# Behavioral Cloning project report

---

**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
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_report.pdf 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 10 layers, including a normalization layer and a cropping layer, 5 convolutional layers, and 3 fully connected layers. Reference(architecture used by NVIDIA, self-driving car).

The model includes RELU layers to introduce nonlinearity (code line 81, 82, 83, 86 & 87), and the data is normalized in the model using a Keras lambda layer (code line 77).

The model also includes Cropping Layer to crop the image (70, 25)

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 85 & 89).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 61-65). 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 (model.py line 98).

Streering angle correction is tuned for left and right images, Initial value was 0.2 and 0.15 produced better result

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of the images from the center, the left and right side cameras

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 reduce mean square error

My first step was to use a convolution neural network model similar to the NVIDIA I thought this model might be appropriate because this model is used in the similar use-case to drive autonomous car.

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 to include Dropout (model.py 85, 89).

Then I added Lambda layer to normalize the data

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

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 76-95) consisted of a convolution neural network with the following layers and layer sizes 10

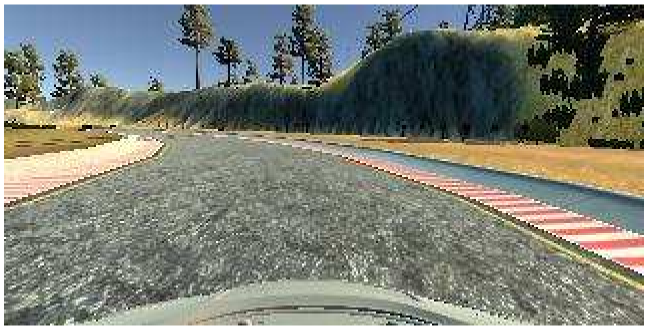
| Layer | Description |
|---|---|
| Lambda | Normalization, std deviation : input shape=(160,320,3) |
| Cropping | Crop image width & height = ((70,25), (0,0)) & input shape=(160,320,3) |
| Convolution2D with RELU | Convolution filter with 24 filter, 5x5 kernel |

| | |
|---|---|
| Convolution2D with RELU | Convolution filter with 36 filter, 5x5 kernel |
| Convolution2D with RELU | Convolution filter with 48 filter, 5x5 kernel |
| Drop Out | Applies Dropout to the input. Rate = 0.5 |
| Convolution2D with RELU | Convolution filter with 64 filter, 3x3 kernel |
| Convolution2D with RELU | convolution filter with 64 filter, 3x3 kernel |
| Drop Out | Applies Dropout to the input. Rate = 0.5 |
| Flatten | Flattens the input |
| Dense | Fully connected layer, output of shape 100 |
| Dense | Fully connected layer, output of shape 50 |
| Dense | Fully connected layer, output of shape 10 |
| Dense | Fully connected layer, output of shape 1 |

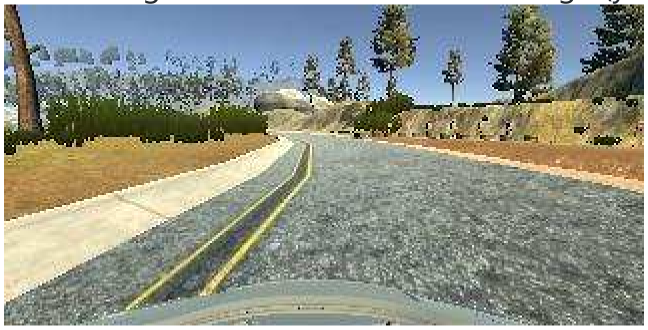
Table: Network Layers

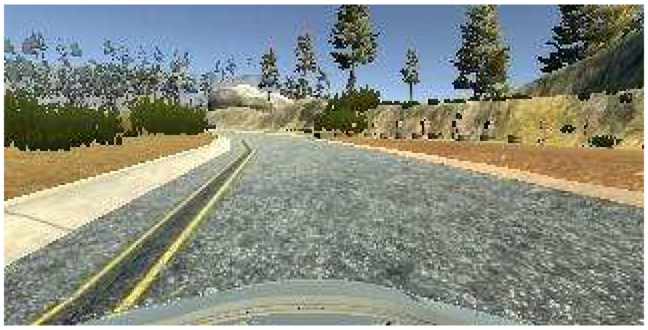
3. Creation of the Training Set & Training Process

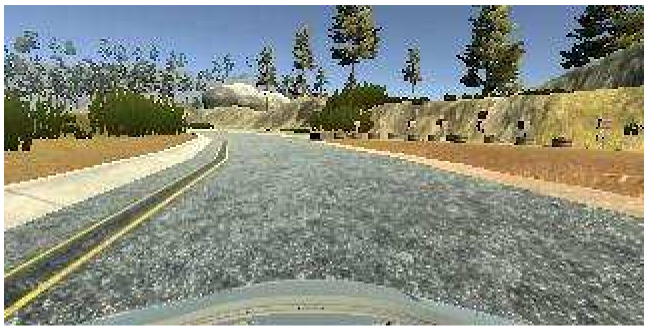
To capture good driving behavior, I first recorded four laps on track one using center lane driving and recovering from if too close to road edge (yellow line). Here is an example image of center lane driving:



Recovering from if too close to road edge (yellow line), examples:

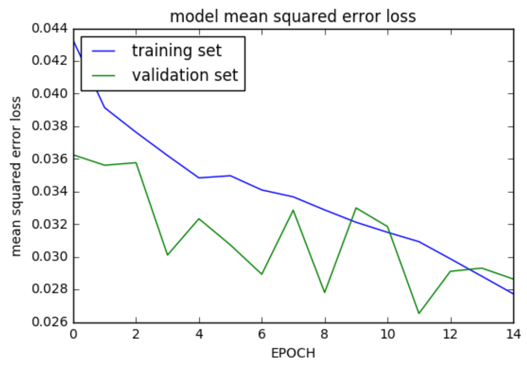






After the collection process, I had 14370 number of data points and randomly shuffled the data set and put 20% of the data into a validation set.

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 15 as evidenced by plotted graph and batch size is 64



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