Name: Gazi Eren Tuncay
ID: 2305522

Middle East Technical University

Department of Electrical and Electronics Engineering

EE583 Pattern Recognition

Term Project Report

1. Introduction

Deep learning is a new-fashion concept that is utilized in many engineering applications, recently. As we discussed in the lectures, we may encounter methods based on this idea in various topics and with different algorithms. For the term project of the EE583 Pattern Recognition Course, I have developed an MLP algorithm that aims to find solutions for the pose estimation problem in the presence of noise.

The pose estimation problem I have handled is about determining the pose of the camera with the information we have in the system. One solution method to this subject is based on the iterative Perspective n-point (PnP) algorithm. That is, given the 3D-space positions and corresponding 2D-image locations of n points and intrinsic camera matrix, the algorithm finds the optimal solution for the camera pose in terms of coordinates in x, y and z-axis, and rotation angles of the axes.

I have developed my program in Python using OpenCV library. Also, I have worked with Google Colab as it provides GPU execution, which significantly accelerates the training of neural networks.

2. Experiments and Results

2.1. Generation of Datasets

Since our aim is to estimate the ground truth pose of the camera, we could make this by determining the possible errors in 6 parameters of the camera pose that is estimated by the PnP algorithm. For this, I have constructed inputs of the MLP as 50 3D-2D point matches and outputs as the pose estimation error in the rotation angles and transformations in x, y and z-axis.

Furthermore, we should consider the randomness of the problem. Since we add Gaussian noise onto the image points, we also need to repeat the executions for a number of Monte Carlo runs. In particular, I have added Gaussian noise with a standard deviation from 0 to 2 with a step size of 0.05 and repeated the process for 100 Monte Carlo runs.

In parallel to all these, I have also created 100 object-image point match groups. In the end, with 40 noise levels, 100 Monte Carlo runs and 100 different creations, I had a set of 400000 vectors. Since this is quite large for this simple problem, I have shuffled the dataset and taken 100000 samples of point matches with corresponding 100000 ground truth pose estimation errors from the PnP algorithm, for the training data. I have also chosen 4000 as the sample number for the validation dataset.

2.2. Building MLP Architecture

Parameters and choices for learning methods are one of the most important factors in the training of neural networks. In my experiments, I have determined an MLP architecture as the reference and made changes in the network to observe the effects. Reference construction consists of 2 hidden layers, which have 50 and 10 hidden units, respectively in the forward direction and each having ReLU activation function. During the training procedure of this neural network, I used Adam optimizer with a learning rate of 0.01 and MSE as the loss function for learning. I have also grouped my data into batches of 50 pair vectors and completed the training for 5 epochs.

For the comparison of results, I have used cumulative training error and validation error plots at the end of each 10 batches during the training. Also, I provide the histogram of the 6 pose error estimations at the end of training for the validation dataset.

2.3. Experiments

There can be different combinations of neural network parameters for well-enough solutions to particular problems. In my experiments, considering the architecture that was described previously as the reference, I had only one controlled variable and observed the results. The following subchapters are related to the effect of changed parameters and discussions.

2.3.1. Number of Hidden Layers/Units

At first, I compared the reference architecture with a deeper one which has more hidden layers and units. For this, I have constructed a similar MLP but having 4 hidden layers, consisting of 50, 100, 50 and 10 hidden units in the forward direction. The results of training and validation errors can be seen in Figure 1.

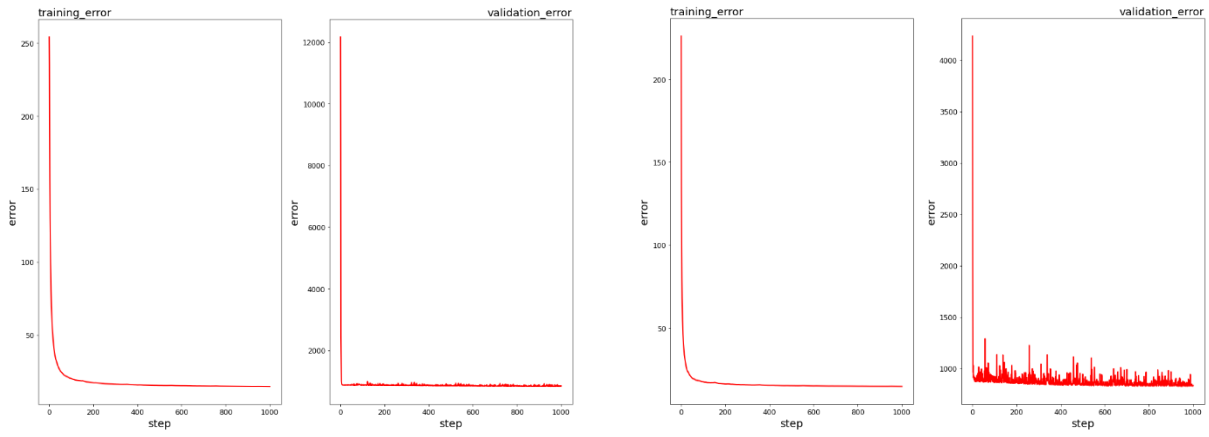

Figure 1*. Training and validation error curve for reference (left) and deeper (right) architectures.*

In the training error curves, it is seen that the one for deeper architecture has a slightly sharper decrease compared to the reference architecture. The plot on the right in Figure 1 also has more fluctuation in the validation error curve. These indicate that the deeper architecture requires more training data for better generalization performance. As another metric, I have also shown the pose error estimations of the validation data at the end of the training in Figures 2 and 3, for 6 parameters of the camera pose.

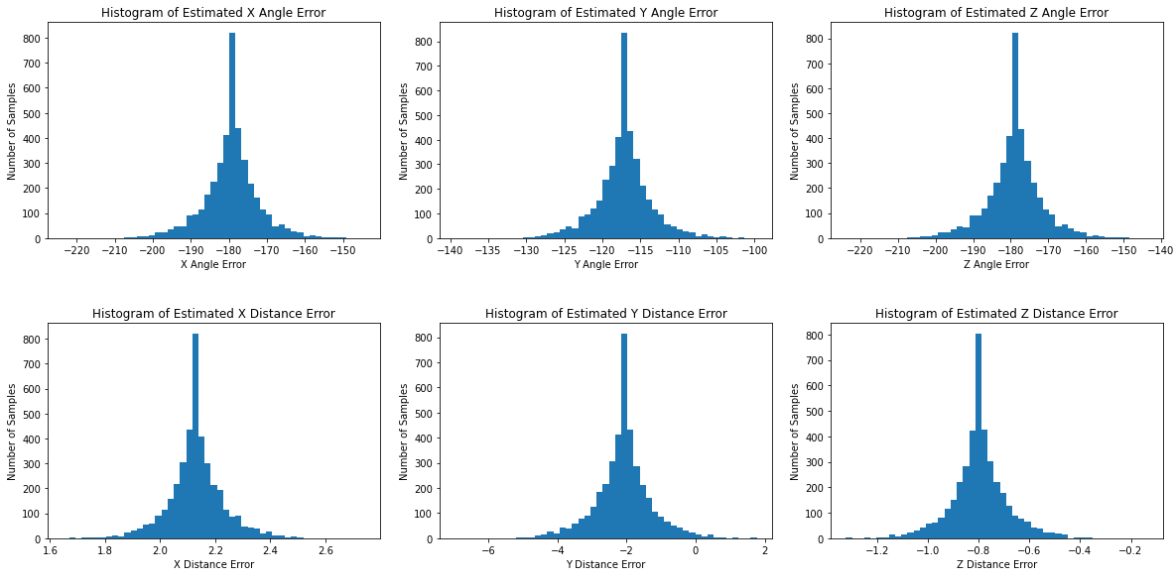

Figure 2. *Distribution of validation data for reference architecture.*

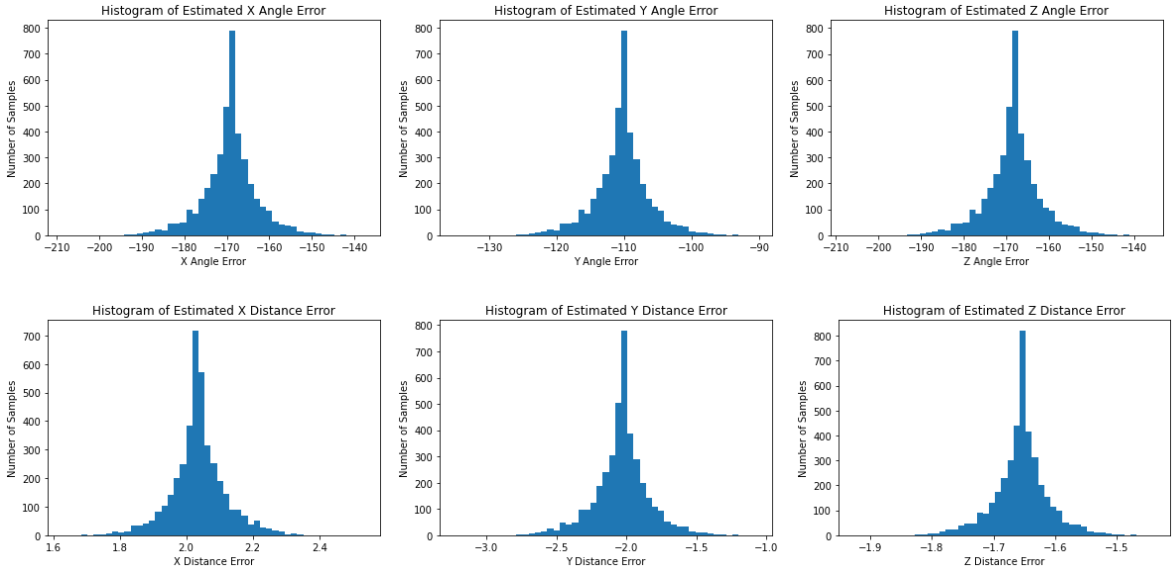

Figure 3. *Distribution of validation data for deeper architecture.*

Although they seem similar for both trainings, it can be concluded that the distributions are more symmetrical around the mean values in the distributions for reference architecture. When more training data is provided for deeper MLP, the validation data distribution could be closer to the expected.

2.3.2. Batch Size

Returning back to the reference architecture, I have changed the batch size from 50 to 10. In this way, I had 5 times more batches of samples and the training procedure took more time. The results of the error curves can be seen in Figure 4.

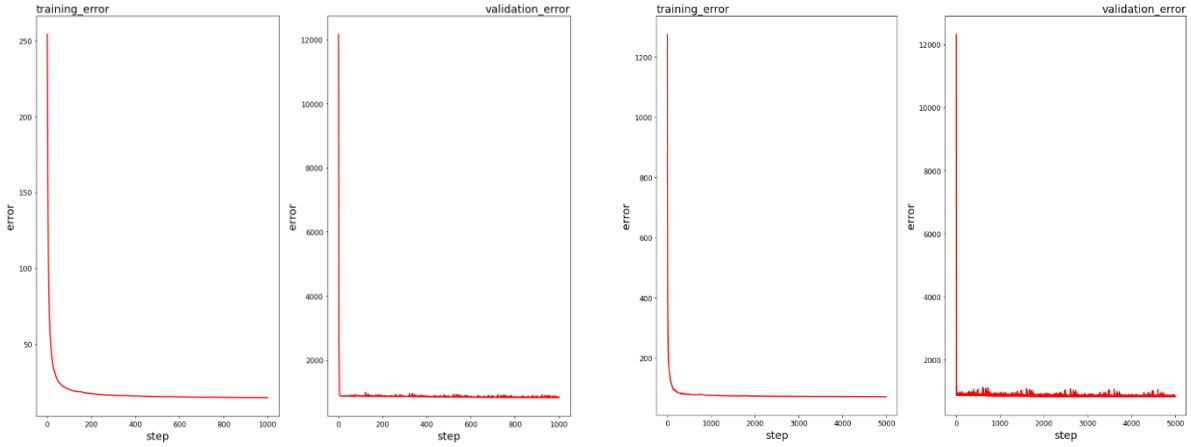

***Figure 4**. Training and validation error curve for batch size of 50 (left) and 10 (right).*

In the performance curves, it is seen that the less batch size increases the number of steps in the training and tightens the curves in time. That is, it comes up with similar training but consumes more time. Also, since each batch affects the parameters of the neural network, the validation error has more fluctuations with less batch size. The histogram of camera pose errors at the end of training can be seen in Figure 5 and 6.

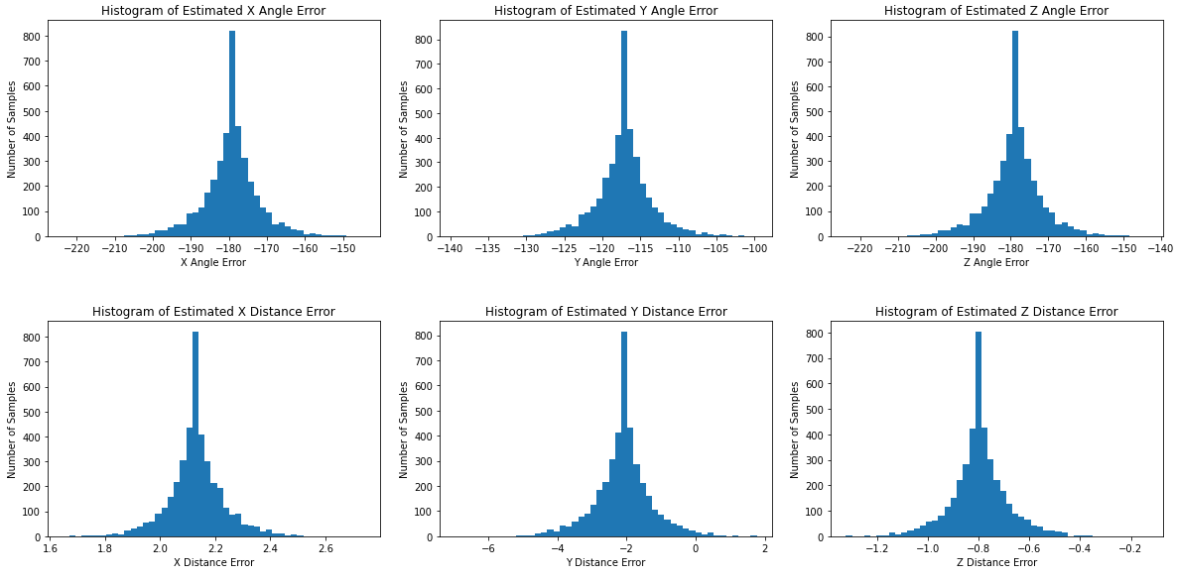

***Figure 5**. Distribution of validation data for 50 batch size.*

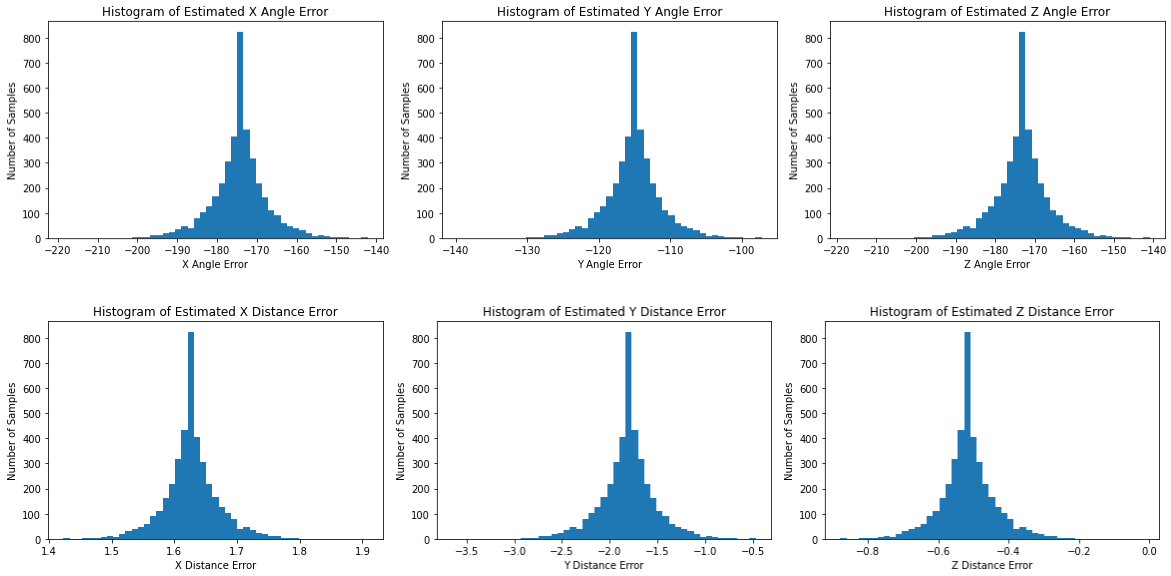

***Figure 6**. Distribution of validation data for 10 batch size.*

As mentioned previously, the learning of the MLP is similar for both batch sizes and the resulting histograms for transformations and rotation angles are similar for batch sizes of 50 and 10.

2.3.3. Learning Rate

In this execution, I have changed the learning rate of the training from 0.01 to 0.1. The resulting training and validation errors for this comparison can be seen in Figure 7.

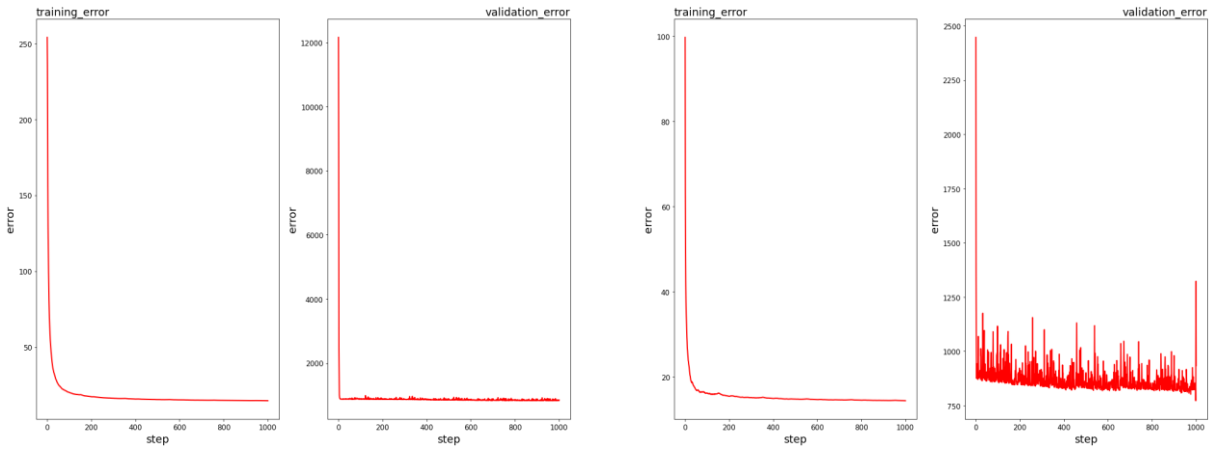

***Figure** 7. Training and validation error curve for learning rate of 0.01 (left) and 0.1 (right).*

The effect of the learning rate is one of the most clearly seen behaviors in the training of neural networks as it is very crucial for optimization. When the learning rate increases, the algorithm may not find the optimum point easily. This characteristic can be seen in the plots in Figure 7. Since the operation is done around some points with a large learning rate, the training error has unexpected disturbances and the validation curve shows more fluctuations. The resulting validation data distribution at the end of training can be seen in Figures 8 and 9.

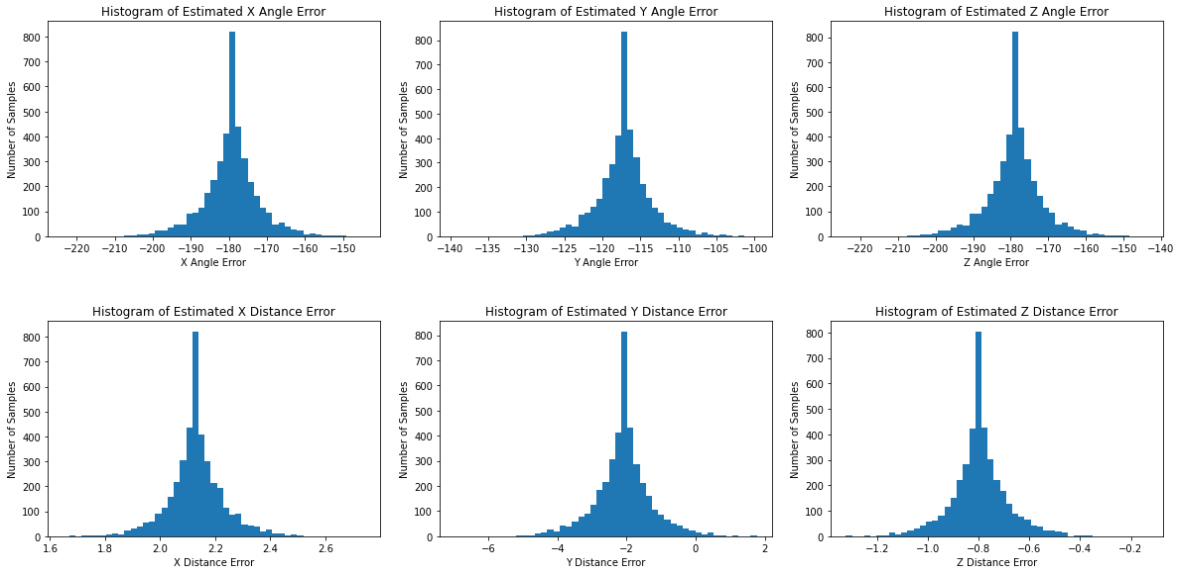

***Figure 8**. Distribution of validation data for 0.01 learning rate.*

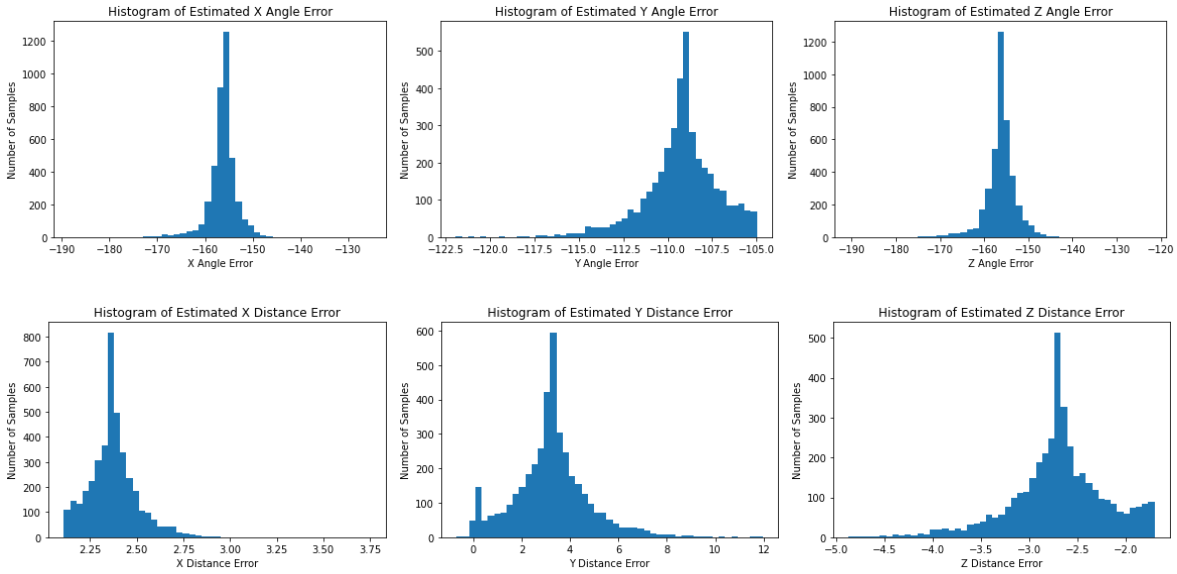

***Figure 9**. Distribution of validation data for 0.1 learning rate.*

As clearly shown in the histograms, the higher learning rate causes wrong learning for some parameters of the camera pose and hence the Gaussian-type distribution is damaged.

2.3.4. Number of Epochs

Another parameter for the training procedure of neural networks is the number of epochs, i.e. how many times the training is done over the whole training dataset. The comparison of the trainings with 5 and 1 number of epochs in terms of error curves can be seen in Figure 10.

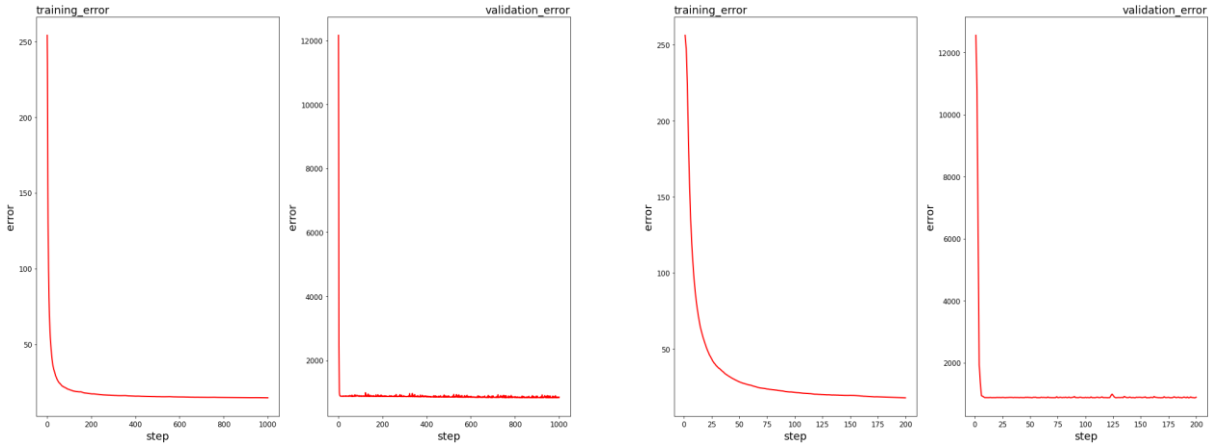

Figure 10. *Training and validation error curve for epoch numbers of 5 (left) and 1 (right).*

When we train our network over the training dataset more, we expect better learning up to some point, before overfitting. This fact is proven in Figure 10 without any doubt. As seen in the training error plots, by choosing the number of epochs as 1, we cut the probability of having less training error at the end. While it is more time-consuming with higher number of epochs, we prefer that since it is a good trade-off for better learning of the neural network. The histograms of the validation data at the end of training for both cases can be seen in Figures 11 and 12.

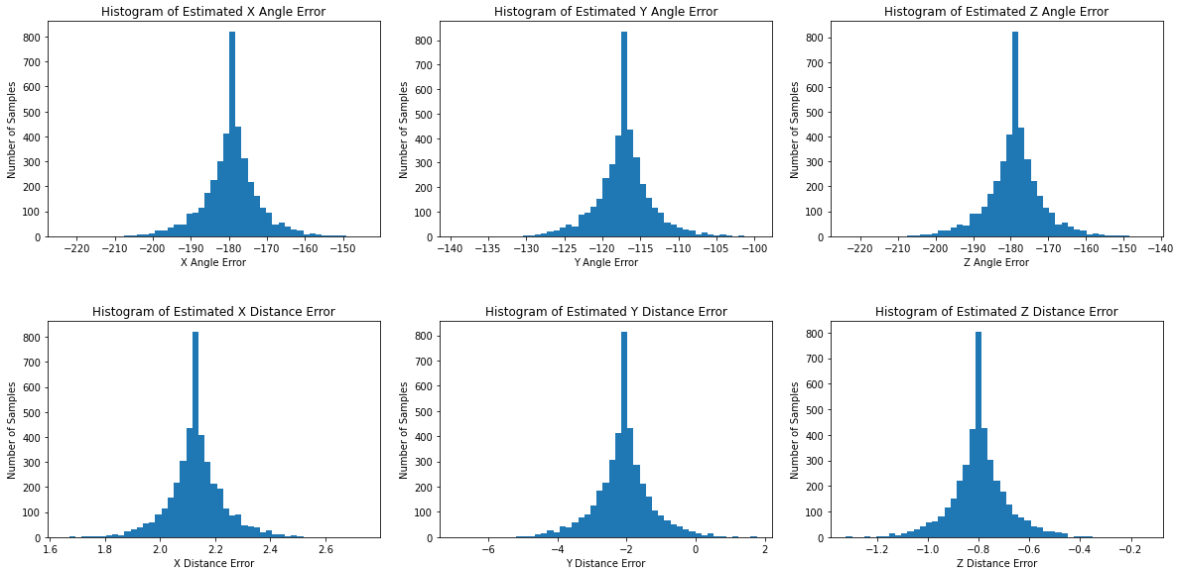

Figure 11. *Distribution of validation data for 5 epoch numbers.*

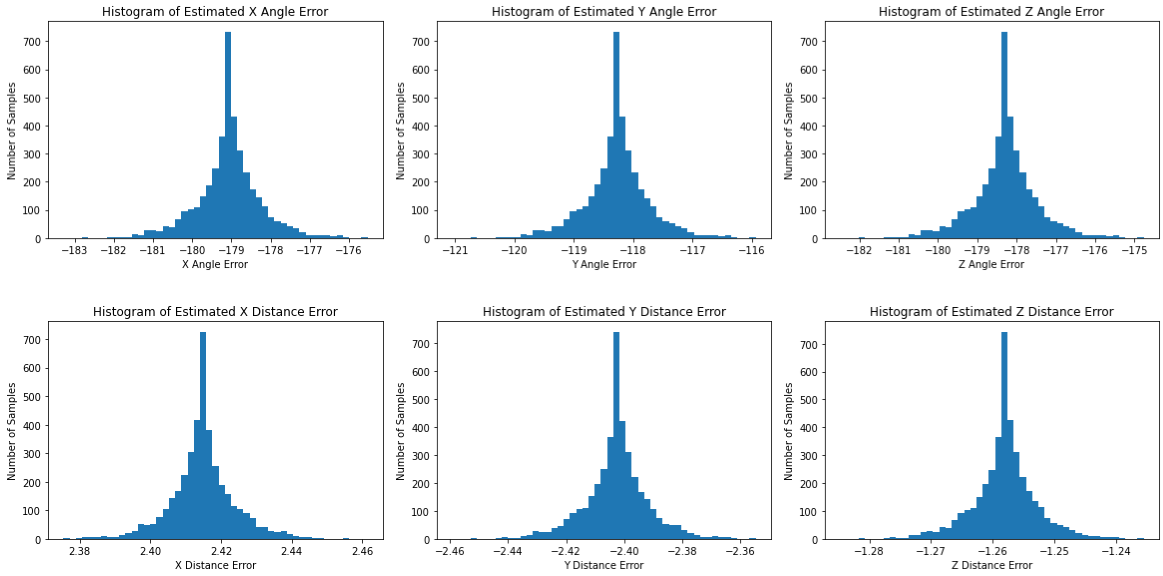

***Figure 12**. Distribution of validation data for 1 epoch number.*

In the histograms of validation data, I was faced with actually an unexpected result. Both cases have similar distributions and the one with 1 epoch has less variance compared to the reference case. At this point, my comment is that the validation of the neural network is problematic and I need to work on that part since I expect better results for 5 epochs.

2.3.5. Optimizer

Another characteristic that I changed in the neural network training to investigate the effect is the optimizer choice. As an alternative to Adam optimizer, I have also used Stochastic Gradient Descent (SGD) optimizer with the same parameters. The performance curves of my network for both optimizers are shown in Figure 9.

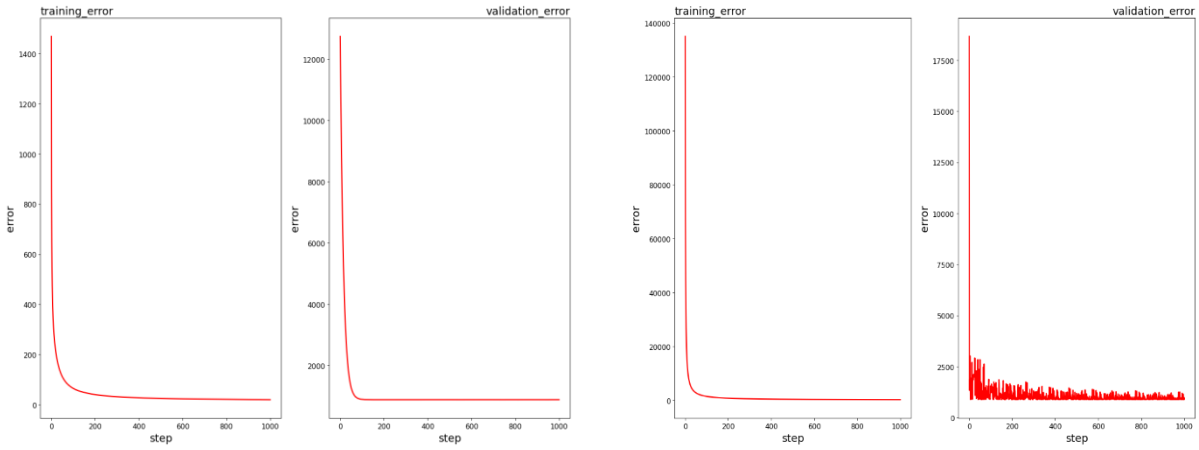

***Figure 13**. Training and validation error curve for different trainings with SGD optimizer.*

At this part, instead of comparing Adam and SGD optimizers, I have decided to show the results of two different trainings with SGD optimizer. In Figure 13, it is seen that this optimizer is quite sensitive to initialization and shows unrelated performances with varying values in different executions. Also, the validation data output at the end of the training was

constant for camera pose errors. Therefore, I considered SGD as not a suitable choice for this problem or this architecture.

2.3.6. Input Data Mapping

As the last experiment over the MLP architecture, I mapped the input data by using sine and cosine functions and completed the training with the corresponding camera pose errors in the datasets. The performance curves for this method is given in Figure 14.

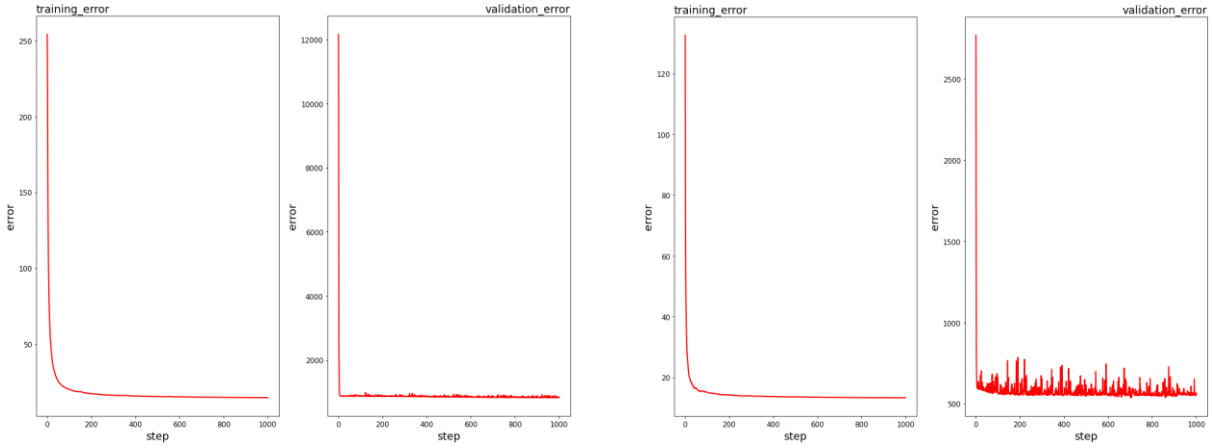

Figure 14. *Training and validation error curve for unmapped (left) and mapped (right) inputs.*

It is seen in the validation error plot that the training with mapped inputs leads to better results whereas the training curve remains the same. As another metric, the output of the validation data at the end of training is shown for 6 camera pose parameters in Figure 15 and 16 for comparison.

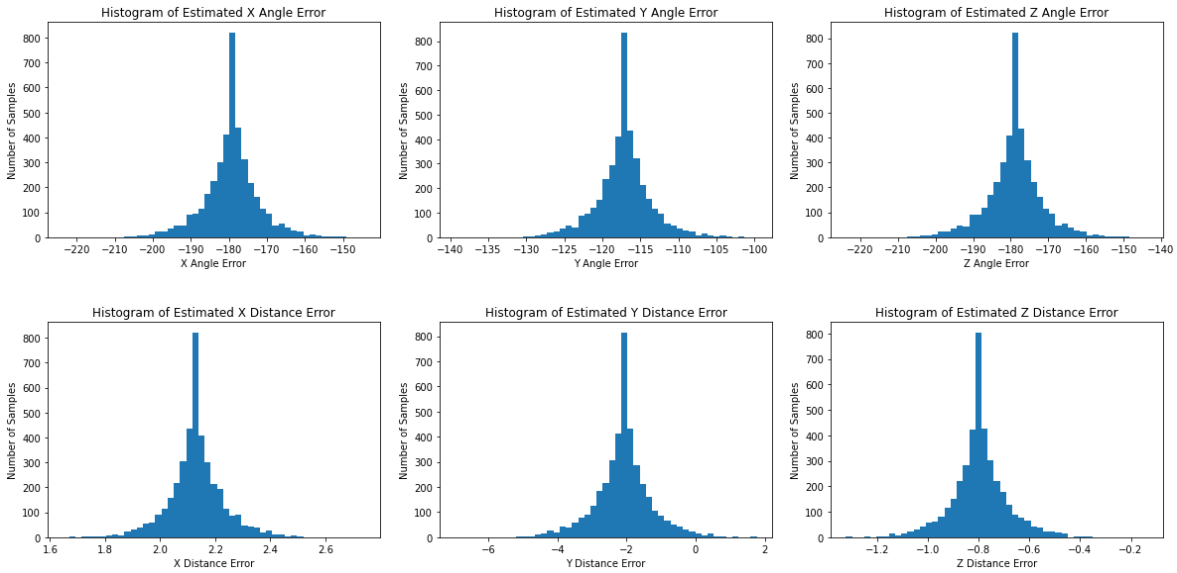

Figure 15. *Distribution of validation data for unmapped inputs.*

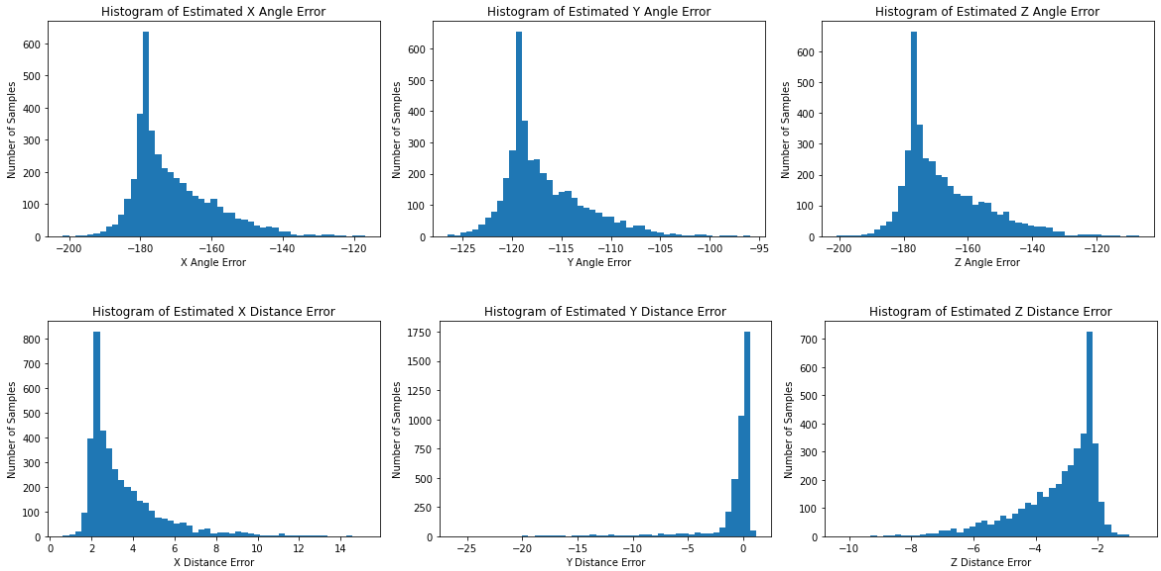

***Figure 16**. Distribution of validation data for mapped inputs.*

In recent studies, the advantage of mapping the inputs is mentioned and it was also seen in Figure 14. On the other hand, histogram plots do not point out the same way. Both the Gaussian characteristic and error margins are obtained unexpectedly disturbed.

3. Conclusion

To sum up, I have done several experiments on an MLP architecture by changing different parameters and observed the results for EE583 Pattern Recognition Course Project. The problem that we handled is the pose estimation of a camera using PnP algorithm and it is highly dependent on random choices or small factors in the datasets. As there are many issues to consider for the training of neural networks, we should pay attention to choosing the parameters of the architectures. I have shown the effects of each parameter change and their possible reasons. Although the results are not the best for the solution of the pose estimation problem, I consider them promising in the future with more emphasis on this study.