

Behavioral Cloning: Make a car drive itself...



Prathmesh Dali

Apr 14, 2018 · 4 min read



Driving Car on Simulator Provided By Udacity

Yeah, as usual, I started with a Complex CNN architecture of dozens of convolutional layers and half a dozens of dense layers as Keras (Kudos to Keras team for making tensorflow so easy) made it easier to create neural networks. I was kidding :-)) I just started with similar to NVIDIA architecture which had five convolutional layers and three fully connected layers and lots of training data by augmenting every image taking a left and right camera images, etc. By the way, I have used dataset provided by Udacity to train my model.

Yeah, it is looking like something big I have achieved right? But do we need complex CNN like NVIDIA CNN architecture which is used for real-life scenarios, hours of

training time and tons of data to run our car on simulator provided by Udacity for this project?

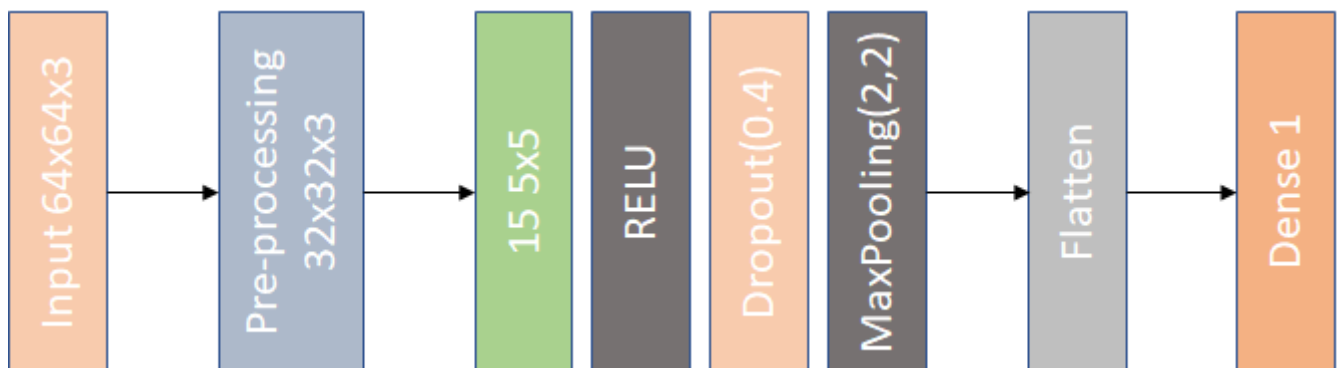
Let's discuss....

What do we need to identify the drivable area and predict steering angle so that our vehicle will run smoothly on the simulator without any fail?

For a car to identify steering angles, we need features like road edges. In CNN such high-level features are extracted at initial layers.

Keeping this in mind I started reducing layers from my NVIDIA CNN model. I tried with three convolution layers and three dense layers. Then I tried with two convolution layers, and two dense layers and my final try was one convolution layer and one dense layer. With the last configuration, I was able to achieve similar performance as my previous trials with a shorter training time of around five mins.

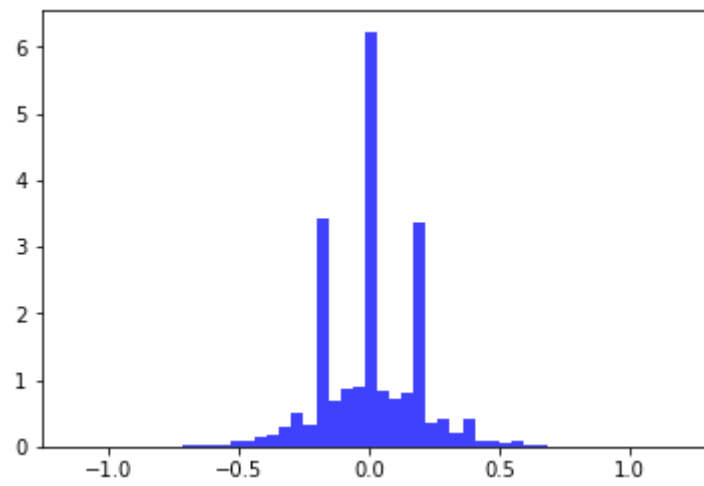
My final model for this project consists of single convolution layer followed by dropout layer, a max pooling layer and single dense layer as shown below



Model used for training CNN for project 3

- Generating Dataset:-

For generating a dataset for this model I have taken all the center camera images and their flipped counterpart by making steering angle -ve of that of the original steering angle of the center image. The flipping of the images is done to balance the dataset so that model will not be biased towards any specific condition. To add recovery data to my dataset, I have taken all right and left camera images by adding offset of 0.4 to steering angle for left camera image and -0.3 to that of the right camera image. The distribution of my generated dataset is as shown below



Steering angle distribution of generated dataset

We can see the dataset is almost balanced. Since most of our track is straight, I have kept more samples of 0 steering angles.

- Processing Images:-

Below image contains some of the samples from images captured by each of the camera










Camera	Image 1	Image 2
Center Camera Images		
Left Camera Images		
Right Camera Images		

Image samples from each camera

To train our model we don't need all the details present in above image such as natural sceneries, sky, surrounding, etc. First ground rule I followed is feed the network with those details which you want it to see. To remove redundant details I cropped image's 80px from top and 20px from bottom.

Sample of the cropped images are below

Cropped Center Camera Image	Cropped Left Camera Image	Cropped Right Camera Image
		

Cropped Images

Then I resized images to 32 X 32 since it will make model faster and we don't need low-level details in this scenario. After normalizing the resized images, those are passed as input to the CNN.

- Overfitting:-

While training model I found that model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

To combat the overfitting, I have added dropout layer and maxpooling layer followed by convolution layer and I reduced number of epoches to 2.

With this model I am able to drive vehicle on both track one and track two.

Parameters I used in my model

- No of epochs= 2
- Optimizer Used- Adam
- Learning Rate- Default 0.001
- Validation Data split- 0.2
- Generator batch size= 32
- Correction factor- 0.4 and -0.3

- Loss Function Used- MSE(Mean Squared Error as it is efficient for regression problem).
- Activation Function- RELU
- Dropout- 40% (after 1st activation layer to combact overfitting)
- Dataset used- Provided by Udacity (No additional images other than original dataset).

Track 1 Video in autonomous mode:-



Track 1

Track 2 Video on autonomous mode:-

Udacity Behavioral cloning track 2 autonomou...



Track 2

The model was able to drive on second track as well without retraining it for second track. Hence we can say that it was generalized enough and not overfitting.

Link to my github repository for this project.

[Machine Learning](#)

[Self Driving Cars](#)

[Convolution Neural Net](#)

[Udacity](#)

[Behavioral Cloning](#)

[About](#)

[Help](#)

[Legal](#)