Behavioral Cloning

Behavioral Cloning Project

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_v5.py*** containing the script to create and train the model
- ***drive.py*** for driving the car in autonomous mode
- ***model_ng.h5*** containing a trained convolution neural network
- ***writeup_report.pdf*** summarizing the results

2. Submission includes functional code

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

```
python drive.py model_ng.h5
```

3. Submission code is usable and readable

The model_v5.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_v5.py is implemented without using a generator, as (at least in my implementation) the generator based version of my code (model_v6.py) is to slow.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

The model includes RELU layers to introduce nonlinearity (code lines 158,162,165,168 and 172), and the data is normalized in the model using a Keras lambda layer (code line 143).

My model is derived from the NVIDIA implementation in lines 157 – 191 of model_v5.py:

```
#layer 1- Convolution, no of filters- 24, filter size= 5x5, stride= 2x2
model.add(Conv2D(24,(5,5),strides=(2,2), activation='relu'))
#layer 2- Convolution, no of filters- 36, filter size= 5x5, stride= 2x2
model.add(Conv2D(36,(5,5),strides=(2,2), activation='relu'))

#layer 3- Convolution, no of filters- 48, filter size= 5x5, stride= 2x2
model.add(Conv2D(48,(5,5),strides=(2,2), activation='relu'))

#layer 4- Convolution, no of filters- 64, filter size= 3x3, stride= 1x1
model.add(Conv2D(64,(3,3), activation='relu'))
#layer 5- Convolution, no of filters- 64, filter size= 3x3, stride= 1x1
model.add(Conv2D(64,(3,3), activation='relu'))
#flatten image from 2D to side by side
model.add(Flatten())
#layer 6- fully connected layer
model.add(Dense(100))
#Adding a dropout layer to avoid overfitting, dropout rate 25%
model.add(Dropout(0.25))
#layer 7- fully connected layer
model.add(Dense(50))
#layer 8- fully connected layer
model.add(Dense(10))
#layer 9- fully connected layer, final layer – only one value (steering) = regression
model.add(Dense(1))
```

2. Attempts to reduce overfitting in the model

The model contains one (1) dropout layer in order to reduce overfitting (model_v5.py lines 182).

Line 213 (« the training and validation») is defining the train/validation-ratio using « validation_split=0.2 ».

```
model.fit(X_train, y_train, validation_split=0.2, shuffle=True, nb_epoch=5)
```

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_v5.py line 201).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used center, lef and right camera images combined with flipping the images.

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 start with a very simple model to see if the implementation of data read, augmentation, definition of the model as such, compile the model is working.

When this was working, I build up a first design approach according David Silvers Youtube Q&Q Video.
Code ist still included for reference in line 144-152 of model_v5.py

This inital implementation was a good start but it tended to overfitting.

To combat the overfitting, I modified the model implementing the NVIDIA pipeline and used dropouts of 25% after this first fully connected layer.

This delivered a perfect result.

A second implementation using a generator was dropped, when it turned out to be very slow. (for reference : model_v6.py)

The final step was to run the simulator to see how well the car was driving around track one.

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 (mode_v5l.py lines 157 – 191 of model_v5.py) consist of

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)

3. Creation of the Training Set & Training Process

As I had lot of problems driving the car within the simulator I used aftertalking to my Mentor the UDACITY provided data.

I finally randomly shuffled the data set and put 20% of the data into a validation set in the model.fit() function :

```
model.fit(X_train, y_train, validation_split=0.2, shuffle=True, nb_epoch=5)
```

Using 5 epochs turned out to be a good value, a further improvement was not visible. As the adam optimizer was used, so that manually training the learning rate wasn't necessary.

4. Execution Pipeline

- *python model_v5.py*
- *start the simulator in autonomous driving more*
- *python drive.py model_ng.h5 run_1*
- *stop the simulator after one complete lap*
- *stop drive.py (CTRL-C)*
- *python video.py –fps=48*

When drive.py drops error (ffmpeg and/or encoder), execute the followin on the console window :

- *conda install ffmpeg*
- *conda install x264==\'1!152.20180717\' ffmpeg=4.0.2 -c conda-forge*