Behavioral Cloning Project

Files Submitted & Code Quality

1. My 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.json containing the model in json file
- model.h5 containing weights of my trained network
- writeup_report.pdf (this file) summarizing the results

2. Submission includes functional code

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

```
python drive.py model.json
```

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

Below is my final model architecture:

Layer (type)	Output Shape	Param #	Connected to
Norm1 (Lambda)	(None, 66, 200, 3)	0	lambda_input_1[0][0]
Conv1 (Convolution2D)	(None, 31, 98, 24)	1824	Norm1[0][0]
Conv2 (Convolution2D)	(None, 14, 47, 36)	21636	Conv1[0][0]
Conv3 (Convolution2D)	(None, 5, 22, 48)	43248	Conv2[0][0]
Conv4 (Convolution2D)	(None, 3, 20, 64)	27712	Conv3[0][0]
Conv5 (Convolution2D)	(None, 1, 18, 64)	36928	Conv4[0][0]
flatten_1 (Flatten)	(None, 1152)	0	Conv5[0][0]
FC1 (Dense)	(None, 1164)	1342092	flatten_1[0][0]
FC2 (Dense)	(None, 100)	116500	FC1[0][0]
FC3 (Dense)	(None, 50)	5050	FC2[0][0]
FC4 (Dense)	(None, 10)	510	FC3[0][0]
output (Dense)	(None, 1)	11	FC4[0][0]

Total params: 1,595,511
Trainable params: 1,595,511
Non-trainable params: 0

I have implemented the above model architecture.

The first layer is Lamda layer to normalize intensities, followed by 3 Convolution layers of 24, 36, 48 filters of 5X5 kernel. And then 2 Convolutions of 64 filters of 3X3 kernel.

These convolutions are followed by 4 fully connected layers and then the output layer.

Each of the convolutions and the fully connected layers had RELU activation function.

1. Solution Design Approach

The overall strategy for deriving the model architecture involved experimenting with transfer learning concepts discussed in the class and several other end to end approaches discussed in our forums.

My first step was to use a convolution neural network model similar to the one that was discussed here , I thought this model might be appropriate because it has subsampling

and few other augmentations which will be generalized well. But unfortunately it did not work out for me, could be because I don't have a GPU machine.

Finally, this helped me allot to come up with this working model architecture.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set.

I found that my first few solutions are not much suitable for a CPU and I was using my own training data.

I spent dozens of hours trying to create test data myself on a keyboard based laptop and realized (very late) that the angles are not smooth with keyboard based data. None of the solutions were working on my data after spending weeks of time.

Finally, I decided to implement my solution just by using the Udacity provided data. And yes, my current model is trained only on the Udacity provided data.

With the current model I just started with 1 epoch with 28K samples and a small batch size of 32, just to check things are ok. To my surprise the car was able to drive half way through, but it started with low loss of 0.01. I went ahead and increased to 2 epochs and that's it, the car is now able to drive smoothly all over the track.

The key, I think, is the correct cropping, resizing and smoothing the angle. I had experimented with several combinations on those numbers. And the YUV transformation is also key.

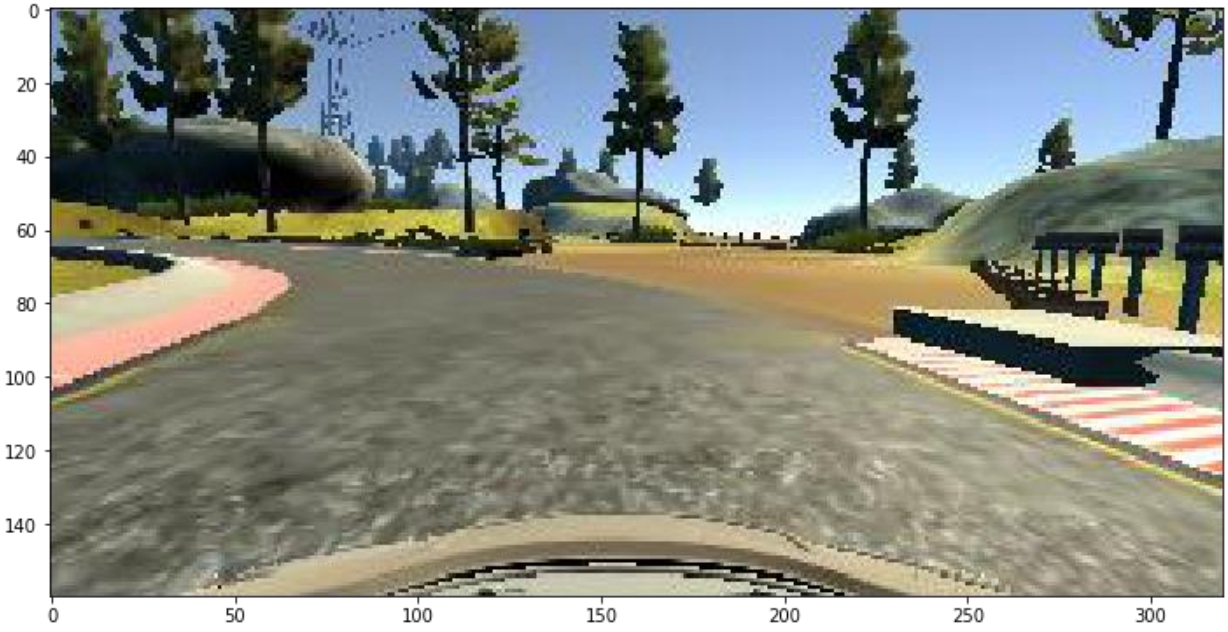
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

3. Creation of the Training Set & Training Process

Like I said, I have created lots of training data along with recovery. But none of my models were working. This all turned out to be waste as I was training with keyboard and the angles were very sharp.

I then decided to stick with Udacity provided data. I have tried several other augmentations but did not use them with this model.

Here is a sample of a center image:



And here is YUV transformation:

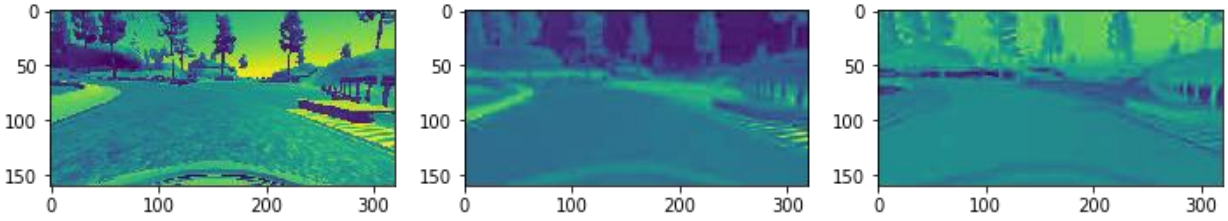


Image after crop to 80X240

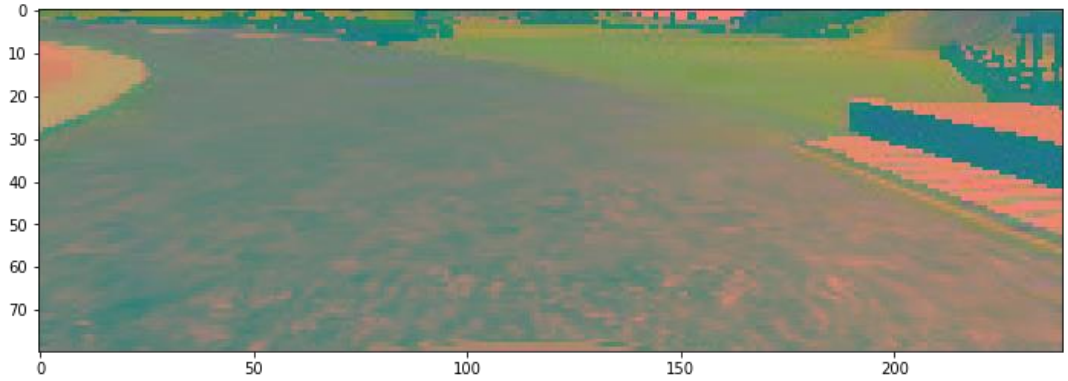
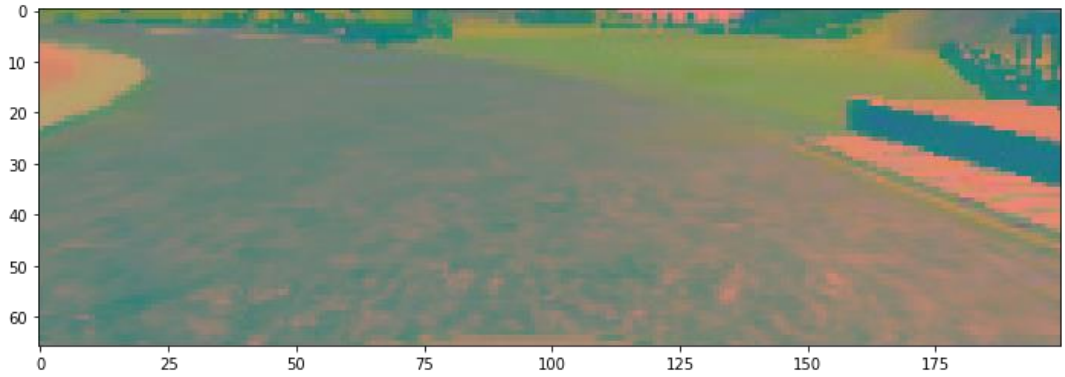


Image after resize to 66X200, this is my input:



With the Udacity provide data, I had 24K images. I have used all the three images (center, left and right) for training. I preprocessed this data by applying YUV transformation, followed by cropping and resizing to (66,200,3) as my input to the model.

I read the csv log into panda dataframe.

I finally randomly shuffled the data set and put 20% 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 with other models that I tried. But with this model the loss and val loss were consistently very low even from the first epch. The ideal number of epochs was 2 as evidenced by testing the autonomous mode. I used Adam optimizer so that manually training the learning rate wasn't necessary.

Note: With this model, adding Dropout/pooling layers did not help. Both the loss and val loss are very low anyway!