**Convolutional Neural Network (CNN)**

One of the methods that we have picked to regress steering wheel angle was the Convolutional Neural Network, a supervised deep learning technique. CNNs are mostly used in image recognition, classification, analysis tasks. A CNN is composed of three main components. An input layer, some hidden layers and an output layer. The purpose of a CNN is that, while a standard neural network is that in CNN, the network forces the filters to learn in order to extract more useful features, which is normally pre-processing that must be done by hand.

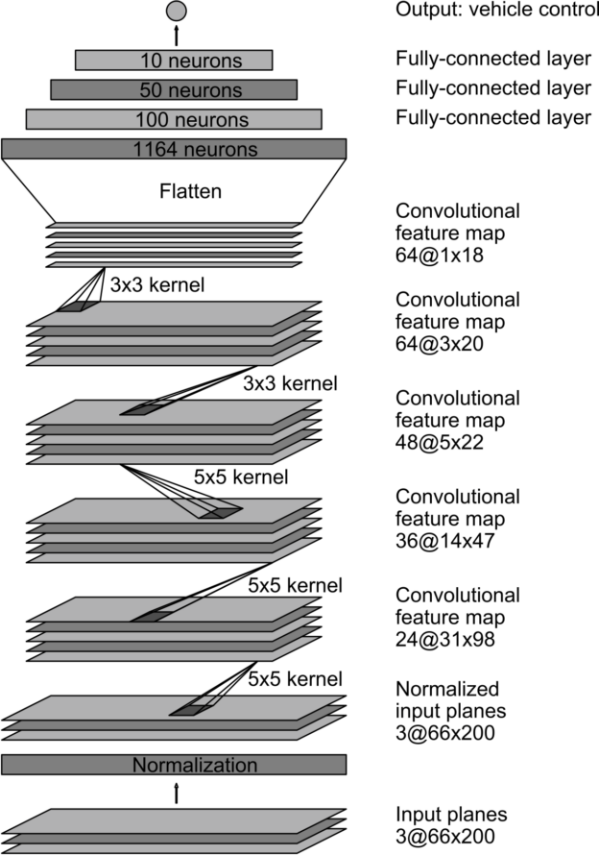
In common convention, the input layer is connected to a convolutional layer, which convolves the image with the specified kernel to filter the image in order to extract some features. Then, if needed, either an average or max pooling layer can be inserted to reduce the data complexity that goes into the next layer. However, pooling reduces the information contained in the image, and because of that, we avoided using any type of pooling. After some convolution layers are stacked after each other, features that are highly explanatory of the data is expected to come out. After the end of the convolution layers, there is usually a fully connected network. That part is a Multilayered Perceptron and discussed in the section with the same name. This activation functions of these fully connected layers are chosen as ReLU, however, at the end of the network, since the task at hand is classification, we inclined to choose linear activation instead of a SoftMax or a ReLU activation.

The conv layers of a CNN are the most important parts of the architecture, and while designing, one must know how the 2D convolutional layer work, on the dimension basis. The backbone idea is that the kernel whose width and length is predefined, moves its window over the image and convolves itself with it and gives the output to the next layer. One huge difference with a fully connected network is that the convolution output is only connected to the relevant area on the next layer, not all the upcoming layers. One should remember that; the depth of the kernel should match the depth of the input data.

In most CNN architectures, unlike the regular convolution, strided convolutions are used. Strided convolution is the same when considering the mathematics aspect with regular two-dimensional convolution, however, the window moves not one-by-one but with the hyperparameter specified. Hence, we can give the output width, height and depth as given below.

Here, W represents the width of the data from the previous layer, H represents the height of the previous layer, S represents the striding done in that dimension, P represents the pooling if there is any, and K represents the kernel dimensions in width or height. Depth of the data is always correlated with the number of filters applied in the layer.

The hard part of designing a CNN is that there are too many hyperparameters to choose from. The number of layers, activation functions of each layer, whether to include pooling, whether to do striding or regular convolution, the kernel sizes and number of filters at each layer should be decided separately. Since, we had no previous expertise on the subject, we have researched previously defined and verified neural network architectures and found the NVIDIA CNN Architecture for Road Recognition, whose structure is given below.



This structure consists of normalizing the input images and feeding them to the network. The convolution section consists of three 5x5 convolutions with a striding of two and the other three conv layers are 3x3 convolutions with no striding. Then, flattening is applied on the data with the height dimension become one. The flattened nodes are then connected to three fully connected layers with ReLU activations with each layer containing 100, 50 and 10 neurons respectively. We couldn’t directly use this architecture since the dimensions of the initial image that we have put into the network is not correlated with this architecture’s input which is 60x200x3.

Hence, we have started to manipulate the dimensions and the layer parameters suiting to our data dimensions. In the preprocessing for this section, we have cropped all image data that we own which consists of 20632 images that are 480x640x3 into 120x320x3. We have cropped the upper XXXXXX pixels of each image which corresponded to mostly the environment and not the road itself. Then, we have used the OpenCV resize function to resize the image without losing much information on the data while decreasing the dimensions, hence decreasing the training time of the network. However, as opposed to the NVIDIA architecture, we haven’t normalized the images because when we tried to normalize them, the images that came out were nearly all black and the features were lost.

After that, we started to configure the network that we will be training on. In all the layers except the last one, we have chosen the activation function of each layer as ReLU activations. This conclusion is reached by searching through the internet. Hence, we first added a 5x5 convolutional layer that contained 12 filters with striding 2. After that the output of the layer turned out to be 58x158x12. Then we have added three of the same filters, one with the number of kernels 24, 36 and 48. At the end of these layers, the activation map dimension has become 5x18x48. After that, we have added a dropout layer whose dropout rate is 0.25 to avoid overfitting to the training data. Then we have added two 3x3 convolutional layers with no striding in order to drop the dimension to 1x14x64. After the completely dropped one dimension to one, we flattened the neurons and added a fully connected layer structure with five layers that each have 600,400,100, and 10 neurons. After that, since we are doing regression, we have connected the last layer to a linear activation layer. The summary of the network we have created is given below.

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

Layer (type) Output Shape Param #

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

conv2d\_5 (Conv2D) (None, 58, 158, 12) 912

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

conv2d\_6 (Conv2D) (None, 27, 77, 24) 7224

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

conv2d\_7 (Conv2D) (None, 12, 37, 36) 21636

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

conv2d\_8 (Conv2D) (None, 5, 18, 48) 15600

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

dropout\_1 (Dropout) (None, 5, 18, 48) 0

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

conv2d\_9 (Conv2D) (None, 3, 16, 64) 27712

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

conv2d\_10 (Conv2D) (None, 1, 14, 64) 36928

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

flatten\_1 (Flatten) (None, 896) 0

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

dense\_1 (Dense) (None, 600) 538200

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

dense\_2 (Dense) (None, 400) 240400

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

dense\_3 (Dense) (None, 100) 40100

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

dense\_4 (Dense) (None, 10) 1010

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

dense\_5 (Dense) (None, 1) 11

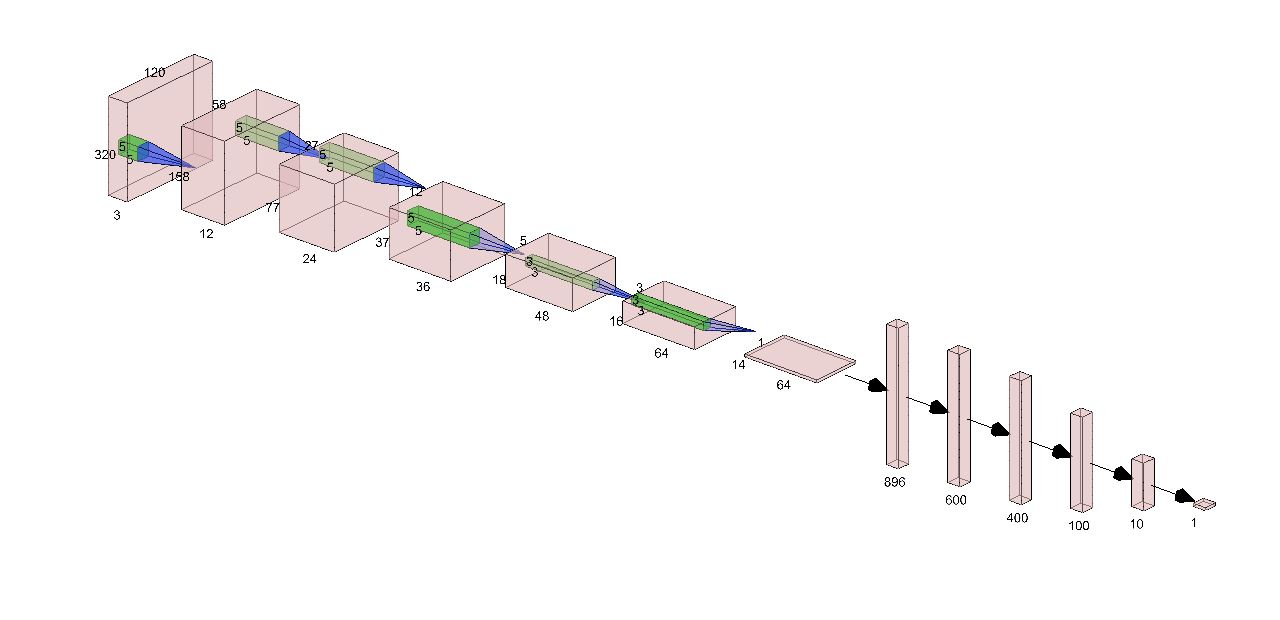
=================================================================

Total params: 929,733

Trainable params: 929,733

Non-trainable params: 0

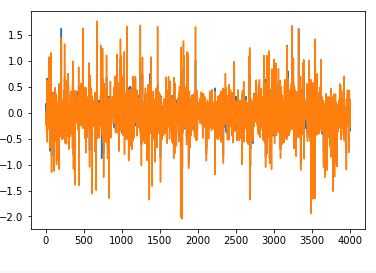
The constructed model is given below as visualized. While constructing the model, the TensorFlow’s Keras library is used.



**Fig X:** The Constructed CNN Model

One important note is that, this configuration is constructed by trial and error most of the time, where the layer properties are changed in order to get the best accuracy on the validation data. Hence, we have split the data into two as training and test data where training had 16632 images and the test set had 4000 photos, all picked randomly from the data set. Then we have trained on the data with a batch size of 32 and with 32 epochs, cross validating over the training with a validation percentage of 0.2 over the training set. We have picked the loss function for the network as a mean squared error loss. Then we have put the test data that we have reserved into the trained network and observed the outcome.

The test labels and the predicted labels of the CNN are below.



**Fig X+1:** The test labels and the predicted labels plot on top of each other

These results show that the predicted results and the real label data are very close to each other. However, we need to observe some metrics in order to decide if the algorithm that we have implemented is valid. Hence, we check the R2 score and the RMSE (Root Mean Square Error).

R2 score is a metric that explains how much of the data variance is explained by the model created and calculated as below. It is used instead of accuracy in regression type problems.

Where, total variance is the variance of the original dataset from its mean, unexplained variance is the variation of the original dataset from the model that we have constructed. In the code, we use the Pearson correlation coefficient and take its’ square to find the R2 score. Furthermore, in the original challenge from UDACITY, to observe efficiency RMSE is used, as given below.

When we calculated these statistics, the R2 score was 0.792 and the RMSE was 0.14. These scores show that we were able to explain 80% of the variance in the data. Overall, we were wrong 0.14 radians from the true labels, which is 8.02°. The results was not as accurate as the algorithms from the original UDACITY challenge, however, they are acceptable in terms of learning the road curvature.