**Behavioral Cloning**

Contents

[**Behavioral Cloning** 1](#_Toc477833235)

[1 Reference to Project Code 2](#_Toc477833236)

[1.1 Files Included in Submitted Project 2](#_Toc477833237)

[1.2 Functional Code 2](#_Toc477833238)

[1.3 Code Usability and Readability 2](#_Toc477833239)

[2 Model Architecture 3](#_Toc477833240)

[2.1 Solution Design Approach 3](#_Toc477833241)

[2.2 Attempts to Reduce Overfitting 3](#_Toc477833242)

[2.3 Model Parameters Tuning 3](#_Toc477833243)

[2.4 Final Architecture 4](#_Toc477833244)

[3 Creation of Training Set and Training Process 6](#_Toc477833245)

[3.1 Initial Capture 6](#_Toc477833246)

[3.2 Recovering from Sides 6](#_Toc477833247)

[3.3 Second Capture 7](#_Toc477833248)

[3.4 Data Augmentation 7](#_Toc477833249)

[3.5 Data Preprocessing 7](#_Toc477833250)

[3.6 Generator 8](#_Toc477833251)

# Reference to Project Code

Here is a link to my [project code](https://github.com/pepere/Behavioral_Cloning/upload/master).

## Files Included in Submitted Project

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 (this memo)

## Functional Code

Using the Udacity provided **simulator** and my **drive.py** file, the car can be driven autonomously around the track by executing the following commands:

* activate carnd-term1
* python drive.py model.h5

## Code Usability and Readability

The **model.py** file contains the code for training and saving the convolution neural network. It shows the **pipeline** I used for training and validating the model, and it contains **comments** to explain how the model works.

# Model Architecture

## Solution Design Approach

I first opted for a **convolution neural network** similar to the **NVidia model**. I thought it might be appropriate after reading the company’s memo (see [link](https://images.nvidia.com/content/tegra/automotive/.../end-to-end-dl-using-px.pdf)) and looking for feedback on the model on the web.

In order to test my model, I split my images and steering angles into **training** and **validation** sets.

Then I experienced **overfitting**, after noticing a low MSE (mean squared error) on the training set and a high MSE on the validation set. To prevent this phenomenon, I modified the model by including **Dropout** and **L2 Regularization** steps.

I finally ran the simulator to see how well the car was driving around track one. The vehicle fell off the track in some (dangerous) spots. To improve the driving behavior, I **augmented the training** data and recorded **shoulder recovery** data.

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

## Attempts to Reduce Overfitting

As mentioned here above, I tried the three following approaches to prevent **overfitting** in my model:

* **Dropout**: as mentioned here above, after each convolutional and fully connected layer (except for the final one), I **dropped 20%** of the training data (*see example on line 382 in model.py*).
* **L2 Regularization**: as mentioned above, each convolutional layer is followed by a L2-Regularization step, in order to ignore excessively **large weights** (*see example on line 380 in model.py*).
* **Train / Validation split**: the model was trained and validated on different data sets, to ensure that the model was not overfitting. **20%** of the training data was randomly selected for validation (*see lines 426-430 in model.py*).

## Model Parameters Tuning

The model used an **Adam optimizer**, so manually training the learning rate was not necessary (*see line 420 in model.py*).

## Final Architecture

My model consists of the following layers (*see lines 371-416 in model.py*):

| **Layer #** | **Sub Layer** | **Description / Comment** | **Output** |
| --- | --- | --- | --- |
| **0** | Input | 66x200x3 images (160x320 RGB images cropped and resized) | N/A |
| **1** | Normalization | Normalization: pixels / 127.5 - 1.0 | 66x200x3 |
| **2** | Convolution 5x5 | 5x5 filter, 2x2 strides, valid padding. | 31x98x24 |
| **2** | L2 Regularization | To prevent overfitting by penalizing large weights (1% of data) | 31x98x24 |
| **2** | Activation | ReLU (to introduce nonlinearity) | 31x98x24 |
| **2** | Dropout | 20% of data dropped | 31x98x24 |
| **3** | Convolution 5x5 | 5x5 filter, 2x2 strides, valid padding. | 14x47x36 |
| **3** | L2 Regularization | To prevent overfitting by penalizing large weights (1% of data) | 14x47x36 |
| **3** | Activation | ReLU (to introduce nonlinearity) | 14x47x36 |
| **3** | Dropout | 20% of data dropped | 14x47x36 |
| **4** | Convolution 5x5 | 5x5 filter, 2x2 strides, valid padding. | 5x22x48 |
| **4** | L2 Regularization | To prevent overfitting by penalizing large weights (1% of data) | 5x22x48 |
| **4** | Activation | ReLU (to introduce nonlinearity) | 5x22x48 |
| **4** | Dropout | 20% of data dropped | 5x22x48 |
| **5** | Convolution 3x3 | 3x3 filter, 1x1 strides. | 3x20x64 |
| **5** | L2 Regularization | To prevent overfitting by penalizing large weights (1% of data) | 3x20x64 |
| **5** | Activation | ReLU (to introduce nonlinearity) | 3x20x64 |
| **5** | Dropout | 20% of data dropped | 3x20x64 |
| **6** | Convolution 3x3 | 3x3 filter, 1x1 strides. | 1x18x64 |
| **6** | L2 Regularization | To prevent overfitting by penalizing large weights (1% of data) | 1x18x64 |
| **6** | Activation | ReLU (to introduce nonlinearity) | 1x18x64 |
| **6** | Dropout | 20% of data dropped | 1x18x64 |
| **7** | Flattening | Main function used: **flatten** class from tensorflow.contrib.layers. | 1,164 |
| **8** | Fully connected |  | 100 |
| **8** | Activation | ReLU (to introduce nonlinearity) | 100 |
| **8** | Dropout | 20% of data dropped | 100 |
| **9** | Fully connected |  | 50 |
| **9** | Activation | ReLU (to introduce nonlinearity) | 50 |
| **9** | Dropout | 20% of data dropped | 50 |
| **10** | Fully connected |  | 10 |
| **10** | Activation | ReLU (to introduce nonlinearity) | 10 |
| **11** | Fully connected |  | 1 |

# Creation of Training Set and Training Process

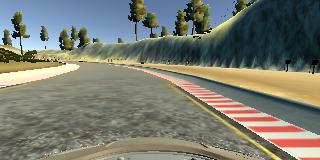
## Initial Capture

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:



## Recovering from Sides

I then recorded the vehicle **recovering from left and right shoulders** back to center, so that it would learn to drive in the center of the road. These images show what a shoulder recovery looks like, from the **right side** of the road:







## Second Capture

Then I repeated this process on **track two** in order to get more data points:



## Data Augmentation

I augmented the data in two ways:

* **Horizontal flip** (*see lines 221-264 in model.py*): I used this technique to convert left turns into right turns, and vice versa. To avoid overloading the model, I only selected images sharp turns (associated steering measurement > 0.5). Associated steering measurements were flipped accordingly.
* **Horizontal translation** (*see lines 270-333 in model.py*): horizontal translation was used to artificially create more images. To avoid overloading the model, only 20% of training images were randomly selected for translation. Associated steering measurements were recalculated accordingly.

## Data Preprocessing

After the collection process, I had **46,298** number of data points. I then preprocessed this data by:

* **Cropping** images (*see lines 110-111 in model.py*): in order to avoid overloading the model with meaningless data, I cropped images by 45 pixels at the top and 22 pixels at the bottom (*based on 160x320 scale*).
* **Resizing** images (*see lines 114-115 in model.py*): I resized images **from 160x320 to 66x200** to fit to my NVidia model.
* **Normalizing** image data (*see line 376 in model.py*): I normalized images by recalculating pixels as **xnorm = x / 127.5 - 1.0**.

## Generator

Using a generator (**fit\_generator syntax**) was a necessity to feed such a **large volume** of data into this model (*see lines 455-457 in model.py*).

The ideal number of **epochs** was **50**. This can be visualized by plotting the training loss vs. the validation loss by epochs (*see lines 470-477 in model.py*).