Behavioral Cloning Project –

Final Submission

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

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
- writeup_report.pdf for 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

My model consists of convolutions neural network with 5x5 and 3x3 filter sizes each and depths of 24, 36, 48.

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

The model flattens the output of last convolution layer and then passes the flattened output through the 4 Dense layers of size 100, 50, 10 and 1.

I am also using mean square error (***MSE***) loss function and ***ADAM*** optimizer to train the network.

```
# Nvidia's CNN architecture
model = Sequential()
model.add(Cropping2D(cropping=((70, 25), (0, 0)), input_shape=X_train[0].shape))
model.add(Lambda(lambda x: (x / 255.0) - 0.5))
model.add(Dropout(0.2))
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(Dropout(0.2))
model.add(Convolution2D(64, 3, 3, activation="relu"))
model.add(Convolution2D(64, 3, 3, activation="relu"))
model.add(Flatten())
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))

model.compile(loss='mse', optimizer='adam')
model.fit(X_train, y_train, validation_split=0.2, shuffle=True, nb_epoch=4)
```

2. Attempts to reduce overfitting in the model

The model contains 2 dropout layers in order to reduce overfitting.

3. Model parameter tuning

I used an Adam optimizer so that manually training the learning rate wasn't necessary.

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. While training the network, I considered all left, center and right camera images of each frame, by correcting steering angles for left and right camera images

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

Model Architecture and Training Strategy

An idea about the sequential convolutional neural network we can read in the blogpost by Nvidia: https://devblogs.nvidia.com/deep-learning-self-driving-cars/

1. Solution Design Approach

The overall strategy for deriving a model architecture was to keep the car running on center of the road and record the data. I have tried much to find that works.

My first step was to use a convolution neural network model similar to the LeNet Architecture used in the tutorial. This model was Ok, but the car had difficulty to run smootly.

After that I tried nvidia architecture. In that architecture I cropped and downscaled my images so that the model can learn faster. But in this architecture I was getting a low mse on training set but high mse on the validation set. This implied that the model was overfitting.

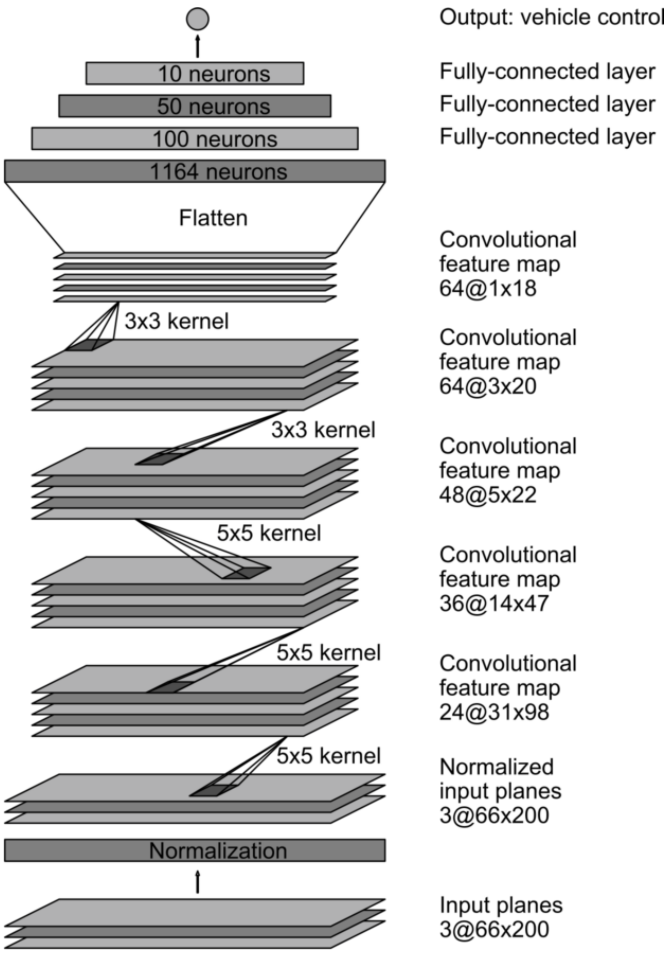
I recorded few laps in backwards in order to have more data und improve the driving behavior and I also flipped the images

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 (model.py lines 55-73) consisted of a convolution neural network similar to the Nvidia's Self Driving car architecture.

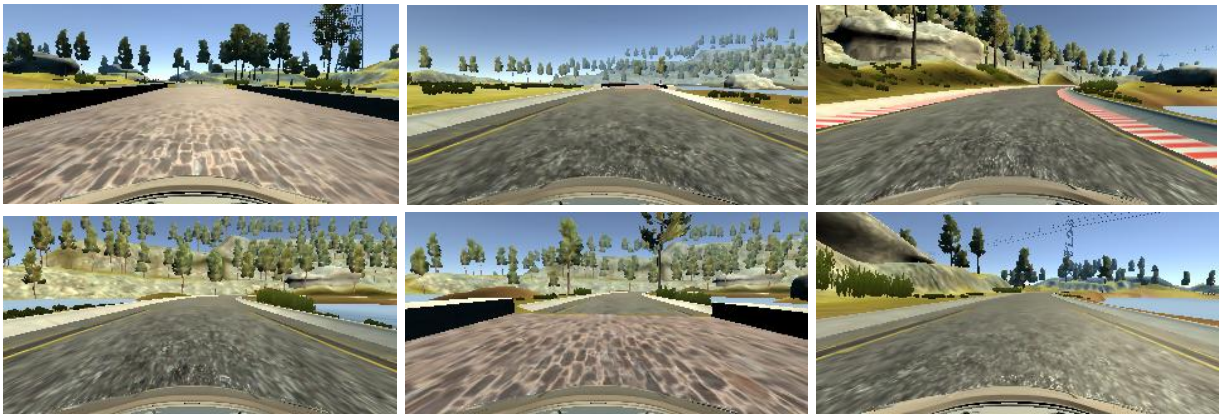
Here is a visualization of the architecture:



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 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 keep itself in middle of road on steep turns. These images show what a recovery looks like.

To augment the data sat, I also flipped images, so I provide more data I thought that this would help the car to stay in middle of the roads while turning on different curves. Also, I used left and right camera images to train the network by correcting the steering angle by +- 0.20 degree.

I then preprocessed this data by normalizing it and cropping it. Also, converted the data from BRG to RBG

I finally randomly shuffled the data set and put 20% of the data into a validation set.

I had issues with the car making the right turn, so I went back and recorded a few recoveries from left to right. This helped.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. I used 4 epochs, as the loss rate dropped really fast.

I used an adam optimizer so that manually training the learning rate wasn't necessary.

Conclusion

By collecting around thousands training images from both track 1 and track 2 and using Nvidia neural network architecture with some added dropout and batch normalization layers it was possible to train the model which can be used to autonomously drive the car in both tracks.

The car sometimes fails, maybe I think, I need to collect more training data or improve the model, could help to build even better model which would be robust on both tracks.