

Homework #1

Supervised Deep Learning

Monaco Saverio - 2012264

Neural Networks and Deep Learning - Professor: A. Testolin

Abstract—As for the first homework, the main models of Supervised Deep Learning are investigated. For the regression task, it is needed to effectively approximate a noisy unknown 1-dimensional function. The classification task instead consists in building a Convolutional Neural Network for the FashionMNIST Dataset. For both tasks, more advanced techniques are further explored and compared.

I. REGRESSION TASK

The task of Regression in a Neural Network framework consist in approximating a scalar function:

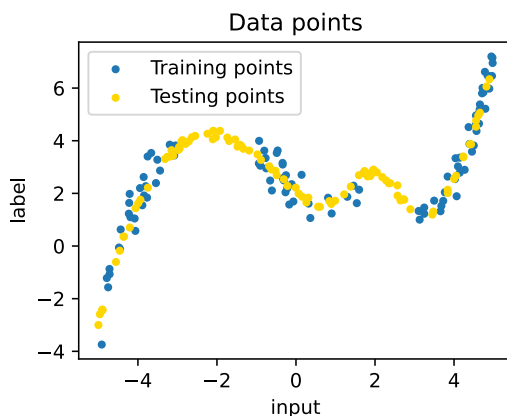
$$f : \mathbb{R} \rightarrow \mathbb{R}$$

through the use of a Network. Usually, the deeper the network is, the more complex patterns and behaviours of the target function it can grasp, however it might also encounter overfitting.

For the current exercise, training and testing points are generated according to a theoretical and unknown function, plus some noise:

$$\hat{y} = f(x) + noise$$

The data for the task is the following:



Data seems to be generated from a grade-5 polynomial function. It is worth noticing that the training dataset is not optimal, compared to the testing dataset: it is noisier and it has missing values, namely for $x \in [-3, -1]$ and $x \in [2, 3]$. The latter obstacle will be the most hard problem to overcome, since it requires the model to properly generalize on those two areas it has never been trained on.

A. Methods

For the Regression task, it was implemented a Python class: `RegFC` to generate a Fully Connected Neural Network (FCNN) model with the following parameters:

- `Ni`: dimension of input vector (int);
- `No`: dimension of output vector (int);
- `Nhs`: number of neurons in each hidden layer (list of int);
- `activation`: torch activation function;
- `dropout`: dropout rate after each hidden layer (float);
- `lr0`: initial Learning Rate for the optimizer (float);
- `reg_term`: Regularization term for the optimizer (float);

Both `Ni` and `No` must be 1, being the model a 1-D Regressor, while the depth of the hidden layers was kept general: for example `Nhs = [10, 20, 10]` will create a 1-10-20-10-1 model.

Loss function and optimizer were instead kept fixed:

- Loss function: `nn.MSELoss()` (L2 norm);
- Optimizer: `optim.Adam()`;

The models were evaluated by the Validation Error implementing *k-fold cross validation* using the `sklearn` function

`sklearn.model_selection.KFold`, for all models the number of folds was fixed to 5.

To obtain optimal hyperparameters a Grid Search was manually implemented with nested for loops and keeping the model with the lowest validation error.

Parameters of Grid-Search

Lr	1e-2, 1e-3
Reg. terms	1e-2, 1e-1
H. layers	[1000,1000], [1000,1000,1000]
Dropout	1e-2, 1e-1
Act. fun.	nn.Tanh(), nn.Sigmoid()

B. Results

1) Fully Connected Neural Network:

The best parameters found using the Grid Search method were the following

Best parameters

Lr	1e-3
Reg. terms	1e-2
H. layers	[1000,1000,1000]
Dropout	1e-2
Act. fun.	nn.Tanh()

Epochs: 1000

MSE on Test set: 0.27585697

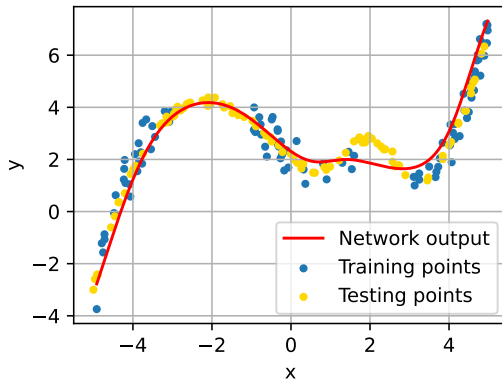


Fig. 1: Performance of FCNN on both sets

Overall, one could say that the implementation of the Regressor is successful, capturing the general tendency of the data, furthermore the model does also perform accordingly on the first patch of missing data (namely for $x \in [-3, -1]$), however it does not achieve the same goal on the other interval. This failure is mostly to blame on

the training data being noisy and having missing patches on the most unpredictable parts of his range (namely two local maxima).

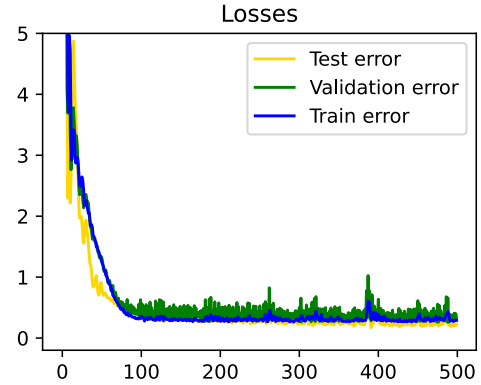


Fig. 2: Train, Test and Validation loss curves

The figure above displays the learning curve for all 3 sets: Training-set, Validation-set e Test-set. The model seems to reach a plateau right after 100 epochs and then it shows swift bumps during the remaining epochs. Those spikes can be explained by the fact that the sets are limited (only 100 samples for training and test set).

In addition one can see that for the second half of the training process, the test error is slightly lower than the training error, this odd result is due to the fact that the test set is less noisy.

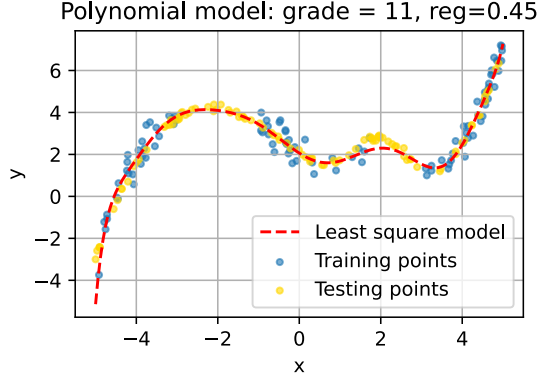
2) *Polynomial model*: Lastly, the FCNN was compared to a much simpler model: a *Polynomial model* fitted to the training dataset using the method of least squares.

In this particular case, in which data is clearly generated by adding noise to a polynomial function, a Polynomial model can be considered a viable alternative, however it restricts our hypothesis space to a much smaller space as it will not well generalize to other problems.

The best model was found by the use of a grid search of the parameters:

- `grade`: maximum grade of the fitting polynomial function;
- `reg`: regularization term;

applying the least square method on the training dataset and choosing the model with the lowest error on the test dataset:



However, as one can see, the grade of the best model may be too high and it will not generalize well for points outside the range $[-4, 4]$.

Being not a machine learning model, it is not wise to operate with training and test set, however in order to fairly compare the two models it was preferred perform both Grid Searches equally.

II. CLASSIFICATION TASK

The objective of classification is to obtain a *rule* that outputs the most probable label (belonging to discrete space \mathcal{L}) given a set of parameters X .

The second task consists in implementing a Convolutional Neural Network to build a Classifier on the FashionMNIST dataset[1]. Each sample of data is a 28×28 grayscale image and its possible label is an integer between 0 and 9 (each number represent a type of clothing or accessories such as T-shirt/top, Trouser, Pullover).



Fig. 3: Samples of FashionMNIST

The size of the set for this task is broader, having 60,000 samples for the training set, and 10,000 samples for the test set.

A. Methods

As for the previous task, the most optimal models were found using a custom made Grid Search and evaluating them using K-Fold Cross-Validation, with $K = 5$.

B. Results

1) *Basic Convolutional Network*: The structure of the model implemented is the following:

- *First convolutional layer*
 - `in_channels = 1;`
 - `out_channels = channels[0];`
 - `kernel_size = 5;`
 - `stride = 1;`
 - `padding = 2;`
- A max pooling layer with kernel size 2
- *Second convolutional layer*
 - `in_channels = channels[0];`
 - `out_channels = channels[1];`
 - `kernel_size = 5;`
 - `stride = 1;`
 - `padding = 2;`
- another max pooling layer with kernel size 2
- *A final layer*
 - `input_size = 7 \times 7 \times channels[1]`
 - `output_size = 10`

As input during the initialization, one must specify the `channels` array (of two integers).

For the Grid Search, different initial learning rates, regularization terms, channels arrays, dropout rates and activations were tried out, while the losses was kept `nn.CrossEntropyLoss` and the optimizer was `optim.Adam`

Best parameters

Lr	1e-3
Reg. terms	1e-4
channels	[6,12]
Dropout	1e-1
Act. fun.	nn.ReLU()

Epochs: 100

Accuracy on Test set: 90.11 %

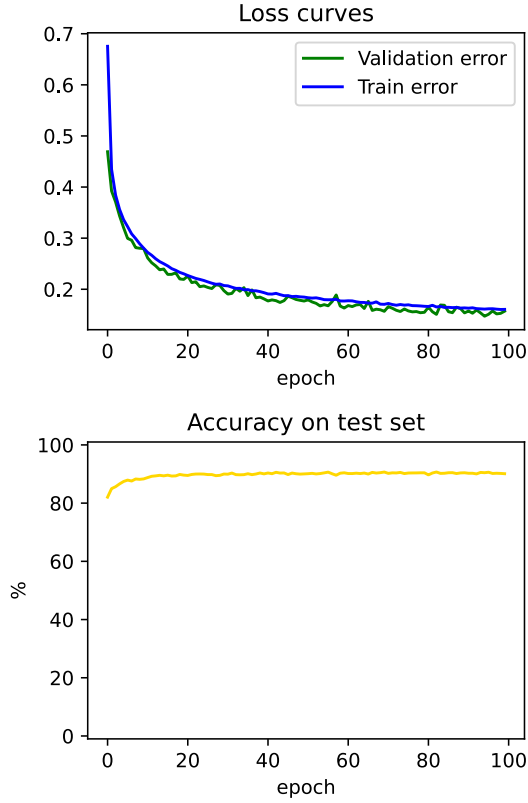


Fig. 4: First image: Loss curves on Validation and Training Set.
Second image: Accuracy of the model on the test set throughout its learning

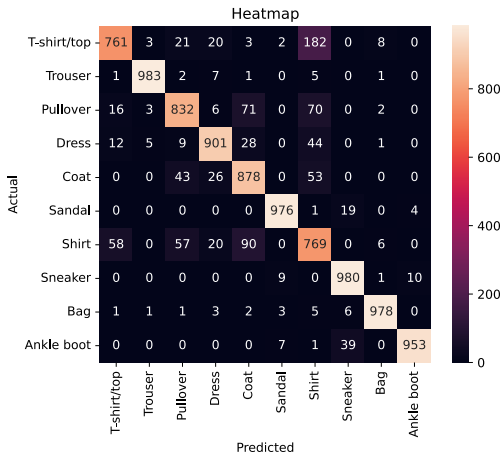


Fig. 5: Confusion matrix of the model

From the confusion matrix above, it is clear that the implementation of the classifier is overall convincing, however there it does not well distinguish between T-shirt/top and Shirt. This kind of behaviour was although expected since those two classes are aesthetically close.

2) *Data augmentation*: Augmenting the training data mainly helps a convolutional neural network obtain rotation invariance and generalize better overall. Data augmentation was applied on the training set by applying the following transformations on the dataloader:

```
augmentation =
    transforms.Compose(
        [transforms.ToTensor(),
         transforms.RandomVerticalFlip(p=0.5),
         transforms.RandomCrop(28,padding=2)] )
```

Since having a larger training set required longer training time per epoch, it was opted not to perform a grid search and to stick with the most optimal parameter found with the model of the last section:

Epochs: 100

Accuracy on Test set: 87.55 %

In this case, data augmentation does not help the model reach an higher accuracy. This is most probably due to the fact that the images on the test set are well standardizer, namely the clothes are perfectly in frame and not-rotated, hence, augmenting the data just misleads our model.

3) *Finetuning a ResNet50*: In addition, the pretrained model `resnet50` was modified to support grayscale images and 2 fully connected hidden layers at the end were added as finetuning. Due to the long period of time for the training, no Grid search was performed for this model either.

lr0: 1e-3

Loss: nn.CrossEntropyLoss()

Optimizer: optim.Adam()

Epochs: 100

Accuracy on Test set: ??? %

REFERENCES

- [1] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

III. APPENDIX

A. Regression

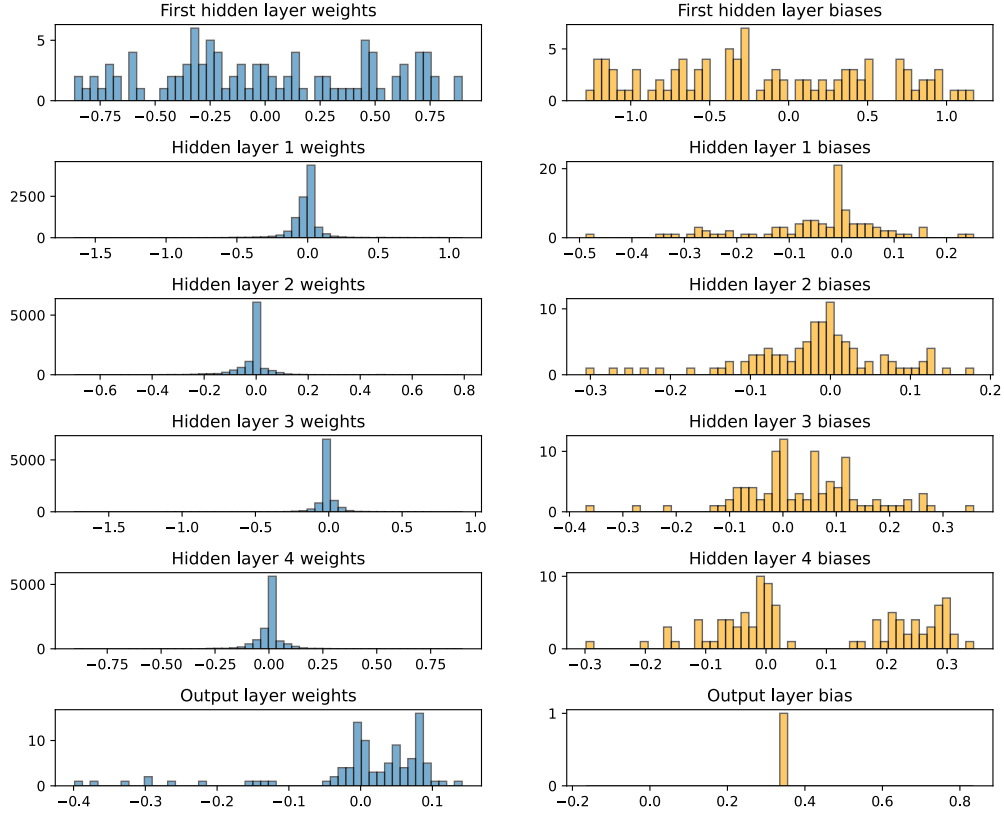


Fig. 6: Histogram of weights and biases of the best regressor found using Grid Search

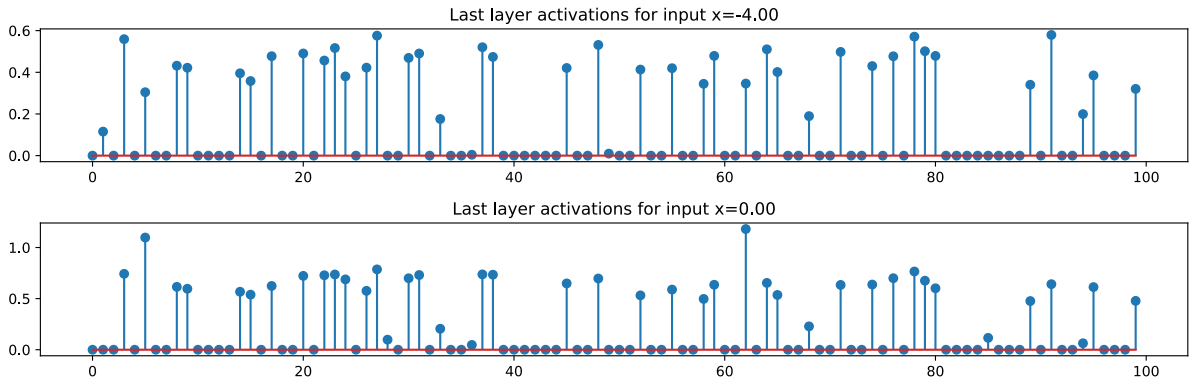


Fig. 7: Activation of the best regressor

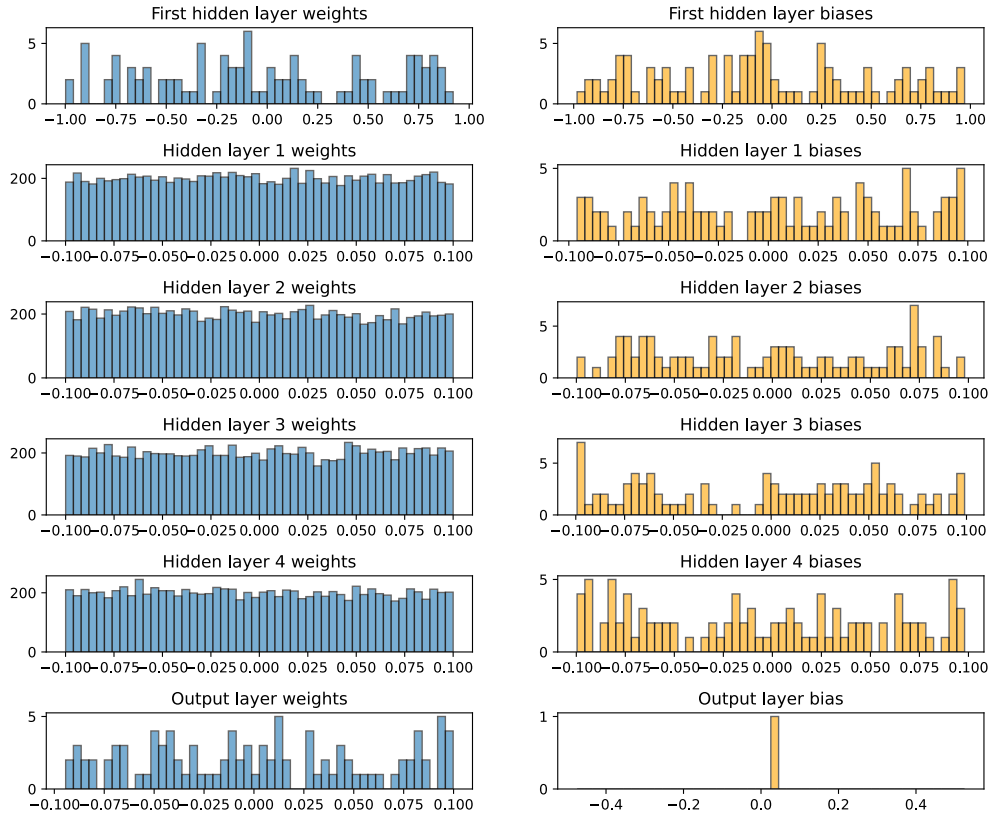


Fig. 8: Histogram of weights and biases of a not trained model

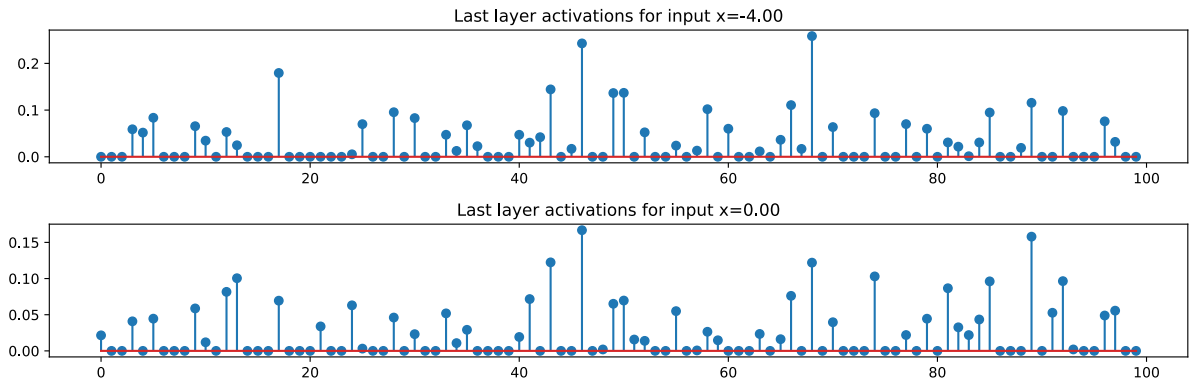


Fig. 9: Activation of the same not trained model

B. Classification

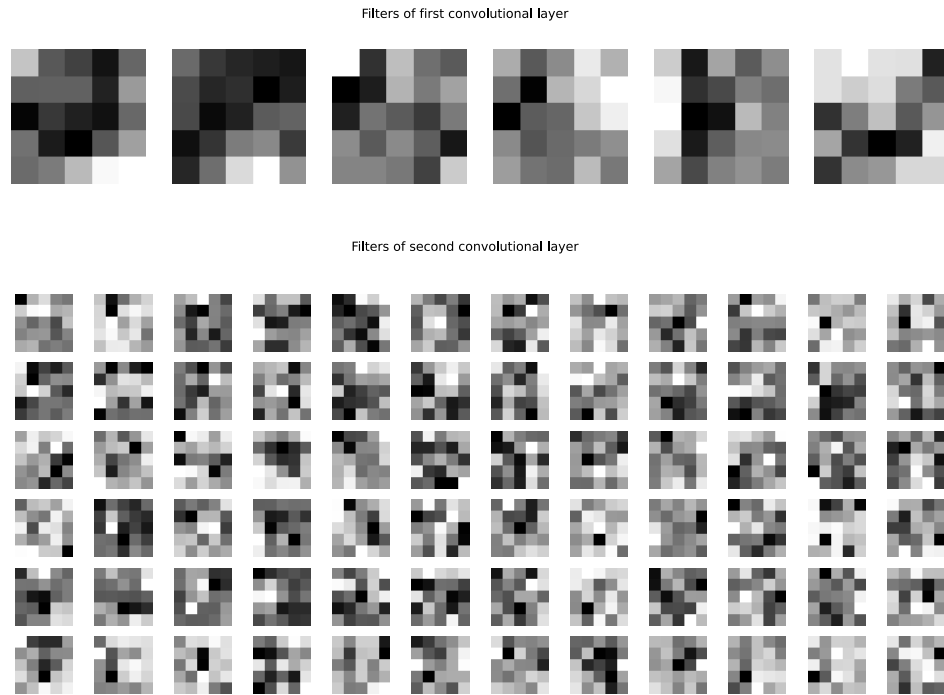


Fig. 10: Filters of both convolutional layers

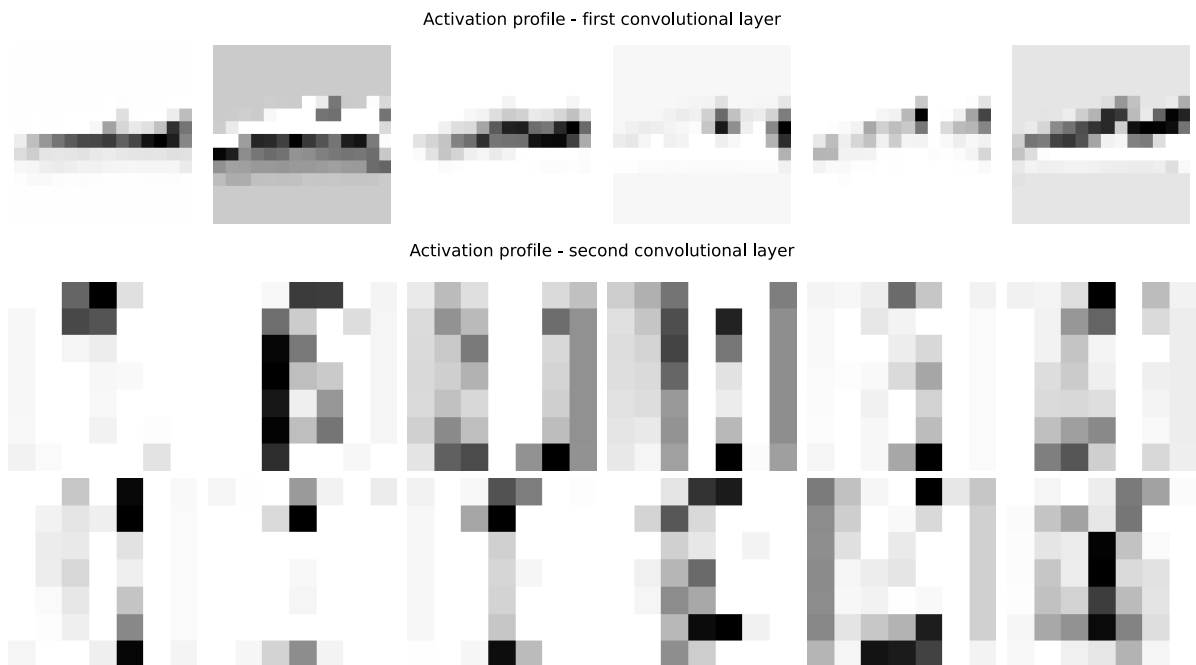


Fig. 11: Activation of both convolutional layers