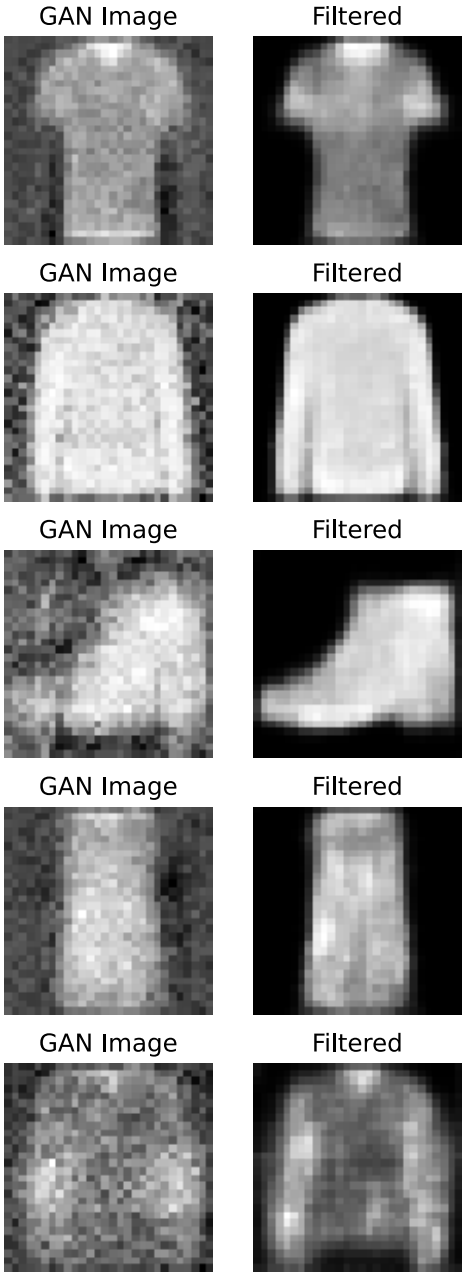

Fig. 9: Generated and filtered images

Although being a questionable choice, the combination GAN+Autoencoder generates new samples that are quite satisfying and for some of them, may fool a human aswell.

IV. Appendix

A. Hyperparameter optimization using Optuna

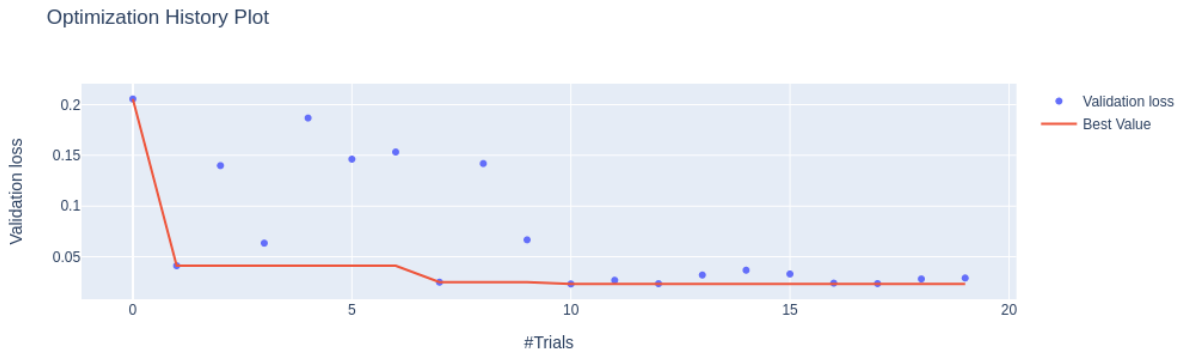

Fig. 10: Best model through trials

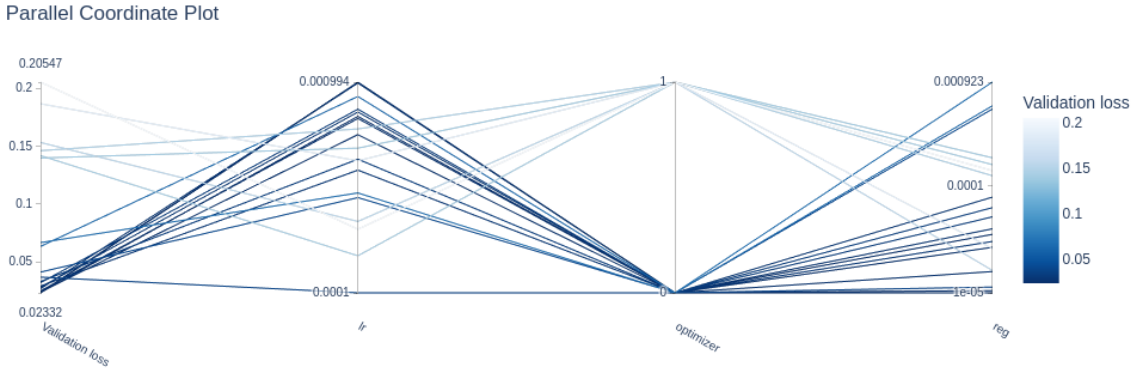

Fig. 11: Overview of parameter combinations and their Validation loss

B. Latent space of Trained and Untrained Network

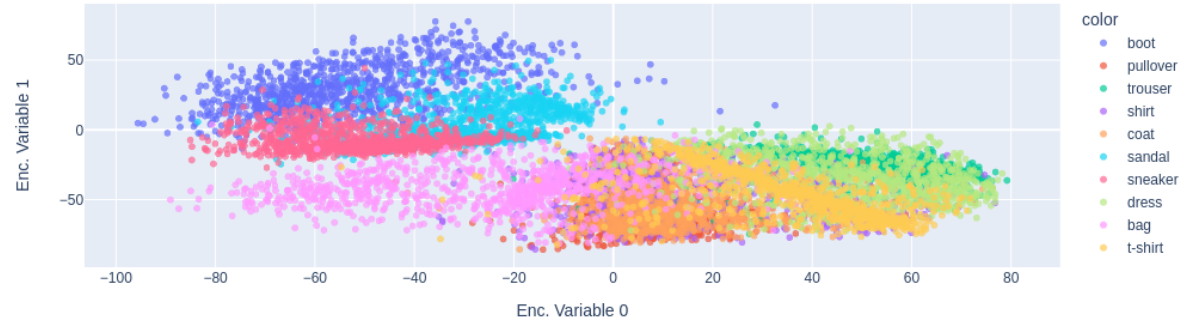

Fig. 12: Latent space of 2-D Trained Autoencoder

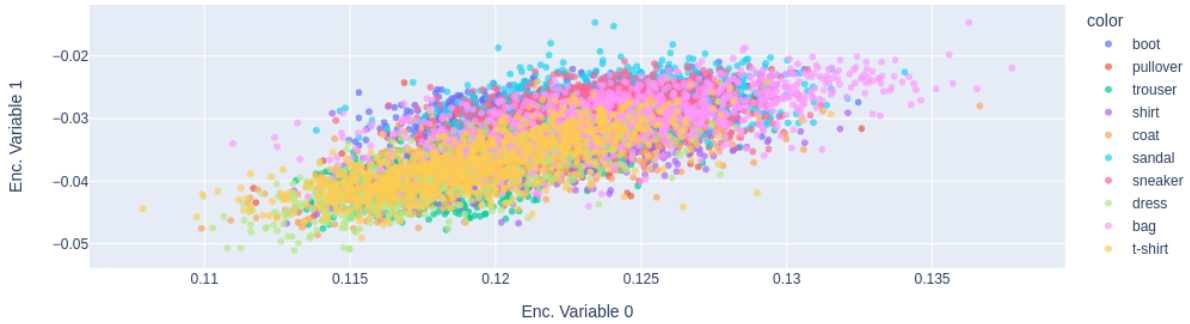

Fig. 13: Latent space of 2-D **Un**trained Autoencoder

Fig.12 and Fig.13 shows respectively the encoded vectors of the images from the test set in latent space of the Trained and the Untrained Autoencoder with latent space dimension equal to 2. The colors of the dots were added separately and represent the class each point belongs to.
The dots in the encoded space of the untrained network are focused randomly in a small neighborhood around zero, while for he trained net, they are more sparsed and each classes are almost separated one another.

C. PCA and t-SNE on Latent Space

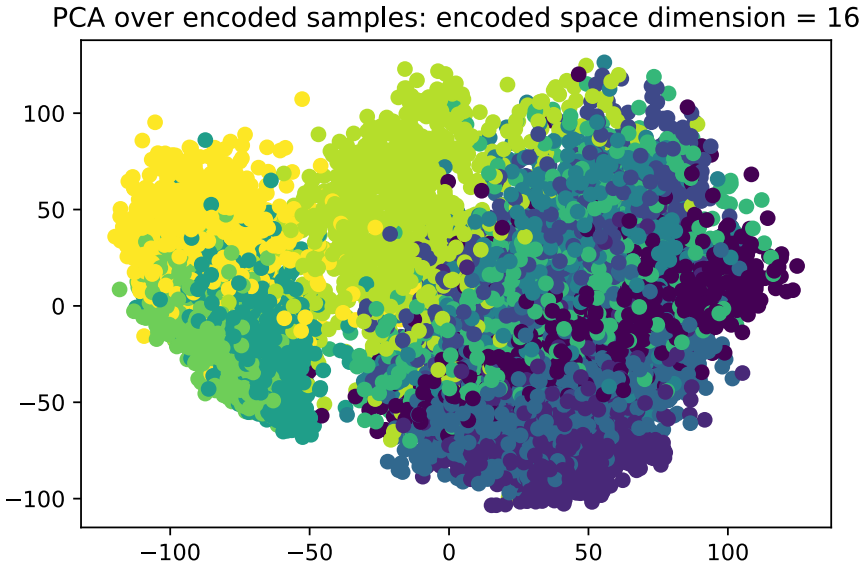

Fig. 14: PCA over 16-dimensional latent space autoencoder

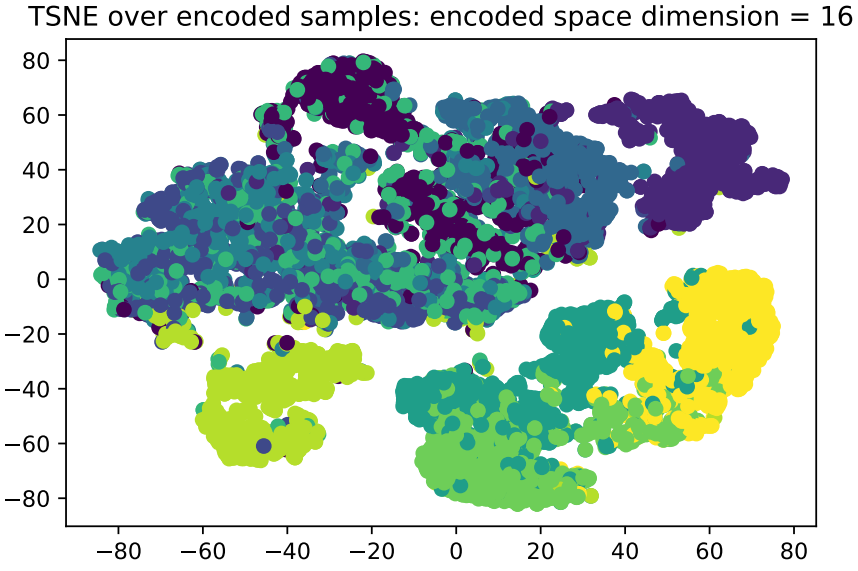

Fig. 15: TSNE over 16-dimensional latent space autoencoder