Behavioral Cloning

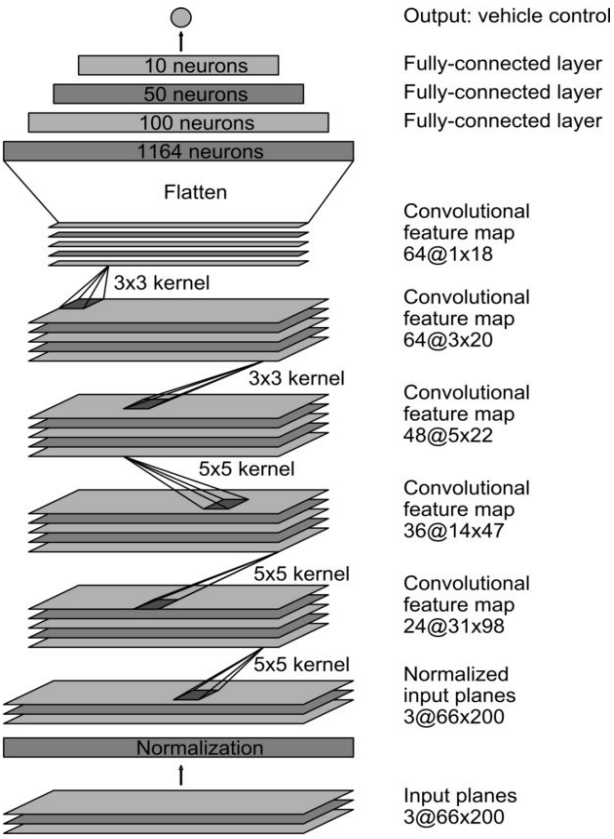
The goals / steps of this project are the following:

- 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

Model Architecture and Training Strategy

1. Architecture

I used NVIDIA model architecture. I thought it could be a good starting point for such project.



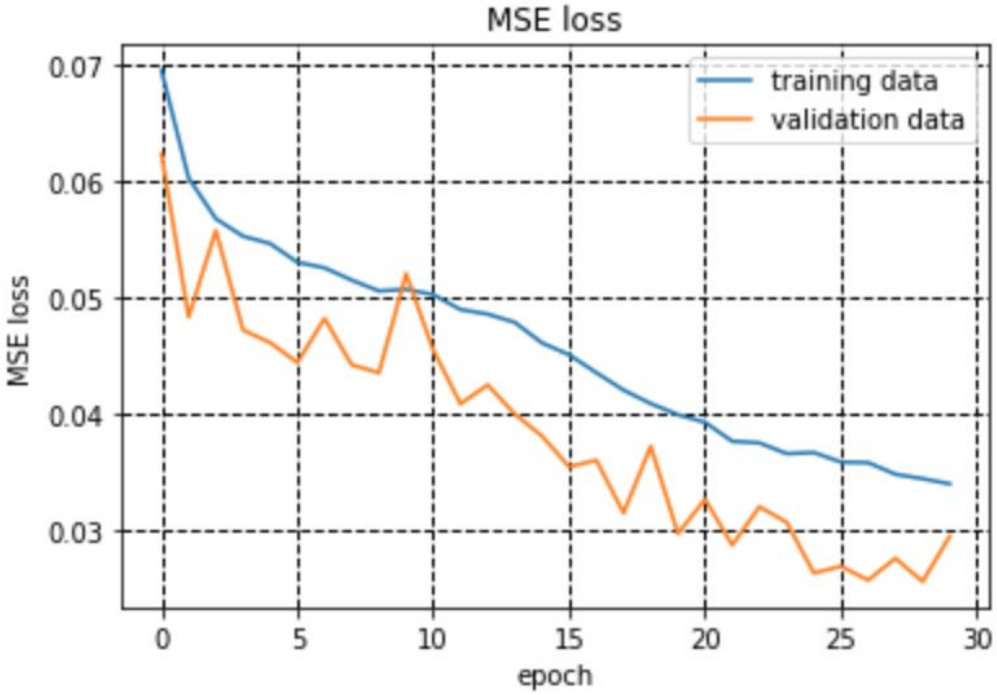
I used ELU for activation, due to fact that it provides better accuracy and faster learning speed. RELU has bias impact on next layer due to fact that their outputs are not balanced around zero.

2. Attempts to reduce overfitting in the model

First training tries showed overfitting case, so I added three dropout layers with 0.5 keep rate. I added final model architecture below.

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 66, 200, 3)	0	lambda_input_1[0][0]
convolution2d_1 (Convolution2D)	(None, 31, 98, 24)	1824	lambda_1[0][0]
convolution2d_2 (Convolution2D)	(None, 14, 47, 36)	21636	convolution2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 5, 22, 48)	43248	convolution2d_2[0][0]
convolution2d_4 (Convolution2D)	(None, 3, 20, 64)	27712	convolution2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 1, 18, 64)	36928	convolution2d_4[0][0]
flatten_1 (Flatten)	(None, 1152)	0	convolution2d_5[0][0]
dense_1 (Dense)	(None, 100)	115300	flatten_1[0][0]
dropout_1 (Dropout)	(None, 100)	0	dense_1[0][0]
dense_2 (Dense)	(None, 50)	5050	dropout_1[0][0]
dropout_2 (Dropout)	(None, 50)	0	dense_2[0][0]
dense_3 (Dense)	(None, 10)	510	dropout_2[0][0]
dropout_3 (Dropout)	(None, 10)	0	dense_3[0][0]
dense_4 (Dense)	(None, 1)	11	dropout_3[0][0]

The plot below show the final loss charts of training and validation data.



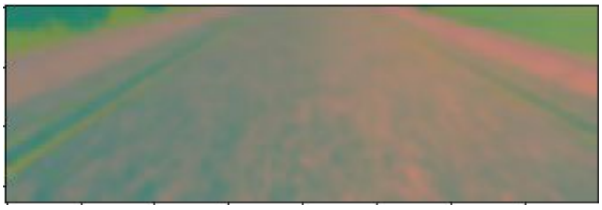
3. Image preprocessing

Image preprocessing unit contains 4 steps:

1. Crop top and bottom of an image, the nn does not need constantly changing landscape info and static car image.
2. Resize image for reducing the nn amount of parameters.
3. Add gauss filter for noise removing.
4. Convert to YUV colorspace.

All these steps were used by NVIDIA. There is a tricky step in drive.py file. I added same function^ and change read mode from BGR to RGB in drive.py function.

Default	

Cropped	
Resized	
Blurred	
YUV	

4. Training strategy

Default provided data could be used for training on the first lap. But when I tried to use data on the second one I realized the fact that I need collect data from it. My first attempt was to try to use the keyboard as a controller, but after several tries and digging forum and the guide, I decided to use the mouse as a steering device. Keyboard generates step function and for good learning, we need a continuous function with a small delta of steering angle. The trained network performed better, but still had problems with sharp turns. Finally, I used ps4 controller, because it could give good speed control and I could perform sharp turn smoothly.

5. Summary

You could find captured videos of the first and second lap in video1.mp4 and video2.mp4 files respectively. I used 18 mph limit for driving. The second lap is very curvy and sloppy so the car needs more time to recognize road.