Behavioral Cloning - Project Report

The goals / steps of this project are the following:

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

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
   - report.pdf summarizing the results
   - video.mp4 containing a video recording of the vehicle driving autonomously using model.h5 around track 1 for 1 lap.

2. Submission includes functional code

   Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around track 1 by executing

   ```>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

4. 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 74-89)

The model includes RELU layers to introduce nonlinearity (code line 79-83), and the data is normalized in the model using a Keras lambda layer (code line 76).
The model also includes a dropout layer and fully connected layers at the end of the network.

5. Attempts to reduce overfitting in the model

The model contains a dropout layer with a dropout rate of 0.5 in order to reduce overfitting (model.py lines 86).
Some data from track 2 was collected and used along with data from track 1 to avoid the model from overfitting track 1.
The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 94). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

6. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 92).

7. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used data from the following situations:

- Center lane driving from track 1
- More center lane driving for the bridge part of track 1, since the bridge scene doesn't come up often
- Right lane driving from track 2, only a part of it since I wasn't able to drive within the lane manually on some hard curves
- Recovering from the left and right sides of the road of track 1
- Center lane driving backwards on track1

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

Model Architecture and Training Strategy

8. Solution Design Approach

The overall strategy for deriving a model architecture was to start off with a tuned neural net architecture and adjust from that.
My first step was to use a simple fully connected layer to see how hard the problem was. The car drove perfectly on track 1 without going off to the side, which surprised me. Then I found that the model had a low mean squared error on the training set but a high mean squared error on the validation set, which implies that it is overfitting track 1.

To combat the overfitting, I added training data from track 2. This made the performance of the model driving on track 1 very awful; it was going off the road almost immediately after start. It was time to find an established neural net.

So I started with a convolution neural network model similar to the NVIDIA End to End Learning neural network architecture. I thought this model might be appropriate because the model was originally designed to get image inputs from the cameras (front, right, left) of a car and output steering wheel angle, which is very similar to our problem setting.

Since the first simple network was already overfitting, using the NVIDIA architecture would make the problem worse, since it has far more parameters, and the same amount of input data. So I added a dropout layer to reduce the overfitting effect.
In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. The car was driving better at the start of track 1. However, it still went off the road at the curves and bridge.

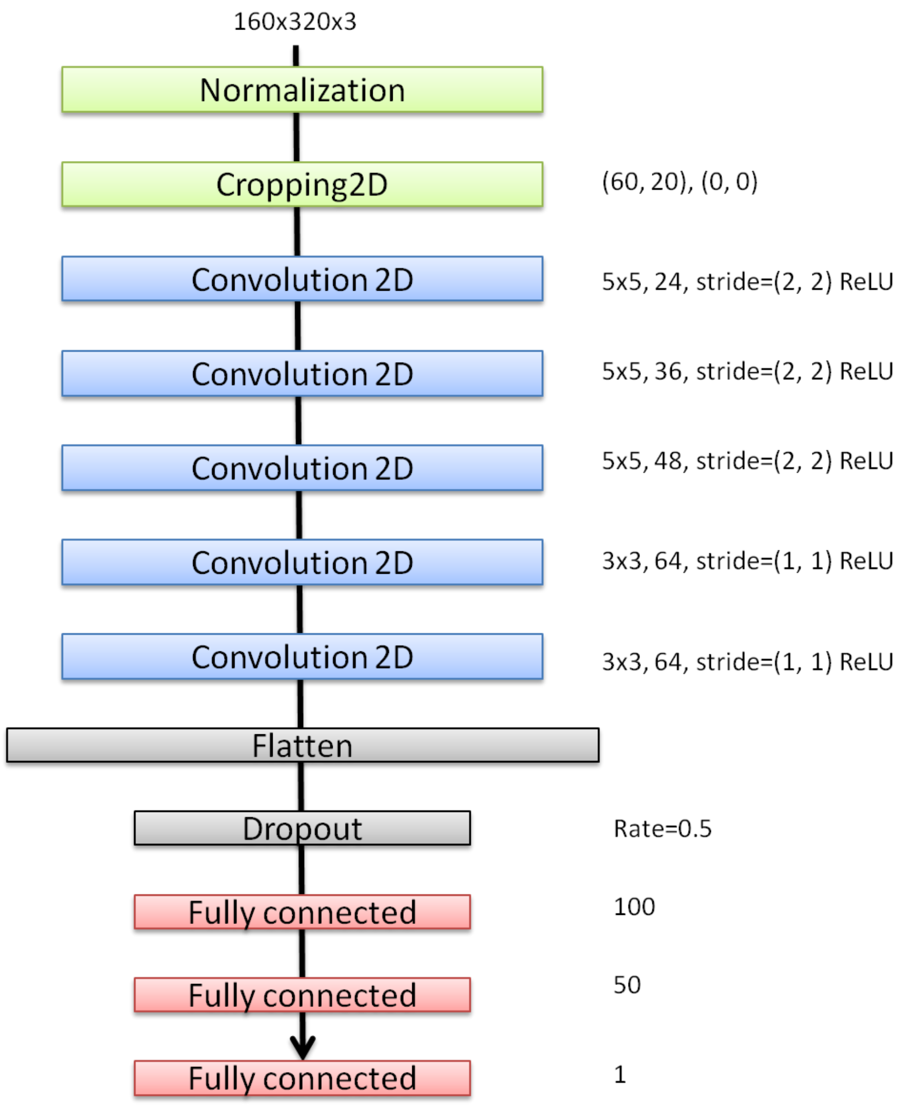
So I decided to add more training data. I added data for driving track 1 backwards. I think maybe it was because there is no left and right lane on the bridge, so the model was not performing well on the bridge. Also the bridge is a special scene in track 1 and it does not appear often. So I decided to strengthen this place and added more training data specifically for this scene. Finally, I flipped all of the input images I can double the amount of training data. Now the car was down to two ~ three places going off the road.

I guess the model might not know how to recovery back to the center of the road once it started going to the side of the road. To improve the driving behavior in these cases, I added some recovery data from the left and right of the road, especially at some hard places, like the bridge and after the bridge.

At the end of the process, the vehicle is able to drive autonomously around track 1 without leaving the road.

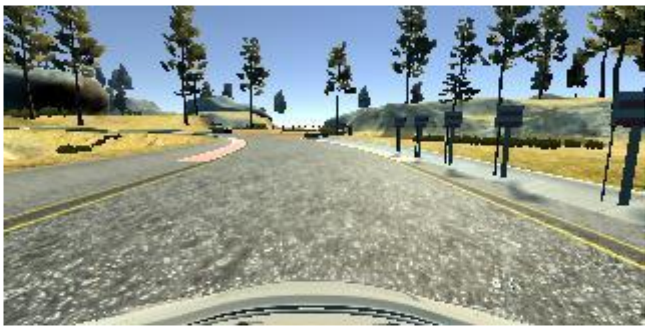
9. Final Model Architecture

The final model architecture (model.py lines 74-89) consisted of a convolution neural network with the following layers and layer sizes.

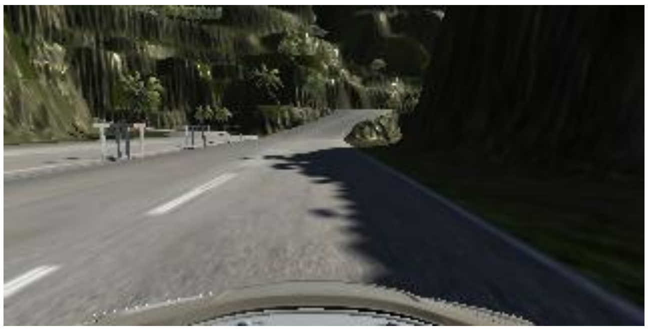


10. Creation of the Training Set & Training Process

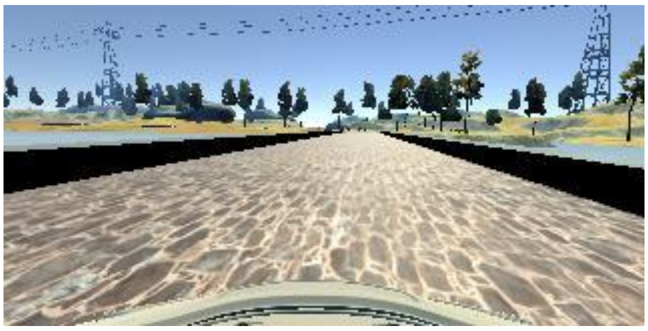
To capture good driving behavior, I first recorded 1.5 laps on track 1 using center lane driving. Here is an example image of center lane driving:



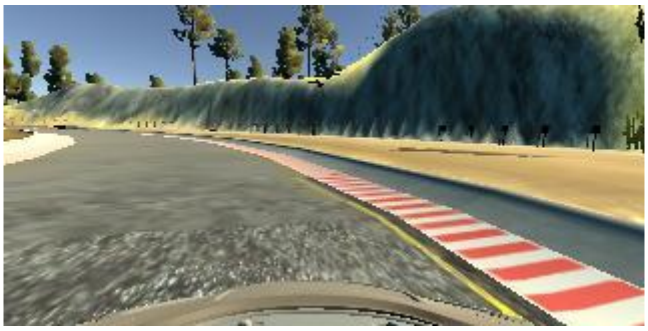
I then recorded some data from track 2, here is an example:

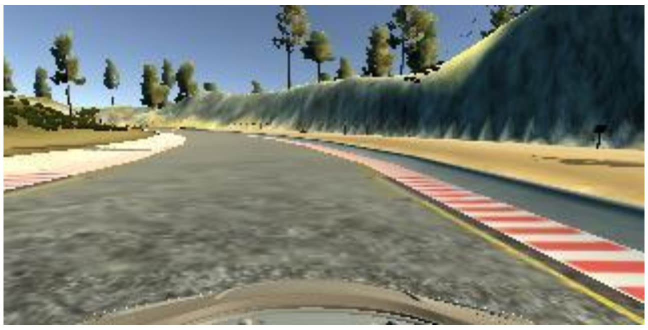


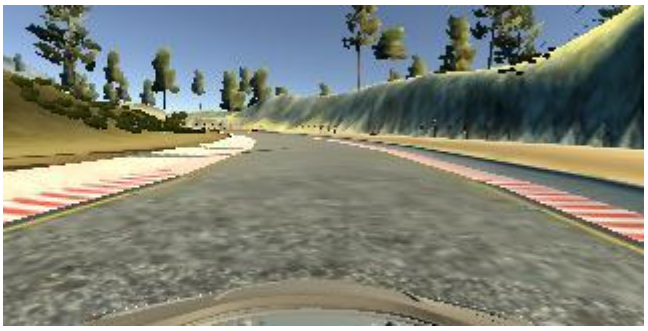
I tried to strengthen my training data with more data from the bridge, here is an example:



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 recover back to the center of the road in case it wandered off. These images show what a recovery looks like starting from the right of the road:







To augment the data sat, I also flipped images thinking that this would not only double the amount of data, but also teach the model to turn in the other direction.

After the collection process, I had 35278 number of data points.
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. I set the number of epoch to 10. It resulted from a balance of time and accuracy obtained. I used an Adam optimizer so that manually training the learning rate wasn't necessary.