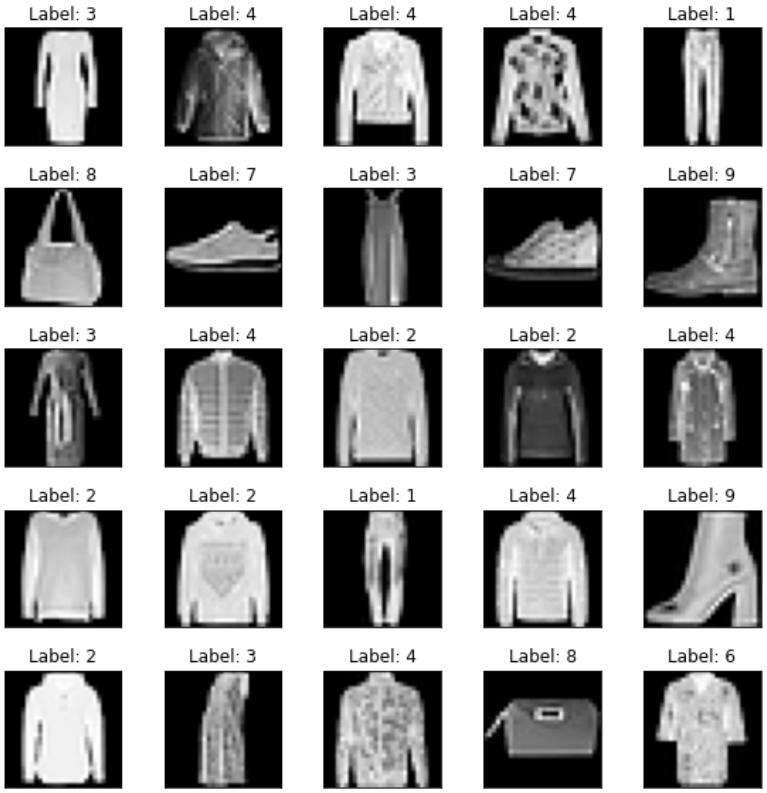
Merve Tunçer 2040508

Neural Network and Deep Learning Homework 2

In this homework, I implemented and tested neural network models for solving unsupervised problems. I used Convolutional Auto-Encoder (CAE) and Optuna analysis for this task. On the FashionMNIST database of fashion clothes images, the CAE will be programmed, optimized, and tested, and it will be able to learn the data encoding of this dataset efficiently. The PytorchLightning and Optuna tools are used to optimize the network's hyper-parameters in order to increase its performance. The Encoder will be used as a pre-trained network to fine-tune a classifier using transfer learning after the best model has been chosen. A Variational Convolutional Auto-Encoder (VAE) is also implemented and its performance is examined, starting with the CAE structure. PCA and t-SNE algorithms are used to investigate the latent space structure, and new samples are generated for both architectures.

1) Method and Results

The Fashion-MNIST dataset is proposed as a more challenging replacement dataset for the MNIST dataset. It is a dataset comprised of 60,000 small square 28×28 pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more.



1.1)Convolutional Auto-Encoder

Convolutional Autoencoder is **a variant of Convolutional Neural Networks** that are used as the tools for unsupervised learning of convolution filters. They are generally applied in the task of image reconstruction to minimize reconstruction errors by learning the optimal filters. The Encoder has the following structure:

- one convolutional layer with 1 input channel, n_channels output channels, a 3 by 3 kernel, padding and stride set to 2 to reduce the image size from 28 × 28 to 15 × 15 pixels;

- another convolutional layer with n_channels input channels, 2·n_channels output channels, 3 by 3 kernel, padding= 1 and stride= 2, that reduces the size of the images to 8 × 8 pixels;

- a third convolutional layer with 2·n_channels input channels, 4·n_channels output channels, 3 by 3 kernel, padding= 0 and stride= 2, that reduces further the size of the images to 3 × 3 pixels;

- after flattening the output of the last convolutional layer, a fully connected layer with 3 × 3 × 4·n_channels = 36·n_channels input units and 64 outputs;

- finally, a fully connected layer with 64 inputs and with encode_space_dim output units.

The Decoder, on the other hand, has a completely symmetrical structure and is composed of:

- a fully connected layer with encode_space_dim inputs and 64 outputs;

- another fully connected layer with 64 input units and 3 × 3 × 4·n_channels = 36·n_channels outputs;

- after unflattening the output of the previous layer, converting it to 3 × 3 images, a transposed- convolutional layer with 4·n_channels input channels, 3 by 3 kernel, padding= 0, stride= 2 and 2·n_channels output channels, that transforms the encoded pictures in 7 × 7 images;

- the second transposed-convolutional layer, with 2·n_channels input channels, n_channels output channels, 3 by 3 kernel, stride= 2, padding and output padding= 1, that changes image sizes to 14 × 14 pixels;

- the last transposed-convolutional layer with n_channels input channels, 1 output channel, a 3 by 3 kernel, stride= 2, padding and output padding= 1 to restore the image size to its original dimension (28 × 28 pixels).

Each layer is activated by a ReLU activation function but the last one has a Sigmoid function. The loss function adopted is the standard Mean Squared Error loss. Optuna is a hyperparameter optimization framework to automate hyperparameter search, which can be applied in Machine Learning and Deep Learning models. Thanks to the fact that it uses sampling and pruning algorithms to optimize the hyperparameters, it's very fast and efficient. Moreover, it allows building dynamically the hyper parameter search space in an intuitive way. In Optuna, the goal is to minimize / maximize the objective function, which takes as input a set of hyperparameters and returns a validation score. For each hyperparameter, we consider a different range of values. The process of optimization is referred as a study, while each evaluation of the objective function is called a trial. After a random search with 20 Optuna trials, the model with the minimum validation loss (value) is picked as the best model and tested, with its final loss and performance shown.

The best hyperparameters results:

Best trial:

Value: 0.016608916223049164 (Test loss)

Best network parameters:

encoded_space_dim: 14

n_channels: 8

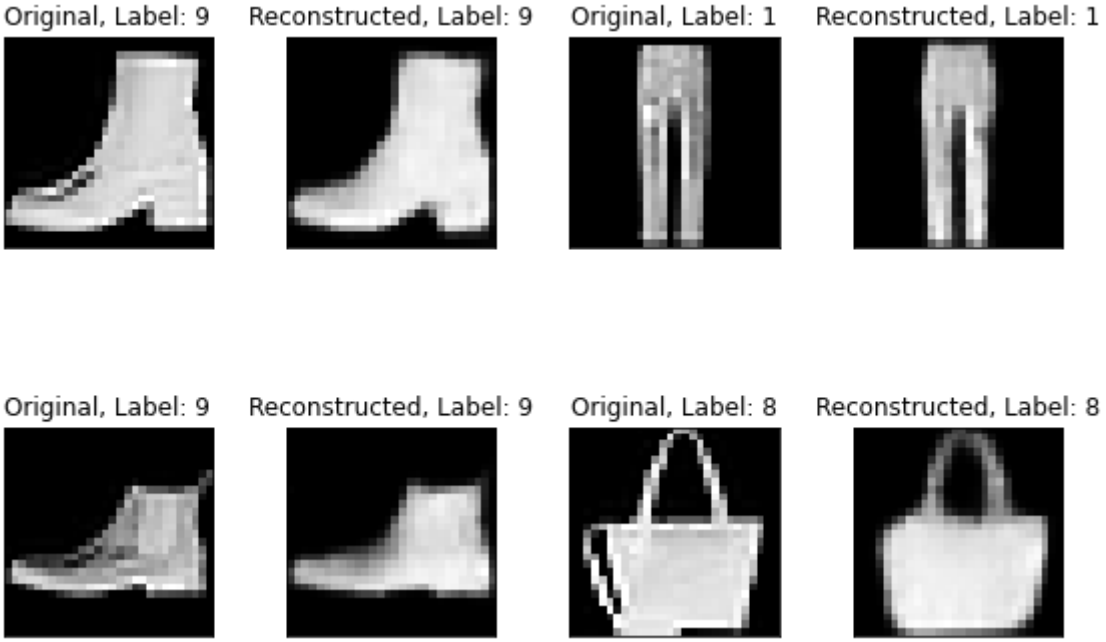
optimizer: Adam

learning_rate: 0.001

weight_decay: 1e-05

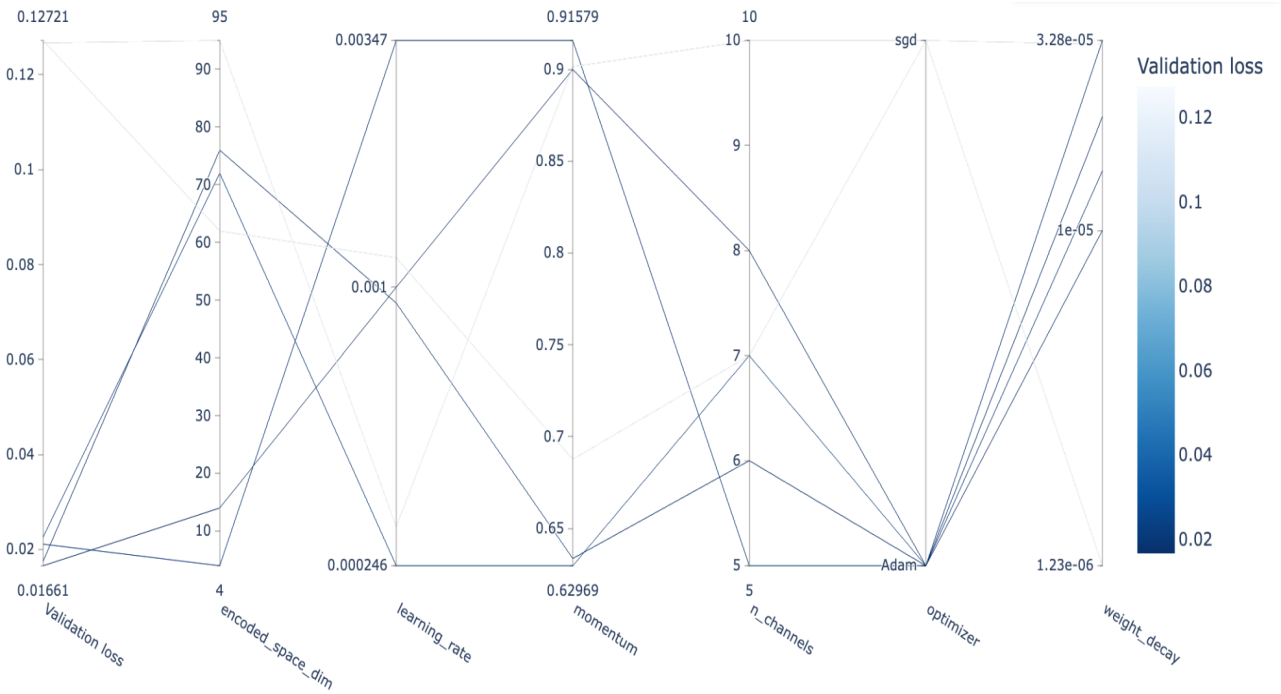
momentum: 0.9

If we look at some reconstructed digits, we can easily see how good the results are. Choosing a random test-set digit and feeding it to the CAE creates images that look like the one seen in Figure of examples of CAE reconstructed test-set. The shape of the rebuilt digits is completely comparable to that of the input ones, with a minimum small margin of "blurriness": the CAE is able to detect the major features and properly reconstruct the original dataset.



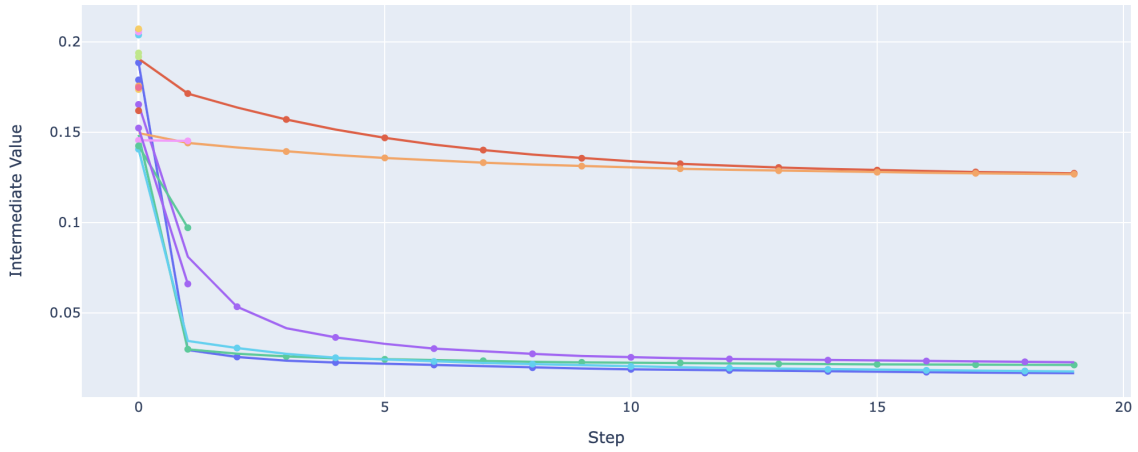
Some examples of CAE reconstructed test-set

Below is the Parallel Coordinate graph of all considered optimization history:



Parallel coordinate plot, which highlights what set of parameters performed better

Plot of validation loss vs. epoch number for the non-pruned trials in the Optuna search:



1.2)Variational Auto-Encoder

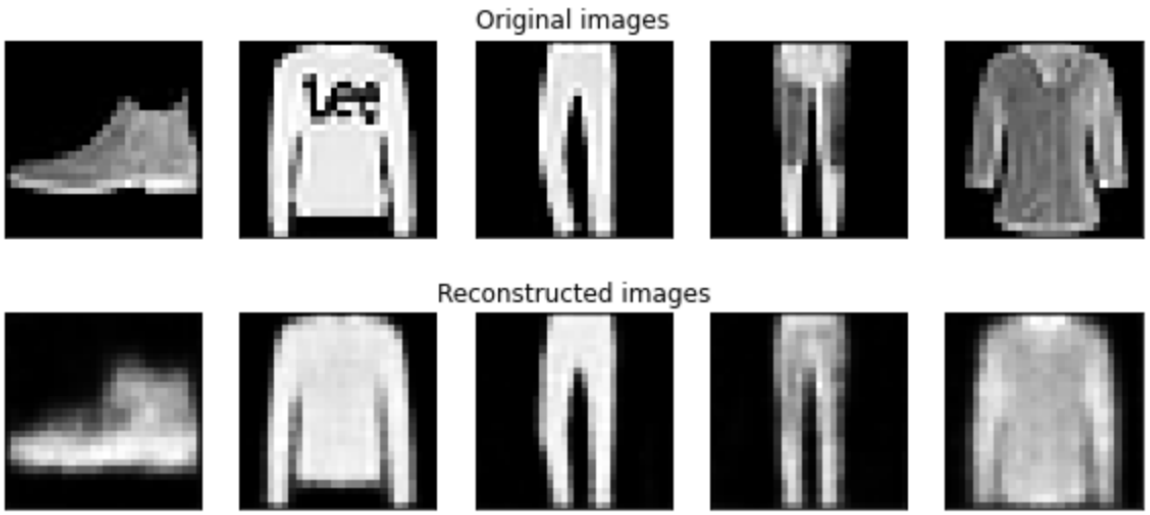
The standard autoencoder can have an issue, constituted by the fact that the latent space can be irregular. This means that close points in the latent space can produce different and meaningless patterns over visible units. One solution to this issue is the introduction of the Variational Autoencoder. As the autoencoder, it is composed of two neural network architectures, encoder and decoder. But there is a modification of the encoding-decoding process.

- Instead of considering the input as a single point, we encode it as a distribution over the latent space. The normality of the encoded distribution is chosen so that the encoder can be trained to return the mean and covariance matrices.
- We sample a point from that encoded distribution in the second phase.
- The sampled point can then be decoded and the reconstruction error calculated.
- We backpropagate the reconstruction error through the network. We need to use a reparameterization method to make the backpropagation work because the sampling procedure is a discrete process, not a continuous one:

$$z = E(z) + \epsilon \times \sqrt{V(z)} \text{ where } \epsilon \sim N(0, I_d)$$

VAE architecture is the encoder and decoder networks contain three convolutional layers and two fully connected layers. To have more robust features in the latent space, some batch normal layers are included. The encoder, unlike the usual

autoencoder, returns mean and variance matrices, which we utilize to get the sampled latent vector. In the VariationalEncoder class, we obtain the Kullback-Leibler term.



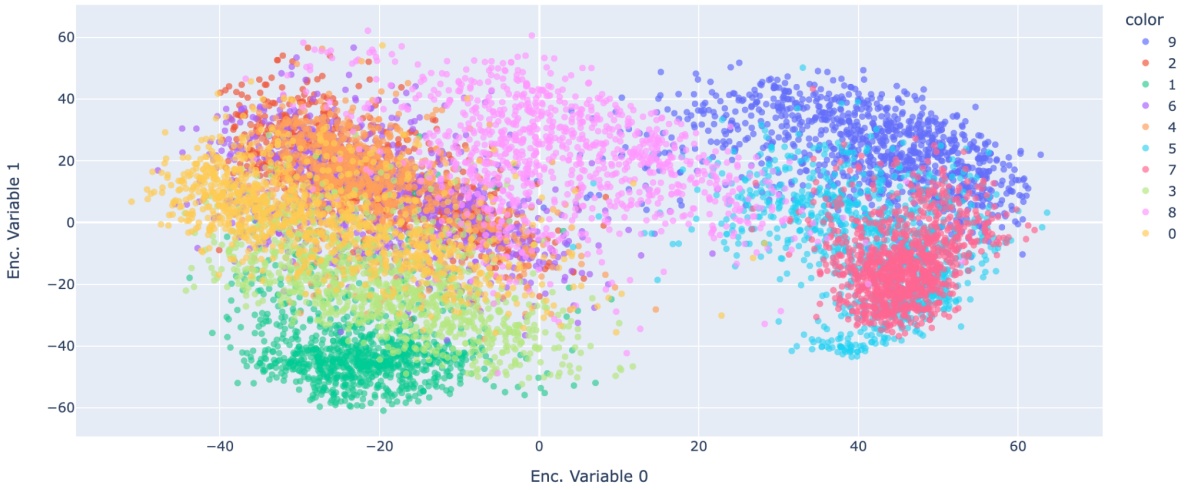
These are the results obtained after 30 epochs. We see that there is a high similarity between the input images and their reconstructions, even if there are still some imperfections. The reconstructed digits lose some details and are blurred compared to the original ones.

1.3) Encoded Space Analysis and sample generation

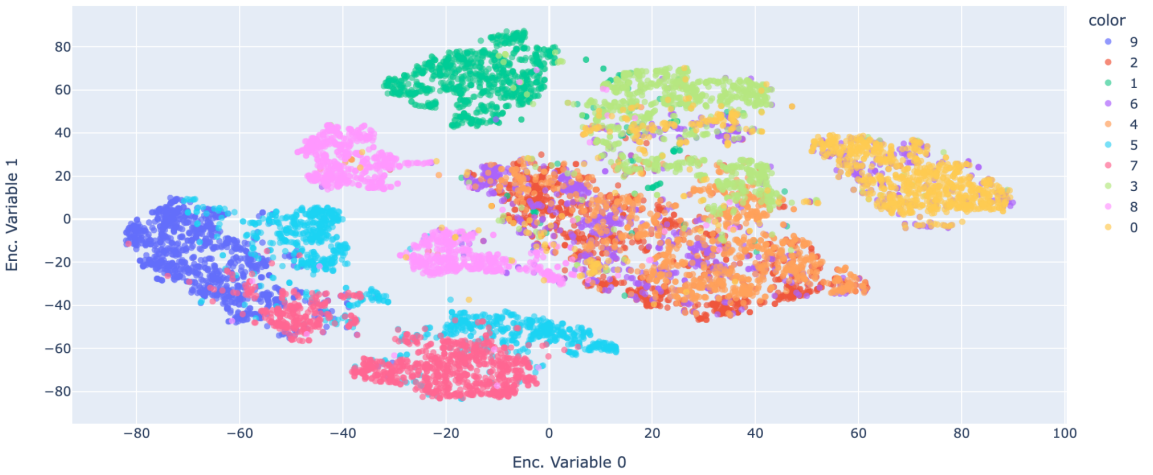
In order to get a better understanding of how the network learns and encodes relevant data characteristics, this section will create a visual representation of the so-called latent space, or the space between the encoder and the decoder, using two different techniques: Principal Component Analysis (PCA) and t-distributed Stochastic Neighbour Embedding (t-SNE). Moreover, to see the properties of the latent space, both the CAE and the VAE will be used to generate new images from the latent space vector.

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. The method of **t-distributed Stochastic Neighbor Embedding (t-SNE) is a method for dimensionality reduction**, used mainly for visualization of data in 2D and 3D maps. This method can find non-linear connections in the data.

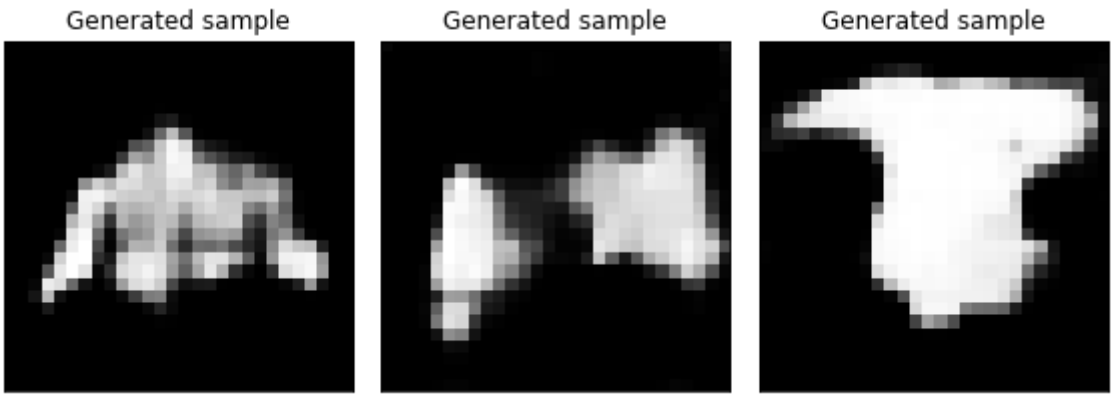
Convolutional Autoencoder PCA and t-SNE results are follows:



Convolutional Autoencoder PCA



Convolutional Autoencoder t-SNE



Samples generated by the CAE

VAE PCA and t-SNE results are follows:

VAE PCA results

VAE t-SNE

The most striking feature of these figures is that the VAE, being a generative model, performs significantly better than the CAE, and that in fact produces images that can totally be interpreted. The CAE, on the other hand, has a discrete (and not a continuous) latent space, and hence is better at reconstructing images similar to the ones he received in input, but not at creating completely new samples. In any case, the images generated by the CAE has some features that remind the shape of a image, but are not clearly interpretable.

VAE generating samples

1.4) Fine tuning

Weight fine-tuning of a classifier is performed in this part utilizing a transfer learning technique in order to perform efficiently the supervised classification task. The first element of the new network is the Encoder trained for the prior task. Two fully connected layers are added in place of the Decoder: the first has encoded space dim inputs, 64 output units, and is activated with a ReLU, while the second has 64 inputs, 10 outputs (corresponding to the 10 classes), and is activated with a LogSoftmax function to generate the probability classes. The supervised task's loss function is Negative Log Likelihood, which, when combined with the last layer activation function, allows the network to accurately assign labels to each input based on the sample's chance of belonging to a specific class. Only the fine tuner is trained before being put to the test on the test set, with accuracy as the only metric.

I got %78 accuracy for fine tuning. I got %90 accuracy for my first homework model. As can be seen, the model of the first homework works more successfully, the fine tuning model could not approach the success of the other model.