Neural Networks and Deep Learning 2021

Homework 2: Unsupervised Deep Learning

Matteo Grandin

January 2022

1 Introduction

The goal of this homework is to implement and test a convolutional autoencoder, testing different hyperparamters using grid or random search. It is also required to fine tune the unsupervised models using a supervised classification task. The latent space should be explored and used to generate new samples.

Hyperparamters were tuned using a manually implemented grid search, in a similar way to the previous homework. Dropout and weight decay were considered for regularization. For the supervised tuning different strategies were considered. The first approach was to use a 10-dimensional latent space and force it to represent the class output, this strategy had the advantage of making generating new samples from a specific class very easy, but had the main downside of calculating a softmax in the middle of the network thus making training very hard, moreover the encoder and decoder are basically retrained and the encoder becomes a simple classifier. The second approach was to use a single fully connected layer, a matcher, from the latent space to the classification output, this approach solves all the previous downsides but makes picking new samples difficult in higher dimensions. For this reason it was first implemented using a 2-dimensional latent space where samples can plotted on a plane. Finally, this approach was extended to higher dimensional latent spaces, the problem of generating samples from a specific class was solved by calculating the mean and std of the encoded samples for each class. The

report will focus on this last approach. The latent space was visualized both using PCA and tSNE.

For the second part a simple GAN and a Auxiliary-Classifier GAN have been implemnted and tested on the MNIST datasets. The code is organized in notebooks; the training, grid search and cross validation are coded manually in order to have more control over the software and have a deeper understanding of what the code is doing. Almost all the code testing is done on a local machine using a dedicated graphic card (NVIDIA GeForce GTX 950M).

2 Methods

2.1 Convolutional Autoencoder

Both the encoder and the decoder are subdivided in layers. The encoder is composed of 3 convolutional layers with Relu activations, followed by 2 fully connected layers with dropout and Relu activations. No activation function was used for the final layer. The decoder is specular with respect to the encoder: 2 fully connected layers with dropout followed by 3 transposed convolution to recreate the output image.

A simple grid search has been manually implemented to compare the hyperparameters combinations, and k-fold cross validation has been used to validate the models. The implementation is very similar to the one of the first homework: also in this case the network weights are not reset at each fold to speed up training and test more combinations. The list of parameters selected for the grid search is the following:

- Encoded space dimension: 10 and 30, smaller sizes were also considered but were discarded to reduce the number of combinations
- Optimizers: Adam, SGD with momentum and RMSprop were also tested but both performed very similar or slightly worse than Adam.
- Learning rates: 5×10^{-3}, 1×10^{-3}, 5×10^{-4}
- Weight decay: 1×10^{-5}, 1×10^{-6}, 0

Dropout with probability 0.5 was used to introduce regularization. Different losses were also tested: Huber loss performed similarly to MSE loss, L1 achieved good results but with a different style of reconstruction. Samples from some of this losses are shown in the Appendix, from now on only MSE will be considered.

In order to use classification to perform finetuning, focus needs to be shifted onto a single combination of parameters: an encoded space dimension of 10, Adam optimzier with 1×10^{-3} as learning rate and no weight decay. The model is initially trained for 150 epochs to achieve good reconstruction capabilities; then, the single layer matcher net is trained for a few epochs (10, using Cross-Entropy loss on the one-hot encoding of the labels), without updating encoder and decoder, in order to initialize to good values the single layer weights. The final step is to train everything together, encoder, decoder and single layer matcher. To do this 2 losses are backpropagated into the networks(this is possible since gradients are accumulated and not overwritten): the reconstruction loss (MSE, L1, Huber) and the classification loss (Cross-Entropy). In this way the encoder is optimized for both tasks.

As already mentioned in the introduction, in order to generate new samples, the best samples for each class were selected, based on the confidence of the classifications. The encoded representation of each sample were then averaged to get a prototypical encoding for each class, from it any number of samples can be generated adding Gaussian noise.

2.2 GAN and AC-GAN

A Generative Adversarial Network is a deep learning architecture composed of two models: a generator, that takes as input noise (and class information in the case of an Auxiliary classifier GAN) and generates as output an image in this case; and a discriminator, that takes as input an image and outputs a 1 if the image is from the original dataset and a 0 if it's generated by the generator network (an AC-GAN also outputs class information). The generator is a fully connected 3 hidden layers network, it uses LeakyRelu in the internal layers and Tanh in the output layer to output a valid pixel value. The discriminator network is also a 3 hidden layers fully connected architecture

with Dropout and LeakyRelu as activation function. LeakyRelu is very important because the gradients need to propagate from the top of the discriminator all the way to the first layers of the generator, this particular activation function helps in preventing the gradient vanishing problem.

In order to train the network generator and discriminator needs to be trained one after the other: the discriminator is trained by evaluating a real batch of images and a fake batch of images, with the same classification labels and with their respective real/true labels, both the classification loss and the discrimination loss are propagated backward. The generator is trained by generating a batch of images and using the loss of the discriminator evaluated against a batch of real labels, this implementation is not the original version described by J. Goodfellow, but is extremely effective in practice.

Training requires a lot of epochs and it's extremely sensitive to hyperparamters, learning rate needs to be slow and the losses of the 2 adversarial networks need to follow a very specific pattern in order for the training process to work. Several tricks and several hours were needed in order to get satisfactory results.

3 Results and Network Analysis

3.1 Convolutional Autoencoder

The final model was trained for 150 epochs, the encoded space is 10-dimensional. It achieves a loss of 0.030 86 on the train set and of 0.030 95 on the test set, the trend of the reconstruction losses is shown in Figure 7. After training the single layer matcher, but without finetuning, the model is not a very accurate classifier achieving 33.1% accuracy on the train set and 32.6% on the test set. Example of reconstructions before and after the supervised tuning can be found in the Appendix, it's hard to notice any change in recostruction capabilities. The classification accuracy on the other hand drastically improves to 91.0% on the training dataset and 87.7% on the test data, meaning that there is a slight overfit on the train points. The accuracy is similar but slightly worse with respect to the convolutional network of homework 1 (90%); the learning speeds are

also very similar but it's difficult to derive conclusions since the learning speed is very dependent on hyperparameters like the learning rate.

The encoded space has been analyzed both using PCA and tSNE. PCA is not very insightful because, although some classes are visible and clearly separated, a lot of the complexity is lost in the projection. tSNE is more useful because it shows how some classes are very well separated and others are close together and sometimese partially mixed; this is what we expect since some classes were hard to distinguish also from the convolutional classifier of the first homework (it's difficult for humans too).

The new samples generated are mostly correct, however there is little variation between the samples despite the fact that it's very easy to end up in a different class adding some noise. This is probably a consequence of the type of loss, which is not very well suited for capturing details and small variations.

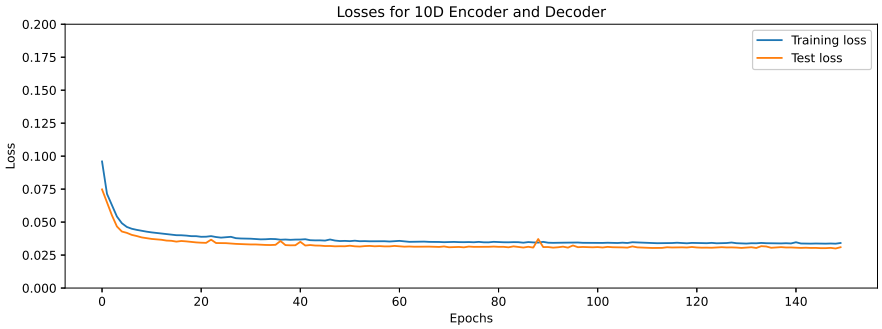

Figure 1: Trend of reconstruction loss

3.2 AC-GAN

The AC-GAN achieves very good results: some of the generated digits images are almost indistinguishable from real ones (shown in the Appendix), and it's possible to generate correct samples for every digit with a good inter-class variation. The classification part of the GAN greatly helps in reducing the problem of mode collapse. The images can sometimes present some artifacts, and the shapes can be discontinuous; this issues could have probably been solved using a convolutional architecture, however, given the difficulties

and the time required for training this possibility has not been tested yet.

From the plot of the losses (Figure 2) it's possible to see the pattern required for an healthy training, where the generator loss peaks at the beginning while the discriminator is at its minimum, and then slowly improves while the discriminator weakens. After a fixed amount of episodes the discriminator is progressively trained more than the generator to achieve better results, the 3 little spikes in the loss correspond to the moments the discriminator starts getting trained more.

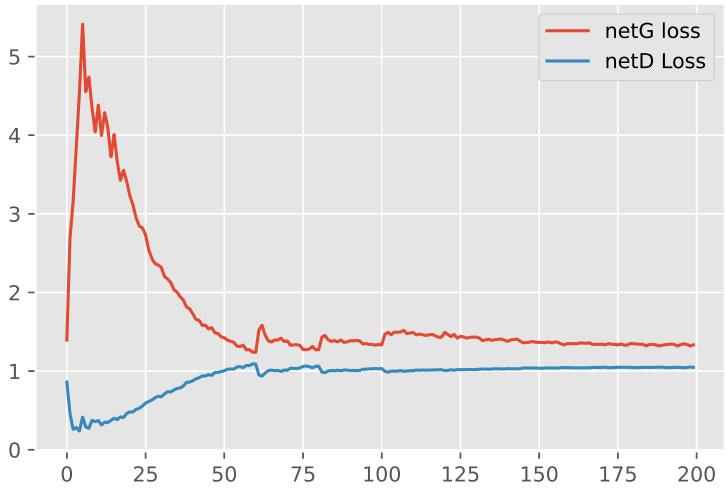

Figure 2: Generator and discriminator losses

4 Appendix

4.1 Convolutional Autoencoder

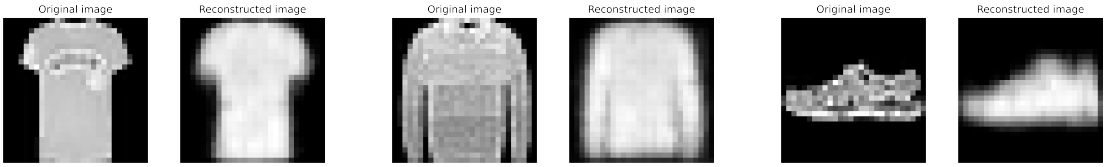

Figure 3: Image reconstruction examples, MSE loss

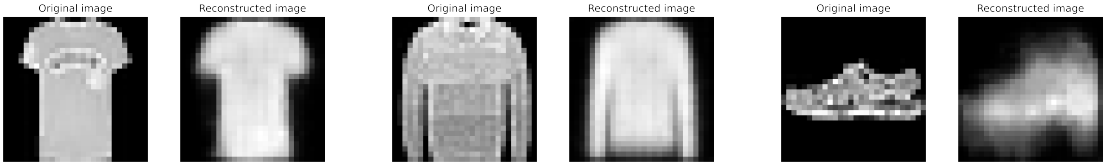

Figure 4: Image reconstruction examples after finetuning, MSE loss

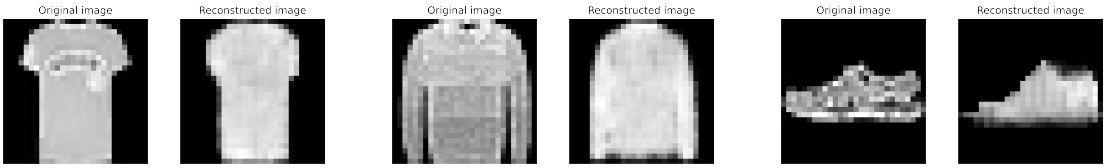

Figure 5: Image reconstruction examples, L1 loss

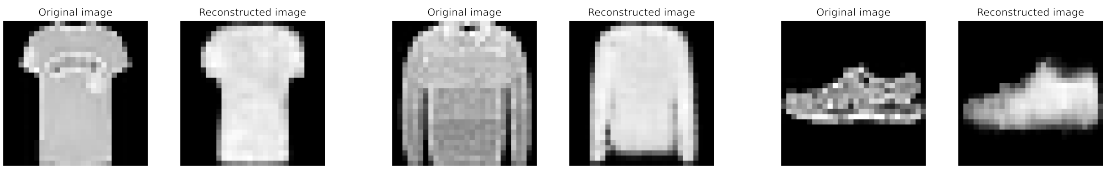

Figure 6: Image reconstruction examples after finetuning, L1 loss

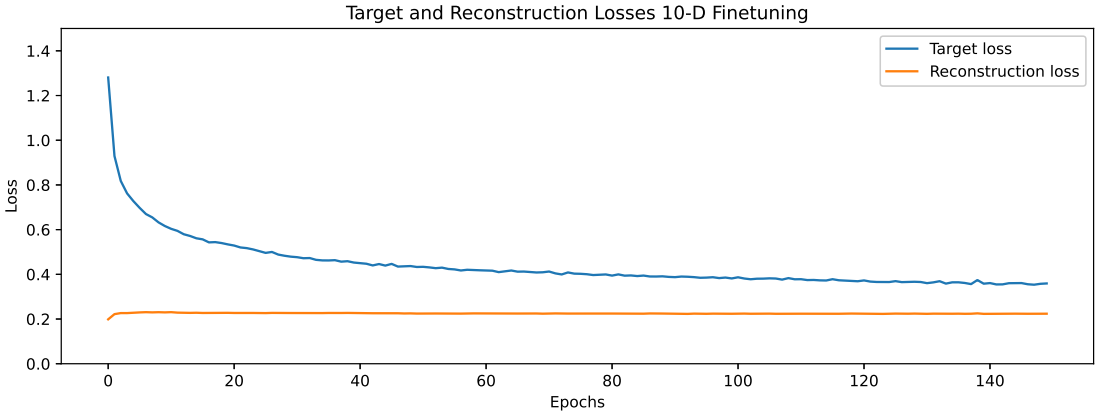

Figure 7: Trend of classification and reconstruction losses during finetuning.

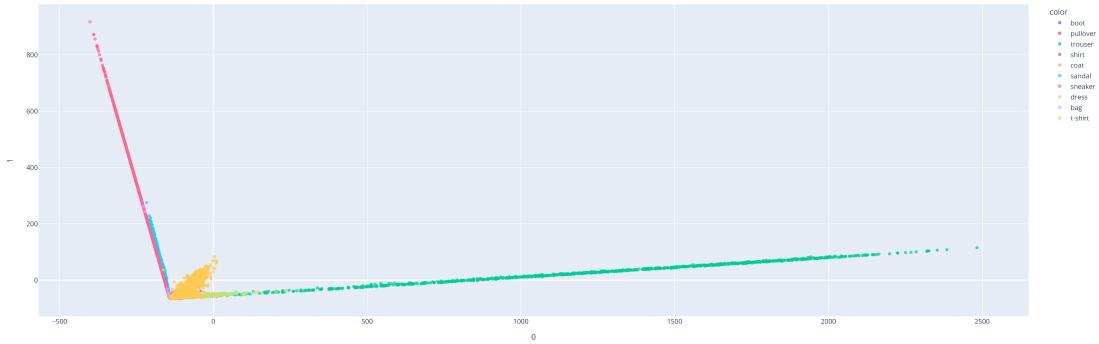

Figure 8: PCA

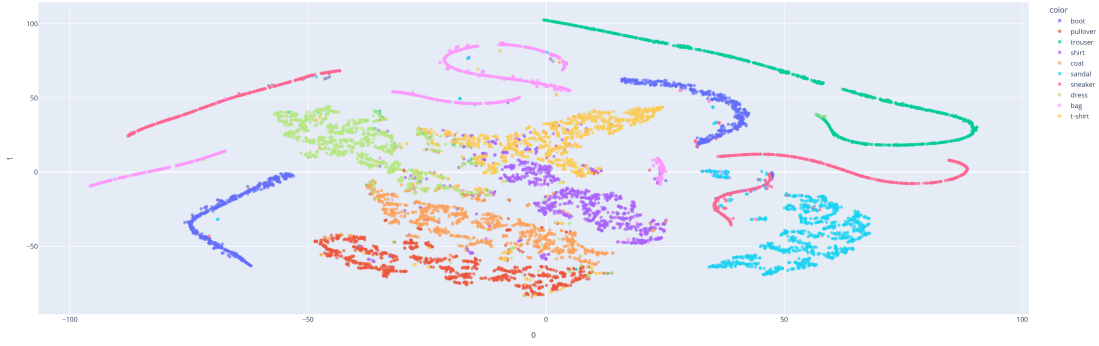

Figure 9: tSNE

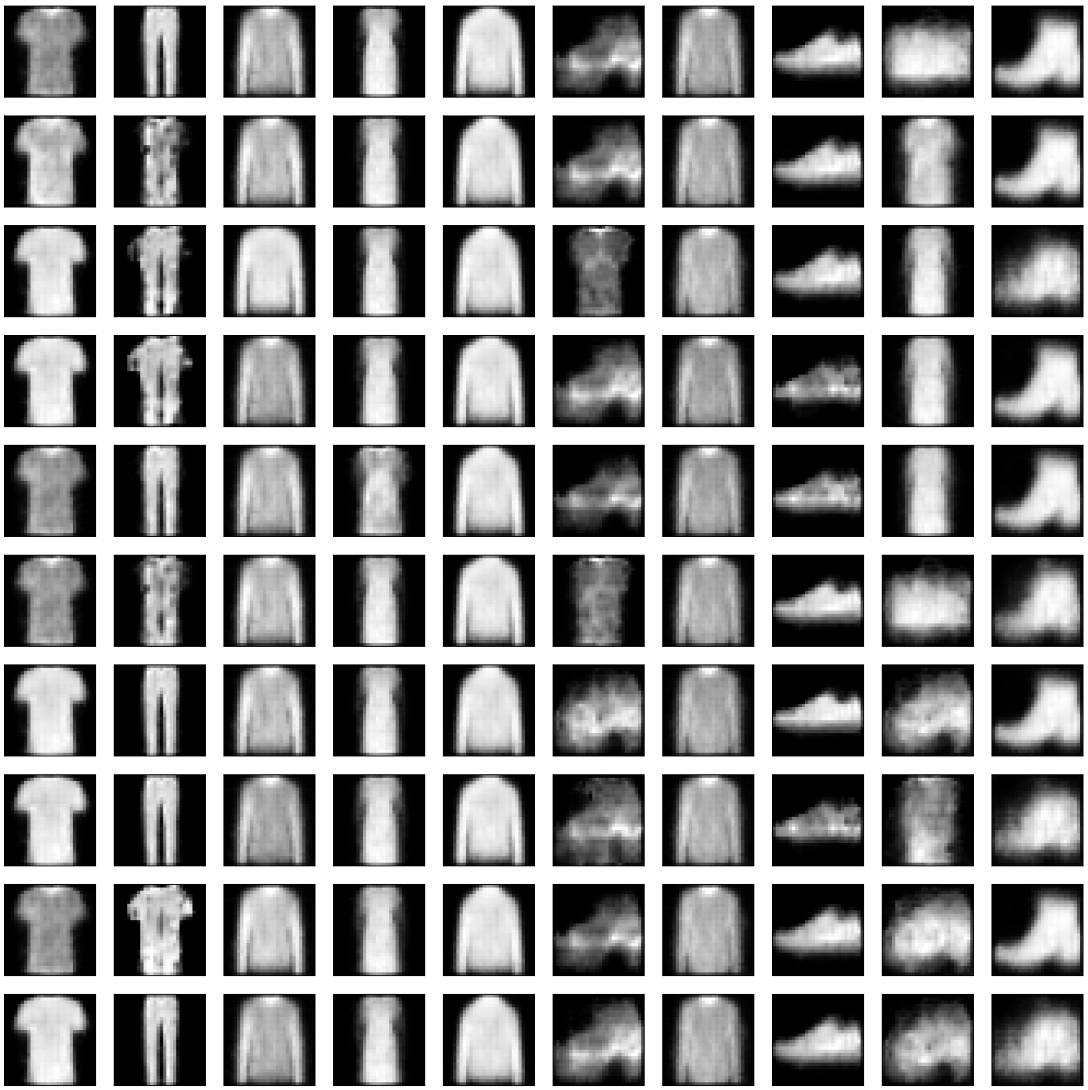

Figure 10: Generated samples for: t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, boot

4.2 AC-GAN

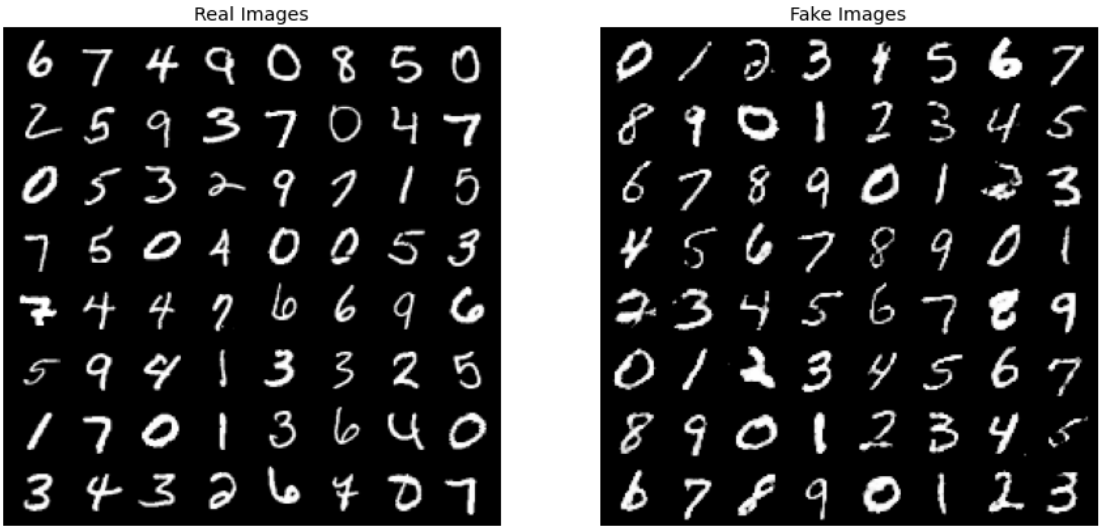

Figure 11: Real vs AC-GAN generated digits

4.3 Additional Images

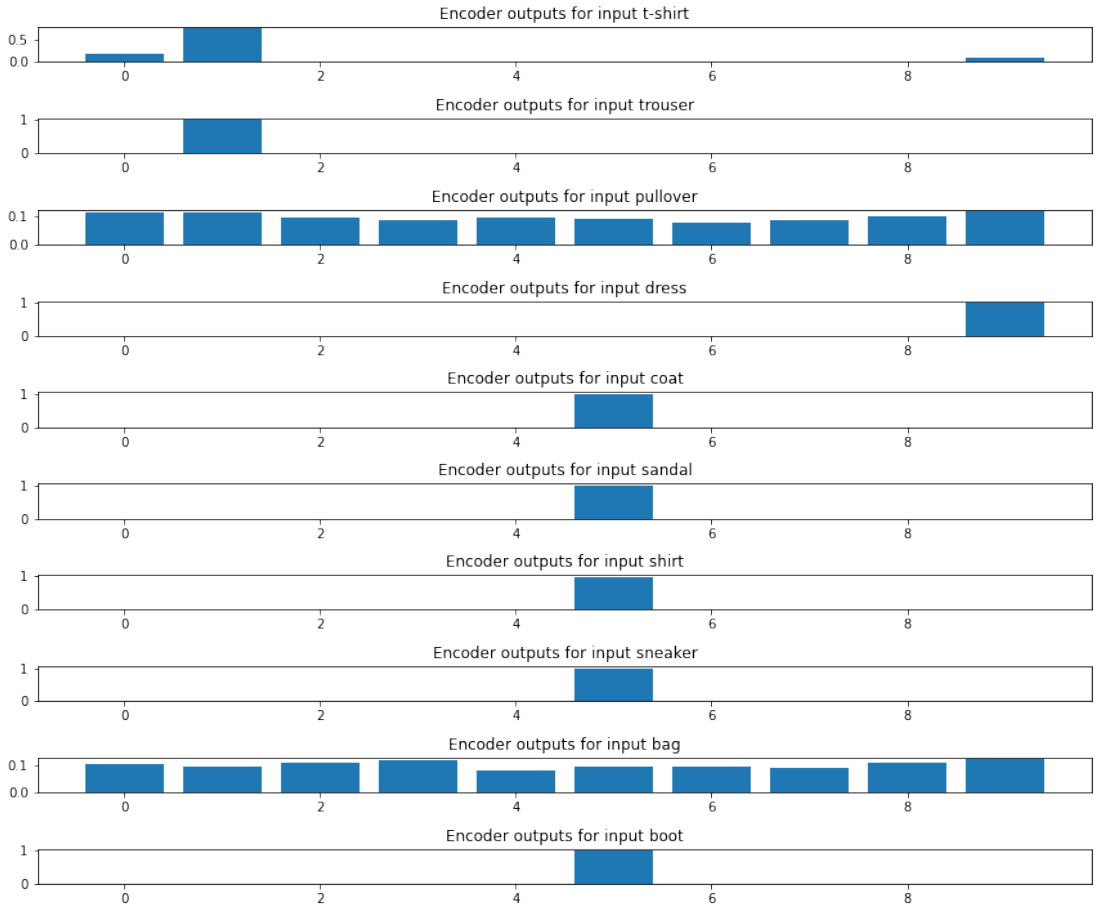

Figure 12: Encoder output activations before tuning (softmax)

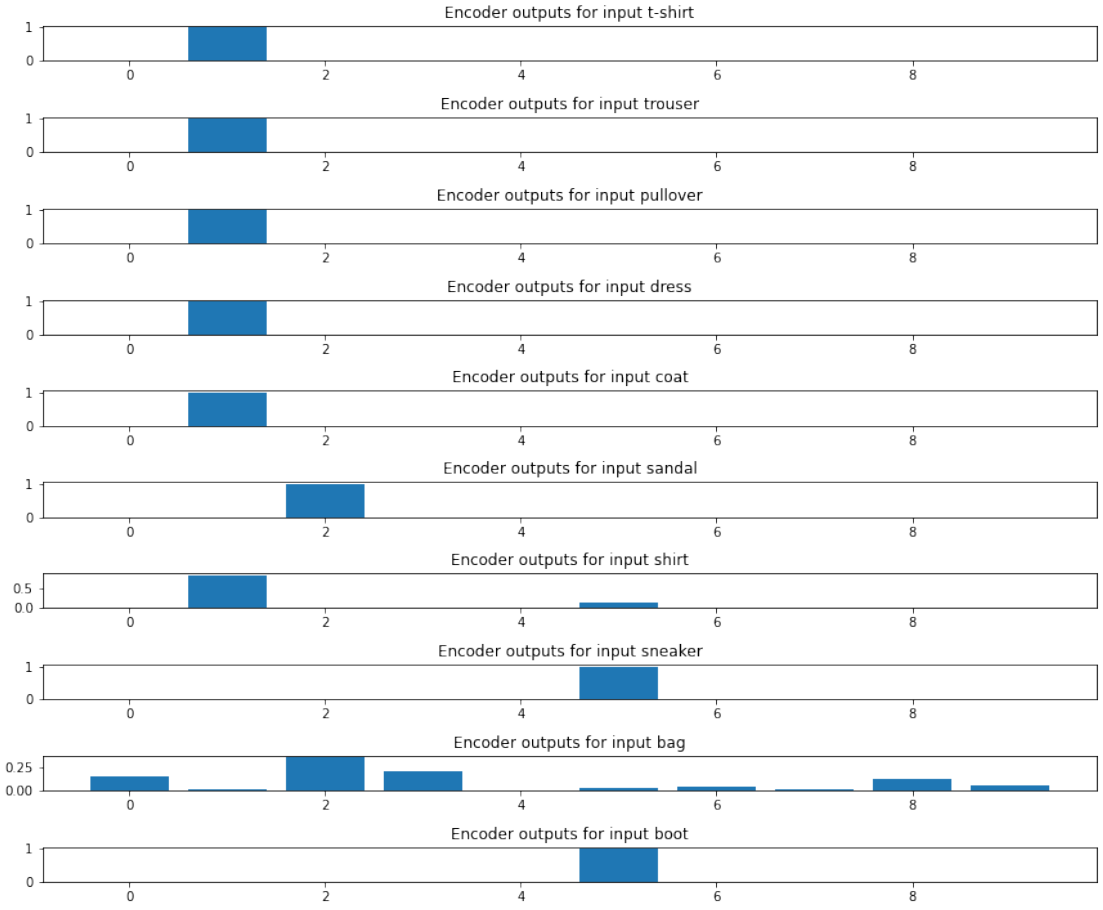

Figure 13: Encoder output activations after tuning (softmax)

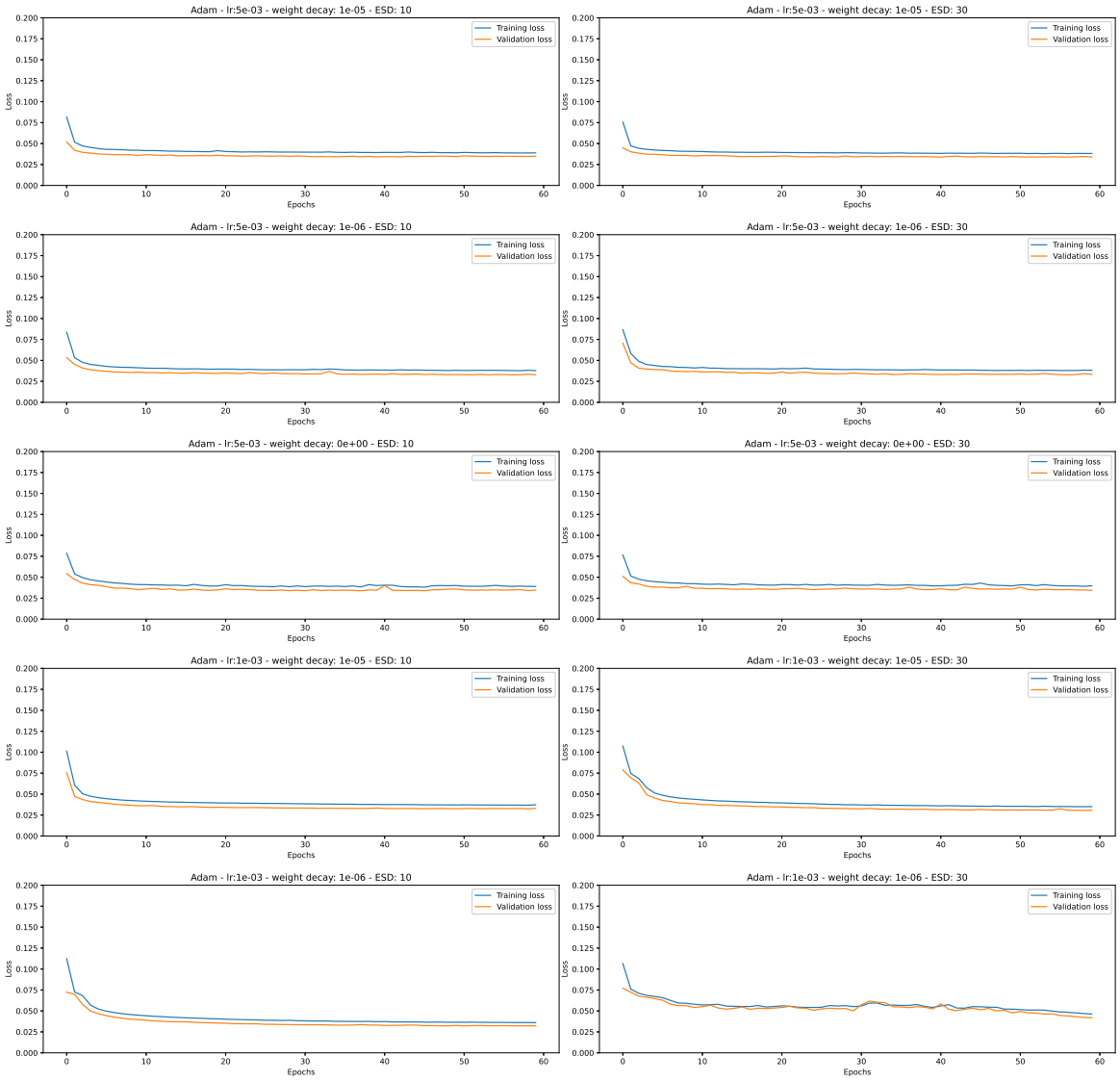

Figure 14: Grid search results 1/2

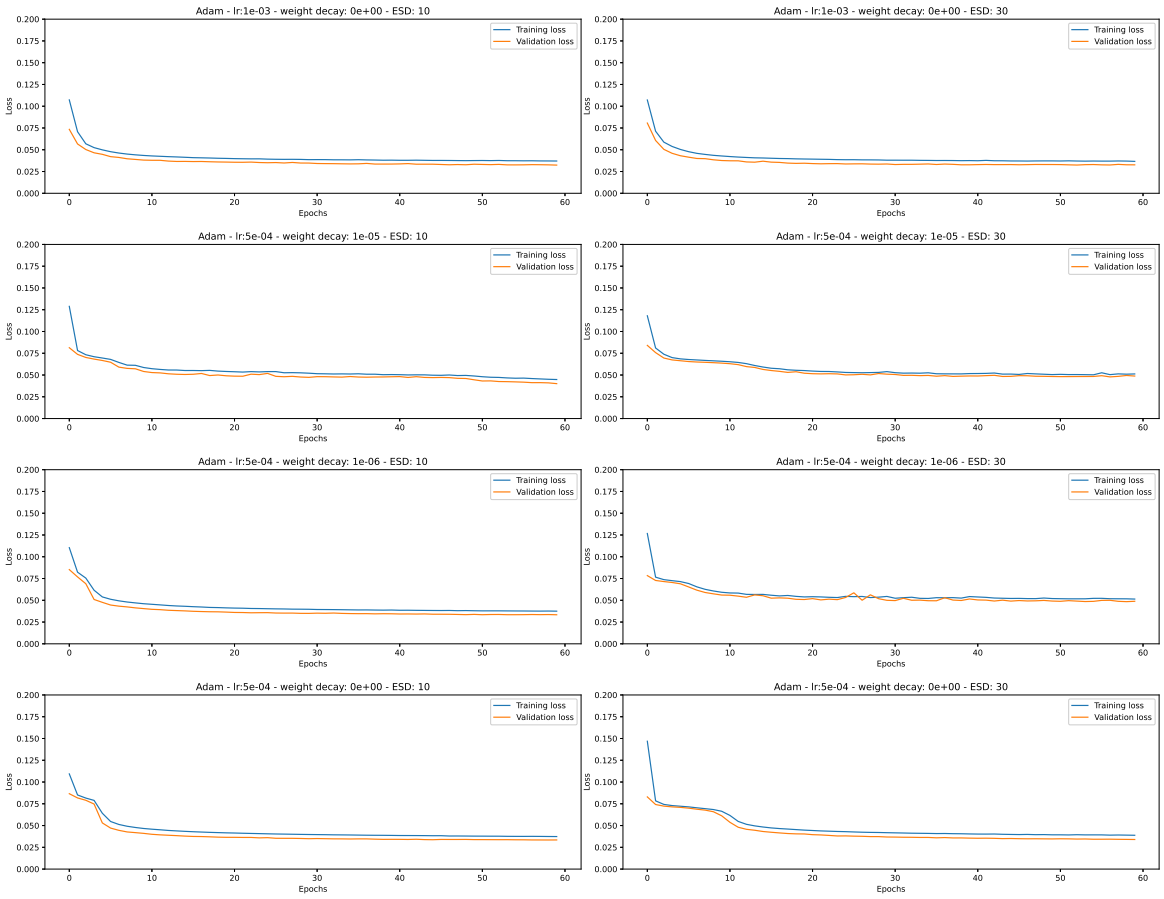

Figure 15: Grid search results 2/2