Neural Networks Homework 2: Unsupervised Learning

Manuel Guatto; email: manuel.guatto@studenti.unipd.it
Mat: 2022574

February 11, 2022

1 Introduction

The main goal in this homework is to learn how to implement and test neural network models for solving unsupervised problems. In particular in the first part we will compare different models that will come from the exploration of advanced optimizers and from the application of regularization methods. Next we will try to improve the model by tuning some hyperparameters. Changing task we will build, thanks to the fine tune, a classifier and compare the result with the one tested in the previous lab. At the end we will explore the latent space and build a variational autoencoder to generate new samples. IN particular this report will follow the following order:

1. implement and test (convolutional) autoencoder, reporting the trend of reconstruction loss and some examples of image reconstruction; explore advanced optimizers and regularization methods
2. optimize hyperparameters
3. fine-tune the autoencoder using a supervised classification task
4. explore the latent space structure with PCA and t-SNE techniques
5. implement and test variational (convolutional) autoencoder

2 Building the Autoencoder

2.1 Autoencoder architecture and best optimizer

First of all we define the autoencoder structure. The autoencoder is a Neural Network that is composed of two main part, the encoder and the decoder. In this architecure that we have implemented the encoder and the decoder are symmetrical. In the autoencoder we are going to define we can find three convolutional layer both for encoder and decoder and two linear layers. The activation function that we have chosen is the ReLU. In the following table we can find the specifics of the net.

Network	Layer	In features	Out features	Kernel	Stride	Padding	Out Padding
Encoder	Convolutional	1	8	3	2	1	0
	Convolutional	8	16	3	2	1	0
	Convolutional	16	32	3	2	0	0
	Linear	3*3*32	128	/	/	/	/
	Linear	128	encoded_space	/	/	/	/
Decoder	Linear	encoded_space	128	/	/	/	/
	Linear	128	3*3*32	/	/	/	/
	Convolutional	32	16	3	2	0	0
	Convolutional	16	8	3	2	1	1
	Convolutional	8	1	3	2	1	1

Table 1: Convolutional Autoencoder Architecture

At the beginning we have fixed the encoded space equal to 2, the learning rate equal to 5e-4 and the loss function is the MSE. First of all we want to compare different optimizers to see the effect of their application to the autoencoder.

In this phase we will consider 3 different optimizers:

1. SGD (Standard one)
2. Adamax
3. Adam with l2 regularization

Is possible to find the samples generated during the training in the appendix, respectively at A.1, A.2 and A.3 . Below we can see the comparison between the three optimizers.

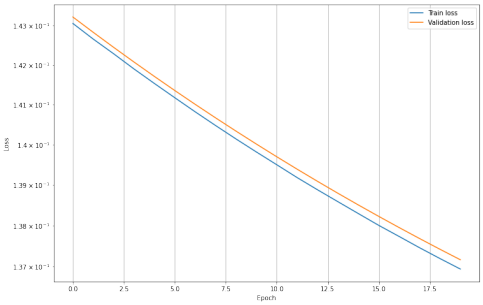

Figure 1: Train and Valid with SGD

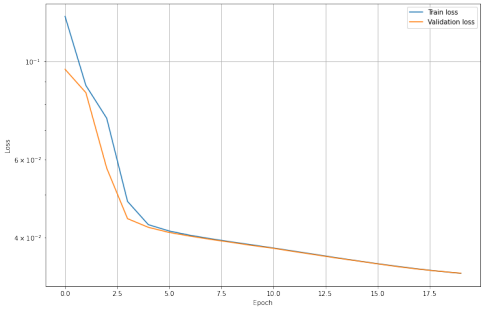

Figure 2: Train and Valid with Adamax

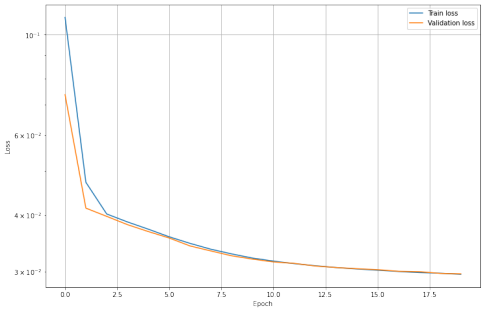

Figure 3: Train and Valid with Adam

Optimizer	Training Error	Validation Error	Generalization Gap
SGD	0.13977	0.13997	0.00019
Adamax	0.04662	0.04378	0.00283
Adam	0.037262	0.03506	0.00220

Table 2: Comparison between optimizers

From the table we can see that with both Adamax and Adam with l2 regularization we have an improvement w.r.t. the training error and the validation error of the network trained with SGD optimizer. During the remaining part of the homework after having seen the results we will use the Adam optimizer.

2.2 Improving the model with regularization and hyperparameters tuning

2.2.1 Batch Normalization

To improve the model we try to add both to the decoder and to the encoder a Batch normalization layer after every convolutional layer present in the network. We can observe the effect of this choice on the resulting image in the appendix A.4. We can see the behaviour of both the training and validation curves in the following image.

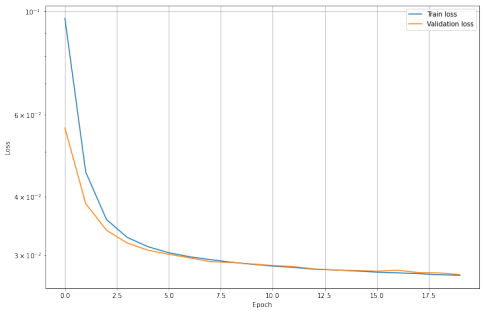

Figure 4: Train and Validation of Autoencoder with Batch Normalization

Optimizer	Regularization	Training Error	Validation Error	Generalization Gap
Adam	l2 + Batch Normalization	0.03327	0.03079	0.00247
Adam	l2	0.037262	0.03506	0.00220

Table 3: Comparison between network with and without batch normalization

From the table we can see that by adding some batch normalization layers to the network we have obtain a reduction of both training and validation error. We also noticed that the generalization gap is increased, but not in a significant way, indeed as shown also by Figure 4 the training error and the validation error are not so far. We can think that since the two are still decreasing we can implement a Early Stopping regularization to find the minimum of both training and validation error but at the same time preventing the overfitting of the network.

2.2.2 Tuning Hyperparameters

To improve the model we have tried to optimize the latent space dimension that previously was fixed to 2 and the learning rate to have a faster convergence. We have used the optuna framework with 5 trials to complete the study. The results are available in the following table.

Trial	Latent Space dimension	Lr	Training Error	Validation Error
1	9	0.04529	0.02110	0.02214
2	126	0.01123	0.01886	0.01748
3	58	0.04175	0.02109	0.02154
4	14	0.04392	0.02111	0.02113
5	3	0.03404	0.02643	0.02628

Table 4: Tuning Hyperparameters: latent space dimension and lr

To complete the task we decided to train with the early stopping regularization method to obtain the minimum train and validation error and to prevent the overfitting.

2.2.3 Training and testing the optimized model with Early Stopping

We fixed the number of early stopping counter equal to 5 and we started to train the network. We reached the following result.

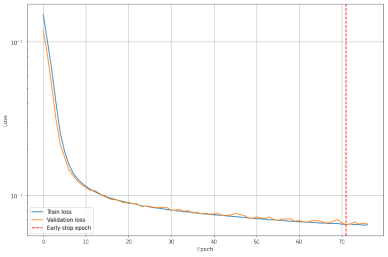

Figure 5: Train and Valid the optimized model with early stopping

Network	Train Error	Validation Error	Gen Gap
Optimized Autoencoder	0.012631	0.01162	0.00100

Table 5: Optimized Network Performances

Testing the network we obtained:
Test Error: 0.00653

3 Fine Tuning

In this section we will exploit the fine tuning part of the homework. We have decided to test and compare two different models:

1. MLP model
    - In this model we have substitute the decoder part of the autoencoder with a mlp network for classification
2. CNN model
    - In this second model we have kept the decoder and we have attached at the end a CNN classifier

Both the models were trained with the early stopping regularization technique to prevent the overfitting. Since one of the task for this section is the comparison between the results of these kind of classifiers and the ones of the first homework we have used both for the MLP and for the CNN the same parameters and architecture of the optimized model of the first homework. Training the networks we obtained the following results.

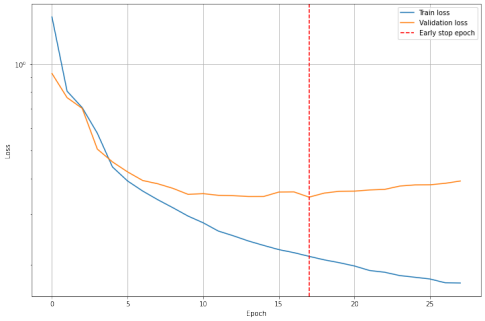

Figure 6: Fine-Tune with MLP

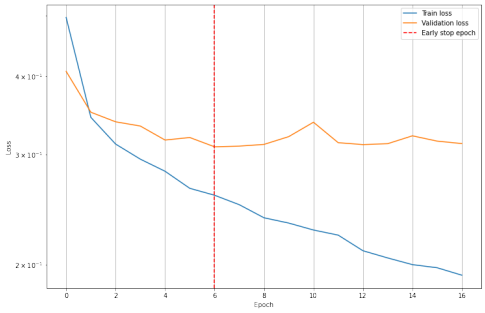

Figure 7: Fine-Tune with CNN

Network	Train Error	Validation Error
Encoder + MLP classifier	0.33930	0.42168
Autoencoder + CNN classifier	0.26041	0.32577

Table 6: Comparison between fine-tune models

Comparing the results with the ones of the previous homework we obtain the table below.

Networks	**MLP**		**CNN**	
	MLP	**Encoder + MLP**	**CNN**	Autoencoder + CNN
Accuracy	86.5500 %	**88.2700 %**	**90.6900%**	89.1900 %

Table 7: Comparison between old and fine-tune models

From the previous table we can see that the results of the comparison between the mlp models are pretty similar even though the "formula" encoder+mlp gains 2 percentage points w.r.t. the old model, also for what concerns the results of the CNN models we see that them are very close. From this results we can conclude two main assumptions:

1. Referring to the MLP model + encoder, the fact that the accuracy is high means that the encoded representation of the images in the latent space let the network classify correctly in a lot of cases this encoded image vector, therefore we can expect that in the PCA we will see the 10 classes very far one from the other.
2. Since also for the Autoencoder + CNN we have obtained a good result this means that the reconstruction of the images is good enough to classify correctly almost all the classes reaching the level of a CNN classifier that elaborates not the reconstruction but original images.

4 PCA and t-SNE

Now we use two techniques to explore the latent space:

- PCA
- t-SNE

The results are the following.

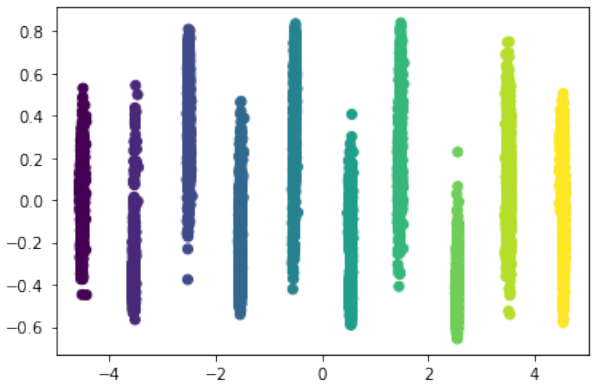

Figure 8: View of latent space using PCA

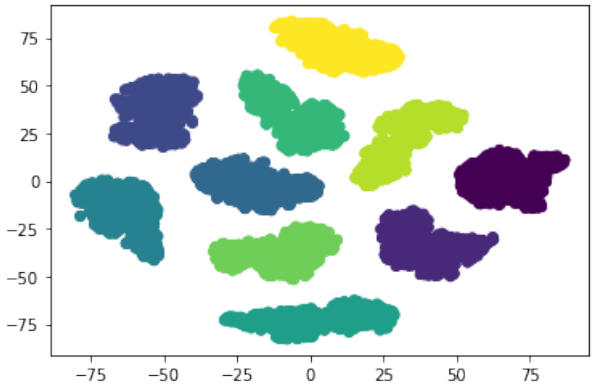

Figure 9: View of latent space using t-SNE

In the previous two images we can see that the 10 classes are well separated and therefore from this follows also the good accuracy in the classifier. Now we generate three samples by passing a random tensor to the decoder. The generate samples are represented in the below images.

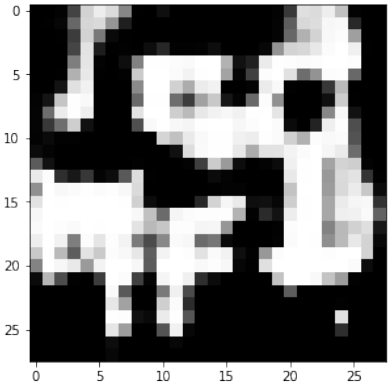

Figure 10: Generated Sample 1

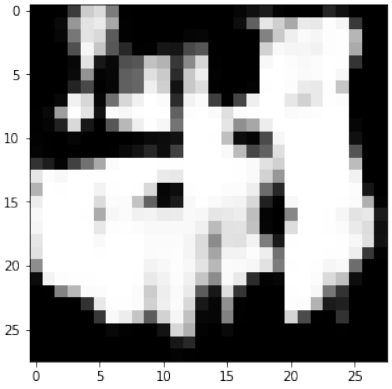

Figure 11: Generated Sample 2

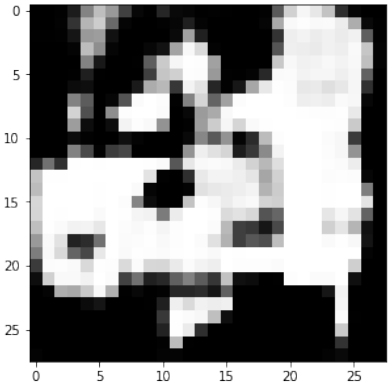

Figure 12: Generated Sample 3

5 Varaitional Autoencoder

The last task in this homework is to build a variational autoencoder. First of all we fix the learning rate at 1e-3 and we use as loss function the Kullback Liebler loss. The architecture of the variational autoencoder is resumed in the following table.

Network	Parts	Layer	In features	Out features	Kernel	Stride
Encoder	Cnn Encoder	Convolutional	1	16	5	1
		Convolutional	16	32	5	1
	Mu Encoder	Linear	12800	256	/	/
	Var Encoder	Linear	12800	256	/	/
Decoder	Linear Decoder	Linear	256	12800	/	/
	CNN Decoder	Convolutional	32	16	5	1
		Convolutional	16	1	5	1

Table 8: Variational Autoencoder Architecture

Training and testing the network we obtained:

```
Epoch 0: Loss 58218.9609375
Epoch 1: Loss 56700.75
Epoch 2: Loss 55962.09765625
Epoch 3: Loss 52488.04296875
Epoch 4: Loss 51630.42578125
Epoch 5: Loss 51998.40234375
Epoch 6: Loss 51904.30859375
Epoch 7: Loss 49402.578125
Epoch 8: Loss 50862.8828125
Epoch 9: Loss 50066.16796875
Epoch 10: Loss 49790.4921875
Epoch 11: Loss 49609.0703125
Epoch 12: Loss 49542.2109375
Epoch 13: Loss 49922.3828125
Epoch 14: Loss 49888.98046875
```

Figure 13: Vae train loss

```
TEST ERROR: 62302.76171875
```

Figure 14: Vae test loss

Trying to generate a new image starting from a image of the dataset we get the following result.

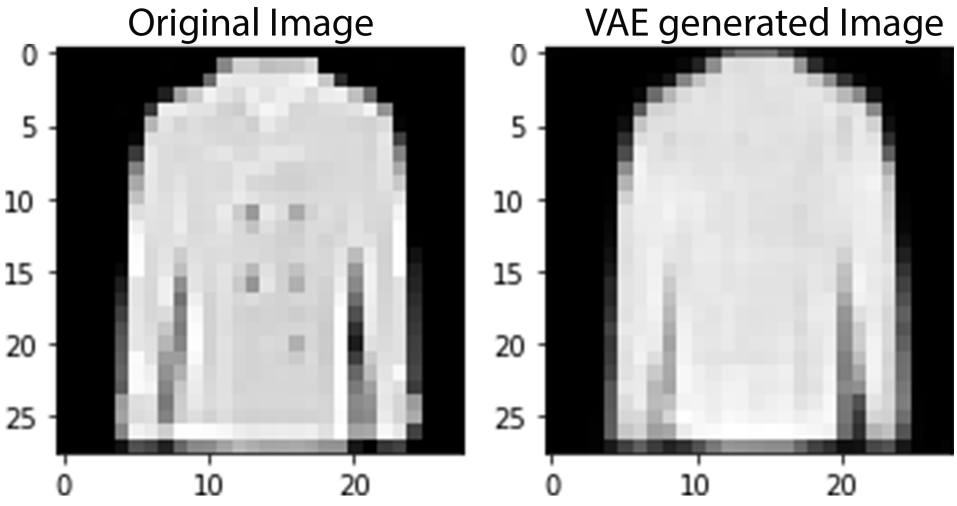

Figure 15: Original Image and VAE generated image

A Appendix

A.1 Autoencoder Trained with SGD optimizer

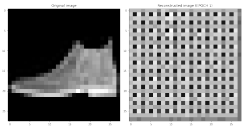

Figure 16: 1st Epoch with SGD

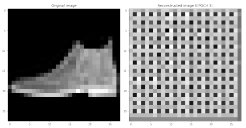

Figure 17: 3rd Epoch with SGD

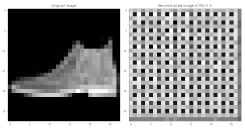

Figure 18: 5th Epoch with SGD

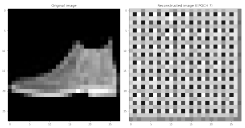

Figure 19: 7th Epoch with SGD

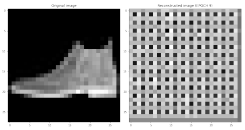

Figure 20: 9th Epoch with SGD

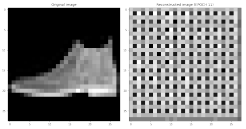

Figure 21: 11th Epoch with SGD

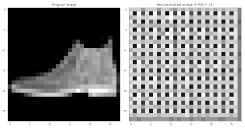

Figure 22: 13th Epoch with SGD

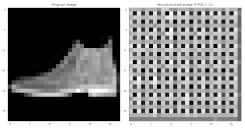

Figure 23: 15th Epoch with SGD

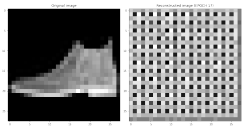

Figure 24: 17th Epoch with SGD

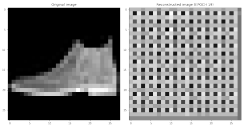

Figure 25: 19th Epoch with SGD

A.2 Autoencoder Trained with Adamax optimizer

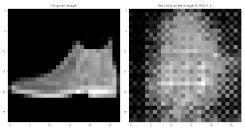

Figure 26: 1st Epoch with Adamax

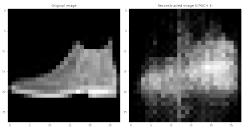

Figure 27: 3rd Epoch with Adamax

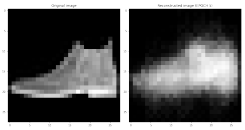

Figure 28: 5th Epoch with Adamax

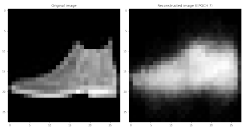

Figure 29: 7th Epoch with Adamax

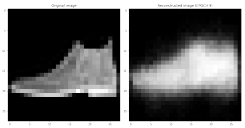

Figure 30: 9th Epoch with Adamax

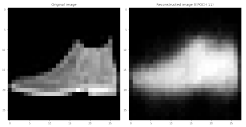

Figure 31: 11th Epoch with Adamax

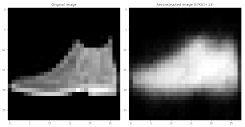

Figure 32: 13th Epoch with Adamax

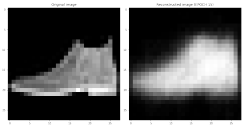

Figure 33: 15th Epoch with Adamax

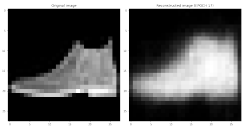

Figure 34: 17th Epoch with Adamax

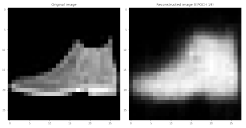

Figure 35: 19th Epoch with Adamax

A.3 Autoencoder Trained with Adam optimizer and with l2 regularization

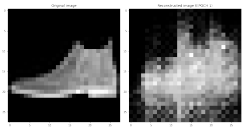

Figure 36: 1st Epoch with Adam

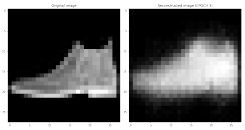

Figure 37: 3rd Epoch with Adam

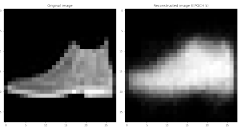

Figure 38: 5th Epoch with Adam

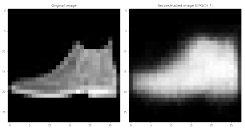

Figure 39: 7th Epoch with Adam

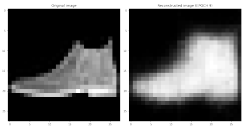

Figure 40: 9th Epoch with Adam

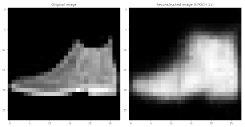

Figure 41: 11th Epoch with Adam

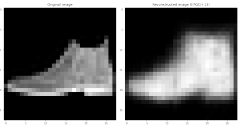

Figure 42: 13th Epoch with Adam

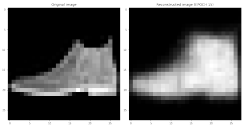

Figure 43: 15th Epoch with Adam

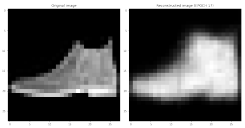

Figure 44: 17th Epoch with Adam

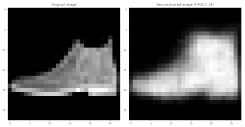

Figure 45: 19th Epoch with Adam

A.4 Autoencoder with Batch Normalization

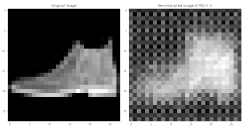

Figure 46: 1st Epoch with Batch Normalization

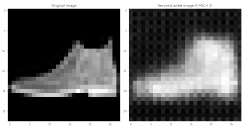

Figure 47: 3rd Epoch with Batch Normalization

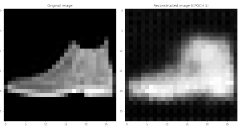

Figure 48: 5th Epoch with Batch Normalization

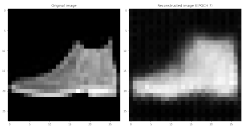

Figure 49: 7th Epoch with Batch Normalization

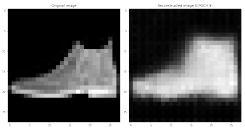

Figure 50: 9th Epoch with Batch Normalization

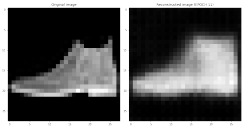

Figure 51: 11th Epoch with Batch Normalization

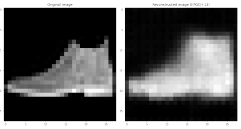

Figure 52: 13th Epoch with Batch Normalization

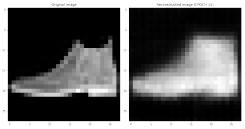

Figure 53: 15th Epoch with Batch Normalization

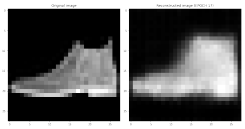

Figure 54: 17th Epoch with Batch Normalization

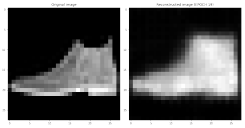

Figure 55: 19th Epoch with Batch Normalization