Unsupervised Deep Learning

Assignment 2

Francesco Manzali

22/06/2021

In unsupervised learning, useful features are learned directly from data, without the need of any labels. Autoencoders are a common neural network architecture for solving such problem. This homework explores different kinds of autoencoders, evaluating their performance through their reconstruction error, and inspecting their learned latent spaces. Moreover, autoencoders are shown to be useful for training supervised models more quickly.

Introduction

An **autoencoder** is a type of neural network constructed to learn latent representations of unlabeled data. It consists of two parts:

- An *encoder*, which maps input samples to their latent representations
- A *decoder*, which (approximately) reverses the encoder mapping.

The main idea is that the network tries to store all the "relevant" information about the inputs in the latent space, so that it can be reconstructed by the decoder. The difference between an input and the reconstruction of its representation is quantified by the *reconstruction error*, which is minimized during training.

In this homework, autoencoders are trained on the MNIST dataset of digits, which is split into 48 000 samples for training, 12 000 for validation and the remaining 10 000 for the final testing.

In sec. 1 a first convolutional autoencoder is tested, and its hyperparameters are optimized. Then, sec. 2 tests the same architecture on the task of reconstructing noisy samples. Sec. 3 presents a variational autoencoder, which is regularized to learn more interpretable representations. Finally, sec. 4 shows how a pretrained autoencoder can help the construction of a small supervised network, which is able to quickly reach a good performance.

All networks are implemented using the PyTorch Lightning library [1], while Optuna [2] is used for the hyperparameters search.

1 Convolutional Autoencoder

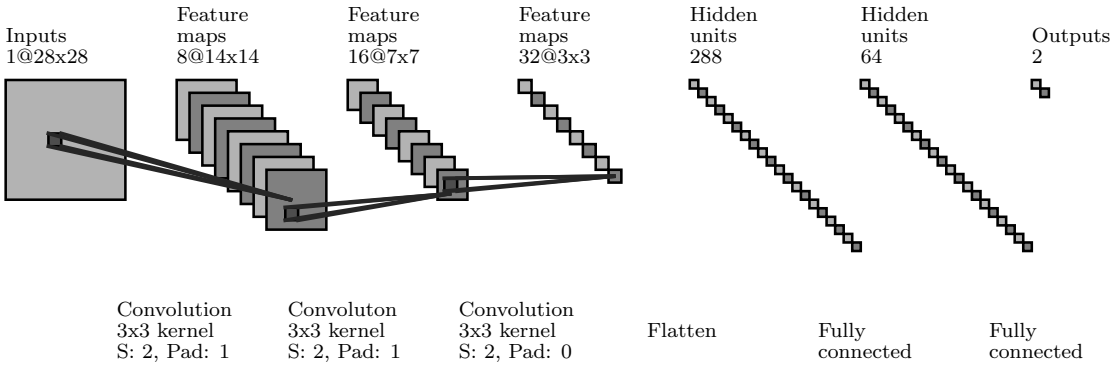
1.1 Methods

The architectures for the encoder and decoder are shown in fig. 1. They consist of a 3-layer convolutional segment, followed by 2 fully connected layers, and they exactly mirror each other.
The ReLU activation is used for all hidden layers, since it helps avoiding the gradient vanishing problem. The encoder's output has no activation, while a sigmoid is applied after the decoder's output, since the dataset is normalized to have values in $[0, 1]$. Dropout is applied after each hidden layer, both in the convolutional and fully-connected segments. MSE is always used as the loss function.

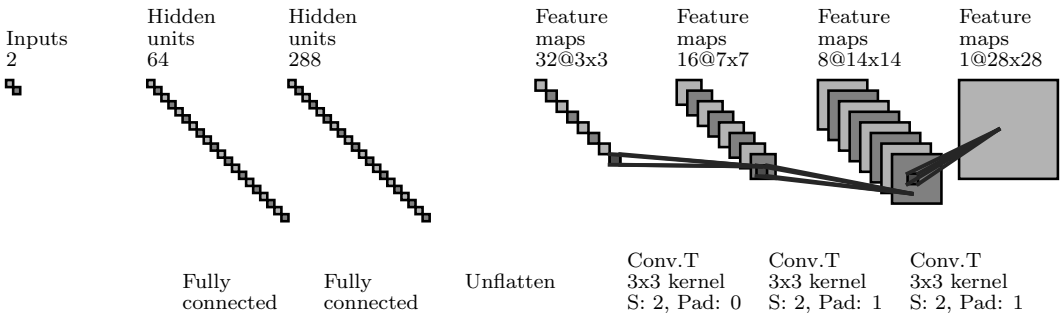
The search space for hyperparameters, and the final values found by optimization through Optuna are shown in tab. 1 (a complete log can be found in the Appendix, in fig. 13). They are used for all subsequent models, if not specified otherwise. Notably, the optimal latent space is not the highest allowed, meaning that few dimensions suffice to capture all the data variability.

	Dim. Latent Space	**Optimizer**	**Learning rate**	**Batch size**	**Dropout**
Search	$[2, 50]$ (uniform)	Adam or SGD	$[10^{-5}, 10^{-2}]$ (log-uniform)	64, 128, 256	$[0, 1]$ (uniform)
Best (100 trials)	33	Adam	5.4×10^{-3}	128	1.2×10^{-2}

Table (1) – Hyperparameters for the convolutional autoencoder.



(a) Encoder architecture



(b) Decoder architecture

Figure (1) – Architecture of the convolutional autoencoder. The size of the encoder output (equal to that of the decoder input) is the dimension of the latent space (2 in the figure).

1.2 Results

The autoencoder is then trained for 50 epochs, using the best hyperparameters found and early stopping. The final MSE loss on the test dataset is 0.0087, which is very good for this dataset. Some examples of reconstruction from the validation set are shown in fig. 2, and are all satisfactory.

A few more pictures are available in the Appendix. First, the learning curves for the reconstruction error are shown in fig. 7. Fig. 9 and 10 show the PCA/t-SNE projection of latent representations of digits. From these, especially the latter, it can be seen that the model learns to cluster together images of the same digit, even if it is not provided any label for them. Finally, an autoencoder is trained with a $2d$ latent space, which allows to directly plot the representations as points in a plane (fig. 11).

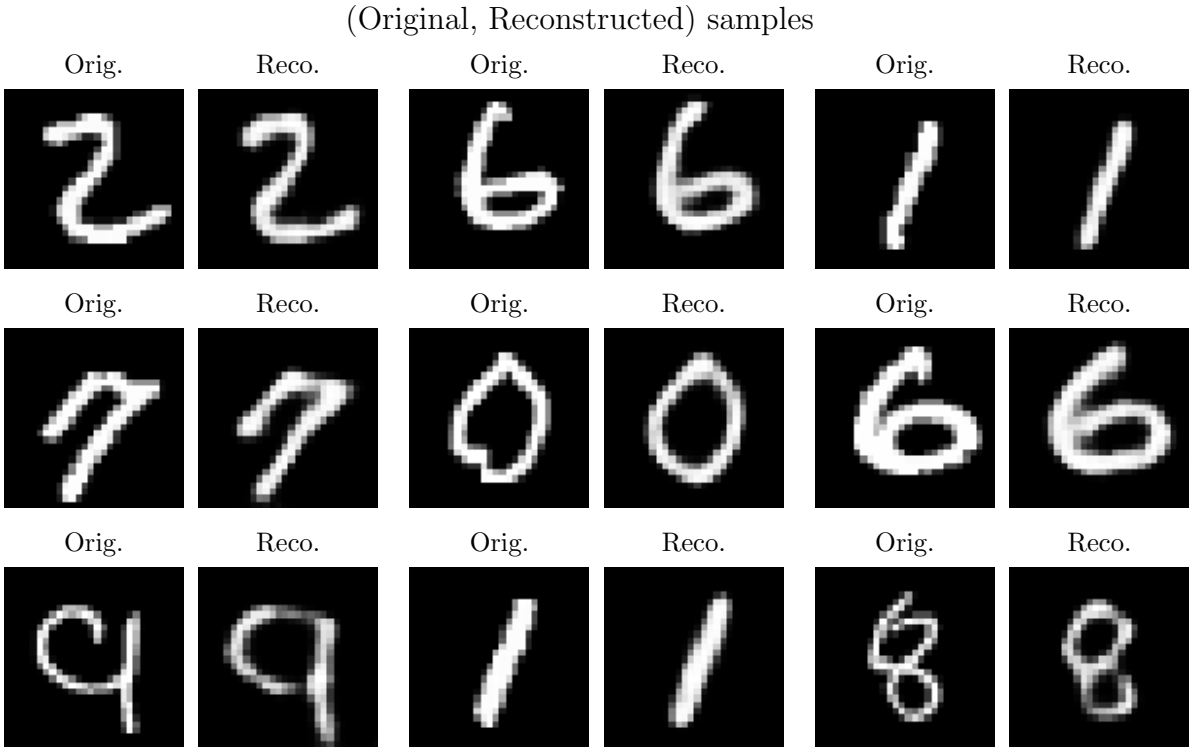


Figure (2) – Samples reconstructed by the CNN autoencoder.

2 Denoising Autoencoder

2.1 Methods

In this application, samples are shown to the convolutional autoencoder with added noise (fig. 3). The architecture is the same described in the previous section, and the reconstruction error still measures the distance between the reconstructed digits and the original ones, without noise. This means that, effectively, the model is trained to *remove* the noise from the input samples.

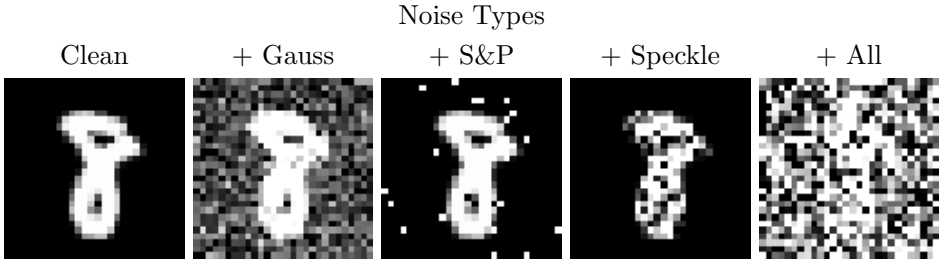


Figure (3) – Types of noise considered for generating the training samples. **Gaussian**: random samples from a normal distribution $\mathcal{N}(0.5, 0.3)$ are added to the image. **Salt & Pepper** (S&P): one out of 20 pixels is set to 1 (salt) or 0 (pepper) with equal probability. **Speckle**: a random filter is constructed by sampling a gaussian $\mathcal{N}(0, .5)$. Then the image is multiplied by the filter, and the result is added to the original image. In all cases, pixel values are clipped between 0 and 1 after applying the noise.

2.2 Results

Training for 50 epochs with the same hyperparameters from tab. 1 allows to reach a test MSE loss of just 0.03. The added noise has clearly made reconstruction much more difficult (fig. 4), and some

digits are not correctly retrieved. Nonetheless, the model is powerful enough to resolve an amount of noise that is challenging even for humans, which is remarkable.

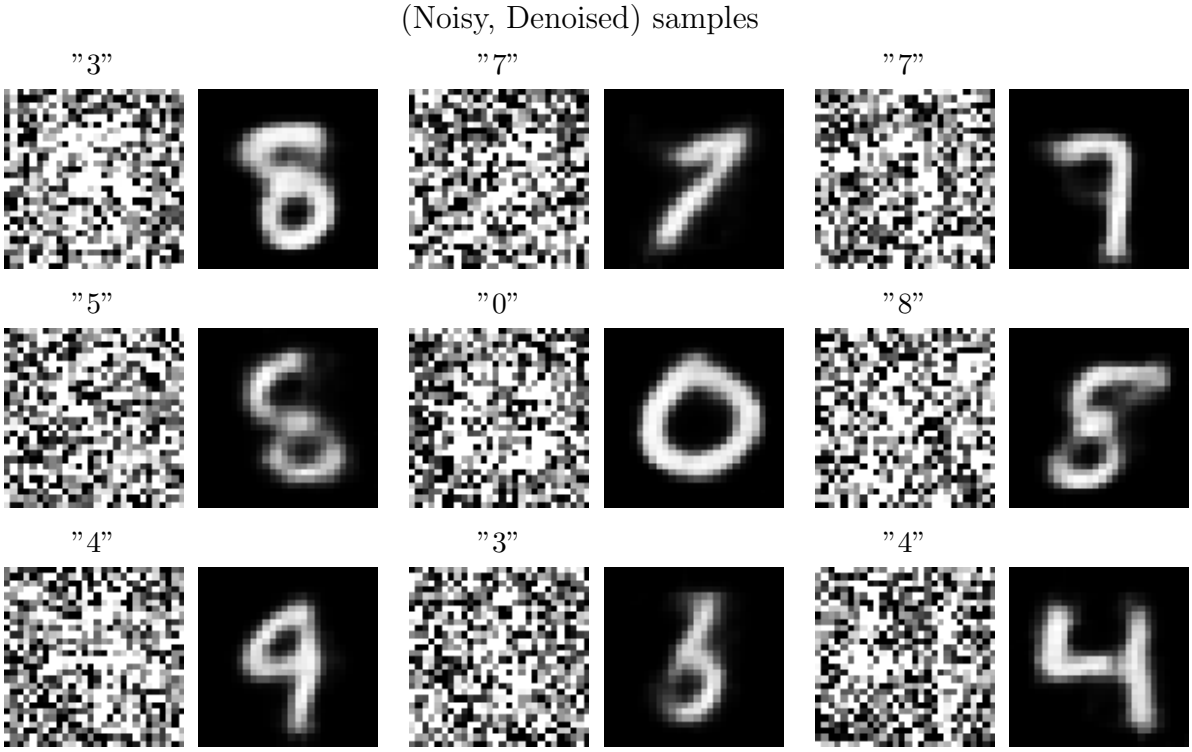


Figure (4) – Samples reconstructed by the denoising autoencoder.

3 Variational Autoencoder

Variational autoencoders use an entirely different process for reconstructing images. In fact, in this case the *encoder* does not output directly a latent vector, but rather the parameters of a probability distribution from which the latent vector can be sampled. The ability to choose that distribution, and to force it to be the "simplest" possible through regularization, allows to obtain an unsupervised learner that is more robust to noise, and learns more interpretable representations.

3.1 Methods

The *encoder* architecture from fig. 1 is changed to output the parameters of a multivariate Normal distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with diagonal covariance matrix $\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_d^2)$, chosen for its simplicity and efficiency. Specifically, the convolutional segment is kept unchanged, but its output is sent to two independent copies of the fully-connected segment: one computes the entries of the mean vector $\boldsymbol{\mu} = (\mu_1, \dots, \mu_d)^T$, while the other returns the logarithm of the variances $(\log \sigma_1^2, \dots, \log \sigma_d^2)^T$. In this way, a sample $\boldsymbol{s}$ can be generated as follows:

$$\boldsymbol{s} = \boldsymbol{\mu} + \boldsymbol{\xi} \odot \exp\left(\frac{1}{2} \log \boldsymbol{\sigma}^2\right) = \boldsymbol{\mu} + \boldsymbol{\xi} \odot \boldsymbol{\sigma} \qquad \boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \mathbb{1}_d)$$

where all operations, such as exponentiation ($\boldsymbol{\sigma}^2$) or multiplication ($\odot$) are element-wise. Note that the randomness is completely contained in the separate $\boldsymbol{\xi}$ factor, and that the expected value of $\boldsymbol{s}$ is differentiable with respect to the network's parameters used to compute $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, allowing to apply backpropagation.

The loss function, taken from [3], consists of a reconstruction error term (MSE) and a regularization term, which forces the learned distribution to be as close as possible to a standard Normal distribution $\mathcal{N}(\mathbf{0}, \mathbb{1}_d)$:

$$\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 + \mathrm{KL}[\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\mathbf{0}, \mathbb{1}_d)] = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 - \frac{1}{2}\sum_{j=1}^{d}(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$

where $\boldsymbol{x}$ denotes an input sample, $\hat{\boldsymbol{x}}$ is the reconstructed sample by the decoder, and KL is the Kullback-Leibler divergence, measuring the distance between two probability distributions.

3.2 Results

The network is trained for 100 networks, with no dropout, Adam as optimizer with $\lambda = 10^{-3}$ and a latent space of dimension 33. Overfitting is mitigated by stopping training if the validation loss has not decreased for 10 epochs.
A few reconstructed samples are shown in fig. 5. The quality is a bit lower than that of the convolutional autoencoder from sec. 1, due to the much more significant regularization.
However, the final representations are more robust: a small change in the input is mapped to a small change in the latent space. This nice property allows to compute "transitions" between images, by starting from a representation vector and moving step by step along the line that connects it to some other target vector. This is shown in fig. 6, where a "0" is gradually transformed to a "9". Interestingly, the middle image looks like a "8", which is intuitively an intermediate shape between the two digits.

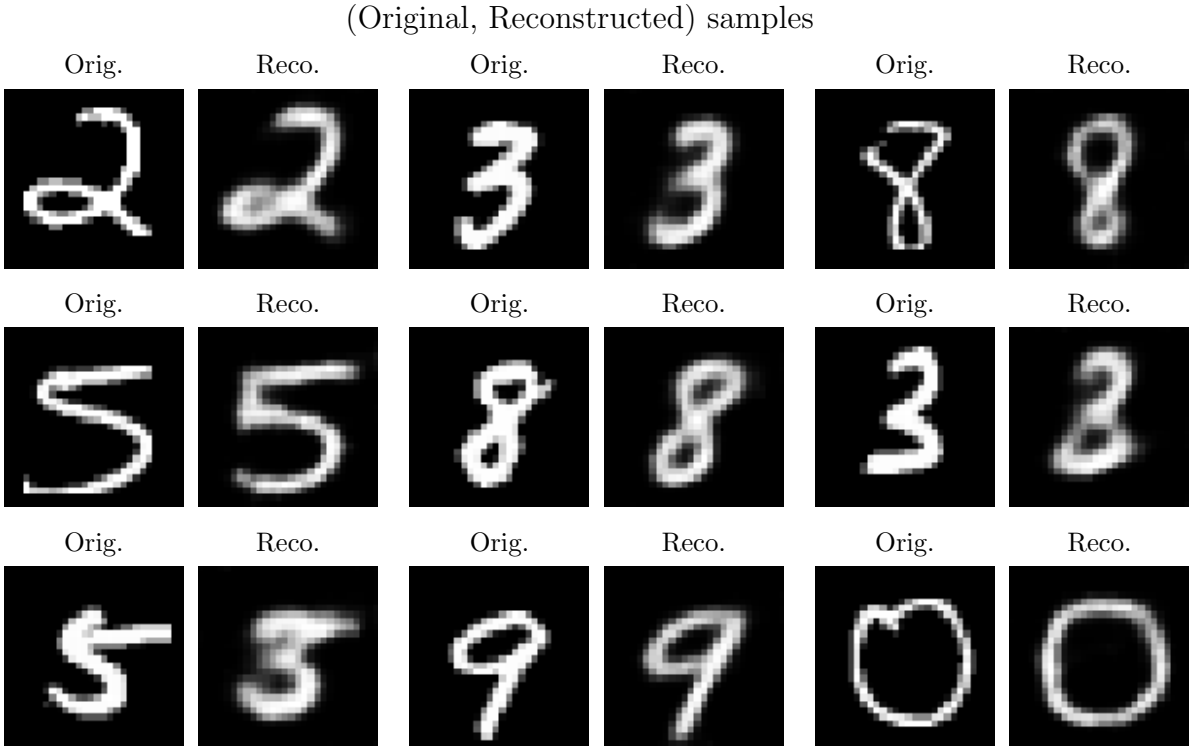


Figure (5) – Samples reconstructed by the variational autoencoeder.

In the appendix, the latent representations from a $2d$ variational autoencoder are directly plotted (fig. 12), so that they can be compared with those from a normal convolutional autoencoder (fig. 11). Note how, in the variational case, clusters are more well separated and "regular".

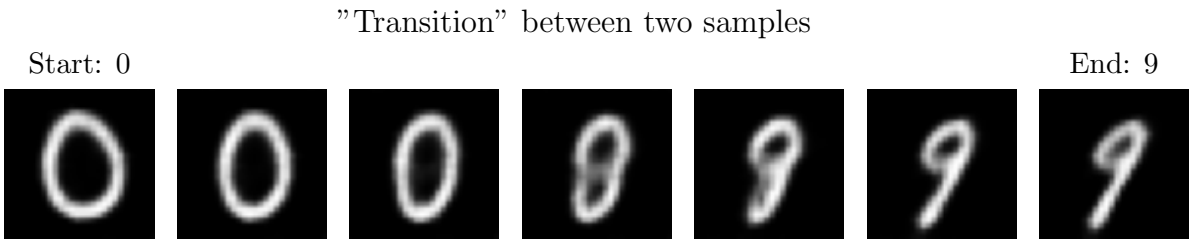


Figure (6) – In a variational autoencoder, a linear interpolation between two latent representations produces a "smooth" transition between the digits they represent.

4 Transfer Learning

A pretrained autoencoder can be *fine-tuned* to a supervised task, such as classification, significantly speeding up the training process, while reaching comparable results to end-to-end training.

4.1 Methods

To test this, the *encoder* part of the convolutional autoencoder from sec. 1 is "attached" to just two fully-connected layers with 128 and 10 neurons each, separated by a ReLU activation. Then, only the parameters of the latter segment are trained, while keeping all the ones from the encoder fixed to their previously learned values.

4.2 Results

The final network is trained for 50 epochs with early stopping, Adam with $\lambda = 5 \times 10^{-3}$ and no regularization. The loss converges after just 22 epochs, with a final train accuracy of 99%, and 98.1% on both the validation and the test datasets. This performance is good for such few trainable parameters (less than 6k), and training is very fast, requiring just a few minutes. Note also that no hyperparameter finetuning was done.
However, the best network from the first homework reached a 99.5% accuracy on the test dataset, albeit requiring a full 100 epochs and an extensive grid search for the hyperparameters. Thus, exploiting a pretrained network is good for reaching acceptable results quickly, but getting the best possible score requires a full training procedure.

The learning curve for this supervised task is shown in fig. 8 in the Appendix.

References

[1] WA Falcon. "PyTorch Lightning". *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* 3 (2019).

[2] Takuya Akiba et al. *Optuna: A Next-Generation Hyperparameter Optimization Framework*. July 25, 2019. arXiv: `1907.10902 [cs, stat]`. URL: `http://arxiv.org/abs/1907.10902` (visited on 06/21/2021).

[3] Diederik P. Kingma and Max Welling. *Auto-Encoding Variational Bayes*. May 1, 2014. arXiv: `1312.6114 [cs, stat]`. URL: `http://arxiv.org/abs/1312.6114` (visited on 06/21/2021).

Appendix

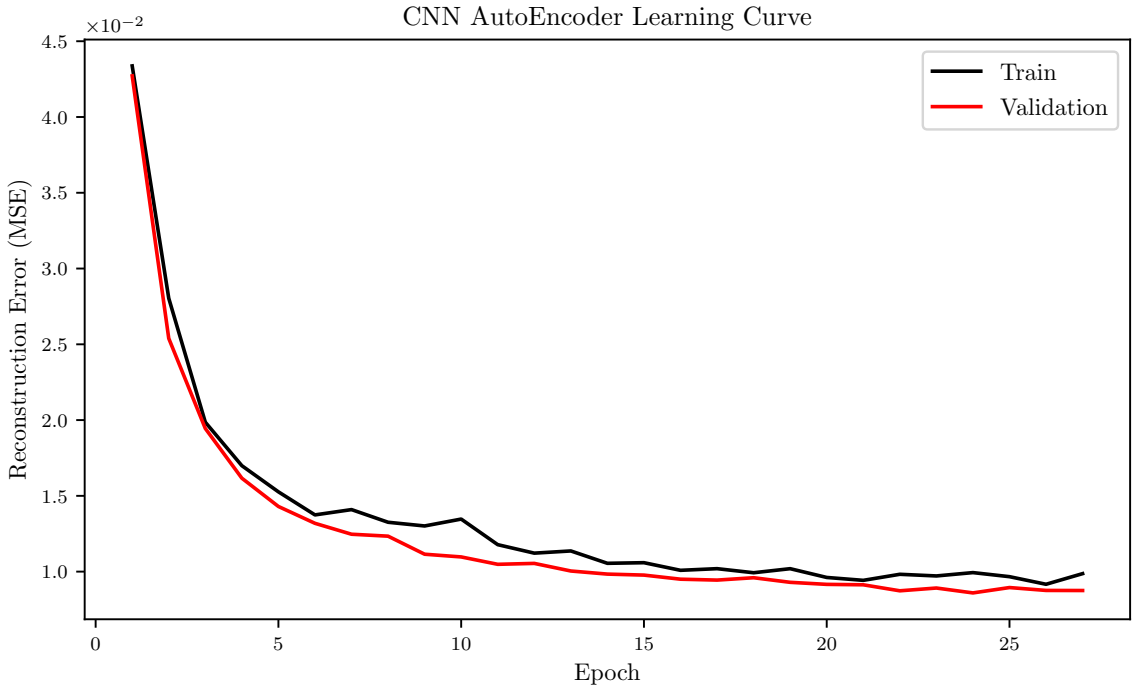


Figure (7) – Learning curves for the CNN Autoencoder with the best hyperparameters found.

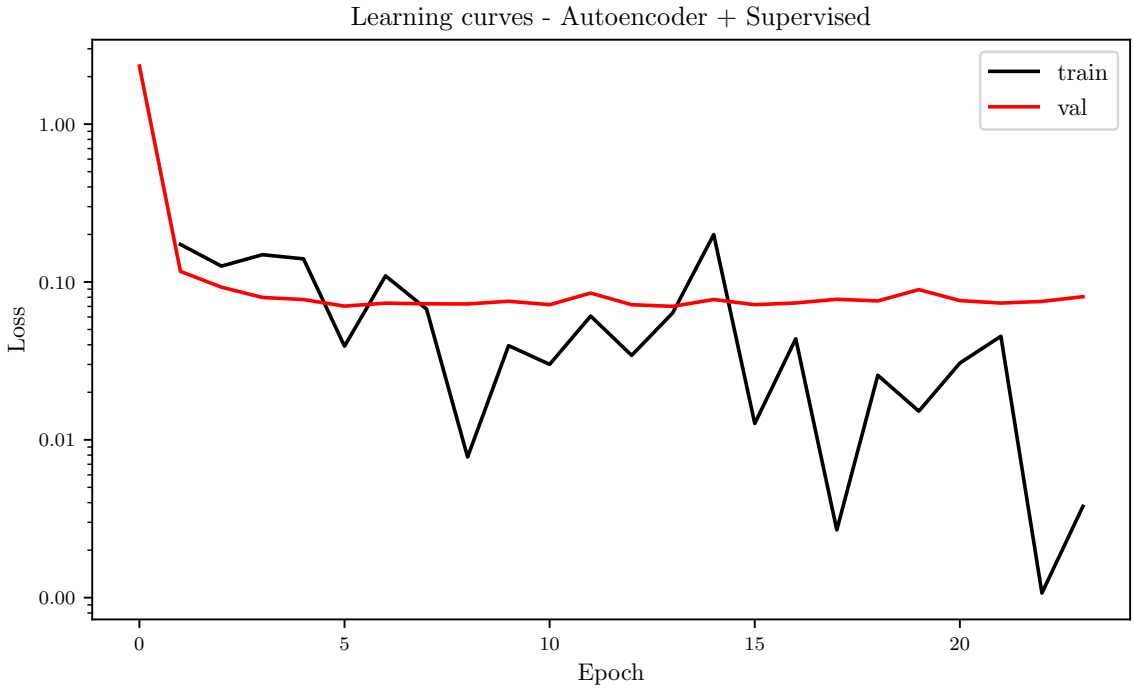


Figure (8) – Learning curves for the supervised classfier trained on top of a pretrained CNN autoencoder.

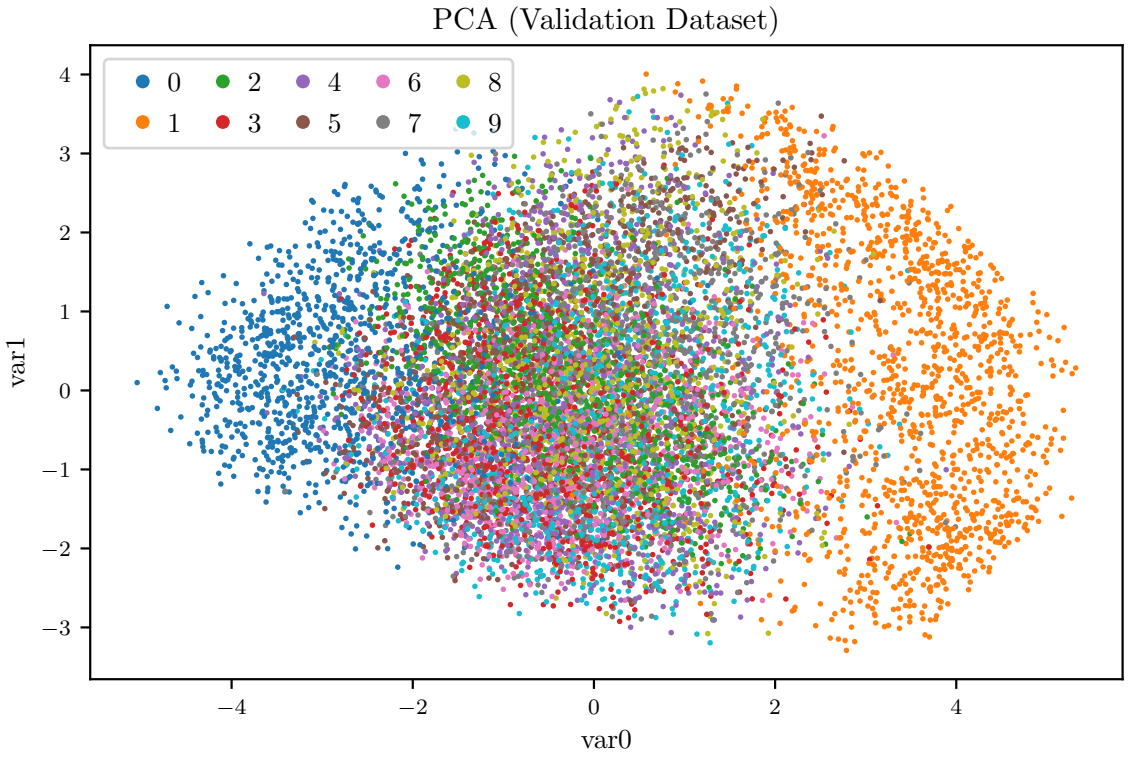


Figure (9) – PCA projection of the latent representations of MNIST digits from the validation set, as learned by the CNN Autoencoder (best hyperparameters).

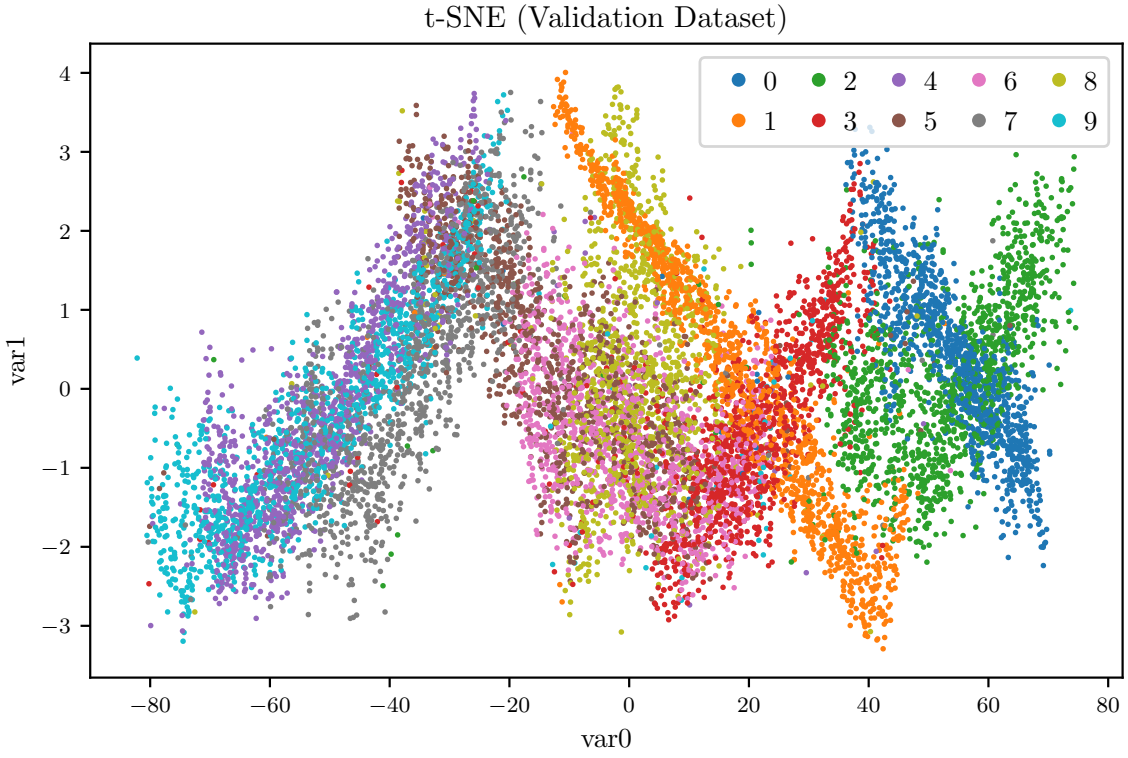


Figure (10) – t-SNE projection of the latent representations of MNIST digits from the validation set, as learned by the CNN Autoencoder (best hyperparameters).

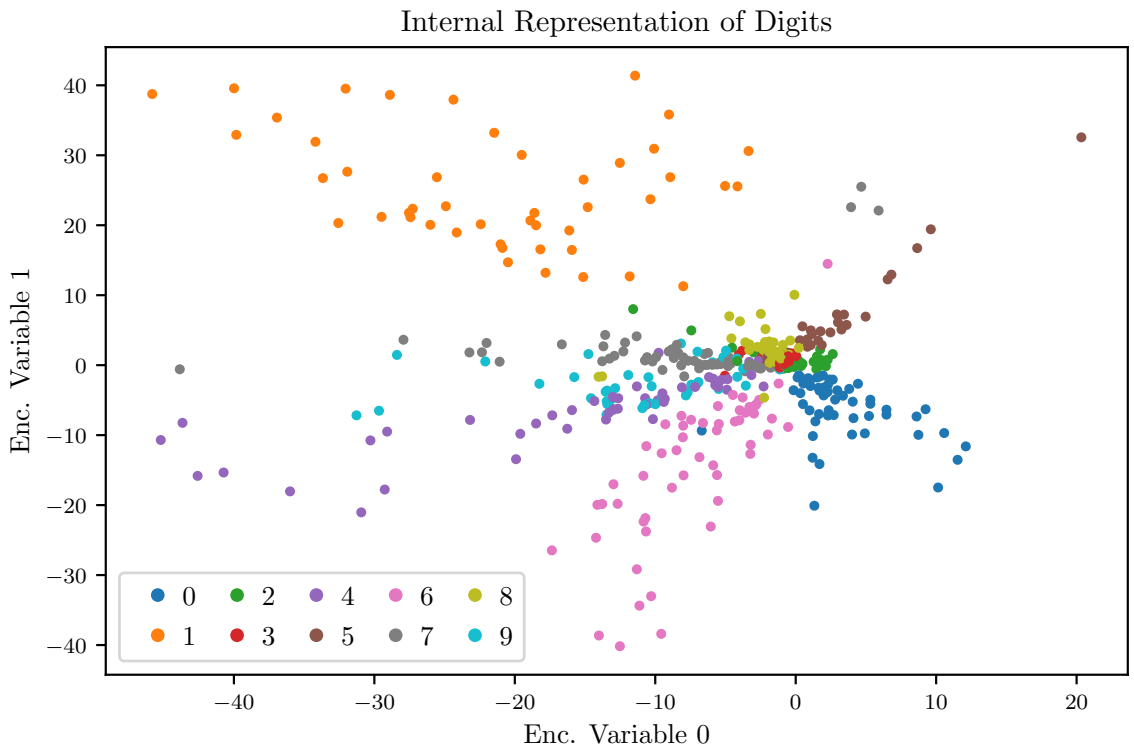


Figure (11) – Direct plot of a few latent representations from a CNN autoencoder with a 2D latent space. Note that images of the same digit form recognizable clusters, meaning that the network is able to learn which images are representing the *same* digit.

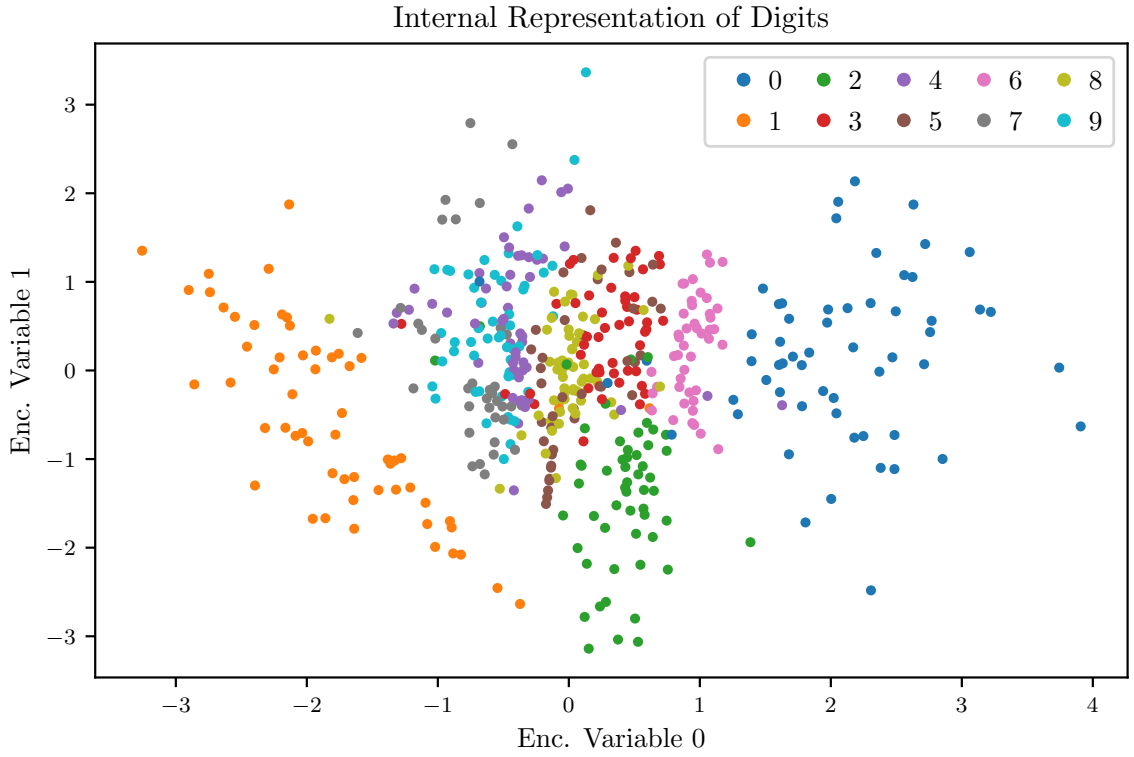


Figure (12) – Direct plot of a few latent representations from a variational autoencoder with a 2D latent space. Note how clusters are more "sparse", and how digits that are visually similar (e.g. 8 and 9) are close together.

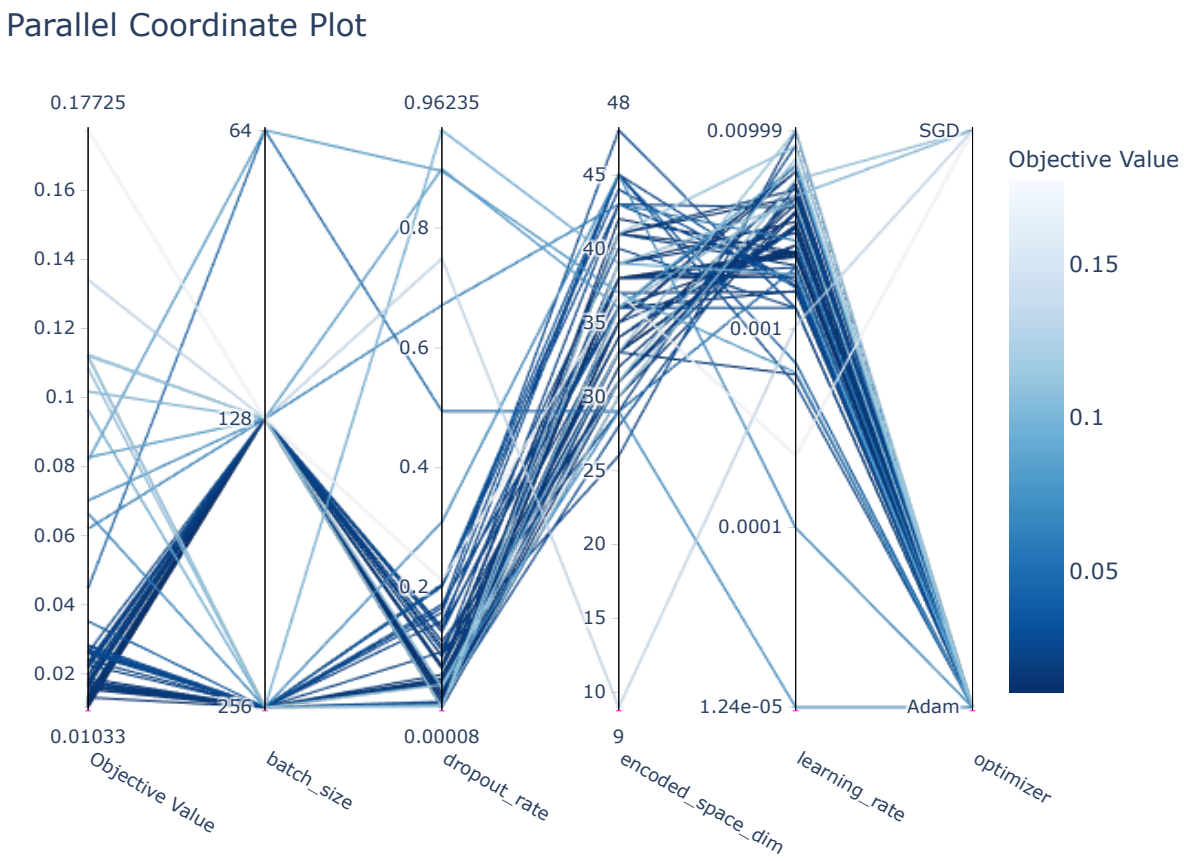


Figure (13) – Log of the hyperparameters search for the CNN autoencoder.