Università degli Studi di Padova

Homework #2 Unsupervised Deep Learning

Student: *Beatrice Segalini* – 1234430

Course: *Neural Networks and Deep Learning* – Professor: *A. Testolin*

1. Introduction

In this homework, the implementation and testing of neural network models for solving unsupervised problems is displayed, in particular, the basic tasks consists of testing and analysing a Convolutional Auto-Encoder (CAE).

The CAE will be coded, optimised and tested on the MNIST database of hand-written digits, and it will be able to learn efficiently the data encoding of this dataset, therefore being able to generate reconstructed samples and to act as a denoiser. To improve its performances, `PytorchLightning` and `Optuna` tools are employed for optimising the hyperparameters of the network.

After selecting the best model, the Encoder will be used as pre-trained network to fine-tune a classifier, using transfer learning. Besides, starting from the CAE structure, a Variational Convolutional Auto-Encoder (VAE) is implemented and its performances tested.

Finally, the latent space structure is studied with both PCA and t-SNE techniques, and new samples are generated form it for bothe the architectures.

2. Convolutional Auto-Encoder

The MNIST database is composed of 28×28 pixels images representing digits, labelled from 0 to 9. The training set includes 60000 samples, while the test one 10000. When the datasets are imported, thanks to the possibility to apply transformations directly when downloading the data provided by Pytorch, the `ToTensor()` tranformation is applied to convert to Pytorch tensors and normalise their values in the $[0, 1]$ interval.

For the hyper-parameters optimisation and the model selection, the training set is divided into two chunks via the `random_split` function: 80% of it (48000 samples) is used for the training process, while the other 20% composes the validation set.

2.1. Methods

Broadly speaking, a CAE is composed of an *Encoder* and a *Decoder*. In this homework, 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 $3 \times 3 \times$ 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, instead, has a totally symmetric 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 $3 \times 3 \times 4 \cdot$`n_channels` $= 36 \cdot$`n_channels` outputs;
- after unflattening the output of the previous layer, converting it to 3×3 images, a transposed-convolutional layer with $4 \cdot$`n_channels` input channels, 3 by 3 kernel, padding$= 0$, stride$= 2$ and $2 \cdot$`n_channels` output channels, that transforms the encoded pictures in 7×7 images;
- the second transposed-convolutional layer, with $2 \cdot$`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).

The number of convolutional channels `n_channels` is an integer chosen in the $[4, 10]$ interval, while the encoded space dimension `encode_space_dim` is picked in the range $[2, 100]$. In this way, the encoder will significantly reduce the original dimension of the data ($28 \cdot 28 = 784$ pixels per image), grasping the relevant features and compressing (encoding) the key information of the inputs. Each layer is activated by a ReLU activation function but the last one, which has a Sigmoid function. The loss function adopted is the standard Mean Squared Error loss.

In addition, during the random search also some regularisation and optimisation methods are investigated, in particular the optimiser is chosen between:

- Stochastic Gradient Descent, with L2 regularisation parameter (weight decay) `weight_decay` sampled from a log-uniform distribution in the interval $[10^{-6}, 10^{-4}]$, `momentum` parameter sampled uniformly in $[0.6, 1.]$ and learning rate `learning_rate` sampled from a log-uniform distribution in the interval $[10^{-5}, 10^{-3}]$;
- Adam optimiser, with the same sampling methods for `learning_rate` and `weight_decay` .

The hyperparameters optimisation is performed through `Optuna` , and the results are displayed in Figures (6), (7) and (8).

Given the size of the dataset and the complexity of the network, the training process can be quite computationally demanding and significantly time consuming, thus the *early stopping* procedure provided by Pytorch Lightining is used.

After having performed the random search with 200 Optuna trials, the one with the minimum validation loss (value) is chosen as best model and tested, showing its final loss and its performances in reconstructing hand-written digits.

2.2. Results

The better performing model is reported in Table (1) and produces a test loss of only 0.00693.

`encoded_space_dim`	`n_channels`	`optimizer`	`learning_rate`	`weight_decay`	`momentum`
67	9	Adam	0.00176	$3.513 \cdot 10^{-6}$	0.88972

Table (1) – Optimised hyperparameters for CAE, found via `Optuna` search over 200 trials.

The goodness of the results achieved can be clearly seen if we observe some reconstructed digits. Picking a random test-set digit, feeding it to the CAE produces pictures similar to the one in Figure (1). One can notice how the shape of the reconstructed digits is totally comparable with the input ones, with a minimum margin of "blurriness": the CAE is able to detect the main features and re-build the original digit almost perfectly.

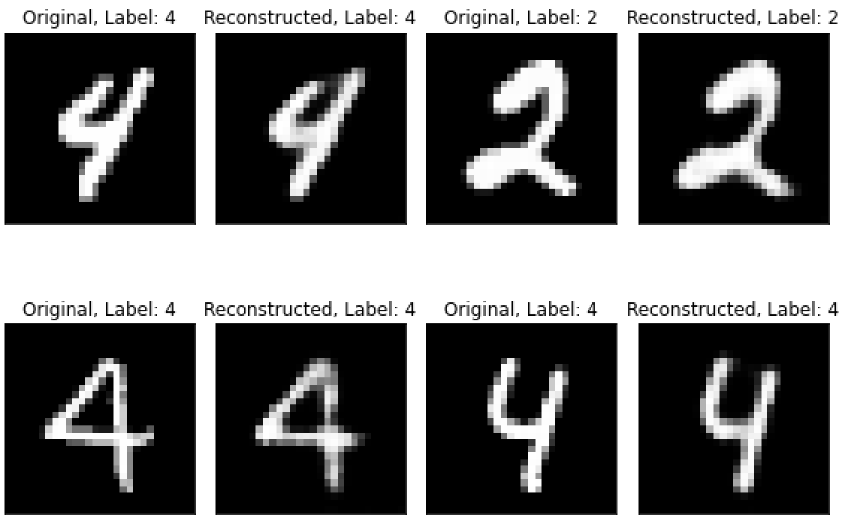

Figure (1) – Some examples of CAE reconstructed test-set digits.

3. Denoiser

In this section, using the CAE optimised previously, the network will be given as input data affected by noise, and it will be able to recover the original picture despite the data "corruption".

3.1. Methods

To generate noisy data, three different kind of noise are defined, taking inspiration from the built-in random noises available in sklearn (`skimage.util.random_noise`): gaussian noise (sampled from a normal distribution of given mean μ and standard deviation σ^2); "Salt and Pepper" noise (which randomly changes elements with a certain probability p, setting them to 1 (salt) or 0 (pepper) with probability p_{SvP}); Speckle noise (add to the original data the same data multiplied by a gaussian distribution of given mean μ and standard deviation σ^2). An example of how the different types of noise act on the digits can be found in Figure (9).

A new data structure `Dataset_noise` is used for producing noisy data and give them as input to the network, which is trained with noisy samples.

3.2. Results

The Denoiser proved to be very effective in reconstructing noisy data, as it is evident in Figure (2). The digits are generally well recovered and, even if their edges are not always perfectly defined, it is generally possible to detect which label they correspond to, task that is impossible to do for a human just looking at the noisy data. On the test dataset, the Denoiser achieves a final loss of 0.02086, which is higher than before but nevertheless acceptable.

4. Fine tuning and transfer learning

In this section, using a *transfer learning* technique, weight fine-tuning of a classifier is performed, in order to perform efficiently the supervised classification task of the hand-written digits.

4.1. Methods

The Encoder trained for the previous task is taken as first element of the new network, freezing its weights. Instead of the Decoder, two fully connected layers are added: the first one has `encoded_space_dim`

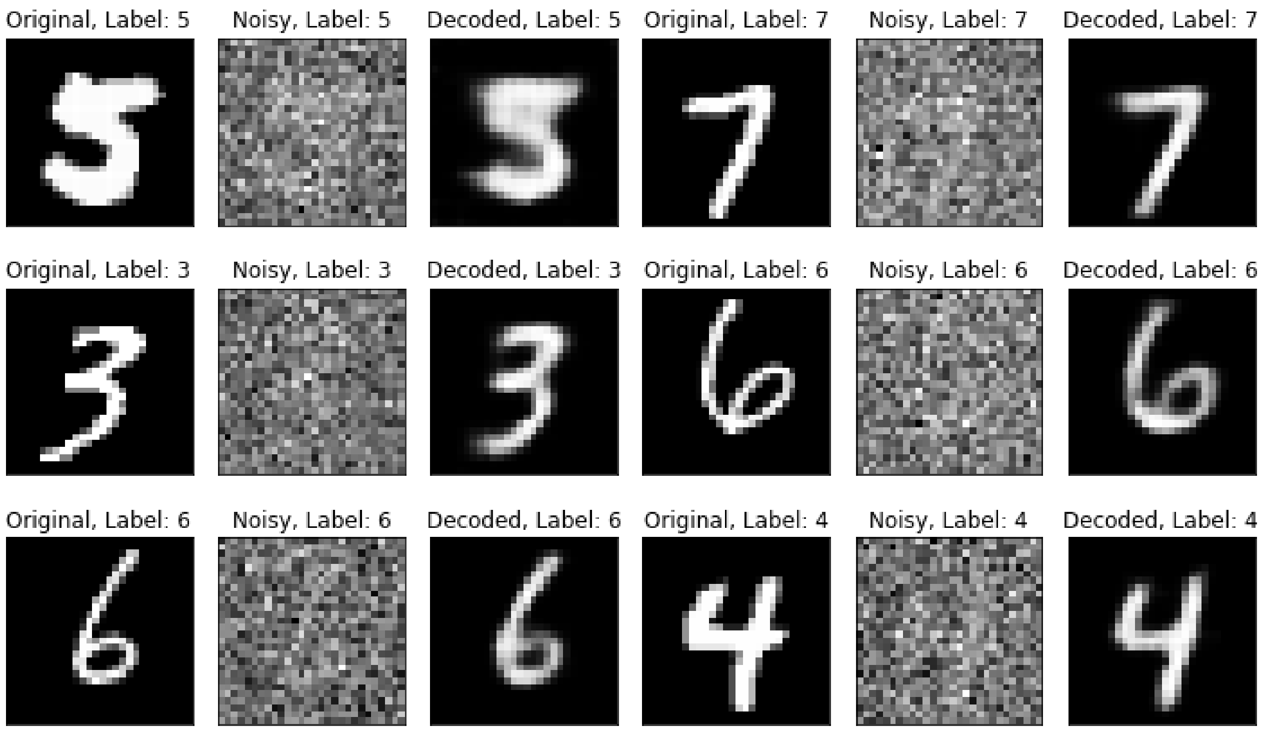

Figure (2) – Reconstruction of noisy data. 6 different digits are shown before the application of noise, with the noise, and after the denoising process performed by the network.

inputs, 64 output units and is activated with a ReLU, the second one has 64 inputs, 10 outputs (corresponding to the 10 classes) and is activated with a LogSoftmax function, in order to generate the probability classes.

The loss function for the supervised task is the Negative Log Likelihood, which combined with the last layer activation function allows the network to properly assign labels to each input, depending on the probability of the sample to belong to a certain class. Only the `fine_tuner` is trained and then tested on the test set, using the accuracy as metric.

4.2. Results

The network trained very quickly both in terms of computational time and in number of epochs required to arrive to convergence. In fact, only 21 epochs are required for the early stopping procedure to cease the training process. Nonetheless, the final test accuracy reaches the incredibly good value of 97.93%, comparable with the results obtained in the previous homework (accuracy of 98.87%).

5. Variational Auto-Encoder

In this section, a VAE is defined, starting from the CAE architecture, by changing the structure of the latent space, i.e. the space between the encoder and the decoder. Differently from the CAE, the VAE is a generative model and the latent space vectors are continuous. This allows the network to generate new images in a more accurate way with respect to the standard CAE.

5.1. Methods

More specifically, if z is the latent vector, the goal of the VAE is to model the data probability distribution $p_\theta(z)$, in order to generate new samples using p_θ as a Bayesian prior.

To do so, the VAE will aim to maximize the log-likelihood of the data, by reducing the reconstruction error between the input and the output of the network. This translates into minimising the quantity:

$$\mathcal{L}_{\theta,\Phi} = -\log(p_\theta(\mathbf{x})) + D_{KL}(q_\Phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$$

where Φ are the learned parameters, and D_{KL} is the Kulba-Leiber divergence.

In order to be able to back-propagate and hence for allowing the network to learn, a *reparameterization trick* is applied, rewriting the latent space vector z assuming that it is sampled from a set Gaussian distributions with means $\vec{\mu}$ and standard deviations $\vec{\sigma}$:

$$z \sim q_\phi(z|x) = \mathcal{N}(\vec{\mu}, \vec{\sigma}^2) \sim \vec{\mu} + \epsilon\vec{\sigma}$$

with $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$.

Hence, the loss to be optimised will be the sum of the *reconstruction* loss ℓ_R (computed with the MSE) and the KL-divergence, that can be rewritten as follows:

$$\ell = \ell_R + D_{KL} = \text{MSE} + \frac{k}{2}\sum_j \left((\mu_j)^2 + (\sigma_j)^2 - 1 - \log{(\sigma_j)^2} \right)$$

The factor k is used as a regularisation term and will be set to 10^{-5} after some trials.

Concerning the practical code, the latent space is composed of 2 entities: `fc_mus` for inferring the μ_j and `fc_sigma` for σ_j, since the distribution assumed to represent the latent space is a multivariate-gaussian with diagonal covariance matrix. Each of the two structures is composed of two fully connected layers, the first with 36·`n_channels` inputs and 64 outputs (activated by a ReLU); the second with 64 inputs and `encoded_space_dim` outputs. The other hyperparameters are the same displayed in Table (1).

5.2. Results

The VAE takes a little longer with respect to the CAE to arrive to convergence (49 epochs), and achieve a final test loss of 0.048.
Differently from what it was observed in Figure (1), here the VAE is not particularly good in reconstructing the samples. It can be noticed that the edges are more smudged and that in some cases it is not perfectly clear what digit is represented.
However, it will perform significantly better in the new data generation task, as shown in the next section.

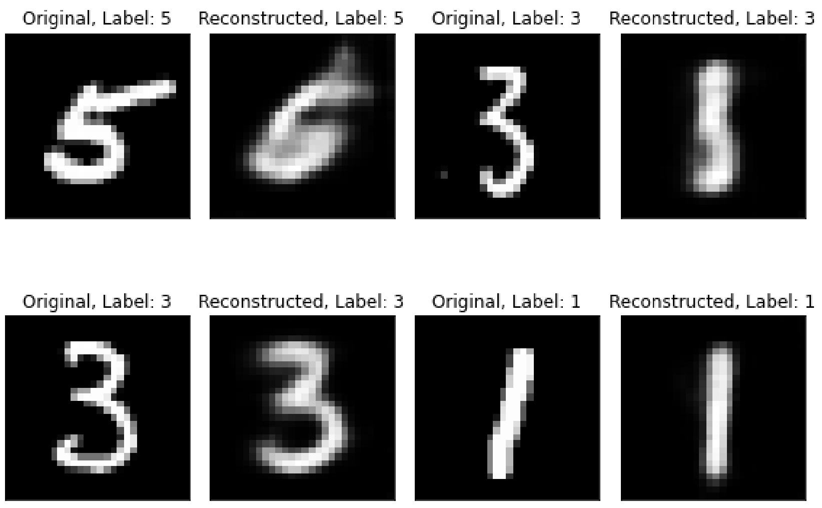

Figure (3) – Examples of VAE reconstructed samples from the test set.

6. Encoded Space Analysis and sample generation

To have more insight on how the network is able to learn and encode relevant characteristics of the data, a visual representation of the so called *latent space*, i.e. the space between the encoder and the decoder, will be produced in this section, 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.

6.1. Methods

Both the techniques employed aim to reduce the dimension of the space in order to infer the most relevant data features. In particular:

- PCA is defined as an orthogonal *linear* transformation that transforms the data to a new coordinate system, such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

- t-SNE is a *non-linear* technique that lowers the dimensionality of data by giving each datapoint a location in a two or three-dimensional space. The t-SNE algorithm constructs a probability distribution over pairs of high-dimensional objects, assigning higher probabilities to points with "similar features", then does the same over the points in a lower-dimensional space, and finally maps in the latter space the original points by minimising the Kullback–Leibler divergence (KL divergence) between the two distributions with respect to the locations of the points in the map.

For implementing both the algorithms, `sklearn` tools are exploited. For both the techniques, the reduced dimensionality (i.e., the number of components `n_component`) is set to 2.

Concerning the generated samples, they are created by generating random gaussian tensors (multiplied by an arbitrary constant for enhancing the visualisation) of the same size of an encoded image, then fed to the decoder of both the VAE and the CAE, which then returns a new image, generated in the latent space.

6.2. Results

In Figure (10) the results of the PCA procedure are displayed. It can be noticed that different digits are not clearly divided into "clusters" and that it is difficult to interpret these data structures.

Differently it happens with the t-SNE analysis, reported in Figure (11): in this case, each digit has its own cluster which belong to a well-defined region of the latent space, with few outliers.

The most interesting task concerning the latent space, however, is the data generation one. In Figures (4) and (5), some generated digits can be observed, produced respectively by the CAE and the VAE.

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 as hand-written digits. The CAE, on the other hand, has a discrete (and not a continuous) latent space, and hence is better at reconstructing digits 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 digit, but are not clearly interpretable.

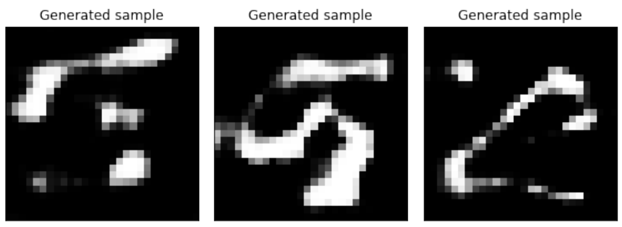

Figure (4) – Samples generated by the CAE.

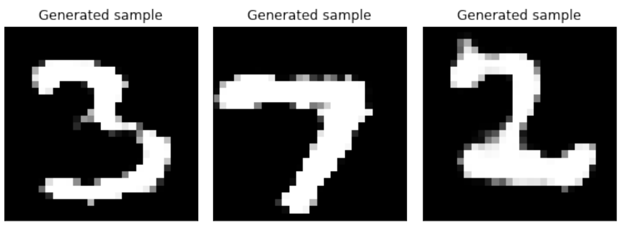

Figure (5) – Samples generated by the VAE.

Appendix

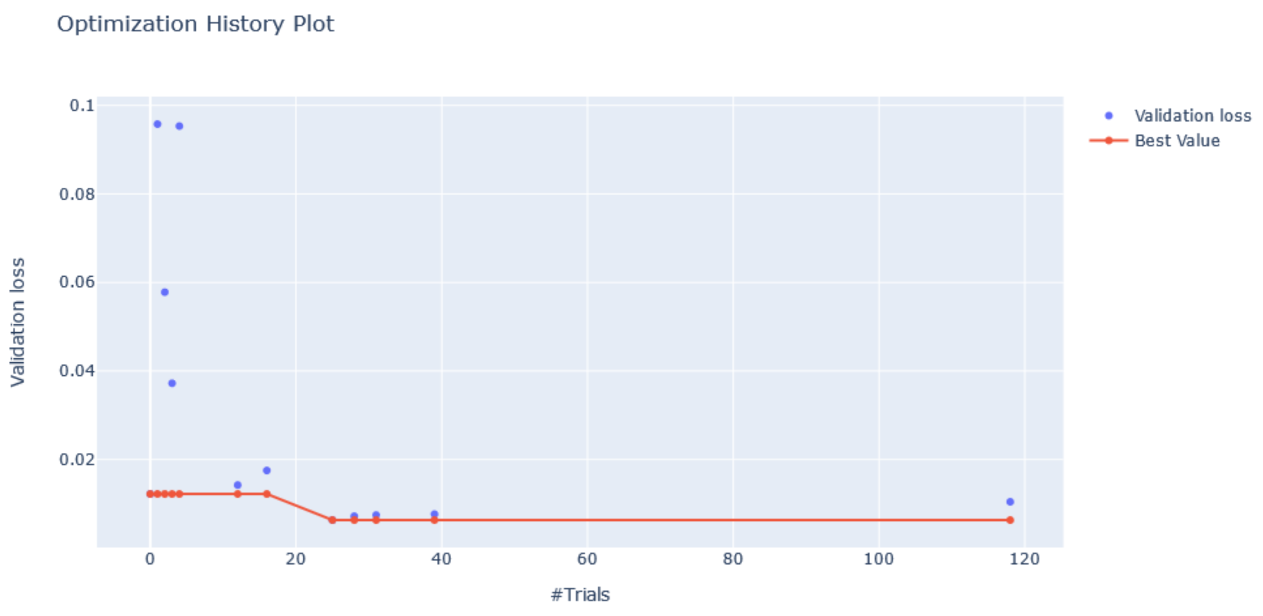

Figure (6) – Optuna optimisation history plot: validation loss vs. trial number. One can notice that several trials were pruned by the algorithm, hence there are few blue points.

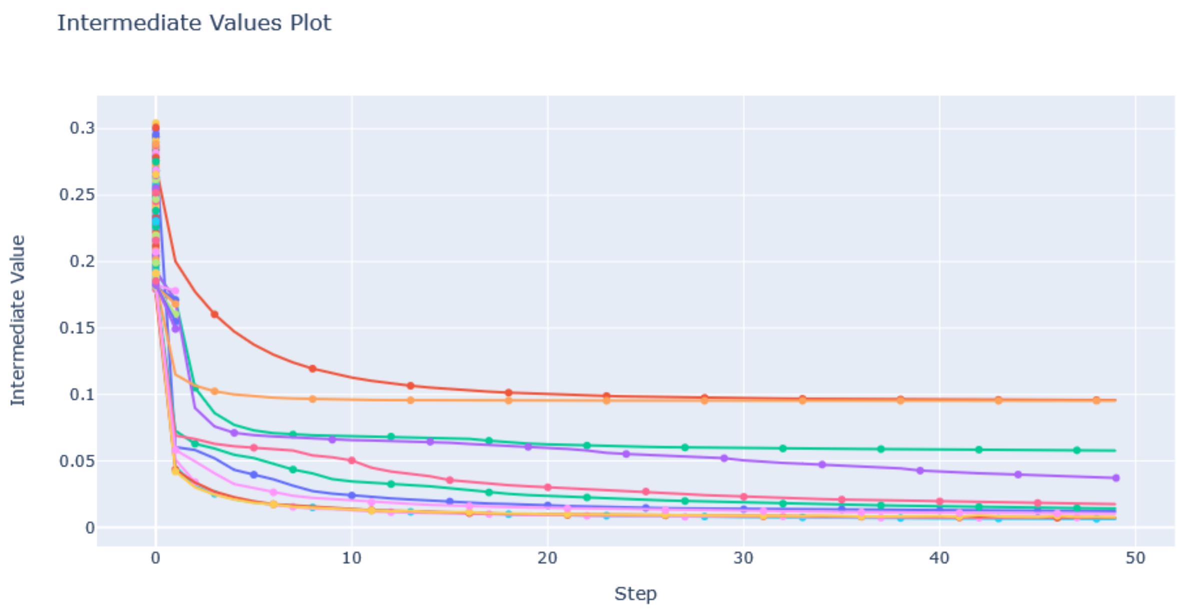

Figure (7) – Plot of validation loss vs. epoch number for the non-pruned trials in the Optuna search.

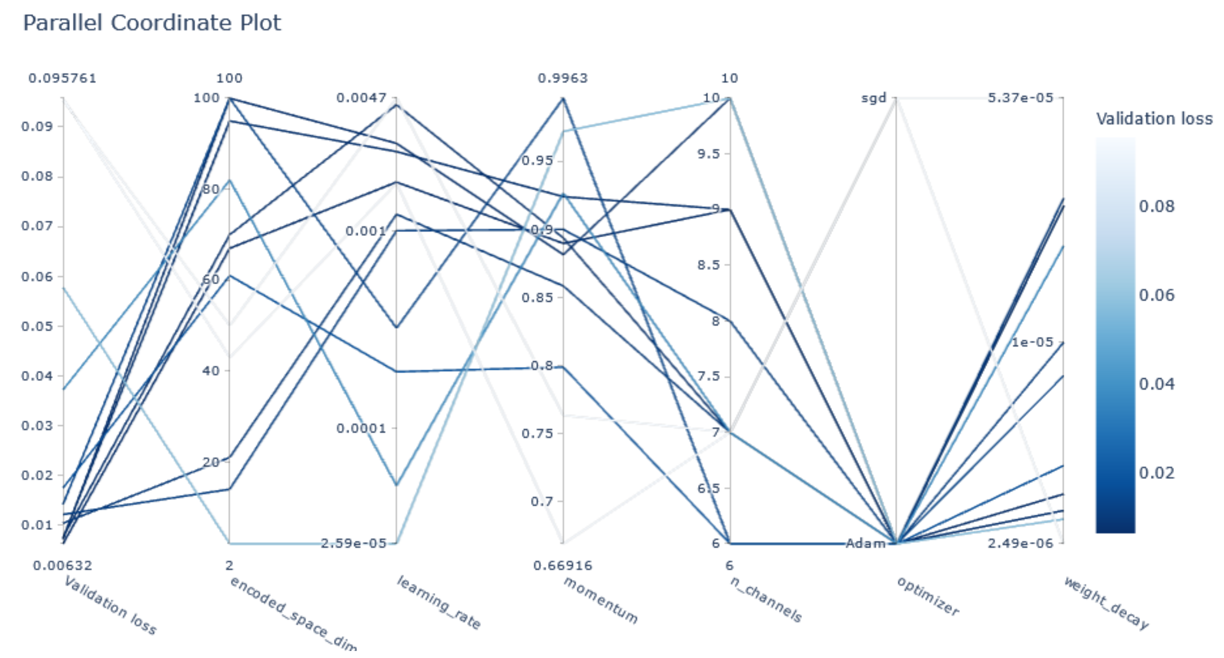

Figure (8) – Parallel coordinate plot, which highlights what set of parameters performed better. The parameter space was covered in a pretty complete way. It is interesting to see how using SGD optimiser leads to worse results.

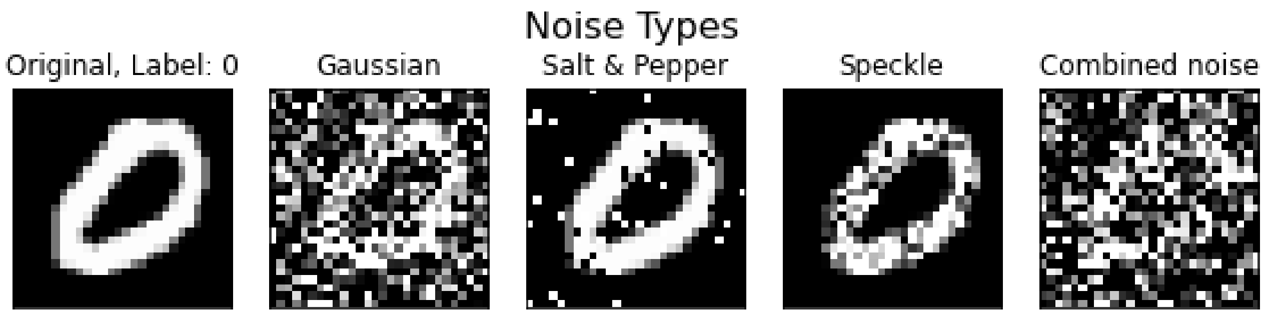

Figure (9) – Action of the different user-defined noise functions on a sample digit. From left to right: original, un-noisy data; application of gaussian noise with $\mu = 0$ and $\sigma^2 = 0.5$; application of Salt and Pepper noise ($p = 0.1$, $p_{SvP} = 0.5$); application of Speckle noise ($\mu = 0$ and $\sigma^2 = 0.5$), and a combination of all the three type of noise (gaussian with $\mu = 0$ and $\sigma^2 = 0.5$, Salt and Pepper with $p = 0.05$ and $p_{SvP} = 0.5$, Speckle with $\mu = 0$ and $\sigma^2 = 1$).

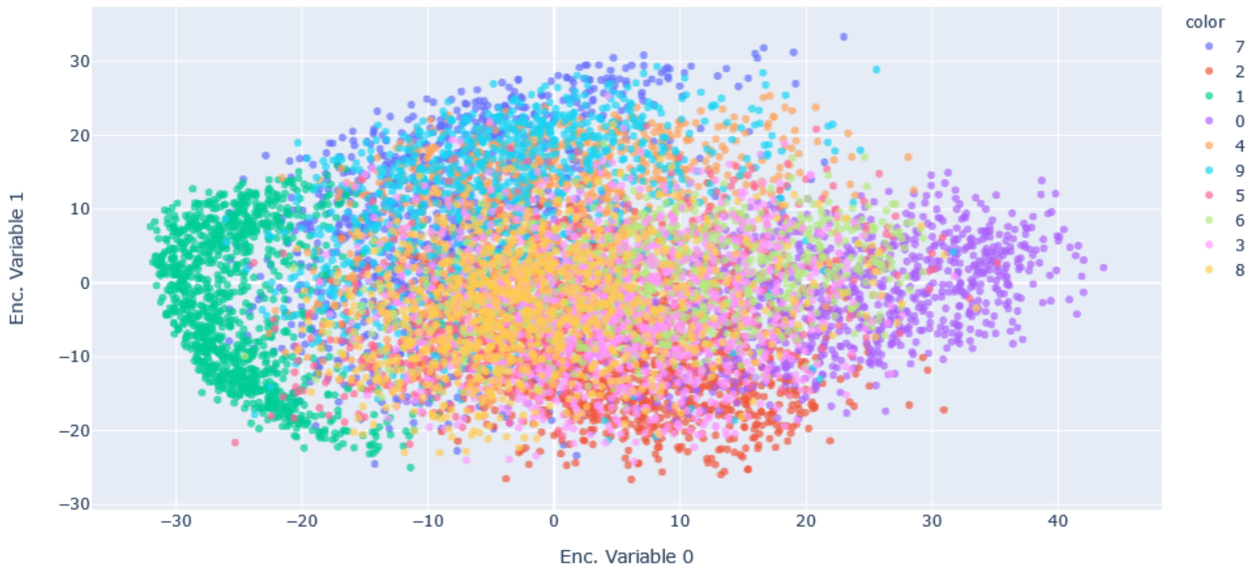

Figure (10) – PCA for the latent space if the CAE.

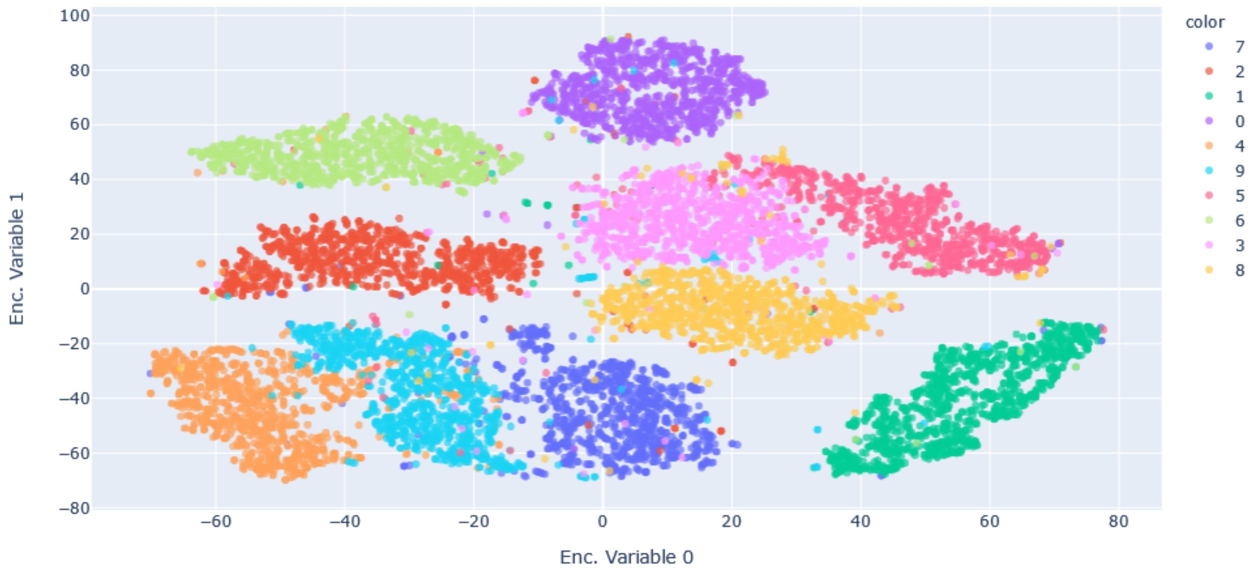

Figure (11) – t-SNE for the latent space if the CAE.