# CarND-Behavioral-Cloning-P3

## Overview

The purpose of the project was to implement a behavioral cloning algorithm using deep neural networks and convolution networks to clone driving behavior. This was accomplished by the following steps:

- Collect driving data using the simulator provided.
- Augment the data to provide a generalized data set for the deep neural network.
- Train and validate the model by splitting the data in to training and validation sets.
- Test the trained model using the simulator by having the car run in autonomous mode.
- Record video evidence to show car completes simulated track without getting off the road.

## Rubrics Addressed

### Required Files & Quality of Code

#### 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 is well commented. The trained model provides autonomous commands to the car allowing it to successfully complete track 1 multiple times.

## Model Architecture and Training Strategy

### 1. An appropriate model architecture has been employed

I adopted a simplified version of the NVIDIA model for this project. The model consists of a lambda layer, a cropping layer, 3 convolution layers, and 4 fully-connected dense layers.

The lambda layer is used to normalize the training data and the cropping layer excludes segments of the image considered irrelevant to training the model.

All 3 convolution layers use 3x3 convolution with a 1x1 stride value and RELU activation. From first to third, the convolution layers have output filters 8, 32, and 64, respectively. Inserted between each convolution layer are max pooling layers to reduce overfitting.

```
178 #1st convolution layer with max pooling
179 model.add(convolutional.Convolution2D(8, 3, 3, activation='relu'))
180 model.add(pooling.MaxPooling2D(pool_size=(2, 2)))
181
182 #2nd convolution layer with max pooling
183 model.add(convolutional.Convolution2D(32, 3, 3, activation='relu'))
184 model.add(pooling.MaxPooling2D(pool_size=(2, 2)))
185
186 #3rd convolution layer with max pooling
187 model.add(convolutional.Convolution2D(64, 3, 3, activation='relu'))
188 model.add(pooling.MaxPooling2D(pool_size=(2, 2)))
```

The first 3 dense layers are activated with RELU functions, with output dimensions of 500, 100, and 20, in sequential order. The last dense layer has an output dimension of 1 and no activation function.

```
#Dense layer with relu and dropout layer
model.add(core.Dense(500, activation='relu'))
model.add(core.Dropout(.5))

#Dense layer with relu and dropout layer
model.add(core.Dense(100, activation='relu'))
model.add(core.Dropout(.25))

#Dense layer with relu and dropout layer
model.add(core.Dense(20, activation='relu'))
model.add(core.Dense(1))
```

### 2. Attempts to reduce overfitting in the model

To reduce overfitting in the model, 80% of the data was used for training and 20% for was used validation. To further reduce the potential for overfitting, max pooling layers are included after every convolution layer. In addition, dropout layers are included after every dense layer, except for the last dense layer.

### 3. Model parameter tuning

An Adam optimizer was employed in this model with the learning rate initialized to 0.0001.

### 4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of training data provided by Udacity and additional data collected while driving the car in trouble areas of the track. Additional training data was created by geometrically manipulating the images.

## Architecture and Training Documentation

### 1. Solution Design Approach

The design approach was to employ the NVIDIA model which had been proven to work for the same application. However, a simplified version of the NVIDIA model was developed for this application. The model was first tested with the training data provided by Udacity and showed promising results with low mean squared errors for both training and validation data. Unfortunately, using only the Udacity training data was not enough to successfully complete the simulation track. For this reason, additional training data had to be collected and appended to the existing Udacity training data.

### 2. Model Architecture

As mentioned earlier the final model consisted of 3 convolution layers and 4 dense layers. This model was a simplified version of the NVIDIA architecture. The convolution layers consisted of 3x3 convolution with a 1x1 stride value and RELU activation with output filters 8, 32, and 64, respectively. The first 3 dense layers are activated with RELU functions, with output dimensions of 500, 100, and 20, in sequential order. The last dense layer has an output dimension of 1 and no activation function. The diagram below illustrates the final model architecture used.

To prevent overfitting, max pooling layers were inserted between each convolution layer and dropout layers were inserted between the dense layers.

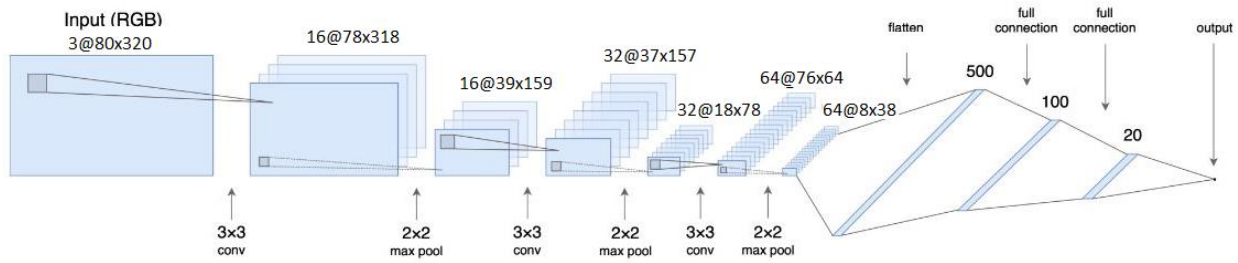


Fig. 2.1. Illustration of model architecture.

## 3. Training Set

The model was initially trained using the Udacity training data provided. Unfortunately, because of the heavily biased 0 angle steering data shown in the distribution below the car was unable to successfully complete the track.

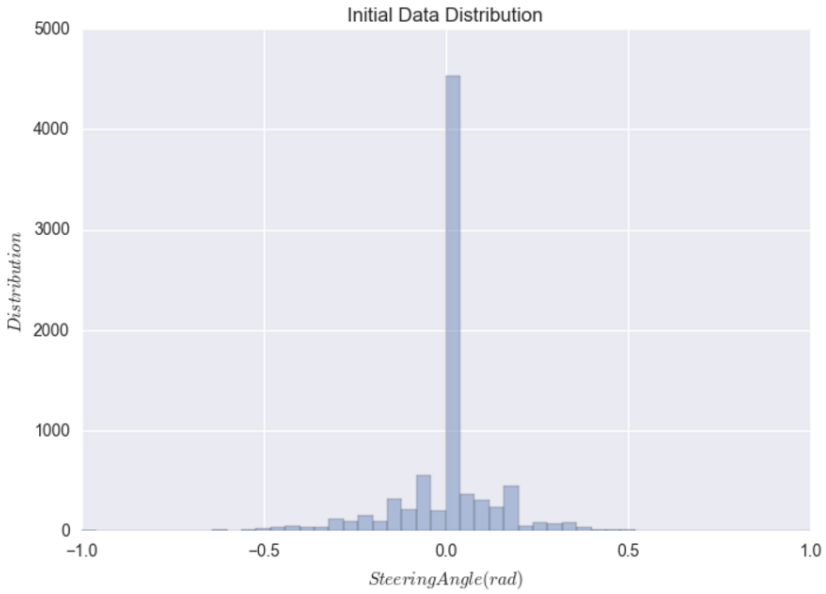

Fig. 3.1. Histogram of Udacity training data set.

To obtain a successful simulation run, a more generalized data set was created by excluding some of the steering angles between -0.1 and 0.1. This resulted in the histogram below, with the frequency of 0-steering data much closer to the frequency of other steering data in the distribution.

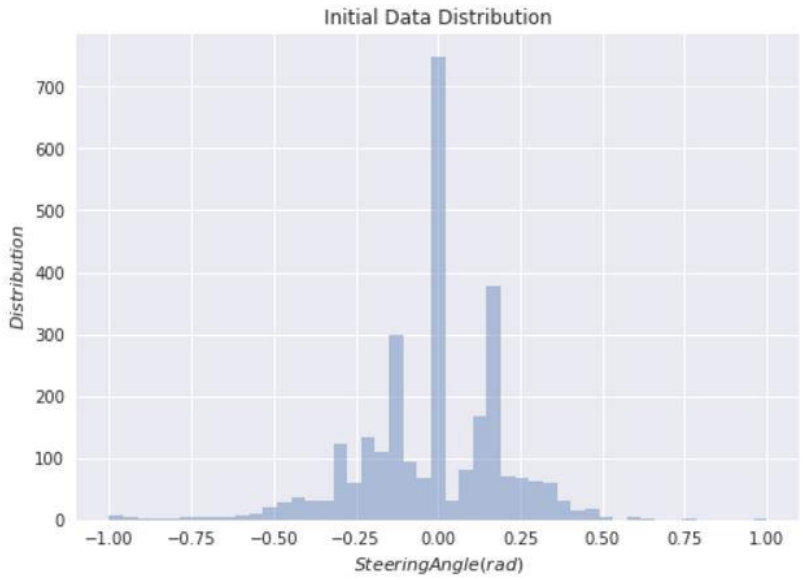


Fig. 3.2. Histogram of modified training data set.

Once the amount of 0-steering data was reduced, images from the center, left, and right cameras were randomly rotated and the steering angles modified accordingly. Manipulating the images provided augmented training data for the model. The center images were randomly rotated and their brightness level altered. While the left and right camera images were rotated clockwise and counter-clockwise, respectively. The code snippets below show how the center camera images were augmented.

```
53      #Declare brightness parameters
54      alpha = np.random.uniform(0.3, 1.0)
55      beta = 30 * np.random.uniform()
```

```
        #Alter image brighness
        img = (alpha*img + beta).astype(np.uint8)
        if rotate == 1:
            ang = np.random.uniform(min_rot, max_rot) #Select rotation angle: -12degrees < ang <12degrees
            M = cv2.getRotationMatrix2D((cols/2,rows/2),ang,1)
            img = cv2.warpAffine(img,M,(cols,rows))
            #if image rotate right (-ang), then steer right
            if ang < 0: steering = steering + (ang/min_rot*right_turn) #right_turn
            #if image rotates left (+ang), then steer left
            else: steering = steering + (ang/max_rot*left_turn) #left turn
        else:
            #if the image is not rotated, keep the original image and steering angle
            ang = 0
            img = img
            steering = steering
```

The figures below provide a comparison of the original images and their augmented versions.

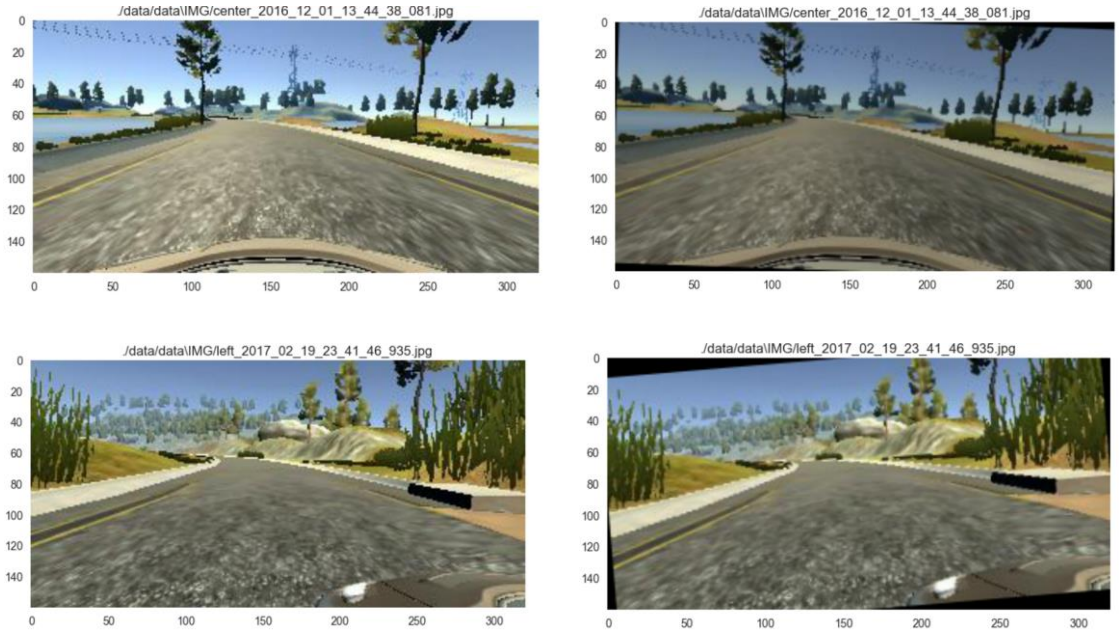


Fig. 3.3. Before (left) and after (right) images showing image rotation and darkening.

After data augmentation, the model was trained with 5 epochs and 8000 data sets per epoch using a generator.

## 4. Conclusion

The final model provided autonomous commands to the car resulting in multiple successful laps around track 1. Data augmentation was crucial to the successful completion of this project. Augmenting the data by rotating the images and altering the

steering angles by a factor related to the image rotation angle provided a robust model. Also, collecting additional data for trouble areas around the track made the model less susceptible to failure.