Behavioral Cloning

Behavioral Cloning Project

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files: * model.py containing the script to create and train the model * drive.py for driving the car in autonomous mode * model.h5 containing a trained convolution neural network * writeup*report.md and writeup*report.pdf summarizing the results

2. Submssion includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing `sh python drive.py model.h5`

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 and 64 (model.py lines 92-96)

The model includes RELU layers to introduce non linearity (code line 92,93,95 and 96), and the data is normalized in the model using a Keras lambda layer (code line 91).

2. Attempts to reduce over fitting in the model

The model contains dropout layers in order to reduce over fitting (model.py lines 94 and 97).

The model was trained and validated on different data sets to ensure that the model was not over fitting (code line 78-79). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 107).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road ...

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to generate lots of training data using multiple runs of the supplied simulator. New beta simulator did not work so had to work with old simulator. Initial runs were around 2 laps with which the model did not work properly. So eventually after struggling few days to generate a good solution eventually generated around 25k line of data points.

My next step was to use a convolution neural network model similar to the Nvidia pipeline. This pipeline was appropriate because it attacks the same problem and hence a good starting point. Eventually modified it a little by removing some convolution layers and adding some dropout layers. But before that to minimize the memory usage, cropped the image 60 pixels from top and 20 pixels from bottom to remove unnecessary areas of image. This helped model predict the image more correctly. Also, applied max min normalization to the image to avoid local optima scenarios.

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track to improve the driving behavior in these cases, I generated more data by starting the recording where the car was at the edge then move to center.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines between 89 and 103). consisted of a convolution neural network with the following layers and layer sizes

Cropping

Normalization

Conv2D 24

Conv2D 36

Dropout 0.2

Conv2D 48

Conv2D 64

Dropout 0.2

Flatten

Full Connected layer with

Dense 1164

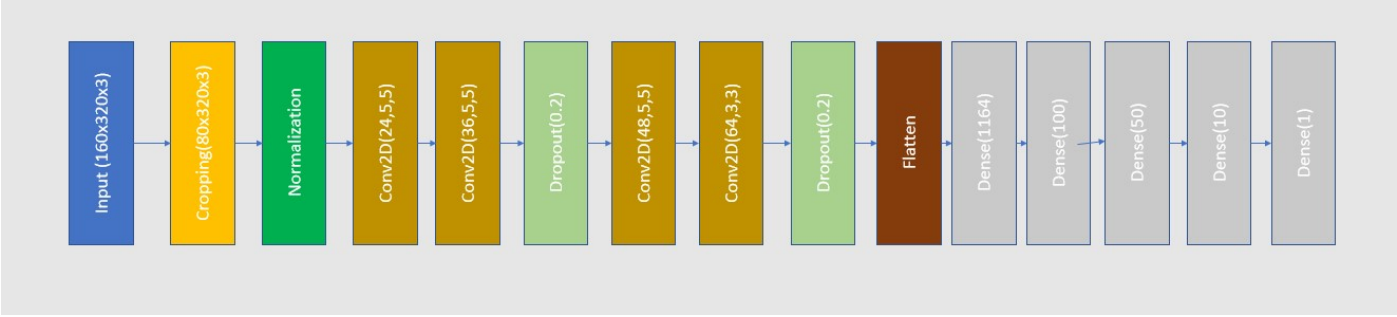
Dense 100

Dense 50

Dense 10

Dense 1

Here is a visualization of the architecture



3. Creation of the Training Set & Training Process

My first step was to load all the three viewpoints plus the steering angle for each data point i.e. center, right and left. Right and left's steering angle were adjusted by subtracting and adding 0.25 angle the center's steering angle.

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to correct if it is in situation were it takes to the edge These images show what a recovery looks like starting from right :

Then I repeated this process on track two in order to get more data points.

To augment the data sat, I also flipped images and angles thinking that this would provide me possibility of generating roughly twice the data points For example, here is an image that has then been flipped:

After the collection process, I had 70-80k number of data points.

I finally randomly shuffled the data set and put 0.2% of the data into a validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 3 as evidenced by showed it was not improving after 2 epochs I used an adam optimizer so that manually training the learning rate wasn't necessary.