**MULTI-LAYERED PERCEPTRON (MLP)**

Multi-layer perceptron is one of the methods that we have used beside the Convolutional Neural Network and Transfer Learning. Multi-layer perceptron is based on the idea of the Perceptron which initially designed as a simple classification algorithm, proposed by Cornell scientist Frank Rosenblatt[1]. The model is inspired from the basic unit of the human nervous system, neurons. The basic idea behind this approach is that each neuron has its own response pattern which can be modelled mathematically, which is called an activation function. The cumulative responses of these neurons are the reason behind understanding and applying deep learning to this type of structure.

One layer in the structure is composed of many neurons. When these layers are placed in series, and each neuron is connected to each other neuron in the previous and preceding layer, the multilayered perceptron structure is complete. Each neuron is connected to the next layer’s neurons with a particular weight and the linear combination of the preceding layers’ neurons and their corresponding weights determine the value of the neuron at the next layer. These weights are updated via back-propagation which tunes these parameters by evaluating the weights according to a specified loss function.

Backpropagation uses a gradient descent approach to train the network and uses the chain rule to take the derivatives of the activations of each output neuron with respect to the all its prior connections. This way, the weights are updated, and the network is trained.

In order to achieve a reasonable outcome that is comparable with the other models, we initially preprocessed the image data. We have first transformed the image into its grayscale representation. Since in this task, road curvature and the main orientation of the image are the most valuable features we want to obtain, we have put these images through a Sobel filter to detect the edges of the images. In the progress report, we have reported two different edge detection algorithms to choose from for this purpose. By trial and error, we concluded that we should use Sobel instead of Canny since adding Gaussian blur smoothens the image data and makes it easier for the gradient kernels to see the edges. This way, we have decreased the training time by lowering the number of dimensions of the data and made it easier for the neural network to learn the road curvature and by this way, we have included a manual filter that enabled this algorithms accuracy to compete with the other two deep learning algorithms. Below are the two example images one is the original and the other is the filtered version of the image.



**Fig XXX:** The original and filtered images

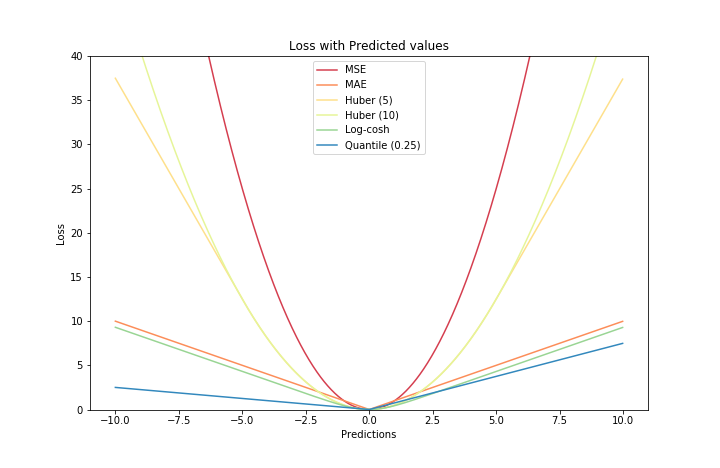
In training the neural network deciding on the hyperparameters such as the number of layers to be used and number of neurons each layer carry, the batch size that is used for training and the number of epochs used for training. Whether a batch normalization will be applied or not. Furthermore, there is no textbook way to design a neural network so the model that is finally constructed has been through many different experimental alterations throughout the design process.

The design has started with a simple neural network that had only two hidden layers but with a high number of neurons at each of those layers. Then, we observed that this approach was insufficient in terms of extracting the features that could explain the data. Hence, we increased the number of layers while decreasing the number on neurons in each of those layers. All the layers that were constructed except the output layer used ReLU as its activation function to introduce a nonlinear mapping between the filtered images and the steering wheel angle. ReLU is used to avoid saturation of the perceptrons and neglects the negative activations. The ReLU function is given below.



**Fig. XXY:** ReLU activation function

When we observed the output of this structure, we have recognized that it was still improvable. So we tried to choose the optimal loss function. We have first selected the loss function as mean squared error, however, we noticed that the mean squared error was too sensitive to instant changes in the data. Hence, we have changed this loss function to “logcosh”, which is basically the same with less reaction to instant changes. The graphs of different loss functions are given below.



**Fig. ANNEN:** The difference between separate loss functions in regression tasks

Also, we added a dropout layer in order to avoid overfitting to the training data one for the input layer to avoid memorizing the input images and one after the second dense layer to have a more general feature space. Since this is a regression task, the output layer is selected as a linear activation to have a continuous representation of the predicted angle.

One other important layer that we have found to be useful is batch normalization, since this layer allows each layer to learn by itself better independent of other layers [https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c] . Batch normalization is placed before the output layer and this increased the R2 score of the model by 6 percent. The final configuration of the network established is given below.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # ================================================================= flatten\_11(Flatten) (None, 38400) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dropout\_5(Dropout) (None, 38400) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_45(Dense) (None, 512) 19661312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ activation\_35(Activation) (None, 512) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_46(Dense) (None, 300) 153900 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ activation\_36(Activation) (None, 300) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dropout\_6(Dropout) (None, 300) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_47(Dense) (None, 128) 38528 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ activation\_37(Activation) (None, 128) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_48(Dense) (None, 64) 8256 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ activation\_38(Activation) (None, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ batch\_normalization\_10(Batc) (None, 64) 256 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_49 (Dense) (None, 1) 65

=================================================================

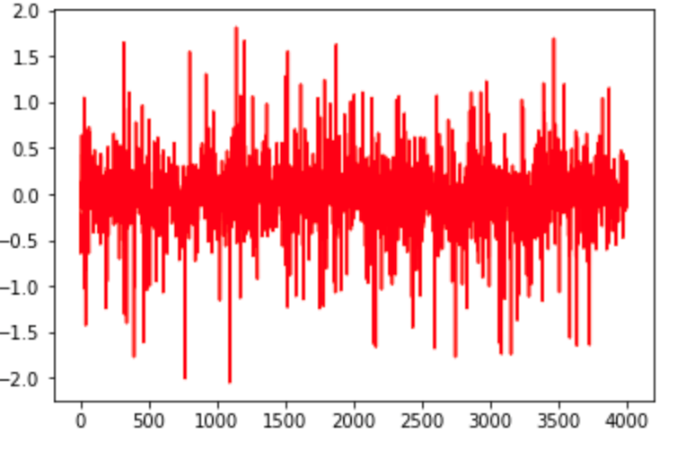
Total params: 19,862,317

Trainable params: 19,862,189

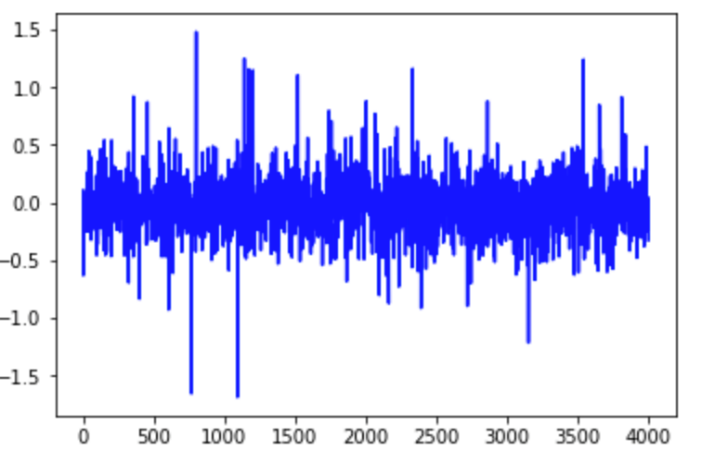
Non-trainable params: 128 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Then, by trial and error, we have configured the batch size and number of epochs. The best working batch size turned out to be 32 and the number of epochs was 100. To avoid overfitting caused by the high number of epochs, a dropout layer is added to the network at two separate locations.

Hence, we split the data the same as discussed in the CNN section and tested on the 4000 test images. The graph showing the results are below for the actual and predicted labels.

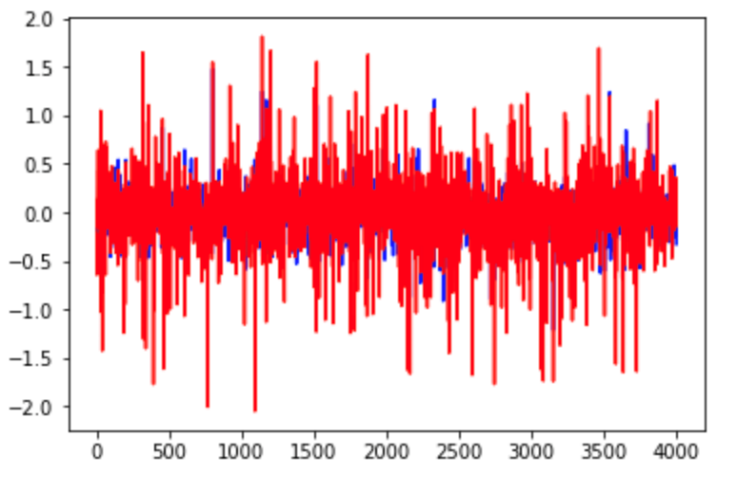


**Fig. ANNEN2:** The Original Test Labels



**Fig. ANNEN3:** The Predicted Test Labels

Here, we observe that although the values have differed from each other, the prediction trend seems to be accurate. In order to observe better, we put these graphs on top of each other and see the correlation in between.



**Fig. ANNEN4:** The predicted and actual labels on top of each other

In order to observe how accurate out model is, we look at the R2 score and the RMSE. The R2 score turns out to be 0.403 and the RMSE becomes 0.33. Here, we see that the model only we constructed only covers 40% of the data variance, which is lower than expected. We see that the overall error is 0.4 radians which corresponds to 23.0902°. This model is highly inaccurate, and we can conclude that the multilayered perceptron is not a good model to solve this problem.