Project 3 - Behavioural Cloning

February 17, 2017

Files in the Project

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.pdf summarizing the results
- run1.mp4 to show that the car made it around the track

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.h5
```

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.

Key Points of Model Architecture

1. **An Appropriate Model Architecture has Been Employed**

   My model consists of a convolution neural network with 3x3 and 5x5 filter sizes and depths between 24 and 64 (model.py lines 75-94)

The model includes ReLU layers to introduce nonlinearity (code lines 79-81, 84-85), and the data is normalized in the model using a Keras lambda layer (code line 76).

2. **Attempts to Reduce Overfitting in the Model**

   The model contains dropout layers in order to reduce overfitting (model.py lines 82, 86, 89, 91).

   The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 59, 90). 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 97).

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, different camera angles and I also augmented the training data.

   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 create a model that wouldn't be too complicated but get the job done. I have come to a consensus after reading a lot of articles on convolutional neural networks that a fairly simple model can perform almost as well as a very "deep" model depending on the amount of data. I thought that architectures like Google's Inception v3 would be over-kill for the task at hand. I believe an architecture that complex and robust would be better suited for millions of data points.

   My first step was to use a convolution neural network model similar to the one that Nvidia used in the *End-to-End Learning for Self-Driving*

Cars paper. I thought this model might be appropriate because it seemed to be along the same lines as this project. Although, they mapped the pixels from just a center camera, I too used the center camera, along with the left and right side cameras to help the car drive around the track.

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 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.

From Nvidia's paper, I didn't see that they had any techniques to reduce overfitting, so to combat this I modified the model so that there are dropout layers with dropout probabilities of 0.2 (code lines 82, 86, 89), and 0.5 (code line 91).

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, the biggest problem was right after the bridge where the curb ends and there is a dirt road. The car would always go on to the dirt road and not continue to turn and follow the paved road. To improve the driving behaviour in this case, I would drive to the dirt road, and then record the car turning away from it.

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 75-94) consisted of a convolution neural network with the following layers and layer sizes

   - *Normalization*
   - *Conv. Layer* - Depth 24, Kernel = 5x5
   - *Conv. Layer* - Depth 36, Kernel = 5x5
   - *Conv. Layer* - Depth 48, Kernel = 5x5
   - *Dropout Layer* - Dropout prob = 0.2
   - *Conv. Layer* - Depth 64, Kernel = 3x3
   - *Conv. Layer* - Depth 64, Kernel = 3x3

- *Dropout Layer* - Dropout prob = 0.2
- *Flatten*
- *Dropout Layer* - Dropout prob = 0.2
- *Dense Layer* - 100 Neurons
- *Dropout Layer* - Dropout prob = 0.5
- *Dense Layer* - 50 Neurons
- *Dense Layer* - 10 Neurons
- *Dense Layer* - 1 Neurons

Here is a visualization of the architecture. See Figure 1

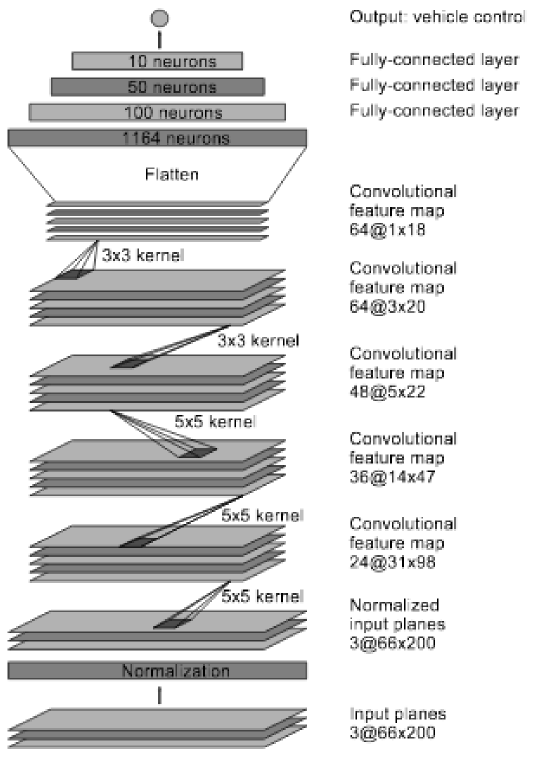


Figure 1: CNN architecture based on the Nvidia paper *End-to-End Learning for Self-Driving Cars*

3. **Creation of the Training Set & Training Process**

   To capture good driving behavior, I first recorded 8 laps on track one and 1 lap on track two using center lane driving. An example image of center lane driving is in Figure 2

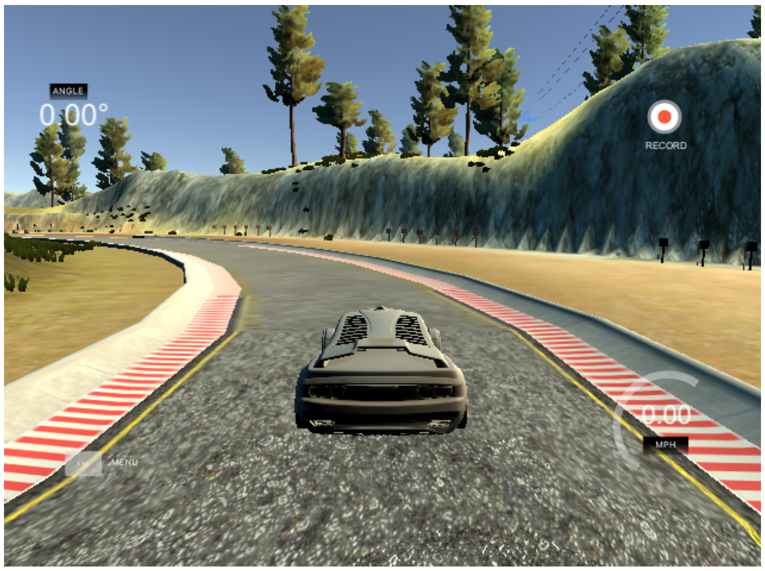
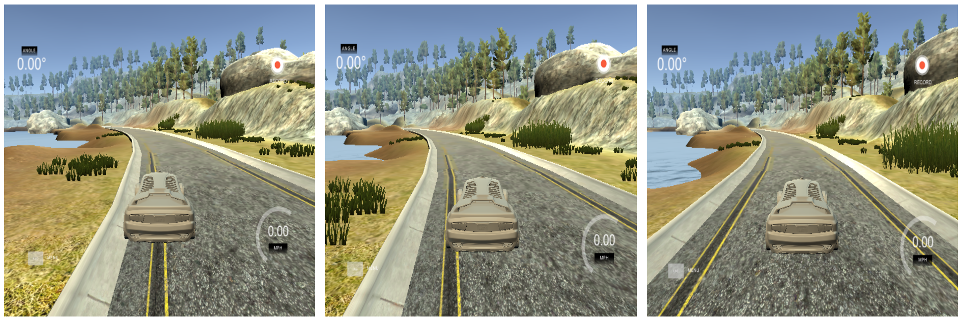


Figure 2: Center Lane Driving

   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 how to recover if it was getting too close to the edge of the road. I found that I needed to do this because at first I drove around the track just down the center, and then the car would near the curb and then it would continue until it went over the curb. The following images show what a recovery looks like starting from the left hand side of the road and ending back in the middle of the road



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

To augment the data set, I flipped the images along the y-axis along with flipping the steering angle so that now it would seem as though I had ”driven” both ways around the track. I added a steering correction of 0.25 to the left and right images so that the car would not get too close to the edge (ideally). I left the predicted ”center” image steering angle un-corrected. I also cropped the top and bottom of the images. I did this because the bottom 25px are just the hood of the car, and the top 70px were just sky and scenery. I felt that the top 70px and bottom 25px would contribute a lot of non-essential information (cropped image show in Figure 3

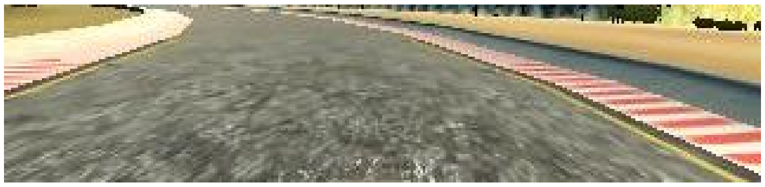


Figure 3: Sample of a cropped image from the dataset

After the collection process, I had 58,172 number of data points. I finally randomly shuffled the data set and put 20% of the data into a validation set, which was not cropped or normalized because I wanted the validation set to represent the test data as much as possible.

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 15 as evidenced by the fact that after 15 epochs, the validation accuracy kept bouncing around the same number. I used an adam optimizer so that manually training the learning rate wasn't necessary.