Homework 1 – Supervised Neural Networks

1. Introduction

The homework consists mainly of a regression and a classification task that are solved using neural networks.

1.1 Regression

For the regression task the goal is to implement a fully connected neural network that approximates a 2D function and to optimize the predictions of the net by adjusting the hyperparameters and using regularization methods. Because of the small amount of training data, I chose to use k-fold cross-validation to get a model with minimal validation loss. To optimize some of the hyperparameters, I perform a grid search for this task, since training a single model is fast and can thus be easily repeated several times. To regularize possible overfitting, I chose to evaluate the regularization methods weight decay and dropout layers.

1.2 Classification

For the classification task the goal is to implement a neural network that can classify digits from the MNIST dataset by labelling the input images correctly with the corresponding digit. To solve this problem I implemented both a fully connected network and a CNN. Since the dataset for this task is quite big, I chose to simple split the training set once into training and validation sets instead of using the computationally more expensive k-fold cross-validation. For opimization of the test accuracy I focused on the CNN. Due to the huge amount of training data and longer runtime for training this model, I decided to perform random search in order to optimize the hyperparameters. Again, I looked into both weight decay and dropout layers to regularize a possible overfitting to the training data.

2. Methods

In this section I describe the model architectures I chose for each of the networks and explain the choice for the hyperparameters.

2.1 Regression

The regression model consists of 3 fully connected layers. The input layer has just one unit, since the input is one-dimensional. The first hidden layer has 128 hidden units, while the second hidden layer has 256 units. Finally, the output layer consists of one unit that predicts the y coordinate for the input. For each layer the model applies a sigmoid as activation function. The loss function to evaluate the results of this model is the mean squared error loss.

To get a model with low validation loss, I use 10-fold cross-validation on the training data and choose the model with the best average validation loss over the last 10 epochs.

To minimize the validation loss further I tried several different optimizers with their standard learning rates and chose the one with the best results to be used for the grid search. These optimizers are Adam, Adagrad, Adadelta and RMSProp.

Then I tried both weight decay and dropout layers as regularization options to see which one works better for this problem and should be used during grid search.

I performed grid search on a subset of hyperparameters to limit the runtime. I focused on the learning rate for the optimizer, the batch size for training and the dropout probability for the dropout layers added between each pair of fully connected layers. Other hyperparameters of this model are the number of folds for cross-validation, which is 10, the number of epochs for training, which is 200, the number of neurons in each hidden layer as well as the weight decay factor, which is set to 0.

2.2 Basic Classification

Also the basic classification model consists of 3 fully connected layers and a softmax output layer. The input size is now 28 * 28, the number of pixels per image. The first hidden layer has 256 hidden units, whereas the second hidden layer has only 128 units. The third hidden layer consists of ten units, one for each possible digit. For each layer the model applies a sigmoid as activation function, except for the third connected layer which does not need an activation function since it is followed by the softmax layer. The loss function to evaluate the results of this model is the cross entropy loss.

The training data is split only once into training and validation datasets.

For this model, I perform no optimization and therefore I only evaluate it with the Adam optimizer with learning rate 0.01. All other hyperparameters like the batch size for training, which is 20, and the number of epochs, which is 50, are fixed.

2.3 CNN Classification

The Convolutional Neural Network architecture is similar to the Google LeNet, since this is the best network to classify images of digits correctly. The input layer is a convolutional layer that takes images with one channel as input and has 6 output channels. It has a square kernel of size (5x5) and stride (1x1). Next comes a 2d max pooling layer with kernel size equal to (2x2). It is followed by another convolutional layer with 6 input and 16 output channels, kernel and stride size are identical to the first convolutional layer. Then there is a second max pooling layer with the same hyperparameters as the first one. Finally, there are 3 fully connected layers with 120, 84 and 10 neurons respectively. The activation function used for the convolutional layers as well as the fully

connected ones is relu. It is not applied to the output layer which is the third fully connected layer. The loss function to evaluate the results for this model is cross entropy loss as for the basic classification network.

The training data is split only once into training and validation datasets instead of applying cross-validation, since the amount of data is enough to still get a high test accuracy.

As for the regression task, to minimize the validation loss I tried several different optimizers with their standard learning rates and chose the one with the best results to be used for random search. These optimizers are Adam, Adagrad, Adadelta and RMSProp.

Then, I tried both weight decay and dropout layers as regularization options to see which one works better for this problem and should be used during random search.

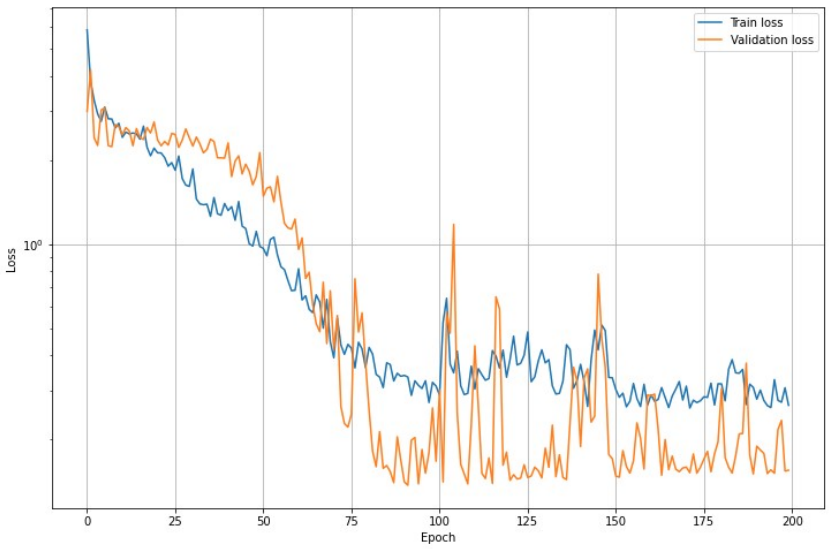
I performed 10 iterations of random search on a subset of hyperparameters to limit the runtime. Grid search is computationally very expensive on a model that already takes some time to train. As before I focused on the learning rate for the optimizer, the batch size for training and the dropout probability for the dropout layers added between each pair of fully connected layers. Other hyperparameters of this model are the number of epochs for training, which is 30, as well as the layer parameters such as number of channels, kernel size, stride and number of neurons.

3. Results

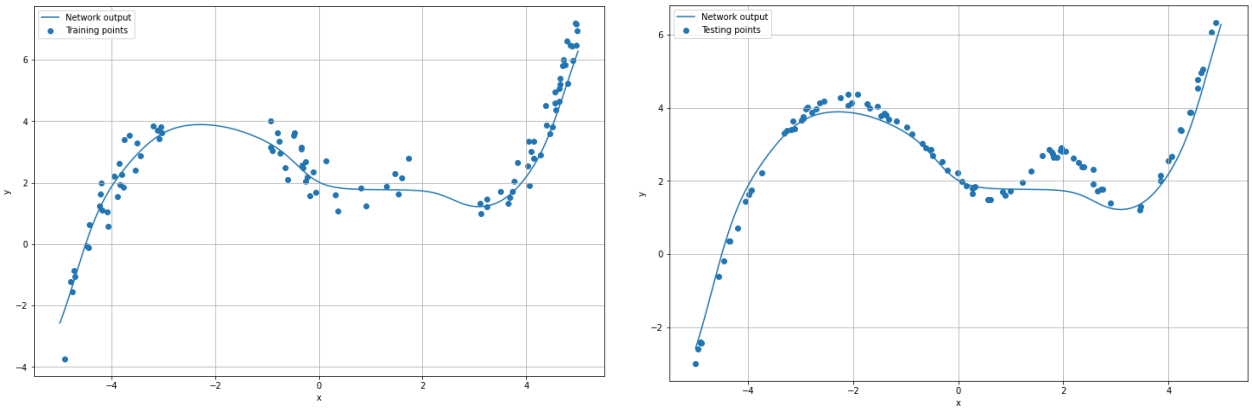
In this section I present the results obtained for the regression model and the CNN.

3.1 Regression

The simple regression model without hyperparameter optimization uses the Adam optimizer with a learning rate of 0.01. After performing 10-fold cross-validation on the regression model described in 2.2, the best model achieves a validation loss of 0.178 and an average test loss of 0.236. The figure below shows the test and validation loss per epoch. Both losses decrease sharply during the first 100 epochs and then seem to stagnate.



Looking at this graph as well as the learned model visualized on top of the training points on the left at the end of this paragraph, it is possible to deduce that the model is fitting the training data well. On the right, the same model is plotted over the testing data. The graph suggests, that the model can generalize well to new data and it also explains the slightly higher test loss compared to the validation loss. Due to the holes in the training data, the network has trouble to approximate the function well in this area, which causes some testing data to be predicted badly.



Compared to the other optimizers the average validation and test losses using Adam are the lowest, with RMSprop coming in second. Hence, I chose the Adam optimizer for performing hyperparameter optimization.

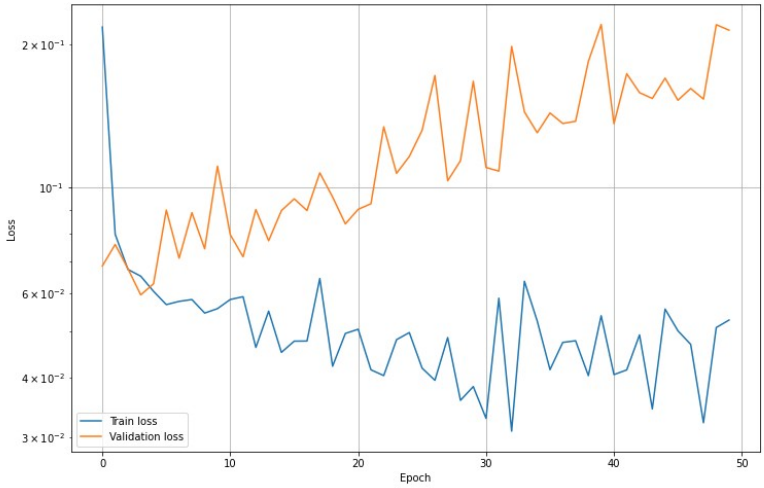
Using weight decay with any factor seems to have a negative impact on the learned model in the sense that it underfits the data badly. Dropout, on the other hand, causes a slight overfitting for a dropout probability of 0.2.

To find optimal hyperparameters, I did a grid search over the batch size for training, the learning rate for the optimizer and the dropout probability. The best performing model reaches a validation loss of 0.747 and a test loss of 0.224 which is slightly lower than the model without grid search. The best choice for the hyperparameters seems to be a batch size of 15, a learning rate of 0.01 and a dropout probability of 0. The grid options for dropout were limited to 0, 0.2 or 0.4 and since 0.2 already meant a slight overfit, this result is not surprising. The regularization methods might not be getting good results in this case because the model does not overfit the data from the beginning so no regularization is needed.

The weight histogram for the network without optimization can be seen in Fig.1 in the Appendix. For the first layer no weights are close to 0, while for the second and the output layer the weights are concentrated around the value 0. Fig. 2 also in the Appendix shows the corresponding activation s of the neurons for different input values. It seems that for inputs 0.1 and 0.9, similar neurons are activated, while the input of 2.5 shows some more noticeable deviations for some neuron activation.

3.2 Classification

The CNN without hyperparameter optimization uses the Adam optimizer with learning rate 0.01, a batch size of 100 and 50 epochs. After training the model, described in 2.3, achieves an average validation loss of 0.214 and an average test loss of 0.192. Looking at the graph below that shows the training and validation losses per epoch, the model is clearly overfitting to the training data even though the test accuracy is pretty high at 97.83%.

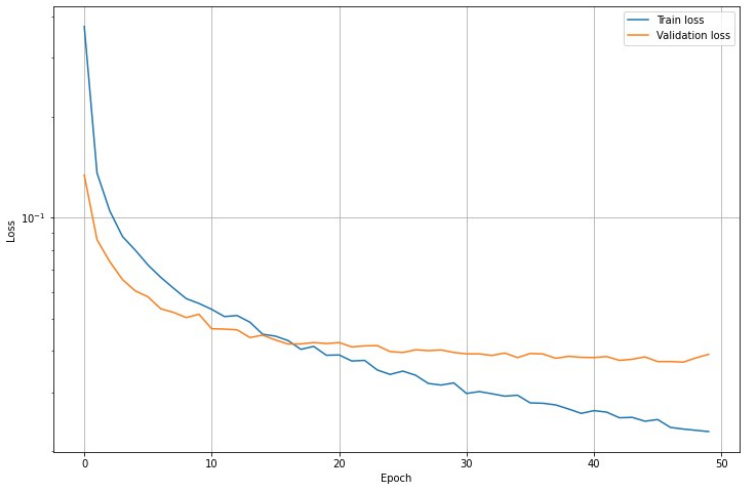


After trying out different optimizers, Adagrad with lr=0.01 achieves the lowest validation and test losses at 0.04 and 0.037 with an improved test accuracy of 98.86%. Adadelta with lr=1.0 achieves a slightly higher test accuracy but also higher validation and test losses. To improve generalization also on unseen data, I chose Adagrad as an optimizer because of its lower loss values. With this optimizer the model overfits the training data noticeably less than before. The model with Adadelta, however, did not show a decrease in overfitting at all.

To regularize the model and further reduce the overfitting, I used the Adagrad optimizer with a weight decay factor of 0.4 and other values. In this configuration, the network always returns 1 as label and hence has a very low test accuracy at 11.35%.

When using dropout layers after the fully connected layers with 0.2 dropout probability, the validation and test losses decrease slightly and test accuracy goes up to 99.04%.

If we look again at the loss graph for this configuration, the model is no longer overfitting on the training data but generalizes well to new data.

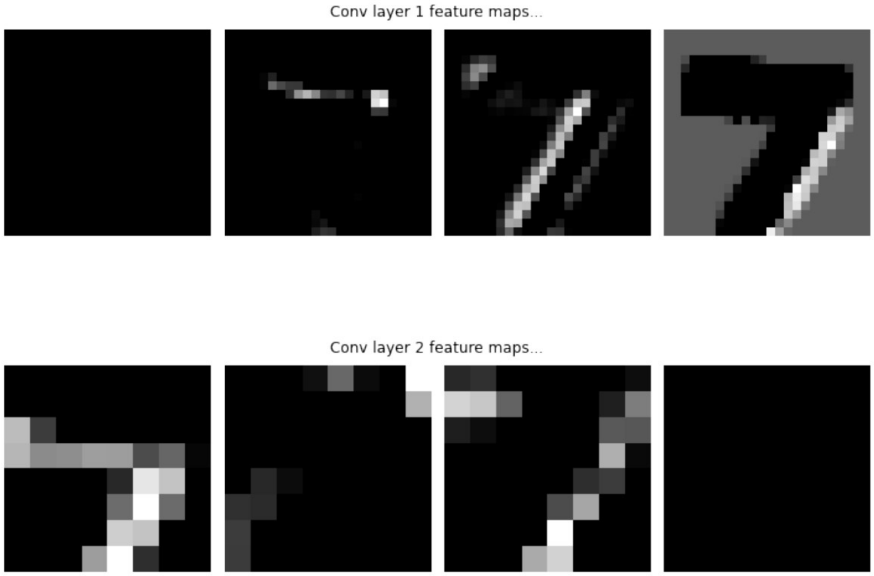


After 10 iterations of random search, the best configuration had a batch size of 200 samples, a learning rate of 0.04 for the optimizer and a dropout probability of 0.422. The validation and test losses at 0.062 and 0.046 are higher than before and the test accuracy is thus slightly lower at 98.94%. The random search did thus not improve the model at all, which could be due to the fact,

that only 10 iterations were performed and the hyperparameter value choice was unlucky, especially since the before achieved test accuracy of 99.04% is difficult to increase much anyway.

When visualizing the weights of the fully-connected layers in a histogram, it can be observed, that most neurons in the first two linear layers have an absolute value of maximally 0.2. The output layer weight distribution shows more variation with a peak at the value 0.1 and many weights with negative sign going approximately till -0.3. The histograms can be seen under Fig. 3 in the Appendix. Fig. 4 in the appendix shows neuron activations of the fully-connected layers for input images at index 0 to 2.

Fig. 5 of the Appendix shows some receptive fields for the first and second convolutional layer in the CNN. It is difficult to discern meaningful features by simply looking at these results. The feature map visualization, however, gives a better insight to the network learning process. The graph below shows four feature maps per convolutional layer. It is possible to discern a 7 or parts that could build a 7 in various maps. This is due to the fact, that the first input image chosen to print the maps is in fact a 7.



All weight and activation analizations have been done on the CNN before hyperparameter optimization and may thus look different than in the jupyter notebook which has since then been adapted to contain cross-validation and random search.

Appendix

Fig. 1 Weight histogram for the regression model

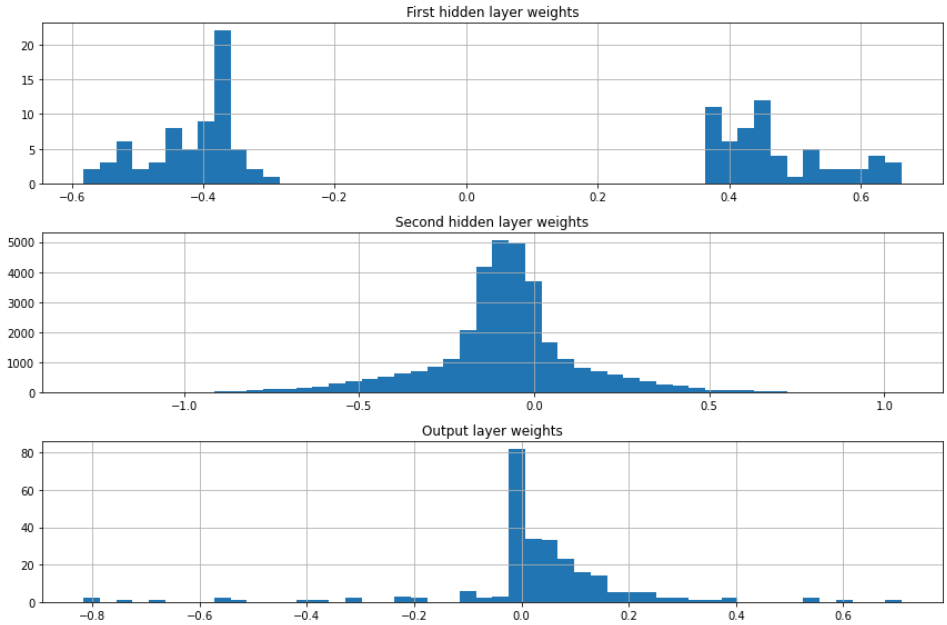


Fig. 2 Neuron activations for the regression model

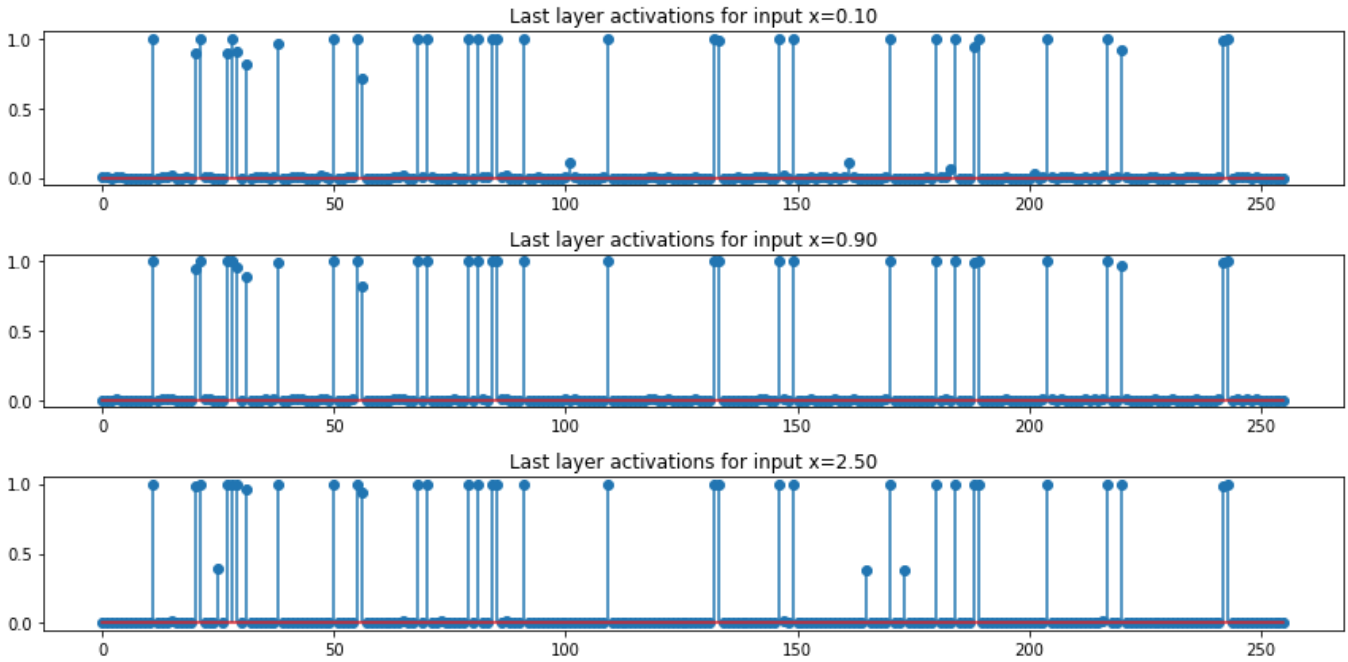


Fig. 3 Weight histograms of CNN linear layers

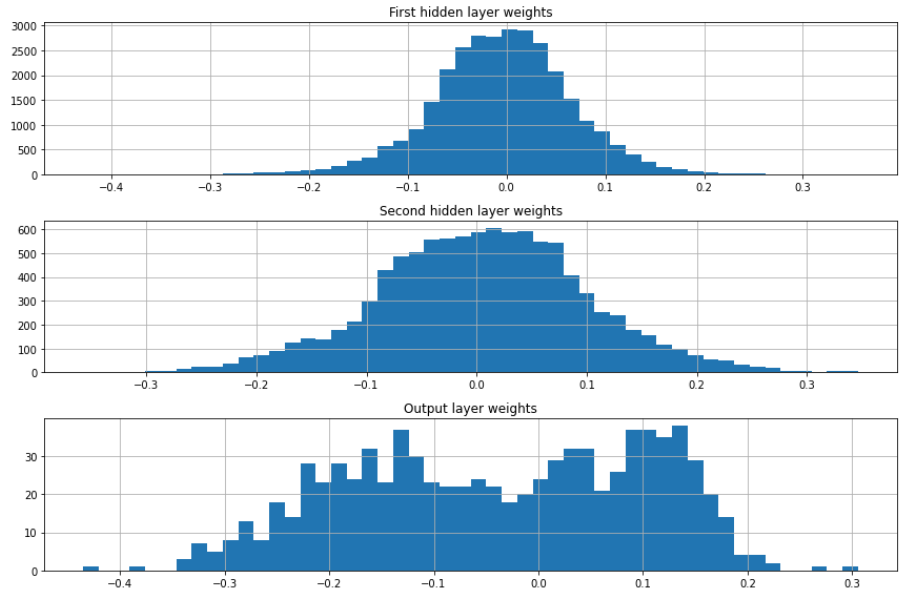


Fig. 4 Neural Activations in CNN linear layers

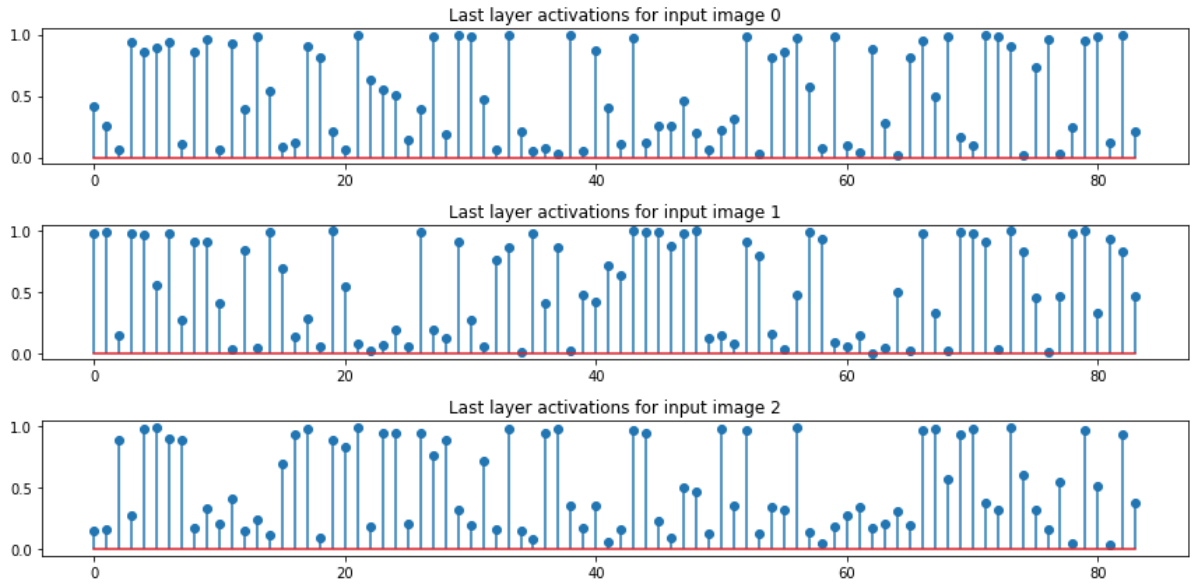


Fig. 5 Receptive fields CNN

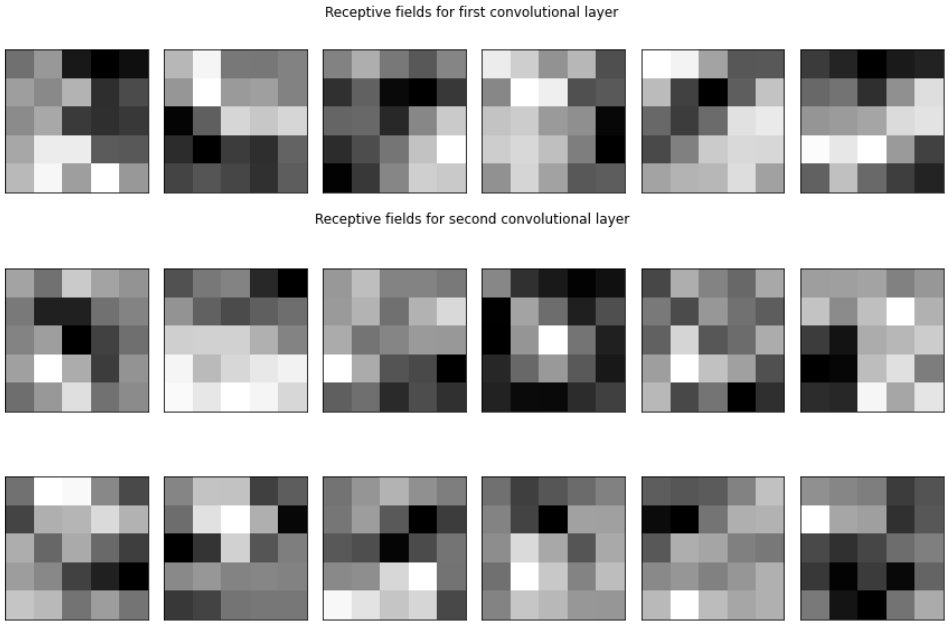


Fig. 6 Loss graph for grid search in the regression model