Name:

Amir Hossein

Surname:

Mohammad Amri

Title:

Training neural networks for

linear regression and & classification tasks

Introduction

In this homework, we have two tasks. The first one is a regression task. In this task, we have a dataset of 100 samples in form of (x, y) and we want to implement a simple Neural Network (NN) and test the trained NN with a test set of 100 samples. Since the dataset is very small and simple a NN with one or two hidden layers might be enough. The parameters that have been selected to be tuned are as follows:

1. Learning Rate ==========> (0.0001,0.002)
2. Optimizer ==========> ["SGD", "RAdam", "Adam"]
3. Number of Layers ======> (1,2)
4. Number of Neurons in layer one======> (16, 128, 8)
5. Number of Neurons in layer two======> (16, 128, 8)
6. Value of momentum in SGD optimizer======> (0.85, 0.98)
7. Dropout error =======> (0,0.1)

Note that a more professional way to optimize the number of layers is to use for loop and add layers to a list and then give this list to nn.sequential as in the function "**oneTwo**". On the other hand, a simpler approach that can be done is that define two different networks one with only one hidden layer and another one with only two hidden layers, optimize each separately and select the network with a better result. In this Homework, the second approach is taken.

In the second part of the homework, we implemented used a neural network to solve a classification problem. The dataset for this part is fashion mnist which is of 60000 images. For training networks on images, mostly Convolutional Neural (CNN) networks are used. Yet, since the dataset is easy to use we can also use a fully connected network. Unlike the previous part number of layers in this part is constant and it is not used as a hyperparameter. Hyperparameters for this part is as follows:

1. number of filters of the first channel
2. number of filters of the second channel
3. Optimizer ==========> ["SGD", "RAdam", "Adam"]
4. dropout probability (a regularization method)
5. number of neurons in the first fully connected layer
6. number of neurons in the second fully connected layer
7. the momentum parameter in SGD optimizer (regularization method)
8. L2 weight (regularization method)

In both tasks, the optimization is done using Optuna.

Another note that should be mentioned is that in the first part, the K-fold cross-validation error is calculated and then this error is given to optuna to be optimized, but in the second part because the time of optimization becomes very high the training dataset is split to train and validation, and optimization is done using error of validation data. Afterward, the k-fold error of the selected model is calculated.

For obtaining **cross-validation loss** first, the training set should be divided into equal subsets and put aside one subset and then the network is trained with the remaining data and then the subset that is not used during the training process is used for validating the loss of the trained network. This is done for all subsets and the final loss is reported by taking the mean of all losses (not that for every subset the network weights should be reset). In this homework, 5-fold cross-validation is used. For sake of this, I used the **K-fold** method from **sklearn.model_selection** library. By using the Kfold method I divided the data into 8 folds for the first task (since the dataset is very small) and 5 folds for the second task, and by using a nested loop, cross-validation loss is calculated. The outer loop selects the validation and train sets and the inner loop trains the model using the training set. Note that because the training set is small I divided the training samples into different batches and all the training samples are given to the network in each training loop.

In the second task, because use of weight decay and dropout simultaneously might not be a good idea. I decided to tune the hyperparameters into two different parts. In the second part, dropout probability is set to zero and weight decay is used for L2 regularization. In this part, the pruning ability of Optuna is activated to stop the trials which are not close to optimum.

Model:

As it was mentioned in the second paragraph there are two ways to optimize the number of layers. The function named **oneTwo** uses a more complex method but it is not used. The class named **Net** is a fully connected network with two hidden layers and it is used to optimize the parameters. There is a function named **initialize_weights** by calling it initialized with Kaiming distribution. Class Net_one_layer is a network with one hidden layer.

The function named **train** takes the network and data as input and does the training proccess

Obj is a function that is given to optuna. In this function the parameters that need to be tuned are defined using **trial.suggest**. In this function, the training loop is done and the error is calculated based on k-fold cross-validation. The same function with the same name is used for the second part one difference, and the difference is that to have a shorter time for training cross-validation error is not used.

Class **EarlyStopping** performs early stopping regularization method. In this class, the loss for every epoch is saved in a variable and if the loss for the next epochs is increasing for a certain number (here it is set to 150) the training loop will break.

Function **weight_reset** is used in k-fold cross-validation to reset the network weights after training the network on each fold of data.

Function **accuracy** takes the original data and predicted data and calculates the accuracy of predicted data. This function is used in the second task.

In the first part of the code for the classification task, before defining data loaders, the mean and variance of the fashion mnist dataset is calculated which are used to normalize the dataset for faster convergence.

Functions **train** and **test_loss** are used for the task of classification for a fully connected network.

The class named **Cnn** is a convolutional network with two convolutional layers followed by two fully connected layers. It takes the number of filters for each of channels one and two, the size of filters, the number of neurons in each fully connected layer, and dropout probability.

In the code, there is a part for exploring data augmentation. Since collecting new data is time-consuming and expensive one possible solution to enhance the accuracy is to perform some transformations on the existing data such as rotating, cropping, adding random noise, flipping horizontally and vertically and etc.

In this part, we used the following transformations on the data: random rotation, random horizontal flip, and random erase.

Then the original dataset is augmented with the transformed dataset. The final length of the augmented dataset is 8000 and a dataset with size of 1000 is used for calculating validation error.

Results

Regression task

The optimization process is divided into four parts. In the first part, early stopping is used during optimization for a network with two hidden layers. In the second part, early stopping is not used for optimizing a network with two hidden layers. The same is done for a network with one hidden layer. In all four parts number of trials of optuna is set to 300.

Part one: optuna for a network with two hidden layers and early stopping

The found parameters are shown in table 1. Then the network with these parameters is trained for 2000 epochs using 90 percent of the training data for training and 10% of this data for validation. The learning curve for train and test error is shown in Figure 1. And predicted points for training and test data are shown in Figure 2.

Part two: optuna for a network with two hidden layers without early stopping

The results are shown in table 2, Figure 3, and Figure 4.

Part three: optuna for a network with one hidden layer and early stopping

The results are shown in table 3, Figure 5, and Figure 6.

Part four: optuna for a network with one hidden layer and early stopping

The results are shown in table 4, Figure 7, and Figure 8.

The **best result** is found in part one with **k-fold error** of **0.252**. Afterward, the network with hyperparameter found in part one (which are shown in table 1) is trained on full training data and the prediction of it for training and test data is shown in Figure 9. In the next part, the network weights were initialized using with **xavier_normal_** distribution and then trained it to see the effect of weight initialization. The learning curve and predicted points are shown in Figures 10 and 11. As it can be seen using this weight initialization is not good for this task.

Visualization of activation of the last layer of the network and weight histograms of both layers before and after training are shown in Figures 12, 13, and 14.

Classification task

In this task first, a network with two fully connected hidden layers is defined and trained for 10 epochs the final test accuracy achieved by this network is 0.854

Then a CNN network is defined with two convolution layers followed by 2 fully connected layers and trained with some arbitrary values and it reached to test the accuracy of 0.898 after 10 epochs.

The Optimization part is divided into two parts in the first part dropout is used besides other hyperparameters and in the second part L2 regularization parameter is used besides other hyperparameters. This is done because using these two regularization methods simultaneously might not result in a good performance.

The hyperparameters that resulted in the lowest validation error are represented in table 5. Next, the k-fold cross-validation for this network is calculated. K-fold cross-validation accuracy is 0.907, and k-fold loss is 0.284. Learning curves for this network are shown in Figure 15. The final test error and accuracy for this network is as follows:

Test error: 0.906**, Test accuracy:** 0.358

From the learning curves shown in Figure 15, we see that after a number of epochs training error is decreasing while validation error is constant which is a sign of overfitting. Therefore, in the next part trained the same network with the L2 regularization method for 50 epochs. The learning curves for this part are shown in Figure 16 and for this part, test error and accuracy are as follow:

Test error: 0.908**, Test accuracy:** 0.26

We see a slight improvement.

In the next section, the data augmentation method is explored. The optimized network is trained on augmented data for 21 epochs and it achieved validation loss of 0.236 and final validation accuracy of 0.922 and final test accuracy of 0.913 we see that this method resulted in a slight improvement of the final test accuracy.

The confusion matrix for the network trained on the augmented data is shown in Figure 17. As it can be seen the lowest accuracy is for ***Shirt*** which are mostly misclassified with coat, pullover, and T-shirt.

In the last part, we have a visualization of two convolutional layers filters and the output of these filters on one sample of the training dataset. The filters of the first and second layers are shown in Figures 18 and 19. One sample of the training dataset which is shown in Figure 20 is fed to the convolution layers and the output of these layers can be seen in Figures 21 and 22.

Finally, in Figures 23 and 24, the weight histograms of the fully connected layer of the network before and after training are shown. As it can be seen most of the weights after training has values close to zero.

In the additional part, **Grad-Cam** has been implemented using PyTorch hooks. In this part, a new model with weights of the model trained model is defined. In this new class after the last layer of the convolution layer, a PyTorch hook is used to retrieve the gradients of this conv layer. Then a sample is fed to the model and gradients of the output with maximum value are achieved then the average of these gradients for every filter is calculated. Afterward, the means are multiplied to the activation of the last layer and again average of them is calculated.

The performance of the grad-cam on the input sample is shown in Figure 25.

Appendix

Table 1. The output of optuna for a network with two hidden layers while using Early stopping

Optimizer	Learning Rate	Neurons of first hidden layer	Neurons of second hidden layer	Dropout	K-fold error
RAdam	0.007248	108	64	0.0014	0.2516

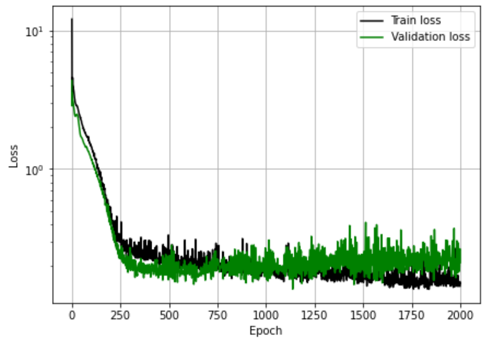

Figure 1. Test and validation error for output of optuna for network with two hidden layers

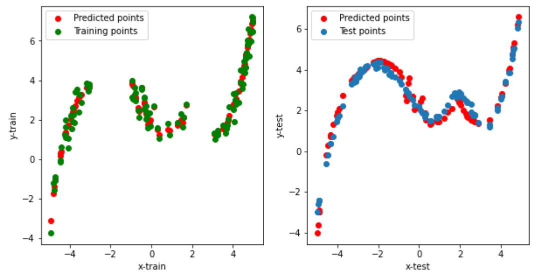

Figure 2. Prediction of trained network with two hidden layers for test and train data (output of optuna using early stopping)

Table 2. The output of optuna for a network with two hidden layers without using Early stopping

Optimizer	Learning Rate	Neurons of first hidden layer	Neurons of second hidden layer	Dropout	K-fold error
RAdam	0.0185	124	80	0.0556	0.252

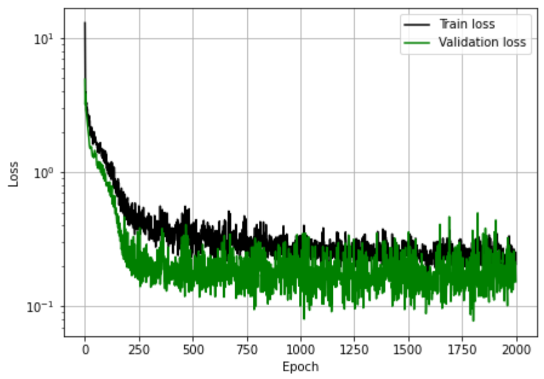

Figure 3. Learning curve of network found with two hidden layers without using Early stopping

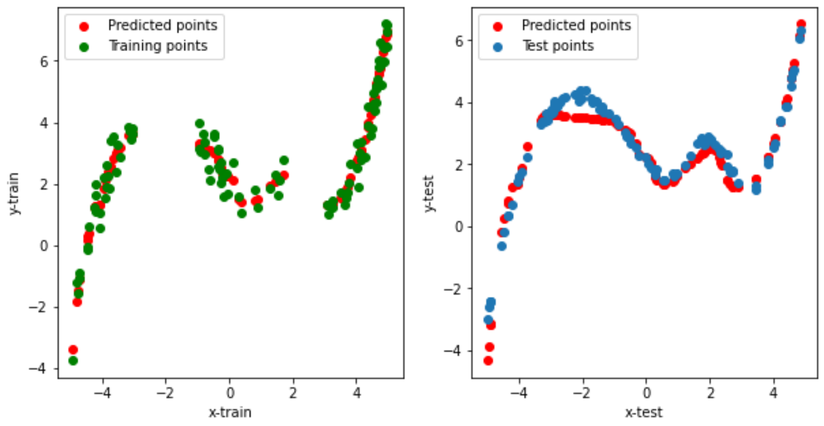

Figure 4. Prediction of trained network with two hidden layers for test and train data (output of optuna without using early stopping)

Table 3.The output of optuna for a network with one hidden layer while using Early stopping

Optimizer	**Learning Rate**	**Neurons of hidden layer**	**Dropout**	**K-fold error**
Adam	0.00988	40	0.0754	0.3704

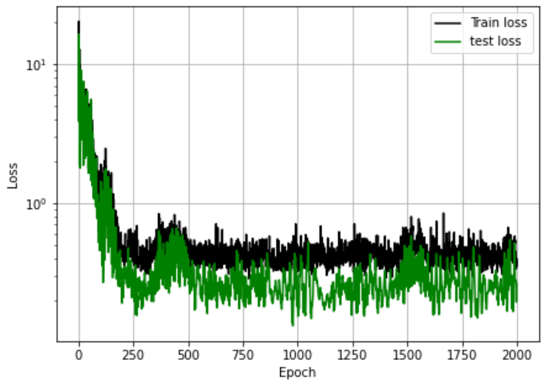

Figure 5. Learning curve of network found with one hidden layer while using Early stopping

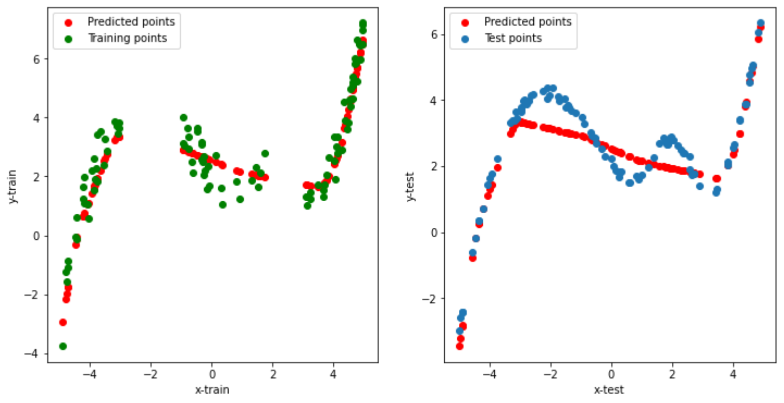

Figure 6. Prediction of trained network with one hidden layer for test and train data (output of optuna while using early stopping)

Table 4.The output of optuna for a network with one hidden layer without using Early stopping

Optimizer	Learning Rate	Neurons of hidden layer	Dropout	K-fold error
SGD	0.00845	76	0.0717	0.28

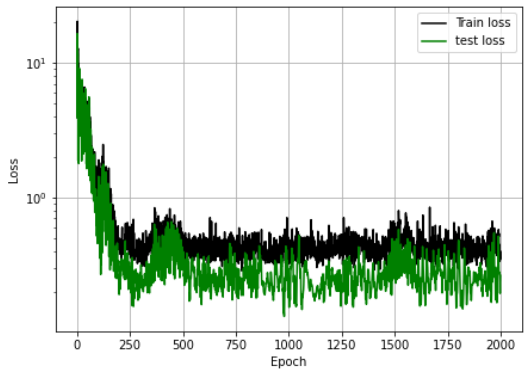

Figure 7. Learning curve of network found with one hidden layer without using Early stopping

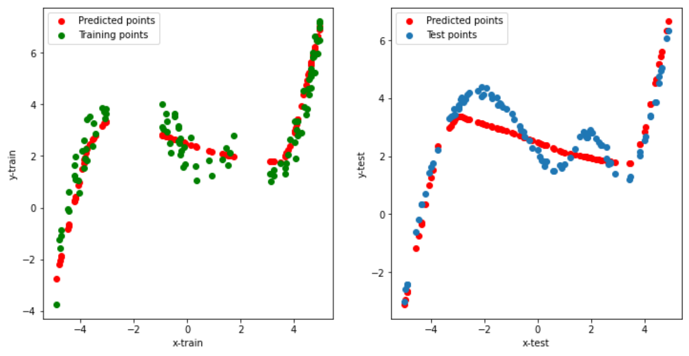

Figure 8. Prediction of trained network with one hidden layer for test and train data (output of optuna without using early stopping)

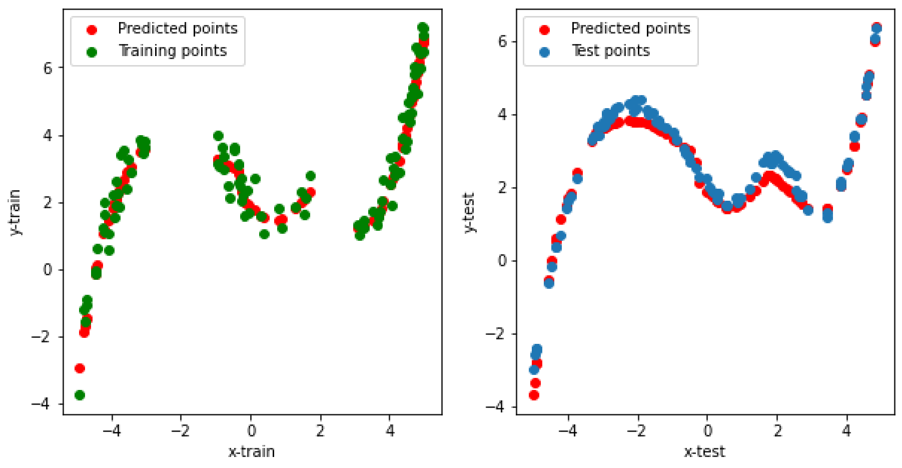

Figure 9. Prediction of trained network with best hyperparameters found (hyperparameters of table one)

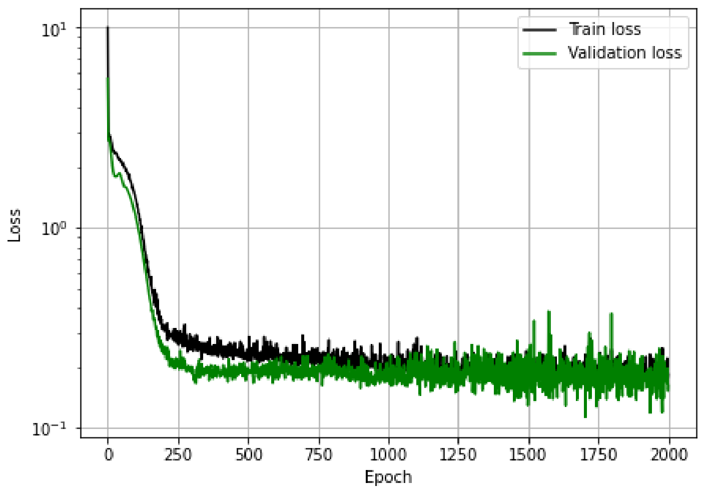

*Figure 10. Learning curve of network with **xavier_normal_** distribution for initializing weights*

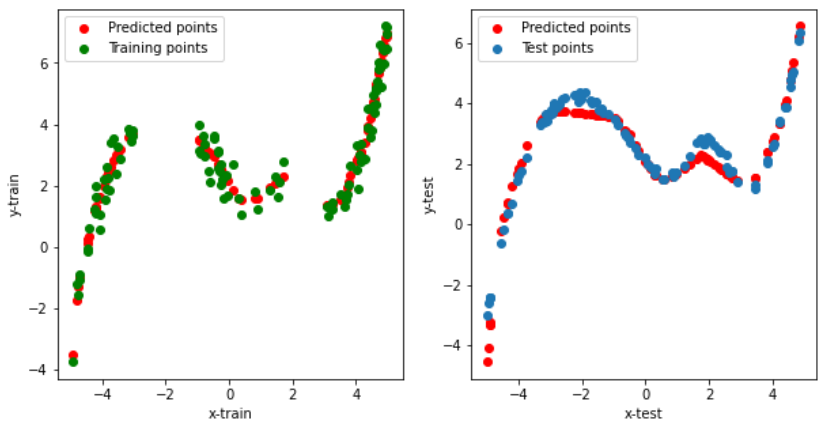

*Figure 11. Prediction of trained network with **xavier_normal_** distribution for initializing weights*

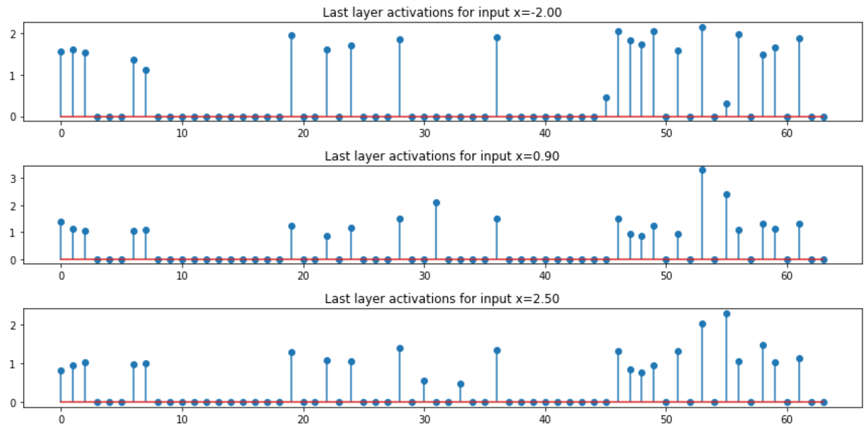

Figure 12. Visualization of activation of last layer of the network

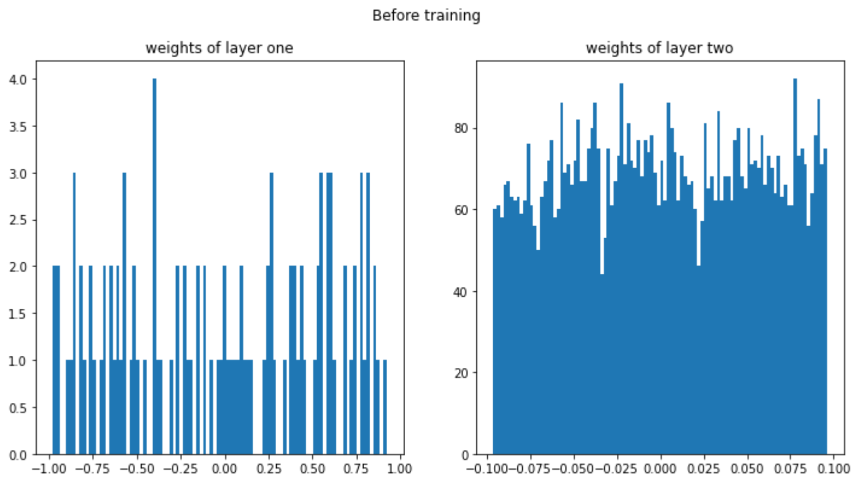

Figure 13. Weight histograms of network before training (initial weights)

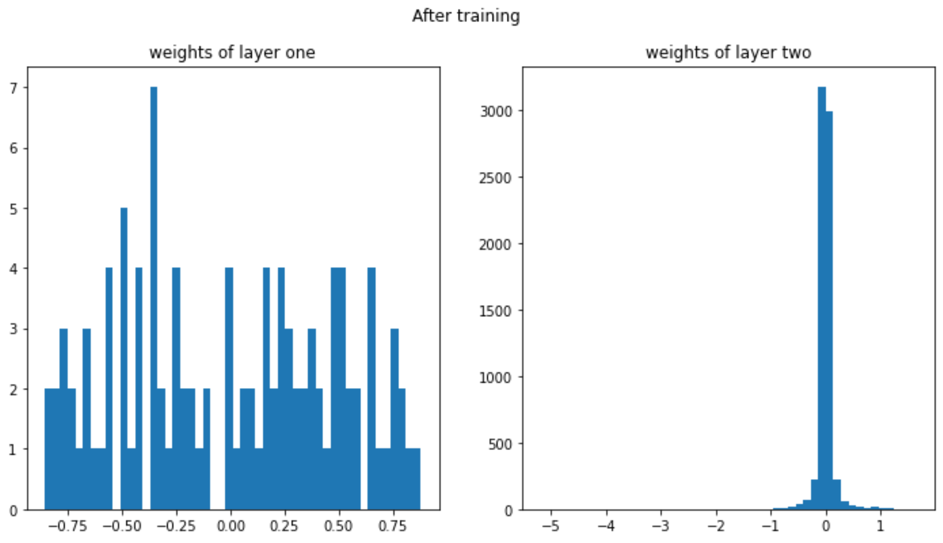

Figure 14. Weight histograms of network after training

Table 5. Optimized hyperparamers of CNN

Optimizer	Learning Rate	filters of first conv layer	filters of second conv layer	Neurons of first hidden layer	neurons of second hidden layer	Dropout	K-fold error
Adam	0.001361	54	50	80	40	0.0556	0.252

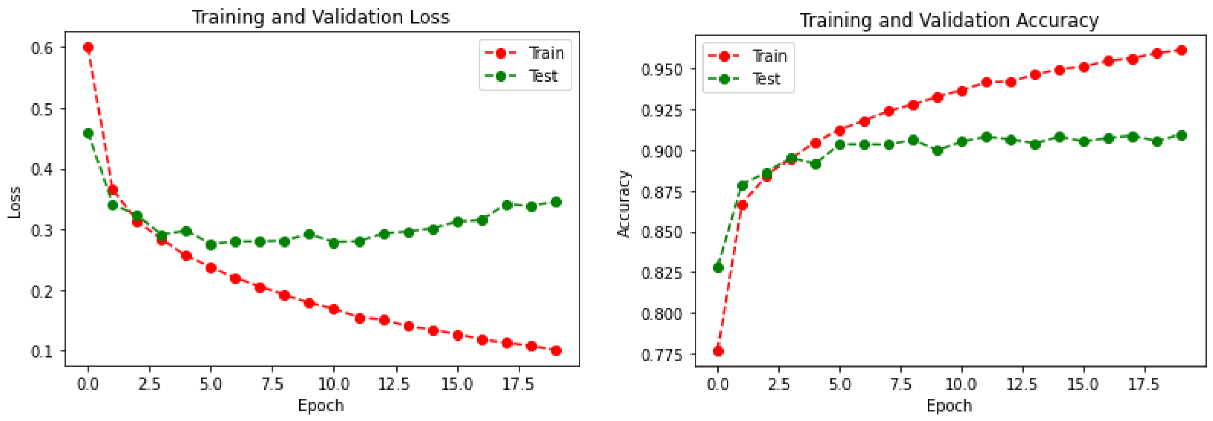

Figure 15. Learning curves of optimized CNN

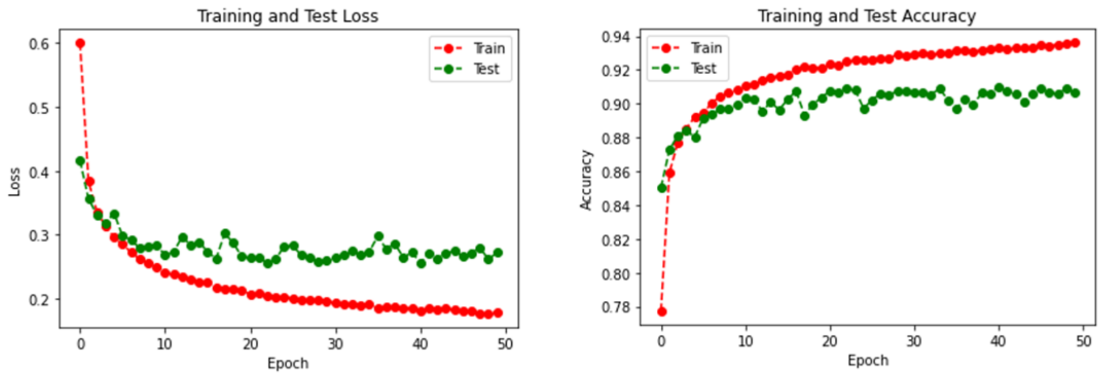

Figure 16. Learnin curve of optimized CNN with L2 regularization

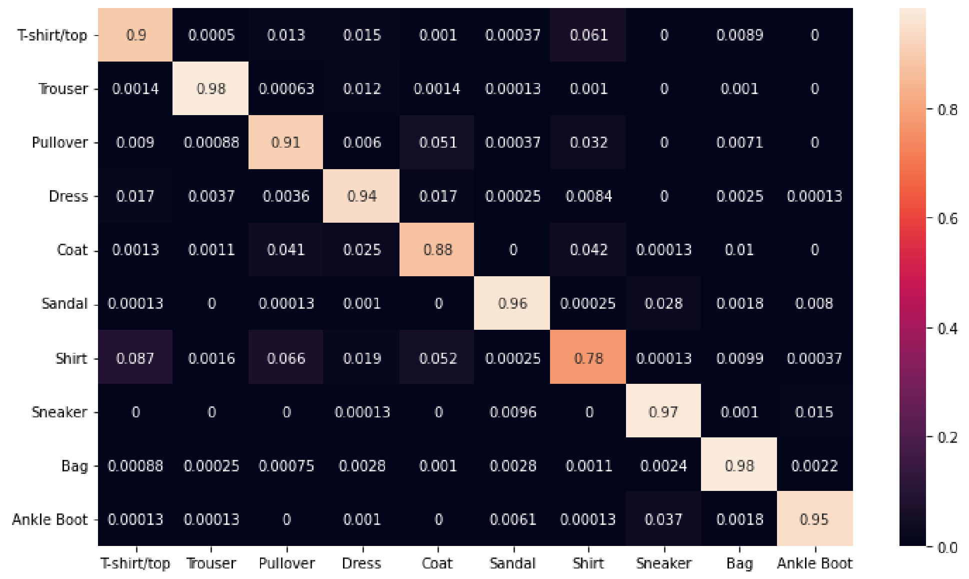

Figure 17. Confusion matrix of augmented data

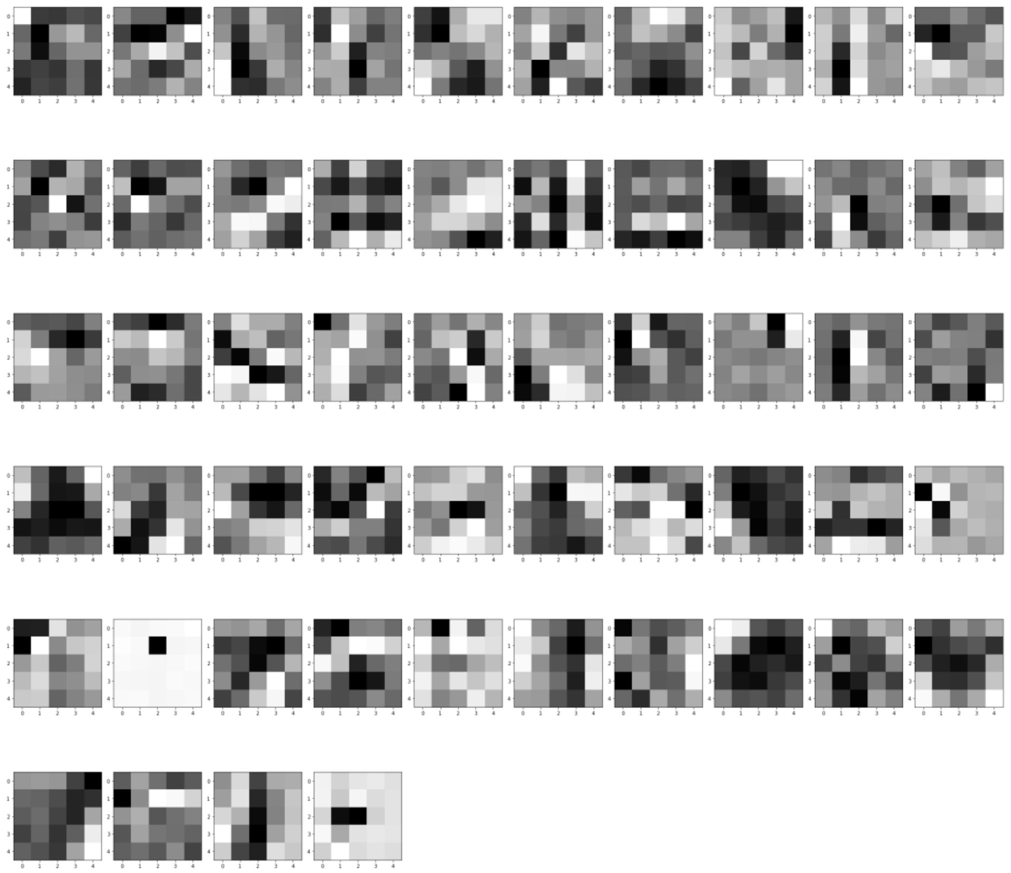

Figure 18. Visualization of filters of first convolutional layer

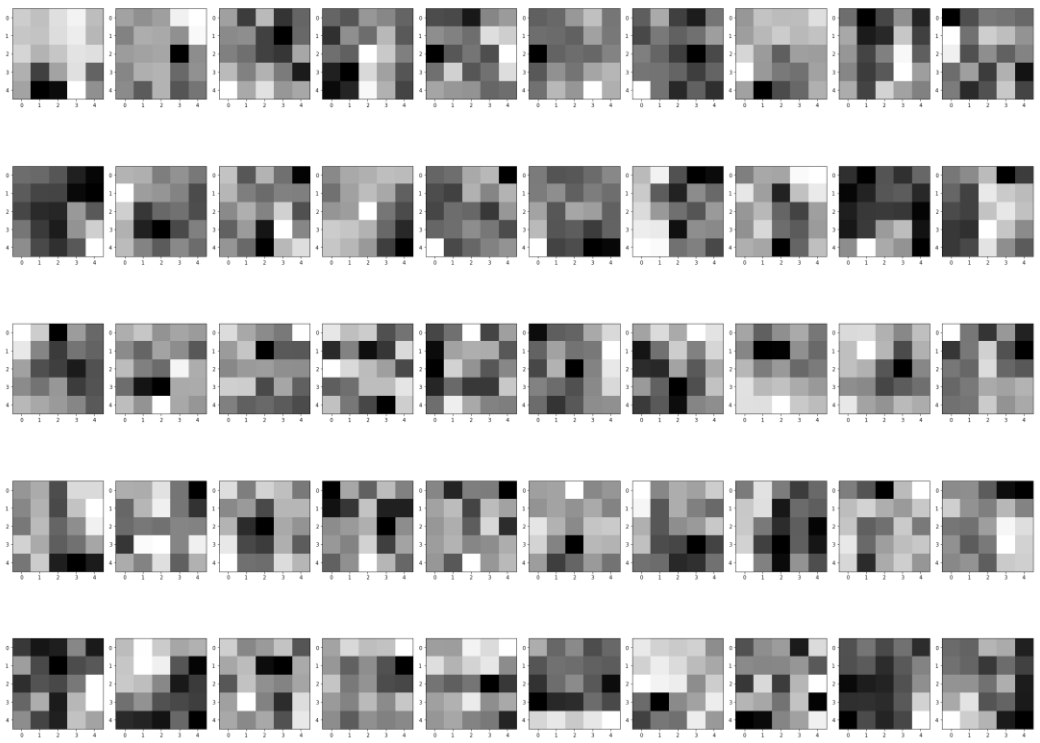

Figure 19. Visualization of filters of second convolutional layer

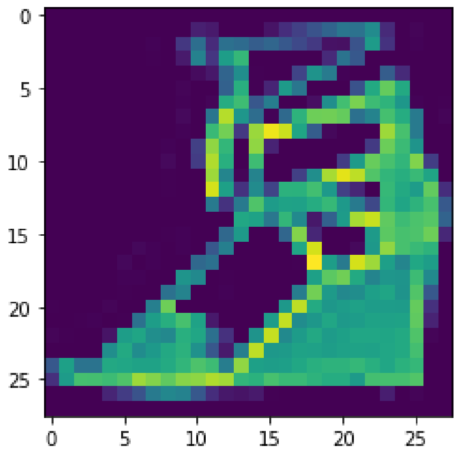

Figure 20. One sample of test dataset

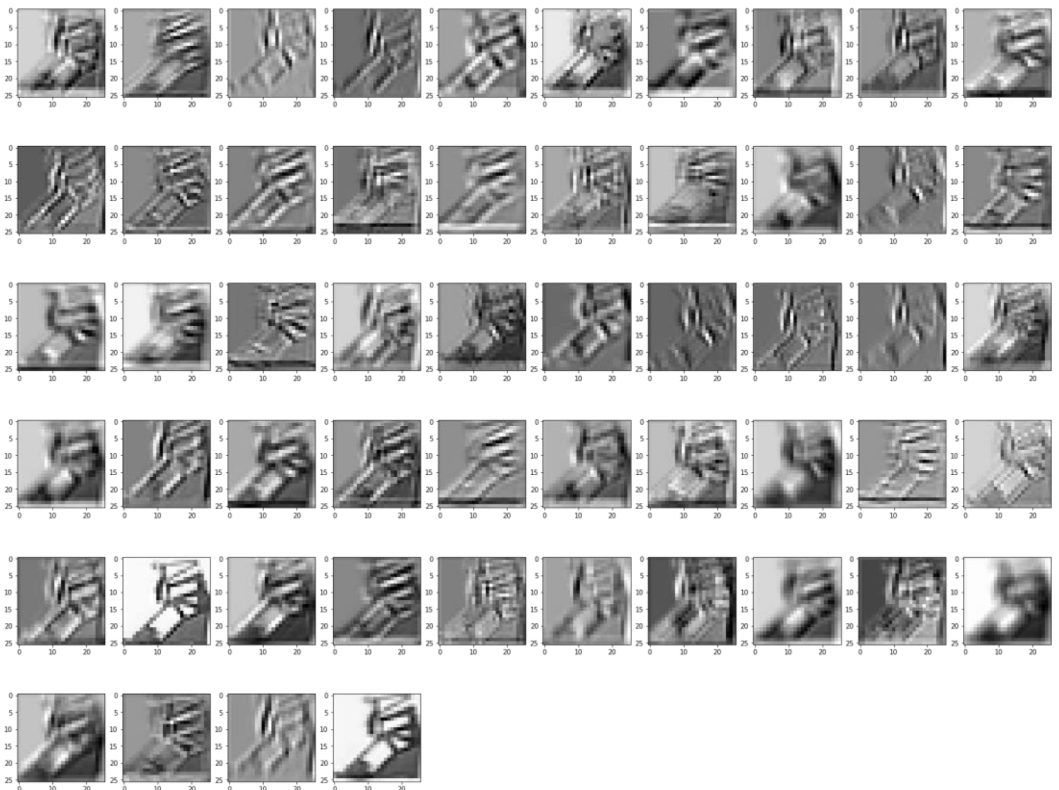

Figure 21. Output of first layer filters of for the test sample

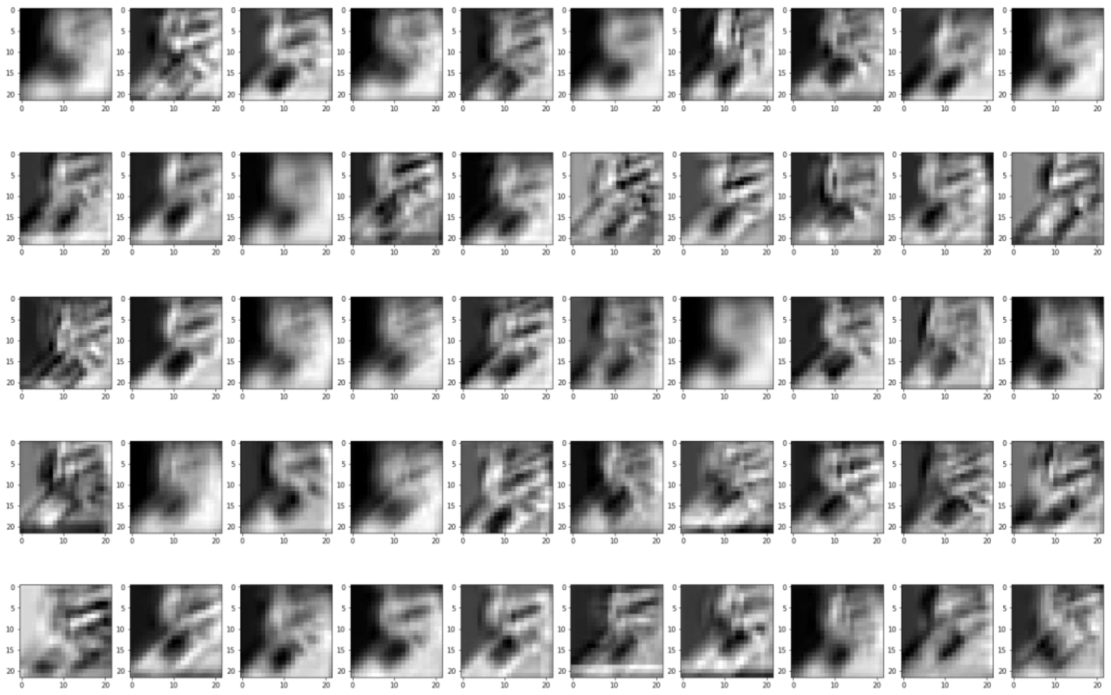

Figure 22. Output of second layer filters of for the test sample

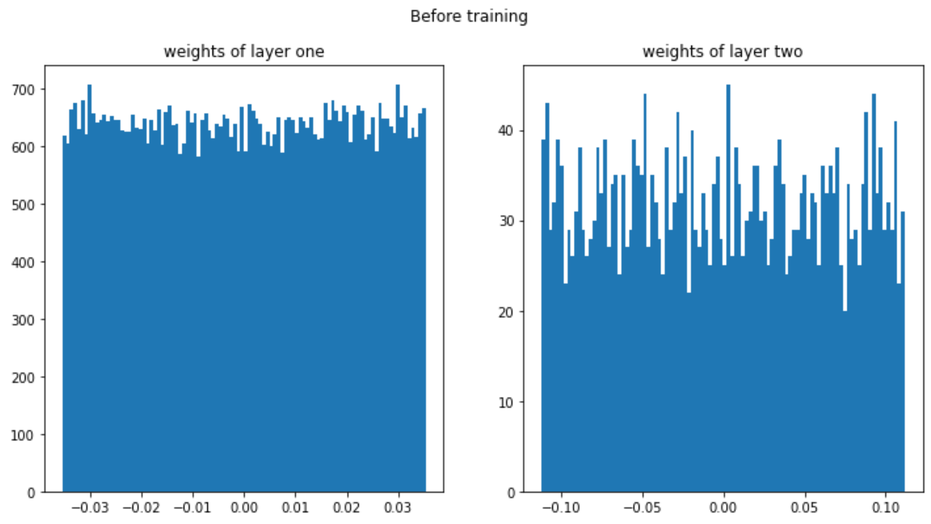

Figure 23. Weight histograms of fully connected layers before training

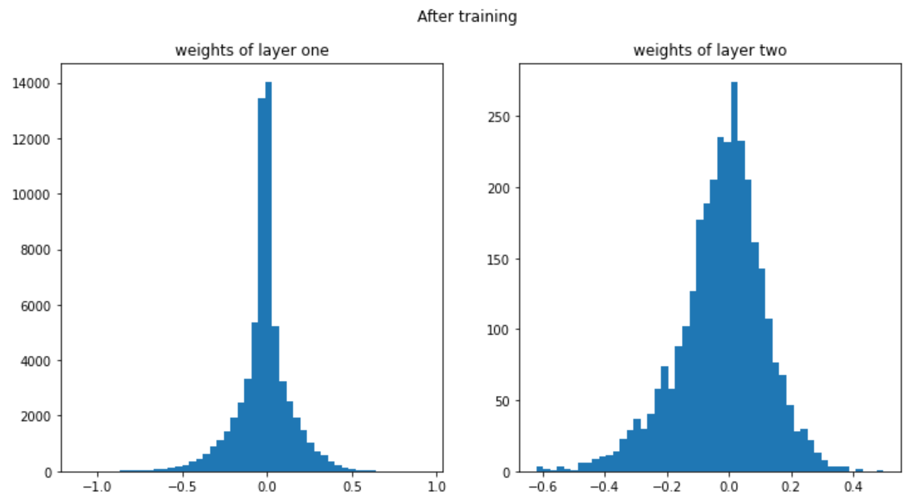

Figure 24. Weight histograms of fully connected layers after training

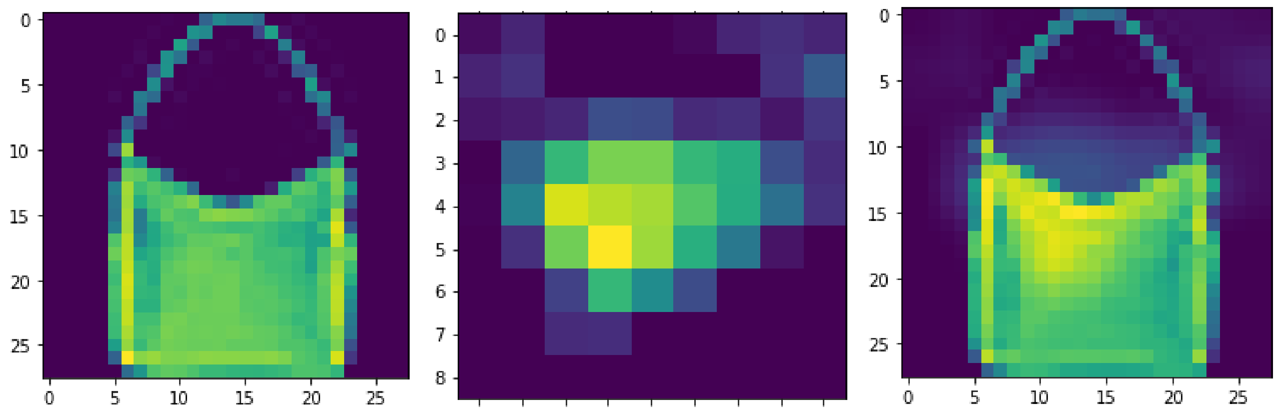

Figure 25. Input of grad-cam and output and both in the same image