Behavioral Cloning

Writeup

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.py containing the script to create and train the model
- •video.mp4 containing the video of autonomous drive outpuy
- •model.h5 containing a trained convolution neural network
- •writeup_report.pdf summarizing the results

•drive.py containing code for driving car

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 run
```

3. Submission code is usable and readable

The train_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

Layer	Description
Input	160x320x3 RGB image
Cropping (line 95)	Crop top 50 pixels and bottom 20 pixels; output shape = 90x320x3
Normalization (line 97)	Each new pixel value = old pixel value/255 - 0.5
Convolution 5x5 (line 100)	5x5 kernel, 2x2 stride, 24 output channels, output shape = 43x158x24
RELU	
Convolution 5x5 (line 101)	5x5 kernel, 2x2 stride, 36 output channels, output shape = 20x77x36
RELU	
Convolution 5x5 (line 102)	5x5 kernel, 2x2 stride, 48 output channels, output shape = 8x37x48
RELU	
Convolution 5x5(line 103)	3x3 kernel, 1x1 stride, 64 output channels, output shape = 6x35x64

Layer	Description
RELU	
Convolution 5x5(line 104)	3x3 kernel, 1x1 stride, 64 output channels, output shape = 4x33x64
RELU	
Flatten(line 106)	Input 4x33x64, output 8448
Fully connected(line 108)	Input 8448, output 100
Dropout(line 111)	Set units to zero with probability 0.5
Fully connected(line 113)	Input 100, output 50
Fully connected(line 114)	Input 50, output 10
Fully connected(line 115)	Input 10, output 1 (labels)

2. Attempts to reduce overfitting in the model

A dropout layer is added after the first fully connected layer to reduce the possiility of overfitting.

The model was trained and validated on different data sets to ensure that the model was not overfitting. 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 with mean squared error, so the learning rate was not tuned manually (model.py line 118).

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 to make the learning model more robust. I drove around the track 4 times to maximise the data collection.

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 implement the

My first step was to use a convolution neural network model similar to the NVIDIA’s Deep Learning for Self-Driving Cars, I thought this model might be appropriate because it is proven and the use case is similar

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 inceased the validation data to 30% of the overall data from 20% and added a dropout layer after the first fully connected layer.

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track, to improve the driving behavior in these cases, I modified the steering angle to 0.65 after testing it with 0.2, 0.4, 0.5, 0.7

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road but is still a little wobbly which needs to be improved

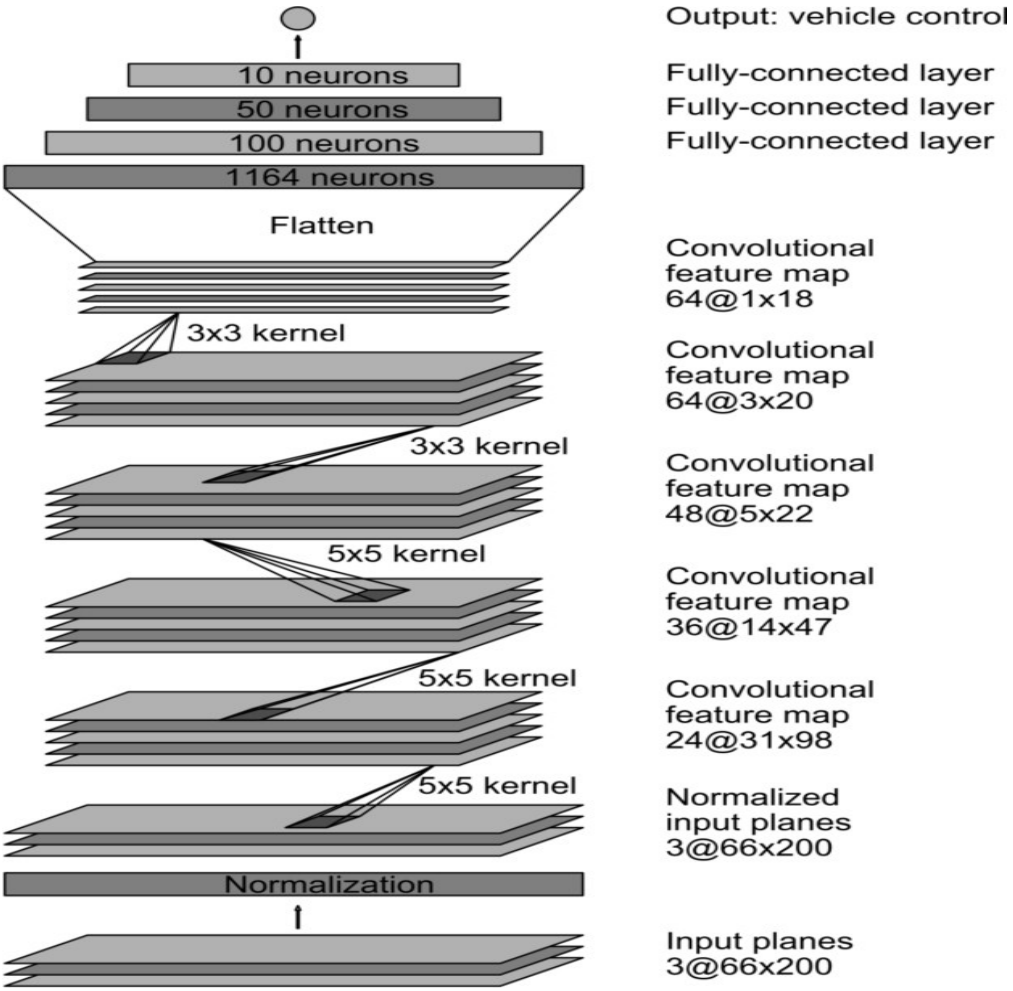
2. Final Model Architecture

The final model architecture is the well proven NVIDIA’s Deep Learning for Self-Driving Cars (model.py lines 95-115) consisted of a convolution neural network with the following layers and layer sizes

Layer	Description
Input	160x320x3 RGB image
Cropping (line 95)	Crop top 50 pixels and bottom 20 pixels; output shape = 90x320x3
Normalization (line 97)	Each new pixel value = old pixel value/255 - 0.5
Convolution 5x5 (line 100)	5x5 kernel, 2x2 stride, 24 output channels, output shape = 43x158x24
RELU	
Convolution 5x5 (line 101)	5x5 kernel, 2x2 stride, 36 output channels, output shape = 20x77x36
RELU	
Convolution 5x5 (line 102)	5x5 kernel, 2x2 stride, 48 output channels, output shape = 8x37x48
RELU	
Convolution 5x5(line 103)	3x3 kernel, 1x1 stride, 64 output channels, output shape = 6x35x64
RELU	
Convolution 5x5(line 104)	3x3 kernel, 1x1 stride, 64 output channels, output shape = 4x33x64
RELU	
Flatten(line 106)	Input 4x33x64, output 8448
Fully connected(line 108)	Input 8448, output 100
Dropout(line 111)	Set units to zero with probability 0.5
Fully connected(line 113)	Input 100, output 50
Fully connected(line 114)	Input 50, output 10
Fully connected(lin	Input 10, output 1 (labels)

Layer	Description
e 115)	

Here is a visualization of the architecture

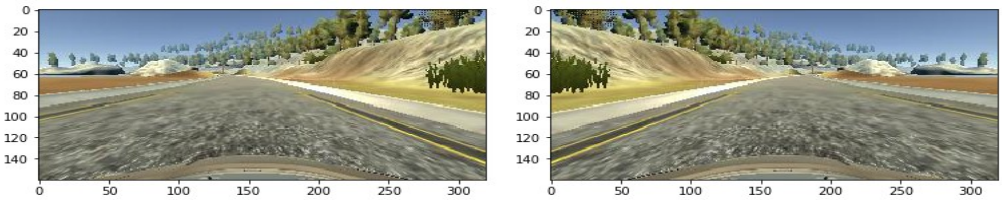


3. Creation of the Training Set & Training Process

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 adjust itself from corners These images show what a recovery looks like the following

To augment the data sat, I also flipped images and angles thinking that this would give a new perpective for the model. For example, here is an image that has then been flipped:



After the collection process, I had 1613 number of data points. I then preprocessed this data combining the center, left, right and flipped image yeilding 4*1613.

For left and right camera, i used and angle factor of 0.65 as this was the value which drove the car on the lanes while other values pushed the car outside the track.

I finally randomly shuffled the data set and put 30% 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 15(after which the loss increased and the final loss is almost the same as in 15th epoch) as evidenced by the loss plot. I used an adam optimizer so that manually training the learning rate wasn't necessary.