Neural Networks and Deep Learning 2021

Homework 1: Supervised Deep Learning

Matteo Grandin

July 2021

# 1 Introduction

The homework is divided in two main tasks. The *regression task* consists in a simple function approximation problem for which a training dataset and a test dataset are provided. The *classification task* consists in a image recognition problem, where the goal is to correctly classifies the image from the FashionMNIST dataset. The regression task is addressed using a simple fully connected network, while a convolutional architecture is used for the classification task. Hyperparameters tuning is implemented using grid-search. The combinations are evaluated using cross-validation on the training datasets, analyzing validation losses and outputs. After tuning, the focus is shifted over a single combination, the nets are tested on the test datasets and performance is evaluated. Finally, networks are analyzed showing weights histogram and activation profiles, in the case of the convolutional network, the receptive fields of the convolutional filters are shown.

The code is organized in notebooks, one for each tasks; the training, grid search and cross validation is 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 Regression Task

### 2.1.1 Network Architecture

For the regression task a simple, 2 hidden layers, fully connected neural network has been implemented. The size of the hidden layers, as well as the type of activation function of the internal layers (the output layer does not need an activation function, we don't want to limit the output) and the dropout probability can all be chosen when initializing the network. These parameters will be some of the ones used in the grid-search.

### 2.1.2 Hyperparamters Tuning

A simple grid search has been manually implemented to compare the hyperparameters' combinations of the architecture. To achieve an effective tuning a lot of values for each parameter should be considered, but due to training time constraints, it quickly becomes very time consuming to explore a lot of combinations. The grid-search code has been written to deal with any amount of values, but a limited amount of combinations has been considered and some combinations has been discarded. With more time the hyperparameters' space could be explored better. A list of the parameters for which different values has been evaluated is shown here:

- Activation functions: Tanh and Sigmoid; Relu was discarded because the given samples are better approximated by smooth functions, Relu generates outputs with sharp corners.
- Loss functions: L1 and L2 losses, Huber loss was also tested and achieved results similar to the MSE loss, therefore it was removed from the combinations.
- Size of the hidden layers: 16 and 32, a smaller network is a way to perform regularization; since we are dealing with a small problem and a simple function, a big and deep neural net would be overkill for the task.

- Optimizers: Adam and SGD with momentum, simple SGD was also considered at first but it performs very poorly versus the alternatives.
- Learning rates: a small range of l.r. from $1 \times 10^{-2}$ to $8 \times 10^{-4}$

Dropout was also considered as an hyperparameter and regularization method but the results has been unsatisfactory, the main reason why is that we are dealing with a regression task and the activation of a single neuron can be extremely important for predicting the output. Dropout makes a lot more sense in a classification task where there are multiple ways to decide the class of an image, this case will be explored in the following sections.

To evaluate the models a classic k-fold cross validation setup has been coded; each model has been trained with a batch size of 8 and for 1000 epochs. Early stopping has been used for the final model training in order to avoid overfitting.

## 2.2 Classification Task

### 2.2.1 Network Architecture

For the classification task a convolutional neural network (CNN) with 2 convolutionals layers followed by 2 fully connected layers has been implemented. In the convolutional part the activation function is Relu followed by a pooling layer. Each linear layer also uses Relu as activation function and it's followed by a dropout layer for regularization. In order to improve the classification capabilities, the integer label corresponding to the class in the FashionMNIST dataset is always converted in a vector of zeros with only the element associated with the label set to 1 (one-hot encoding); it's therefore possible to use the cross-entropy loss and it's much easier to evaluate the accuracy of the model.

### 2.2.2 Hyperparamters Tuning

Hyperparameters tuning has been implemented in a very similar way to the regression task. The main difference is that in this case, the network weights are not reset each fold of the k-fold cross validation, and the model is not retrained on the whole training dataset

at end. Instead, the model is continuously trained through the folds and the effective number of epochs gets multiplied by the number of folds. This has the disadvantage of having a less accurate validation loss, but drastically reduces the training time of each parameter combination and allows for more combinations to be tested. As before, the parameters space is small but the code allows for more exploration with more time. The list of hyperparameters is the following:

- Optimizers: Adam and SGD with momentum
- Learning rates: a small range from $5 \times 10^{-4}$ to $5 \times 10^{-2}$
- Batch sizes: 256 and 1024

Each model has been trained for a total of $20 \times 4$ epochs ($epochs\_per\_fold \times folds$).

# 3 Results and Network Analysis

## 3.1 Regression

The losses and the network output for some of the combinations can be found in the appendix. The parameters for the final training are: $Tanh$ (activation function), 16 (size of the hidden layers), 8 (batch size), Adam optimizer with $2 \times 10^{-3}$ as learning rate. The network achieves a test loss of 0.123 on the test set, from Figure 1 we can see that the test set is very well approximated except in the section where training data points are missing. This was to be expected since training and test datasets are clearly not sampled in the same way from the underline function: the training dataset has a much higher variance and lacks samples in some specific sections; the test dataset, on the other hand, is very low variance and samples are taken more uniformly. This allows to test the generalization capabilities of the model, which in this case are quiet good, it's possible to notice how different parameters' choices strongly overfit the train datapoints and can't approximate the sections without datapoints.

The weights histogram for the 3 network layers and the last layer activations for different inputs are shown in the Appendix. Even with a small network it's hard to

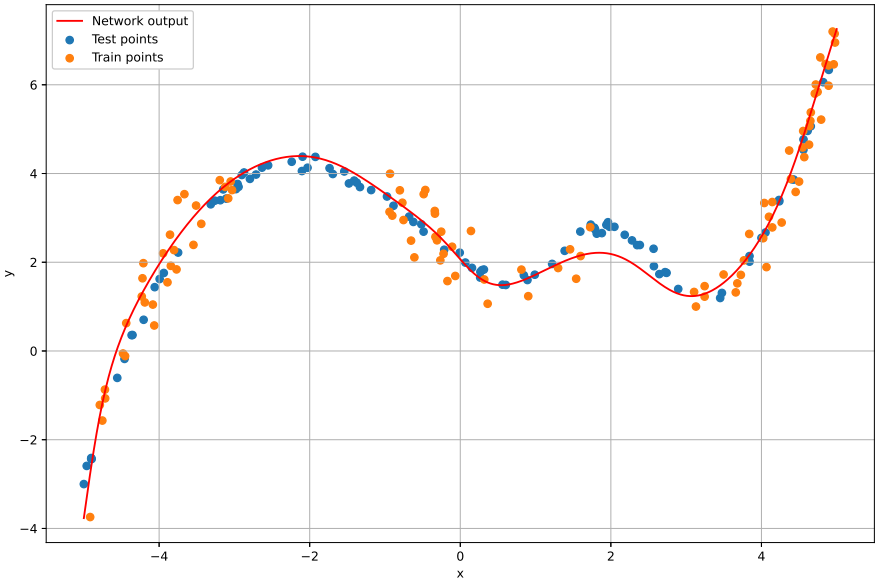


Figure 1: Final regression model output

interpret how the weights influence the output.

### 3.2 Classification

The final model is trained using Adam as optimizer with a learning rate of $3 \times 10^{-3}$. The model trained for 74 epochs before early stopping. It achieves a 90.22 % accuracy on the test set (92.92 % on the train set). Some examples of predictions on the test dataset are shown in Figure 2. The confusion matrices, shown in the Appendix, are very similar for training and test datasets; from this and from the small accuracy difference it can be deduced that the model overfits a little but not very much. The convolutional kernels and the convolutional layers activations are shown in the Appendix; once again it's very difficult to interpret the features encoded. The weights histogram and the linear section activations can also be found in the Appendix, from the linear activations it's possible to recognize some patterns between similar classes.

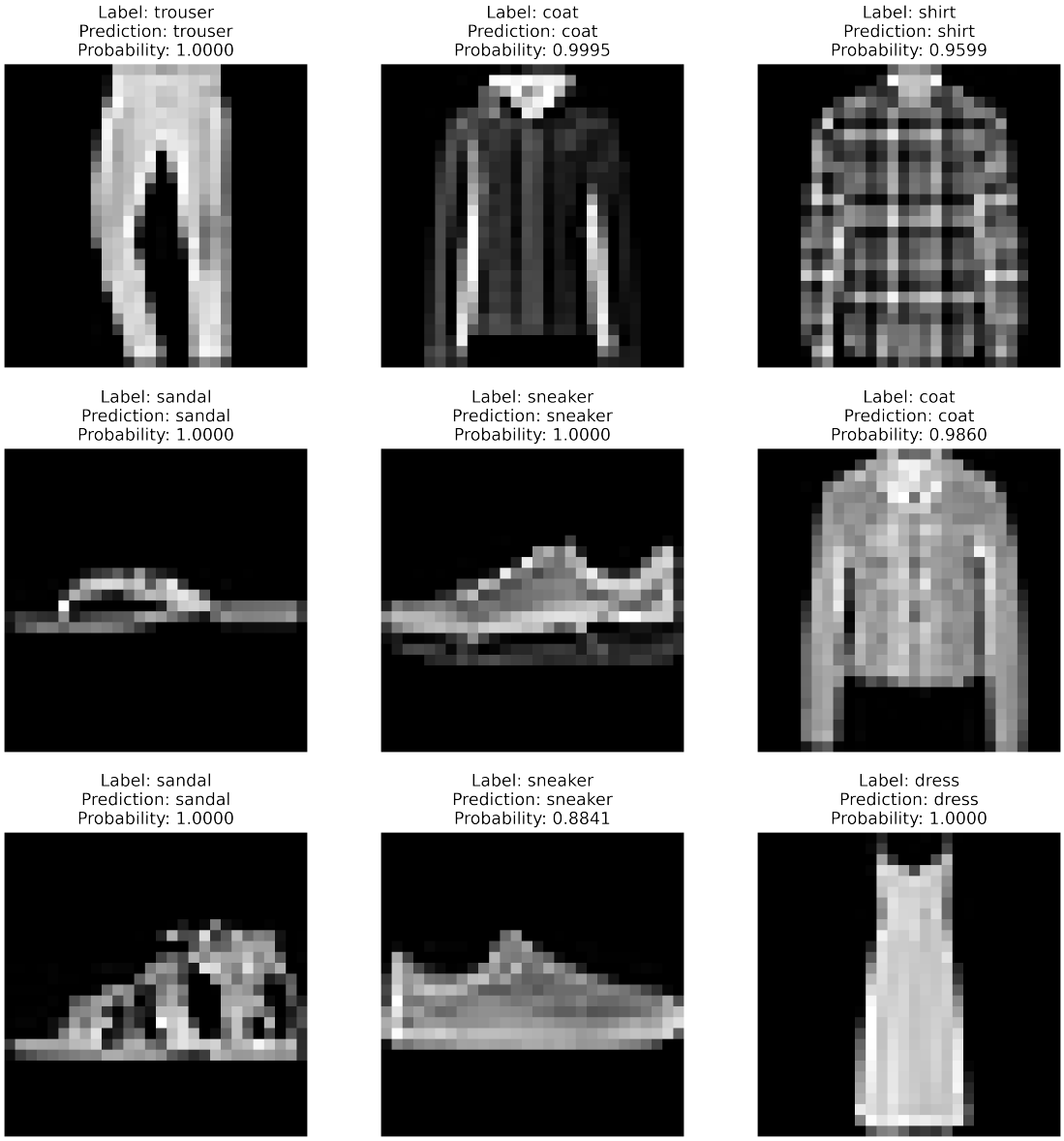


Figure 2: CNN prediction capabilities tested on some examples.

# 4 Appendix

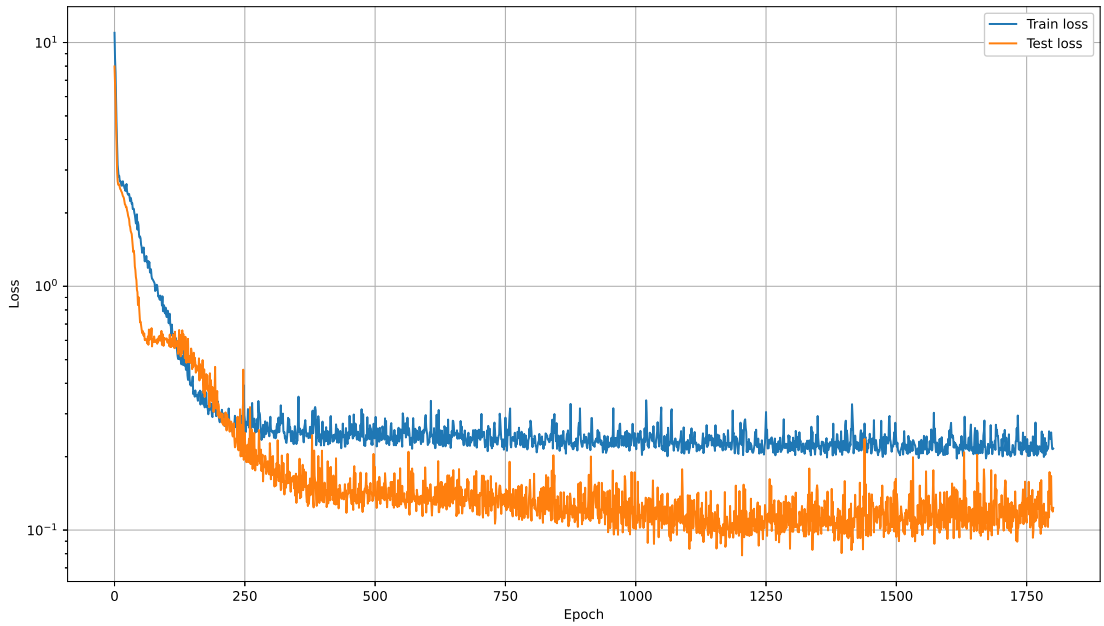


Figure 3: Final regression model train and test losses, early stopped.

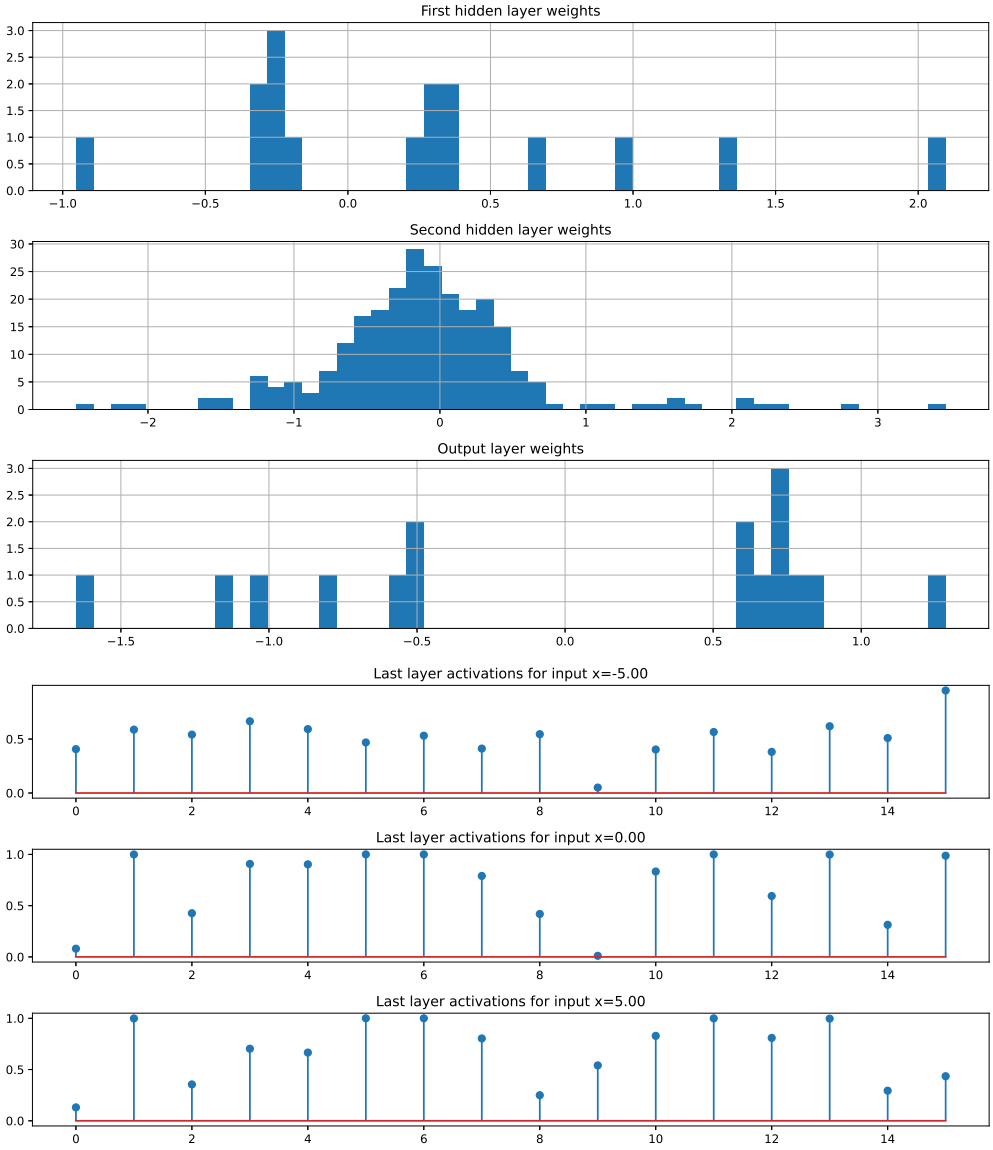


Figure 4: Final regression model weights histogram and last layer activations.

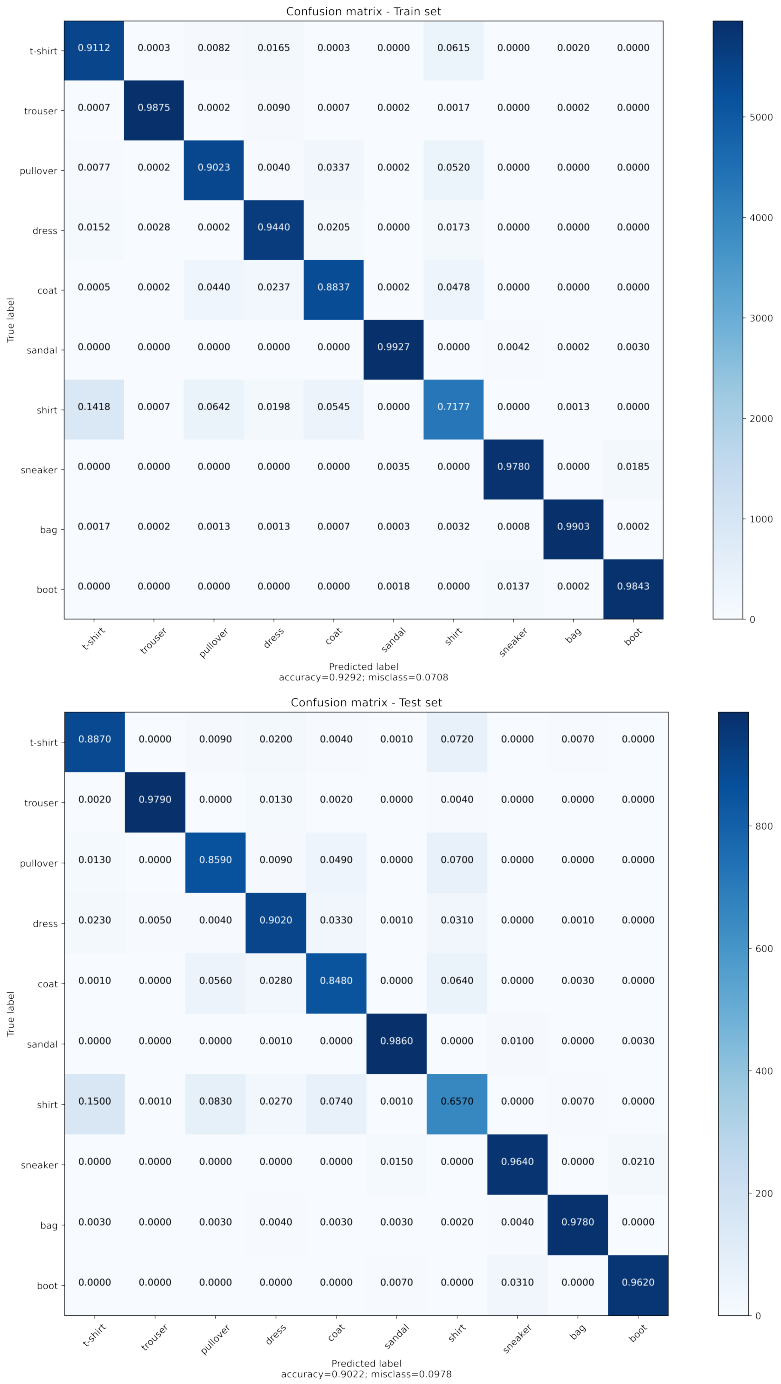


Figure 5: Confusion matrices on training and test dataset. Apparently, shirts can be confused more easily with t-shirts, coats and pullover.

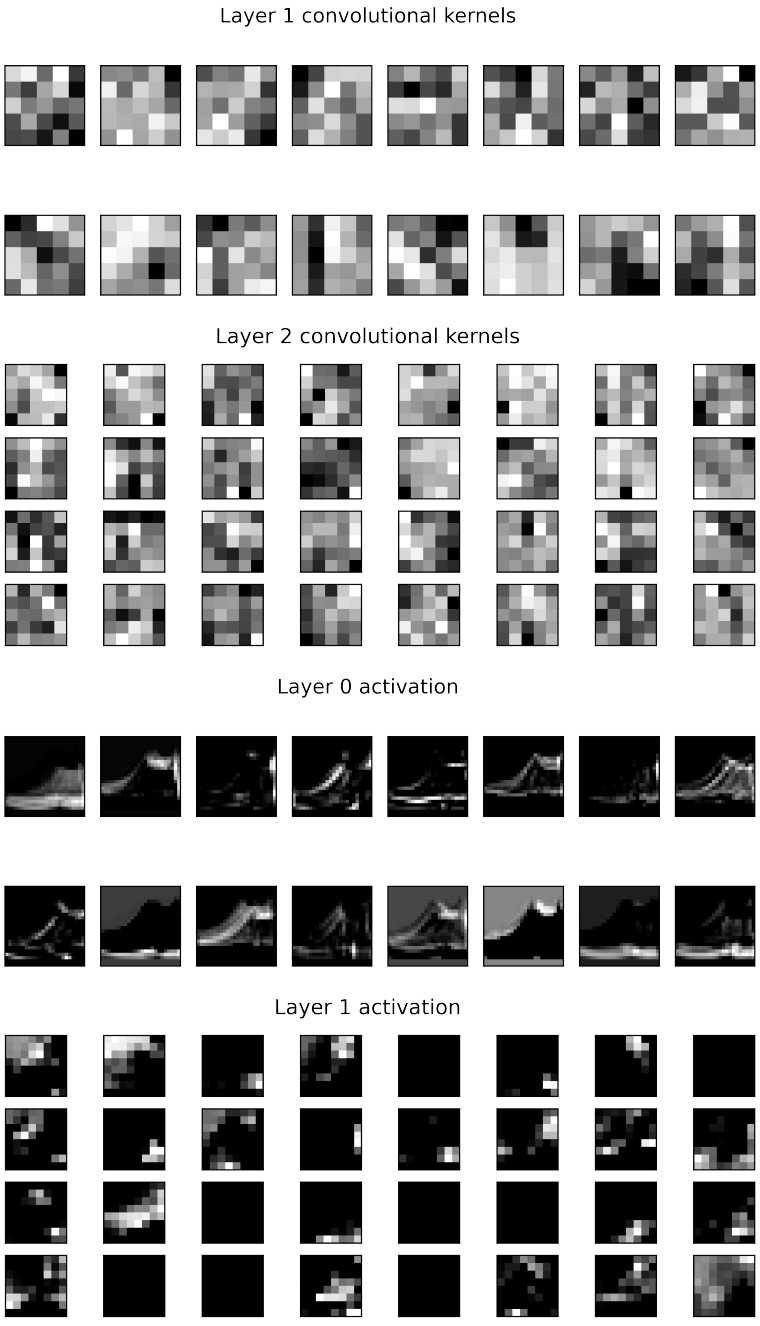


Figure 6: Final classification model convolutional kernels and activations.

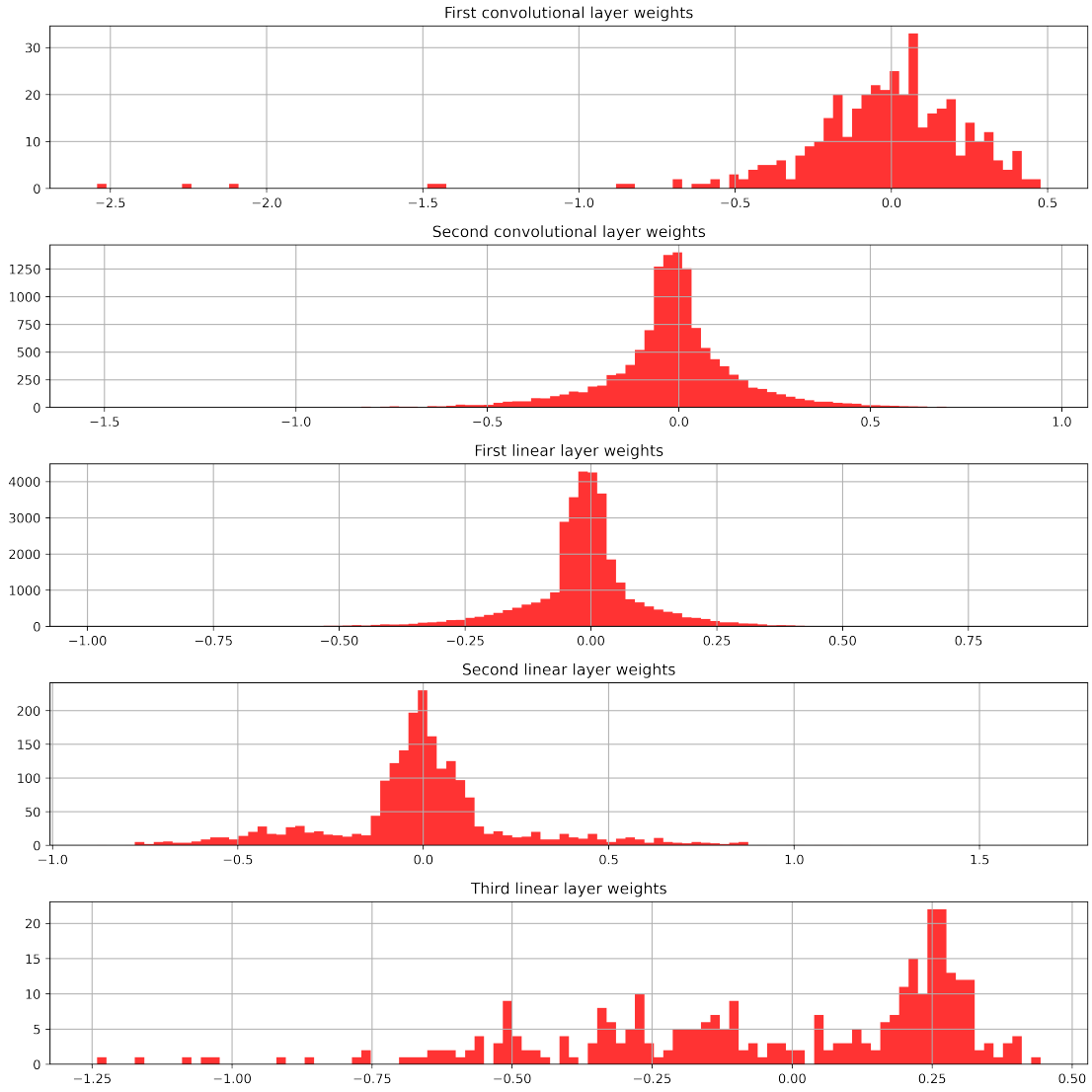


Figure 7: CNN weights histogram (all layers).

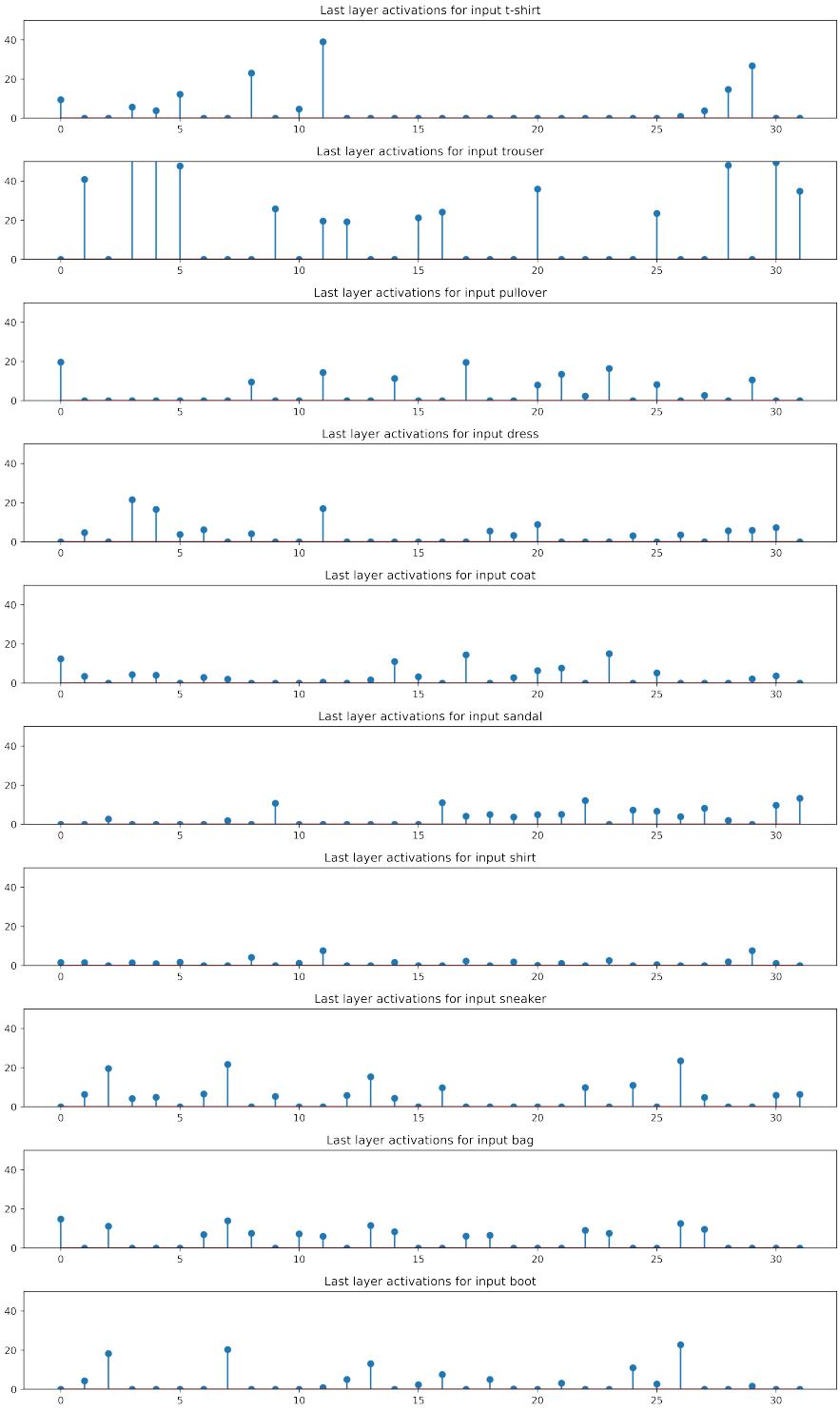


Figure 8: CNN Last layer activations for different classes. It's possible to recognize some patterns

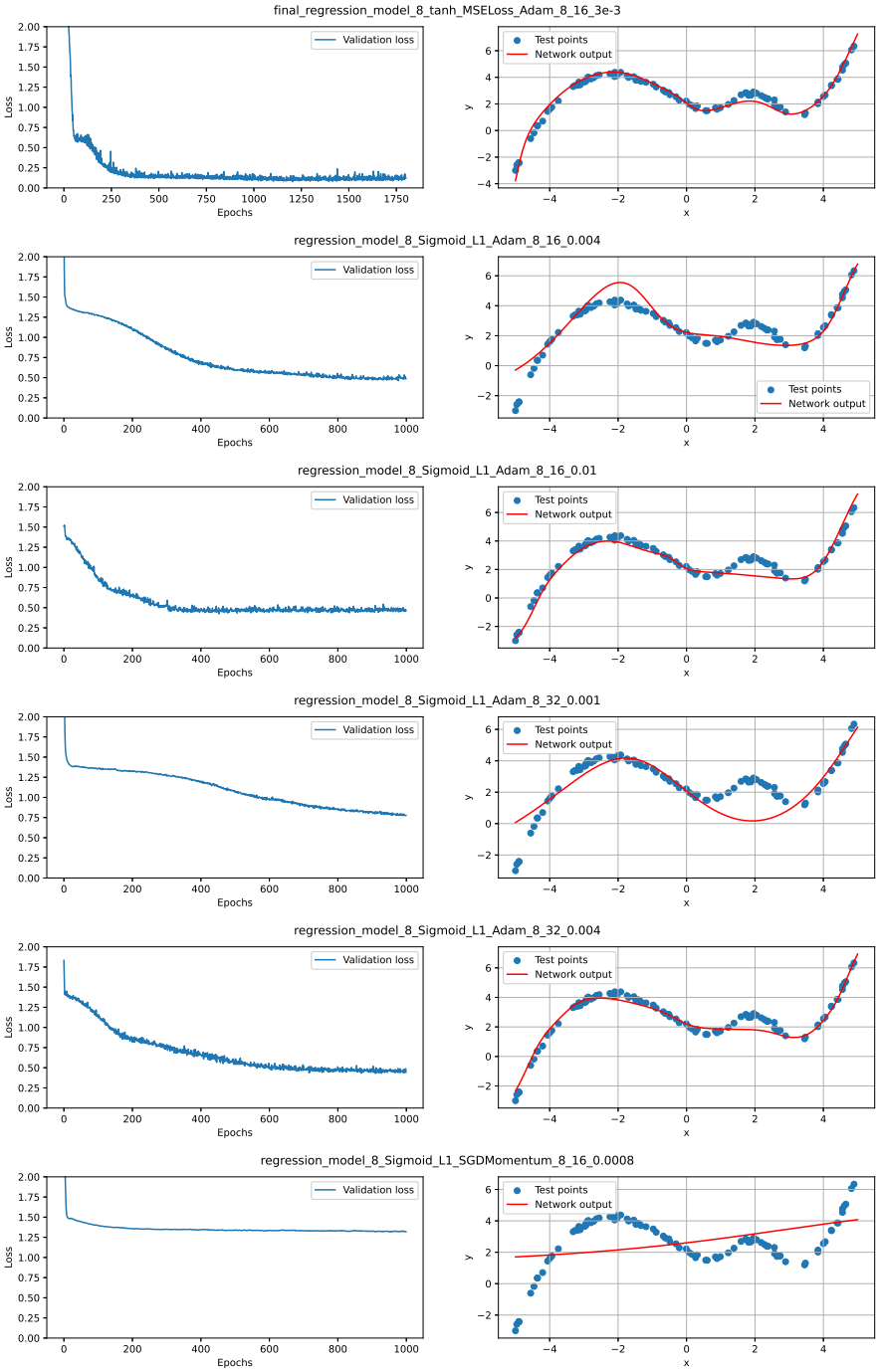


Figure 9: Regression grid search results 1/2

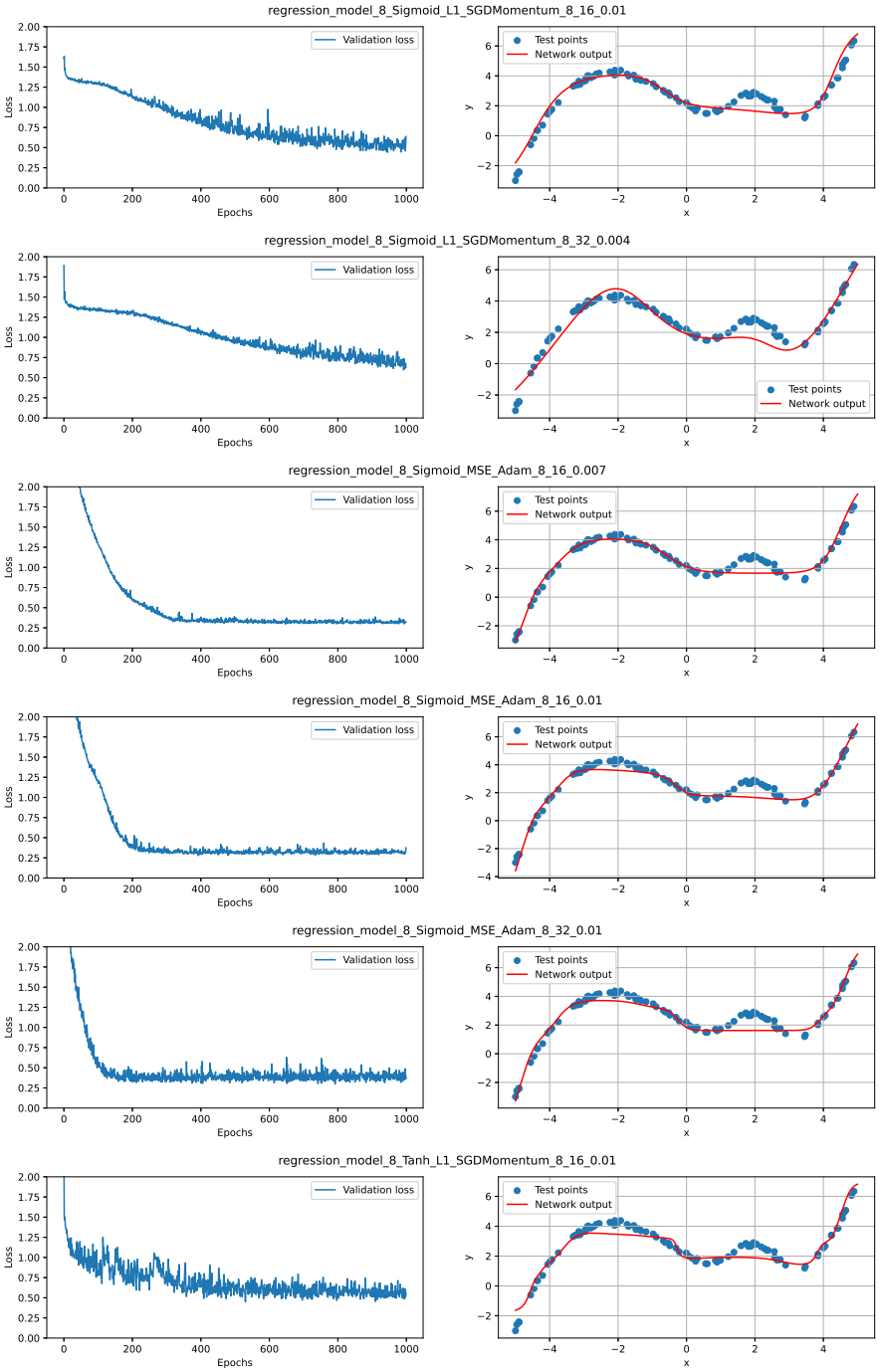


Figure 10: Regression grid search results 2/2

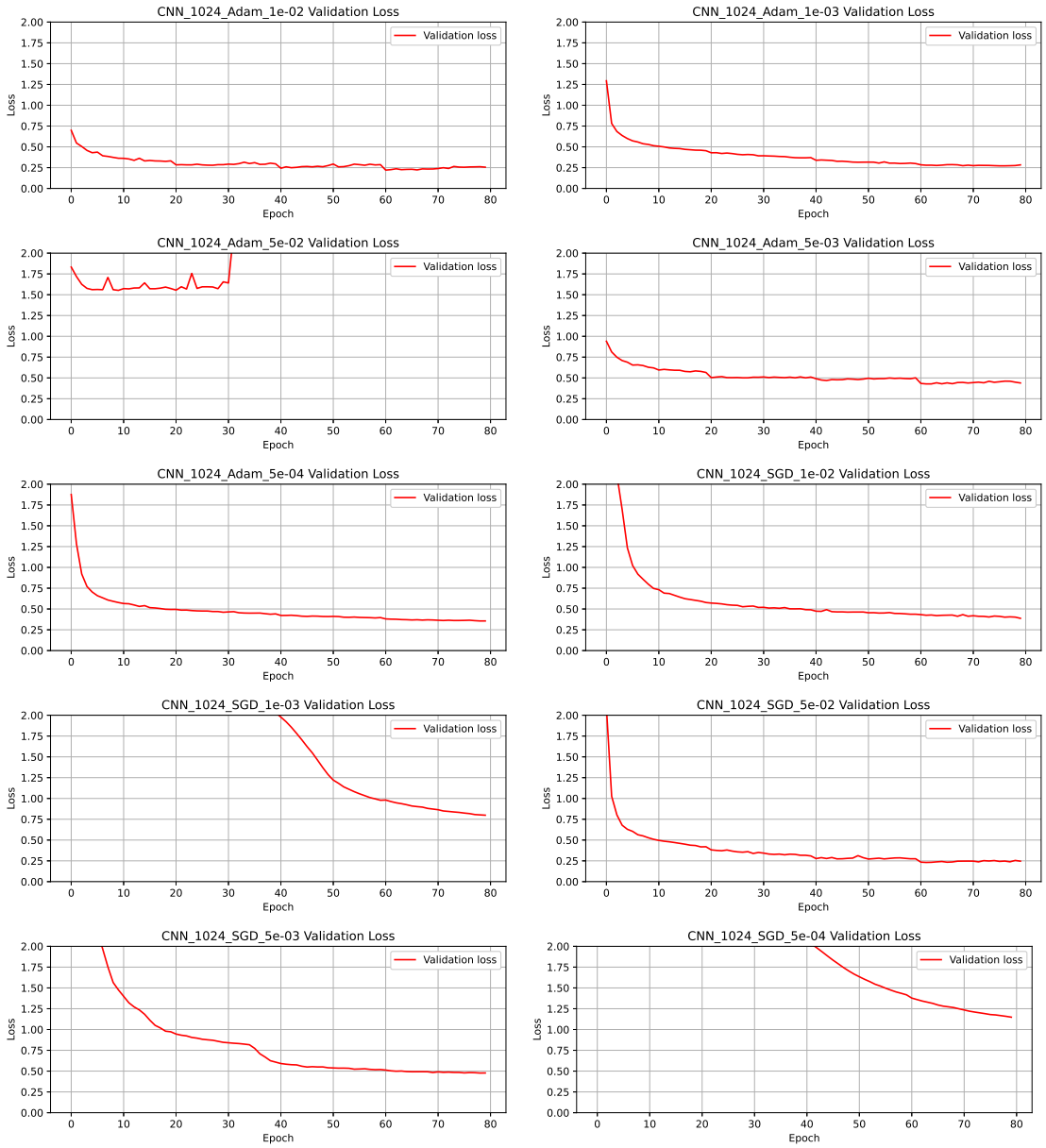


Figure 11: Classification grid search results 1/2

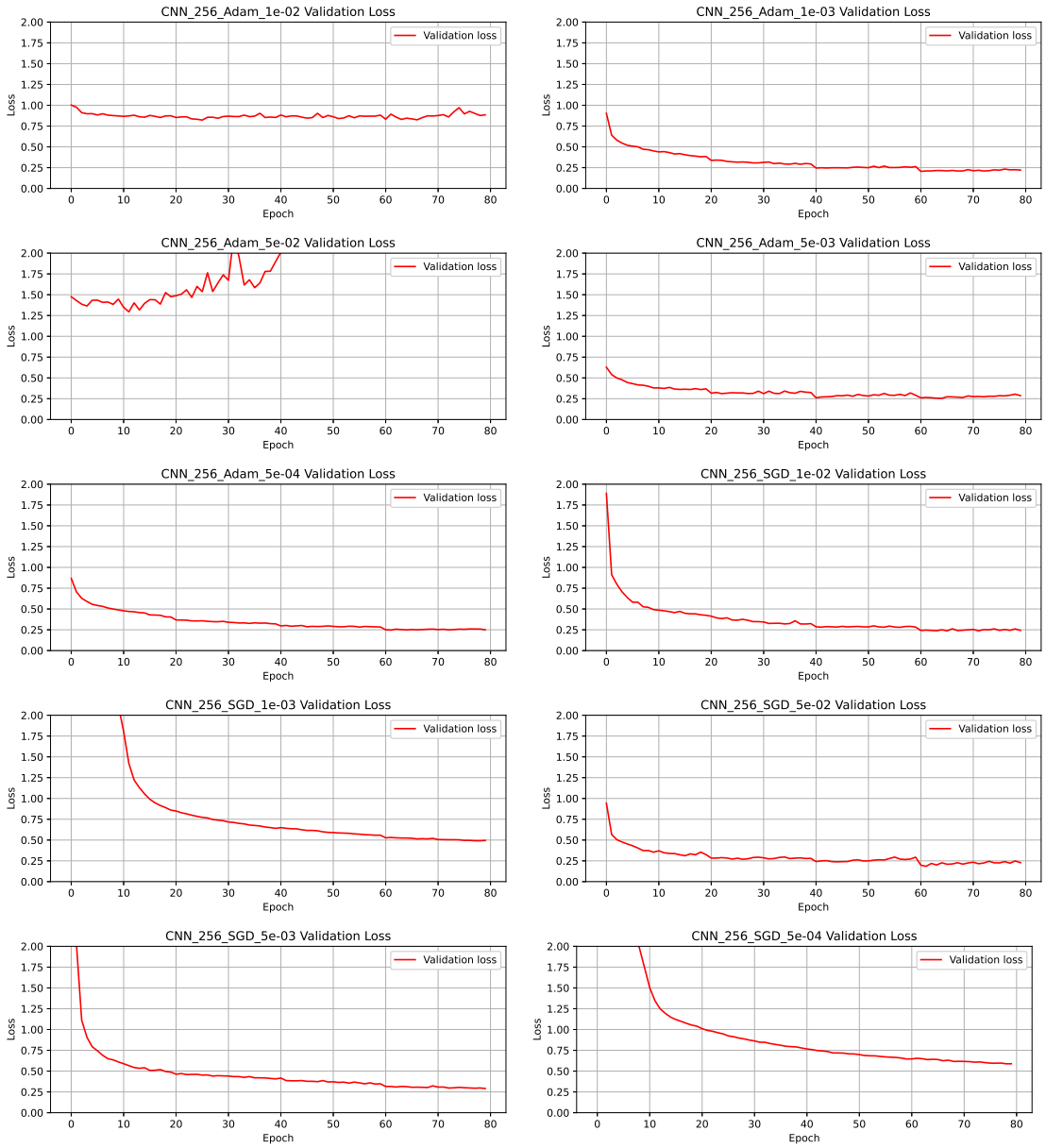


Figure 12: Classification grid search results 2/2