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.ipynb 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 summarize 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

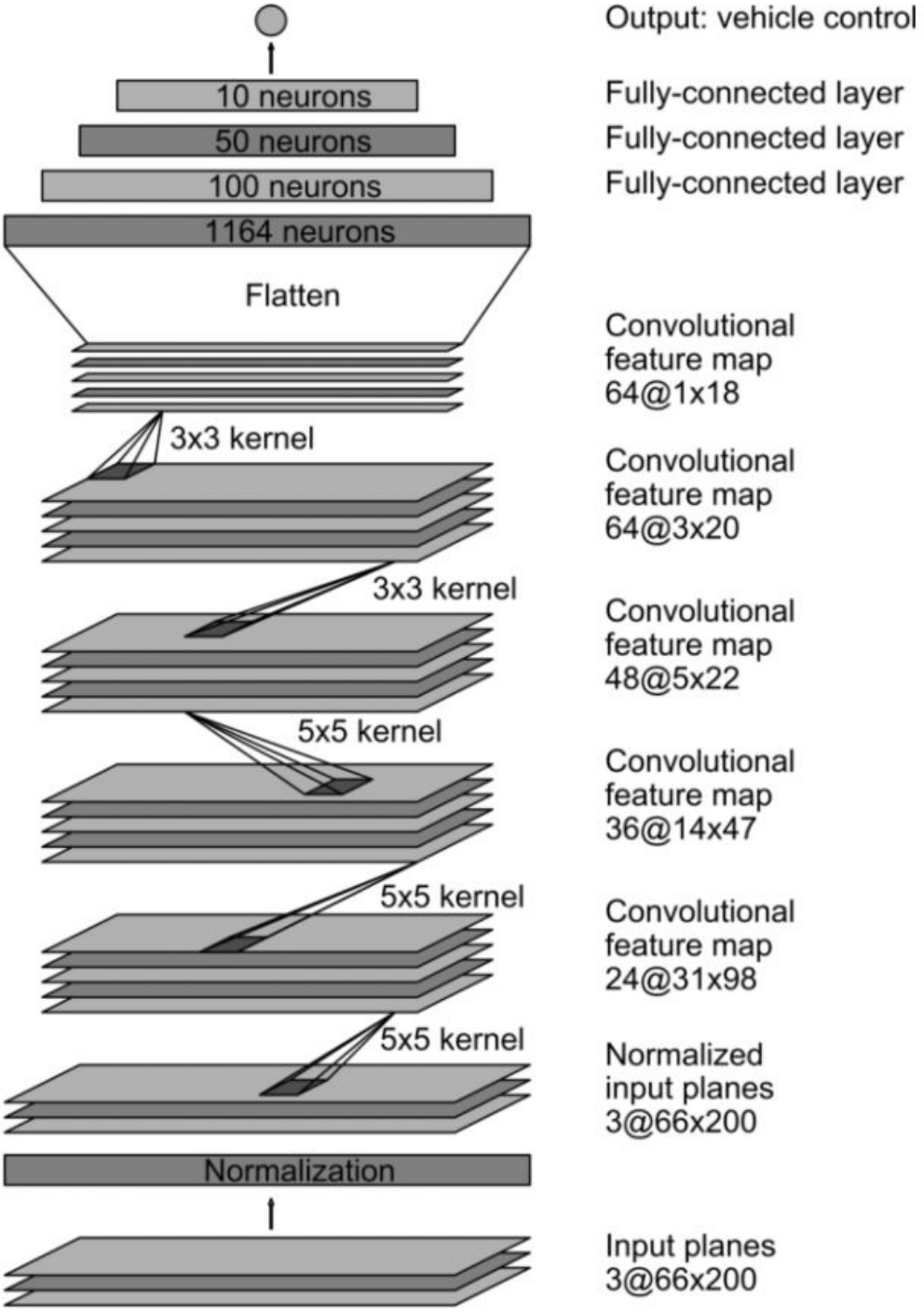
3. Submission code is usable and readable

The model.ipynb 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

1. An appropriate model architecture has been employed

My model was based on the convolutional neural network created by Nvidia for Self Driving.



The model includes RELU layers to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting.

The model was trained and validated on different data sets to ensure that the model was not overfitting, as you could see in this line of code:

model.fit(X_train, y_train, validation_split=0.2, shuffle=True, nb_epoch=3)

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. You can see the implementation in this line of code.

model.compile(loss='mse', optimizer='adam')

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, smooth driving around curves and training data from the Jungle circuit. Driving using mouse, instead of keyboard, really improve the quality of the data and the model worked better. Training data were collected approximately 80% using keyboard steering and 20% using mouse steering.

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

Training Strategy

1. Solution Design Approach
The overall strategy for deriving a model architecture was to start from what was proposed during the class and, then, see what should be improved.

My first step was to use a convolution neural network model similar to the Nvidia neural network.

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.

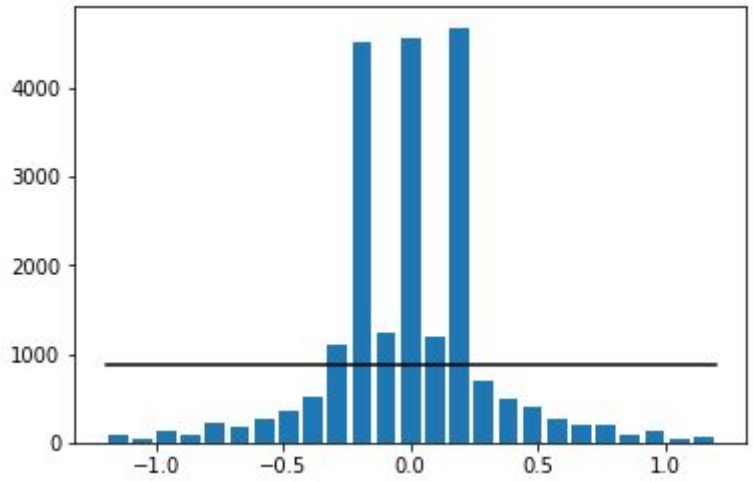
To combat the overfitting, I modified the model adding two Convolutional layer with a keep probability of 0.75, but that wasn't enough.

I run the simulator to understand where the car was acting poorly. I found that was driving poorly when there were curves, so I decided to record data while driving curves.

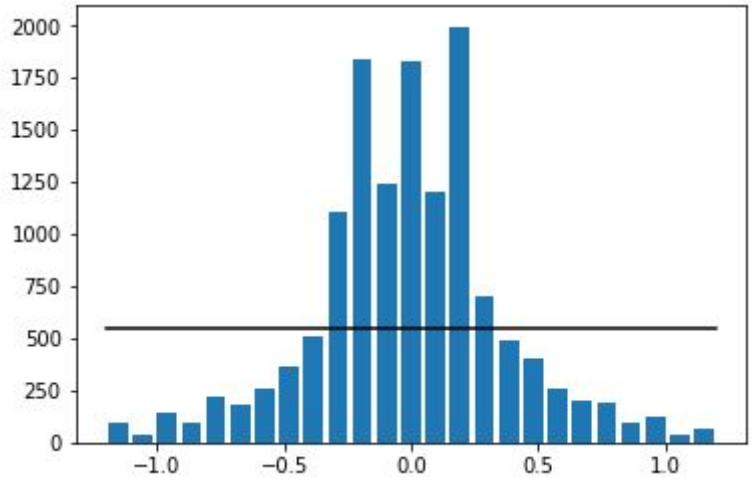
I run again the simulator and I found that was acting poorly where there were shadows, So, I recorded data in the second track, where there are a lot of shadows.

I run again the simulator and I found that the car, sometimes, was not able to recover when was at the limit of the road. So, I recorded data while recovering. I run again the simulator, but it wasn't ok.

Investigating more the dataset, I saw that there were a lot of data with 0 or a small steering angle and fewer data with great steer angles.



So, I decided to cut down by 80% the data that had a steering angle of 0, 0.2 and -0.2. Doing this, data were better distributed around all the steering angles range and obtained a more uniform distribution.



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

This time, it worked and completed an entire lap in track one.

2. Final Model Architecture

Starting from the Nvidia neural network, I added two dropout layer to reduce overfitting. Here is my final neural network:

```
model = Sequential()
model.add(Lambda(lambda x: x / 255.0 - 0.5, input_shape=(160,320,3)))
model.add(Cropping2D(cropping=((70,25),(0,0))))
model.add(Convolution2D(24,5,5,subsample=(2,2),activation="relu"))
model.add(Convolution2D(36,5,5,subsample=(2,2),activation="relu"))
model.add(Convolution2D(48,5,5,subsample=(2,2),activation="relu"))
model.add(Convolution2D(64,3,3,activation="relu"))
model.add(Convolution2D(64,3,3,activation="relu"))
model.add(Flatten())
model.add(Dropout(0.75, noise_shape=None, seed=None))
model.add(Dense(100))
model.add(Dropout(0.75, noise_shape=None, seed=None))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
```

I decided to crop the top 75 lines and the bottom 25 lines of pixels of each image, because they were useless (sky, trees, car)

3. Creation of the Training Set & Training Process

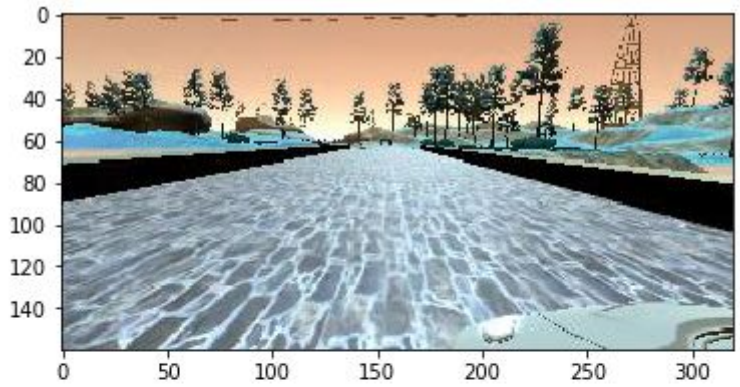
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

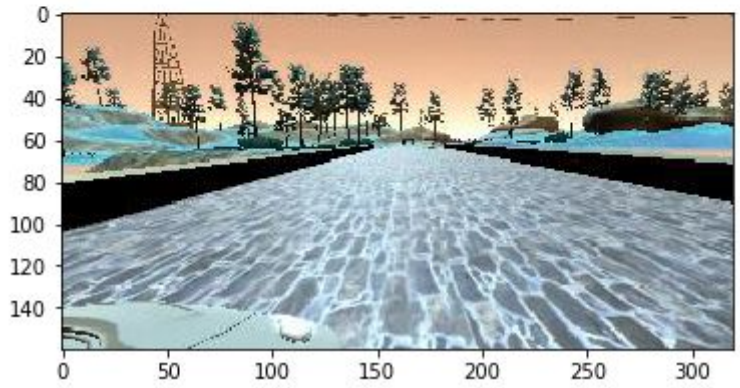
I the recorded the vehicle driving in curves. These images show what a curve look like.

Then I recorded one lap in track two in order to get images with shadows.

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 when he would be going outside the road.

To augment the data sat, I also flipped images and angles thinking that this would help do generalize the training and validation set. For example, here is an image that has then been flipped:





After the collection process, I had 27314 number of data points. I then preprocessed this data by normalizing them and cropping top 75 lines and bottom 25 lines of each image.

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 1.
In the code, I used 3 epochs and you can see that accuracy decrease after the first epoch.
I noticed that using more epochs didn't improve the accuracy of the validation set. I used an adam optimizer so that manually training the learning rate wasn't necessary.

Link to youtube video: https://youtu.be/RX0AxlzyGXU