Data Preparation:

Download data
I used data provided by Udacity: data.zip
data_parser.py
I define a DataParser class that:
* Access the CSV file
* Stores the suffix of filename (e.g. 2016_12_11_22_52_25_418.jpg) in an array
* Stores the recorded steering angles in an array
* Provides left, center, and right images in batches (i.e. a few images at a time) as array

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

The 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.md summarizing the results

2. Submission includes functional code

Using the Udacity provided simulator for mac executed

```
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.

Defines a generator that is called by model's learning function to prepare chunks of data for learning. It grabs a random chunk of left, center, and right images and the corresponding steering angles. For each grouping of steering angle, 1 left, center, and right images, the generator does the following:
* Picks left, center, or right images with equal probability (33%)
* If the left or right images are picked, the steering angle is modified accordingly

* Image/steering angles are sometimes ignored - i.e. the higher the absolute value of the steering angle, the more likely we are to use the image for training
* We flip an image and steering angle with 50% probability

```
(carnd-term1) ppujari (master *) CarND-BehavioralCloning $ python model.py
Using TensorFlow backend.
Running main in model.py
____________________________________________________________________________
_____
Layer (type)                     Output Shape          Param #
Connected to
============================================================================
=========================
lambda_1 (Lambda)                (None, 64, 64, 3)     0
lambda_input_1[0][0]
____________________________________________________________________________
_____
convolution2d_1 (Convolution2D)  (None, 64, 64, 3)     12
lambda_1[0][0]
____________________________________________________________________________
_____
convolution2d_2 (Convolution2D)  (None, 60, 60, 24)    1824
convolution2d_1[0][0]
____________________________________________________________________________
_____
elu_1 (ELU)                      (None, 60, 60, 24)    0
convolution2d_2[0][0]
____________________________________________________________________________
_____
maxpooling2d_1 (MaxPooling2D)    (None, 30, 30, 24)    0
elu_1[0][0]
____________________________________________________________________________
_____
dropout_1 (Dropout)              (None, 30, 30, 24)    0
maxpooling2d_1[0][0]
____________________________________________________________________________
_____
convolution2d_3 (Convolution2D)  (None, 26, 26, 36)    21636
dropout_1[0][0]
____________________________________________________________________________
_____
elu_2 (ELU)                      (None, 26, 26, 36)    0
convolution2d_3[0][0]
____________________________________________________________________________
_____
maxpooling2d_2 (MaxPooling2D)    (None, 13, 13, 36)    0
elu_2[0][0]
____________________________________________________________________________
_____
dropout_2 (Dropout)              (None, 13, 13, 36)    0
maxpooling2d_2[0][0]
```

```
____________________________________________________________________________________________________
____________________________
convolution2d_4 (Convolution2D)  (None, 11, 11, 48)    15600
dropout_2[0][0]
____________________________________________________________________________________________________
____________________________
elu_3 (ELU)                      (None, 11, 11, 48)    0
convolution2d_4[0][0]
____________________________________________________________________________________________________
____________________________
maxpooling2d_3 (MaxPooling2D)    (None, 5, 5, 48)      0
elu_3[0][0]
____________________________________________________________________________________________________
____________________________
dropout_3 (Dropout)              (None, 5, 5, 48)      0
maxpooling2d_3[0][0]
____________________________________________________________________________________________________
____________________________
convolution2d_5 (Convolution2D)  (None, 3, 3, 64)      27712
dropout_3[0][0]
____________________________________________________________________________________________________
____________________________
elu_4 (ELU)                      (None, 3, 3, 64)      0
convolution2d_5[0][0]
____________________________________________________________________________________________________
____________________________
maxpooling2d_4 (MaxPooling2D)    (None, 1, 1, 64)      0
elu_4[0][0]
____________________________________________________________________________________________________
____________________________
dropout_4 (Dropout)              (None, 1, 1, 64)      0
maxpooling2d_4[0][0]
____________________________________________________________________________________________________
____________________________
flatten_1 (Flatten)              (None, 64)            0
dropout_4[0][0]
____________________________________________________________________________________________________
____________________________
dense_1 (Dense)                  (None, 100)           6500
flatten_1[0][0]
____________________________________________________________________________________________________
____________________________
elu_5 (ELU)                      (None, 100)           0
dense_1[0][0]
____________________________________________________________________________________________________
____________________________
dense_2 (Dense)                  (None, 50)            5050
elu_5[0][0]
____________________________________________________________________________________________________
____________________________
elu_6 (ELU)                      (None, 50)            0
dense_2[0][0]
____________________________________________________________________________________________________
____________________________
```

```
dense_3 (Dense)                  (None, 10)            510
elu_6[0][0]
_____________________________________________________________________________
_____________________________
elu_7 (ELU)                      (None, 10)            0
dense_3[0][0]
_____________________________________________________________________________
_____________________________
dense_4 (Dense)                  (None, 1)             11
elu_7[0][0]
=============================================================================
==========================
Total params: 78,855
Trainable params: 78,855
Non-trainable params: 0
_____________________________________________________________________________
_____________________________
BehaviorCloner: train_model()...
Epoch 1/5
 1680/24108 [=>............................] - ETA: 191s - loss: 0.0733
Epoch 2/5
24108/24108 [==============================] - 180s - loss: 0.0205
Epoch 3/5
24108/24108 [==============================] - 180s - loss: 0.0177
Epoch 4/5
24108/24108 [==============================] - 180s - loss: 0.0194
```

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

The model architecture is based on NVIDIA architecture. In order to start with 106 x 320 image, the first convolution layer is implemented with a larger filter (11x11). The model has 5 convolution layers each followed by a RELU activation. The convolution layers are listed as follows:

- 11 x 11 filter, with 24 depth, 3 x 3 stride
- 5 x 5 filter, with 36 depth, 2 x 2 stride
- 5 x 5 filter, with 48 depth, 2 x 2 stride
- 3 x 3 filter, with 64 depth, 1 x 1 stride
- 3 x 3 filter, with 64 depth, 1 x 1 stride

The model has 3 fully connected layers with sizes 100, 50, 10, and 1 output layer of size 1.

This model was selected according to driving performance and training time. In addition to this model, networks with more convolution layers and networks with fewer convolution layers

experimented. The comma.ai model was also included in the study. Since the lowest mse loss function does not guarantee the best driving model, the results are compared on the simulator.

2. Attempts to reduce overfitting in the model

The model contains 6 dropout layers with 0.25 dropout rate in order to reduce overfitting. They are located after the convolution layers 2, 3, 4 and after flatten layer and after fully connected layers 1, 2.

The data split into training data set and validation data set with ratios 80% and 20%.

3. Model parameter tuning

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

4. Appropriate training data

In addition to the recorded data, artificial data is added by flipping images. Only the images having an absolute value of steering angle greater than rthr parameter are flipped and added to training data. For the final model, rthr is set to 0.0. Therefore, the images with zero steering angle value are added only once, and the others are added twice with one of them flipped. The training data then split into training data and validation data.

The final training data consists of four parts. Only the track 1 data is used to train the model, and it contains both clockwise and counter-clockwise driving.

The first part is the provided Udacity data. It contains 8036 images. It mainly provides center lane driving. 3675 of these images are flipped and added again. The number of samples is 11711.

The second part of the data is recorded from autonomous driving. During the study, when a model performed a successful drive it was recorded to use in later training. There four different recording created with this method. The first two recordings are successful autonomous drives of different models on track one clockwise direction. Only the Udacity data is used during the training of these models. They contain 2228 and 2057 images. The last two recordings are successful turns on the different curves of the first track in the counter-clockwise direction. They were recorded with different models. The data used in the training of these models was consist of the Udacity data and the previously recorded second part data. They have 449 and 455 images. All of the 5189 images in this part are flipped and added again, resulting in 10378 samples.

The third part of the data is recovery data and it was recorded when driving in training mode. The strategy was to drive in a curve with full speed without recording; wait until the end of the

road, and only record the last minute turn. There were 327 images in this part. After adding the flipped images it has 620 samples.

The fourth part of the data is extreme recovery data. The car was set in a position that it was about to get out of the road, and it was stopped. Then the steering angle was set to 25.0 or -25.0 degrees to get the car on the road again. The images were recorded with zero or very low speeds. There were 553 images in this part. All of them flipped and added again and the total number of samples in this part is 1106.

The total number of images in the final model is 14105 with flipped images there are 23815 samples.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to build a convolutional neural network to predict the steering angles from camera images to keep the car on the drivable road.

Normalization
I use a Keras lambda function to normalize the data between -1 to 1. Putting normalization in Keras allows the operation to be parallelized in GPUs and I don't have to normalize manually when running the model during testing in the simulator in autonomous mode

Color Transform
There is a 1x1,depth3 convolutional layer. It's job is color space transformation. We could use OpenCV to do a color space transform, but it's not clear what color space or spaces are most useful. Adding color transformation as a convolutional layer allows back-propagation to surmise the most useful color channels. Also, again since it's in Keras, it is more efficient.

Feature Extraction
There are 4 ConvNet layers. Each has:
* 2D Convolution
* ELU activation function
* Max Pooling
* Dropout

For the first two 2D Convolutions, we first do 5x5 to extract large features. Then the later two convolutions, we do 3x3 to extract groupings of features.

For the activation we use ELU instead of RELU, which was talked about in the lectures. With RELU some neurons can become inactive because the negative half of the RELU sends them to 0. ELU uses both positive and negative values and can train systems more consistently:

http://www.picalike.com/blog/2015/11/28/relu-was-yesterday-tomorrow-comes-elu/

I used max pooling to bring down the dimensionality of the training and yield less data.

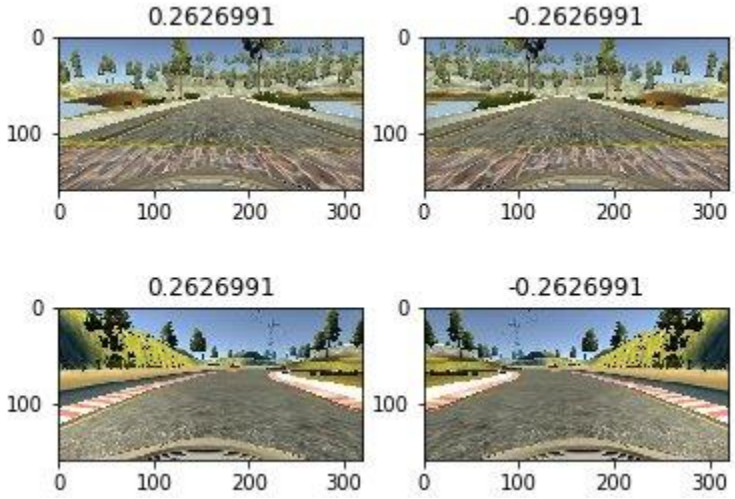
I used dropout to prevent overfitting to the specific images that the system is trained on.

Decision Making
First the data is flattened. Then 3 hidden layers are used, of sizes 100, 50, and 10 neurons. Each of these has a ELU activation function. Lastly, there is 1 output neuron.

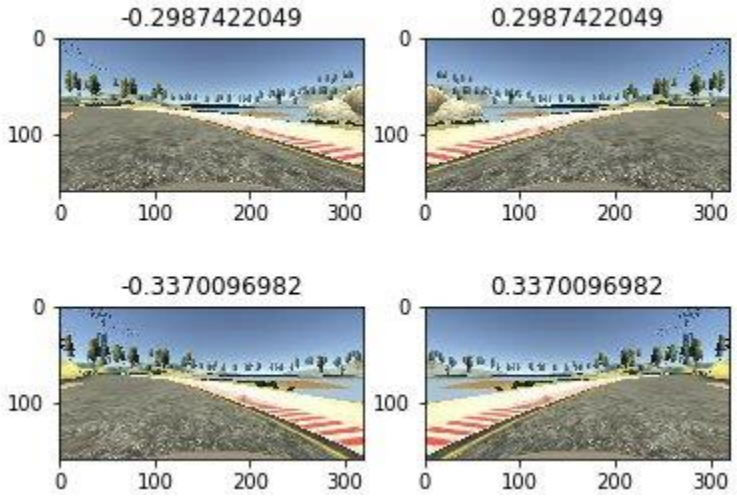
Validation
I validate the model by:
* Create a generator that only returns back center images and steering angles
* Run evaluate_generator() which runs feedforward on the images and compares them the steering angle, resulting in a loss value

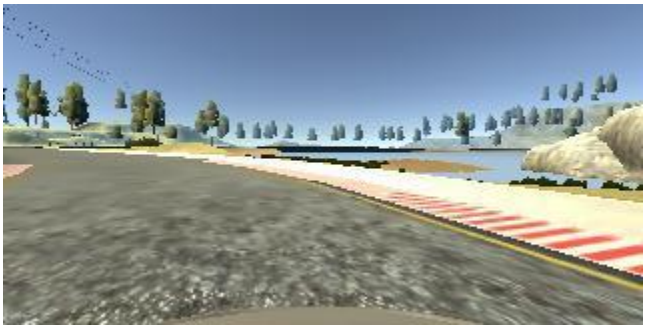
3. Creation of the Training Set & Training Process

I started with the Udacity data. Two examples of center lane driving:

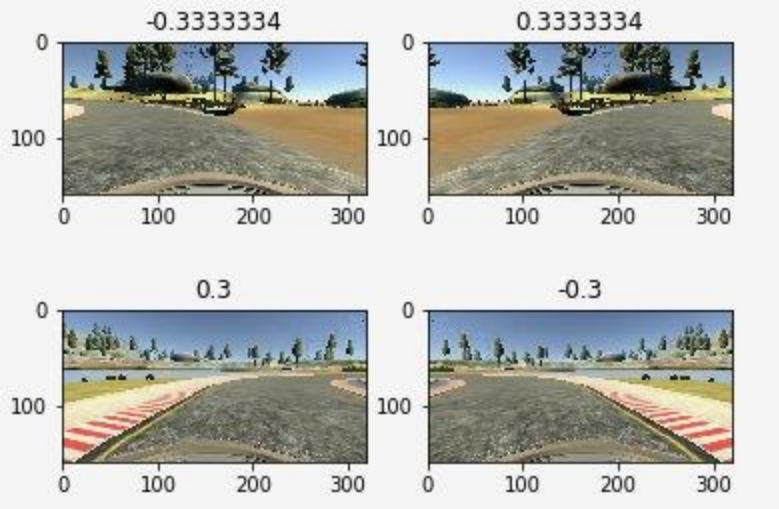


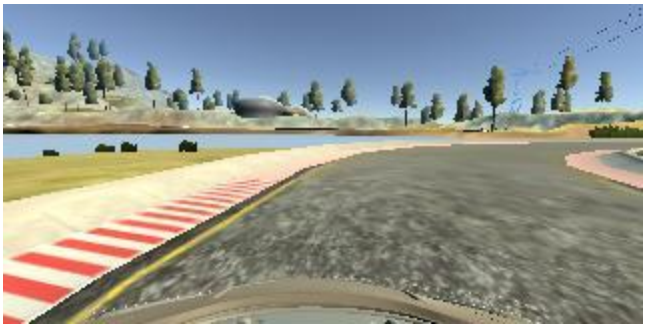
I then recorded the vehicle when it is driving in autonomous mode.



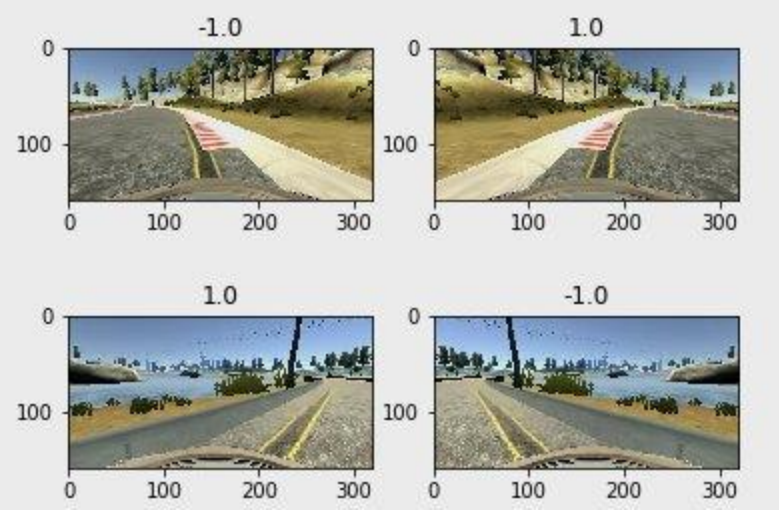


Then I recorded recovery data.





Then I repeated this process on last minute recoveries.



I used this training data for training the model.

Results

The results are listed here

First Track Clockwise

First Track Counter-Clockwise

Second Track Clockwise

Second Track Counter-Clockwise