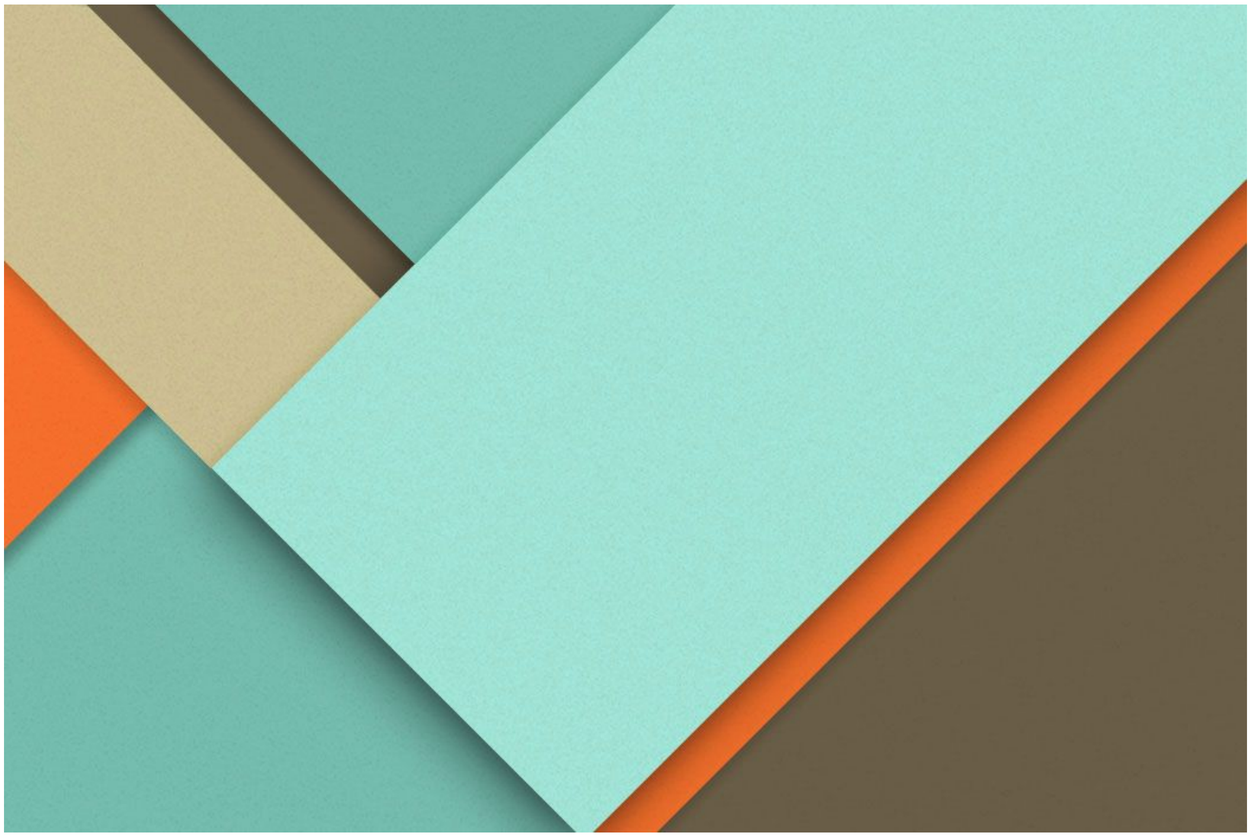


# Behavior Cloning

03.21.2017

Girish Pai
Udacity
Self Driving Car Engineer Nanodegree Student

Behavioral Cloning

Writeup 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 convolutional neural network
- Writeup_report.pdf (this file) 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 final model architecture was borrowed from Nvidia's paper https://arxiv.org/pdf/1604.07316.pdf

This consists of a convolution neural network with 3x3 filter sizes and depths between 24 and 64 (model.py lines 80-84)

The model includes RELU layers to introduce nonlinearity (activation parameter in Keras Convolutional2D layers), and the data is normalized and cropped in the model using a Keras lambda and Cropping 2D layer (lines 78 and 79)

2. Attempts to reduce overfitting in the model

The model contains dropout layer in order to reduce overfitting (model.py line 85).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 21). 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 91).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used the center , left and right camera images to keep the vehicle on the road. For the left and right camera images a correction factor to get the correct steering angle if the car encountered any of the curbs or turns.

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 as follows :

1. Generate data by driving the car myself around track 1 for 2 laps.
2. Split the dataset (images and steering angle) into training and validation sets.
3. Choose a base model.
4. Train the model with the training / validaiton data for fixed epochs till you get sufficiently low validation loss.
5. Save the model and run the simulator in autonomous mode.
6. Update the model and / or do some additional preprocessing to the data and repeat steps 4 and 5 till the car is able to drive successfully for at least 1 lap.

The base model which I chose was similar to that of Nvidia (https://arxiv.org/pdf/1604.07316.pdf). The main reason for this is that the purpose of the model was exactly same as what I was trying to do - make a car drive autonomously - with NVIDIA driving a real car in a real city as opposed to a simulated environment.

To combat the overfitting, I used Dropout layer and also augmented the data by flipping the image and angle. This way the model could generalize better.

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 and was wobbly. To resolve this issue, I had to crop the image to remove the unwanted parts (like sky). This way the model could be trained more efficiently just focussing on the road,  curves, curbs etc.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

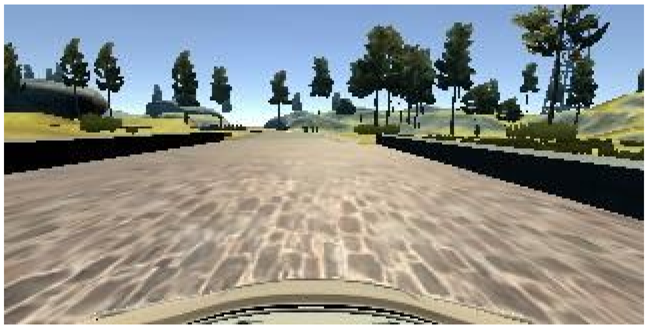
The final model architecture (model.py lines 75-94) was very similar to the NVIDIA one, except the addition of the Cropping layer and Dropout layer.

- Layer 1 - Normalizing layer
- Layer 2 - Cropping Layer
- Layer 3 - Convolutional Layer - 5x5 filters with depth 24 (with subsampling)

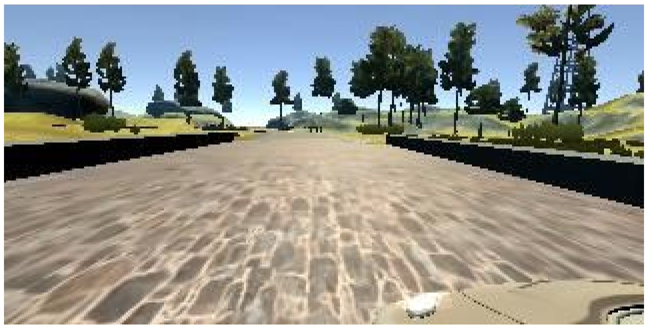
- Layer 4 - Convolutional Layer - 5x5 filters with depth 36 (with subsampling)
- Layer 5 - Convolutional Layer - 5x5 filters with depth 48 (with subsampling)
- Layer 6 - Convolutional Layer - 3x3 filters with depth 64 (with subsampling)
- Layer 7 - Convolutional Layer - 3x3 filters with depth 64 (with subsampling)
- Layer 8 - Fully Connected Layer with 100 hidden units.
- Layer 9 - Fully Connected Layer with 50 hidden units.
- Layer 10 - Fully connected Layer with 10 hidden units.
- Layer 11 - Fully connected output layer with one output.

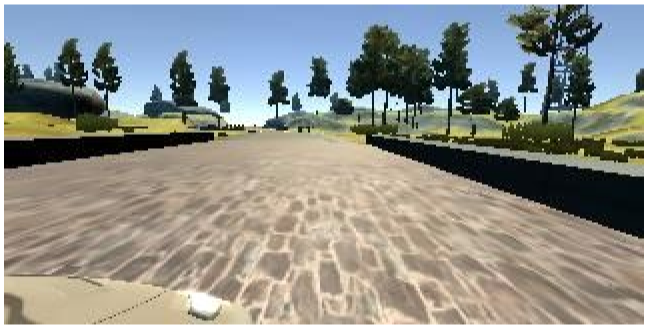
3. Creation of the Training Set & Training Process

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 used the images from left and right cameras and included a correction factor, indicating the steering wheel should turn appropriately at curves, curbs etc. Following are examples of left and right camera images for the center image shown above.





To augment the data sat, I also flipped images and angles. This would help the model generalize better, as the actual drive is clockwise and circular and hence always would be biased to turning left.

After the collection process, I had X number of data points. I then pre processed this data by normalizing the images (dividing by 255) and also cropped the image so as to remove the unwanted parts (like sky).

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 6 after which the validation loss plateaued. Beyond 6 the validation loss started to increase.   I used an adam optimizer so that manually training the learning rate wasn't necessary.