Behavioral Cloning

Behavioral Cloning Project Outline

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 reportCamera Calibration

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. In the file of 'model.py', it can read images data from the data folder and augment the data, preprocess them, define the model architecture (nvidia mode) and adapt the model to the training data and obtain training and verification accuracydrive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_report.md or writeup_report.pdf 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.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

I used the model of nvidia model what I learned in the lesson
Convolution Layer: Filter 24, size 5x5
Convolution Layer: Filter 36, size 5x5
Convolution Layer: Filter 48, size 5x5
Convolution Layer: Filter 64, size 5x5

Convolution Layer: Filter 64, size 5x5
Flatten Layer
Dense layer with output size of 100
Dense layer with output size of 50
Dense layer with output size of 10
Dense layer with output size of 1
(model.py lines 50-63)

I used ELU instead of RELU, because ELU makes the price near zero by making a small negative activation for the negative input and output and linear output for the positive input and output.

Here is a visualization of the architecture(Fig.1)

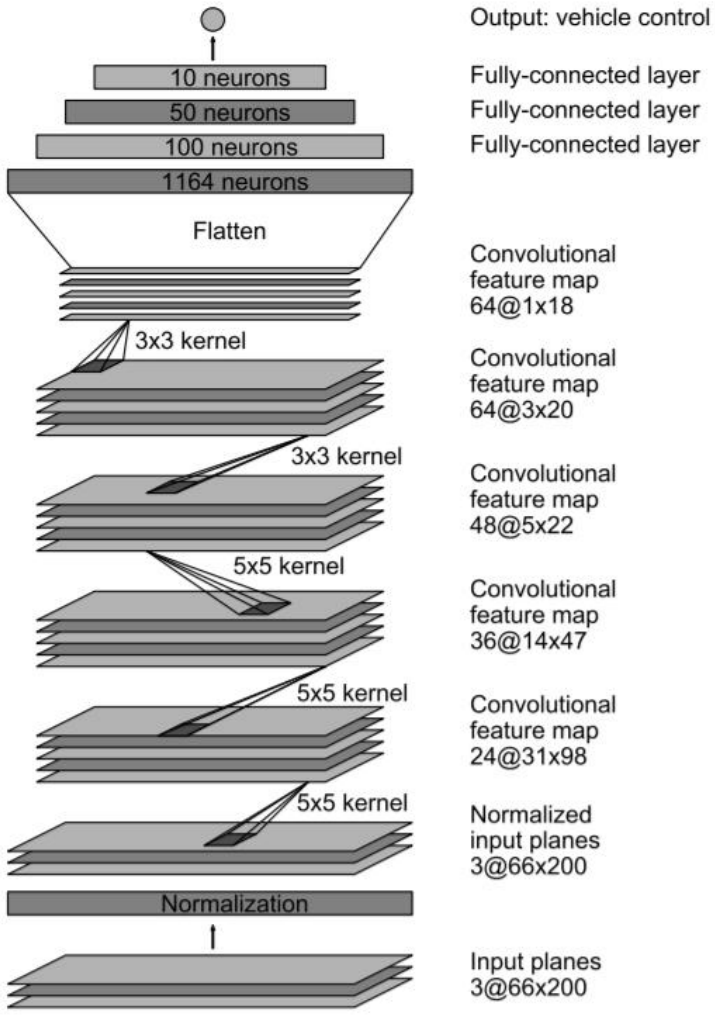


Fig.1

2. Attempts to reduce overfitting in the model

To reduce overfitting I used dropout between the Dense layers (model.py lines 58 and 61).

The model was trained and validated on different data sets to ensure that the model did not overfit. Test the model by running it on the simulator and ensuring that the vehicle can stay on track.

3. Model parameter tuning

I use the adam optimizer, so I cannot manually adjust the learning rate(model.py line 65).

4. Appropriate training data

Select training data to keep the vehicle on the road. I used a combination of center lane driving to recover from the left and right sides of the road and set the correction to 0.2. Left = center + correction, right = center correction. To increase the training data, I used center, left, and right enhancements.

5. Creation of the Training Set & Training Process

To get good driving behavior, I first recorded two laps on the first runway using centerline driving. This is a sample image of a neutral drive(Fig.2):

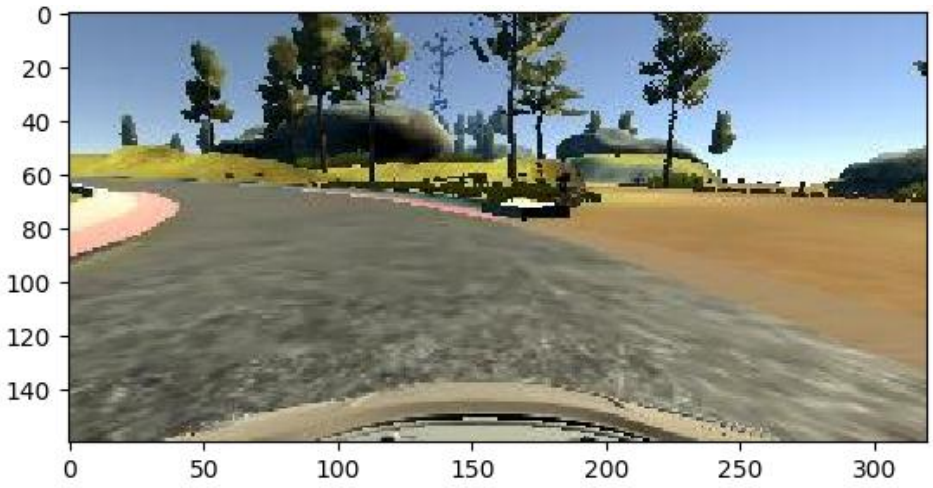


Fig.2(Image from Center Camera)

I then recorded the vehicles that recovered from the left and right sides of the road to the center so that the vehicles could learn how to turn left and right(Fig.3).

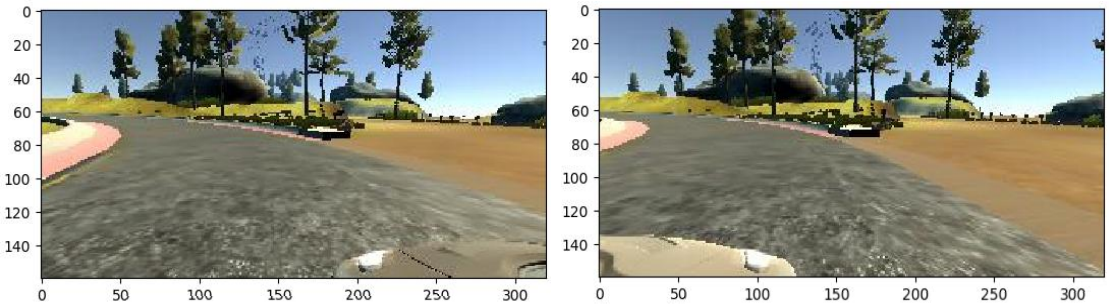
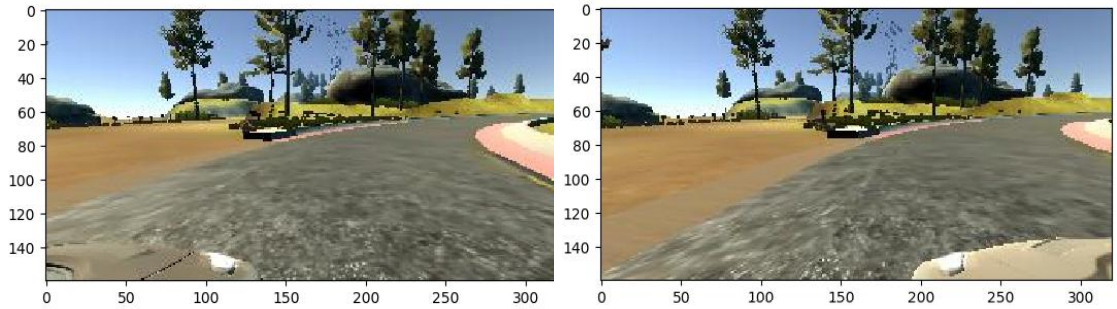


Image from Left Camera

Image from Right Camera

Fig.3

I then repeated this process on the second track to get more data points. To increase the sitting data, I also flipped the images and angles, thinking that this would increase my training data. For example, here is an image that has been flipped(Fig.4):



Flipped Left Image

Flipped Right Image

Fig.4

After the collection process, I had 38572 number of data points. I then preprocessed this data by Lambda and cropping.

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. The ideal number of epochs was 5 as evidenced by validation set. I used an adam optimizer so that manually training the learning rate wasn't necessary.