Traffic Sign Recognition

Hyukpyo Hong February 13, 2017

Behavioral Cloning Project

The goals/steps of this project are the followings

- 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

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 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 a convolution neural network with

- 3 filters, size of 5x5, depth of 24, 36, 48, and stride of 2x2 valid
- 2 filters, size of 2x2, depth of 64, and stride of 1x1 valid

and, linked to fully connected layers. Also, The model includes seven **ELU** layers to introduce nonlinearity.

2. Attempts to reduce overfitting in the model

The model contains five **dropout layers**(rate=0.4) in order to reduce 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, so the learning rate was not tuned manually.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used images of good driving, recovering images from the left and right sides of the road. Those images are recorded by a center, left, and right cameras. And I flip and translated to the right and left to get a more various situation.

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 follow the existing model, and add more layers when necessary.

My first step was to use a convolutional neural network model similar to the Nvidia end-to-end. I thought this model might be appropriate because they already succeed in self-driving with this model.But, since there was no information about what kind of activation layer or dropout layer was used so that I started with a few 'relu' layer.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. Although my first model had shown a low mean squared error on both the training set and validation set, the car of simulator off the road in short time.

To combat the overfitting, I modified the model a lot of time. Actually, MSE factors always show good results but the performance of simulator was not enough. During the test, I realized that relu activation is not so useful in this case because they cannot make negative values, although angle requires negative and positive values.

Then I tested with ELU activation which can produce negative values, and the car of simulator moved more distance. The next problem was the car easily failed on

the curved lane and zigzagged during its driving. So I augmented the imageset for training to make them more focused on the curved lane, and lowered the throttle value, since my laptop is an old model, so it cannot process images quickly which resulted in a zigzag movement.

2. Final Model Architecture

The final model architecture (model.py lines 25-48) consisted of a convolution neural network

Here is a summary of the architecture.

```
carnd@ip-172-31-50-194: ~/SDC-Behavioral-Cloning
Saved model to disk ->'model.h5'

Layer (type)                 Output Shape              Param #     Connected to
====================================================================================
C1 (Convolution2D)           (None, 31, 98, 24)        1824        convolution2d_inpu

batchnormalization_1 (BatchNorma (None, 31, 98, 24)    96          C1[0][0]

elu_1 (ELU)                  multiple                  0           batchnormalization
                                                                   batchnormalization
                                                                   batchnormalization
                                                                   batchnormalization
                                                                   batchnormalization
                                                                   L1[0][0]
                                                                   L2[0][0]

dropout_1 (Dropout)          (None, 31, 98, 24)        0           elu_1[0][0]

C2 (Convolution2D)           (None, 14, 47, 36)        21636       dropout_1[0][0]

batchnormalization_2 (BatchNorma (None, 14, 47, 36)    144         C2[0][0]

dropout_2 (Dropout)          (None, 14, 47, 36)        0           elu_1[1][0]

C3 (Convolution2D)           (None, 5, 22, 48)         43248       dropout_2[0][0]

batchnormalization_3 (BatchNorma (None, 5, 22, 48)     192         C3[0][0]

dropout_3 (Dropout)          (None, 5, 22, 48)         0           elu_1[2][0]

C4 (Convolution2D)           (None, 3, 20, 64)         27712       dropout_3[0][0]

batchnormalization_4 (BatchNorma (None, 3, 20, 64)     256         C4[0][0]

dropout_4 (Dropout)          (None, 3, 20, 64)         0           elu_1[3][0]

C5 (Convolution2D)           (None, 1, 18, 64)         36928       dropout_4[0][0]

batchnormalization_5 (BatchNorma (None, 1, 18, 64)     256         C5[0][0]

dropout_5 (Dropout)          (None, 1, 18, 64)         0           elu_1[4][0]

flatten_1 (Flatten)          (None, 1152)              0           dropout_5[0][0]

L1 (Dense)                   (None, 100)               115300      flatten_1[0][0]

L2 (Dense)                   (None, 50)                5050        elu_1[5][0]

L3 (Dense)                   (None, 10)                510         elu_1[6][0]

L4 (Dense)                   (None, 1)                 11          L3[0][0]
====================================================================================
Total params: 253,163
Trainable params: 252,691
Non-trainable params: 472

None
(carnd-term1) carnd@ip-172-31-50-194:~/SDC-Behavioral-Cloning$
```

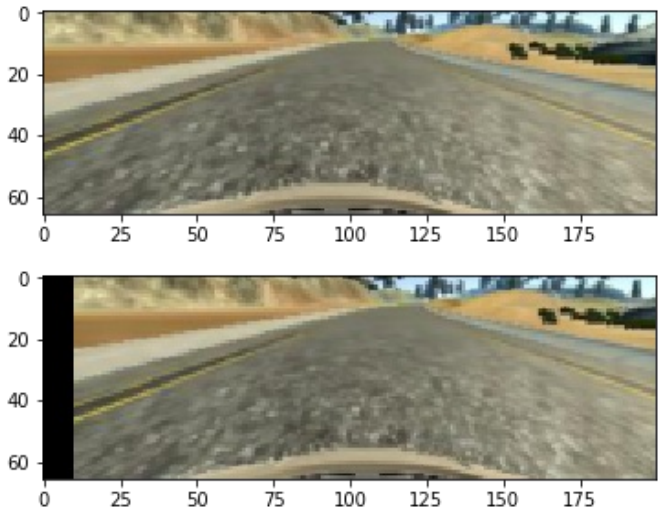
3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded several laps on track one using center lane driving. Here is an example image of center lane driving,

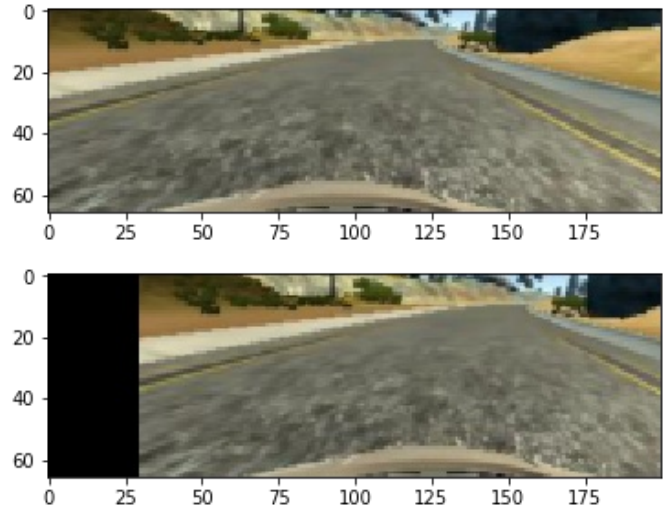
I then recorded the vehicle recovering from the side of the lane to center, so that the vehicle would learn how to recover to the lane. These images show what a recovery recording looks like starting from right to center.

To augment the data set, I translated the image with openCV. If angle value of the image is more than **0.1**, I moved the image **10**px toward the right or left, and add **+-0.15** on original value. Also, if angle value of the image is more than **0.3**, I moved the image **30**px toward the right or left, and add **+-0.35** on original value. Here are sample images

[Left:Original, Angle:0.15 / Right:Move to right 10px, Angle 0.3]

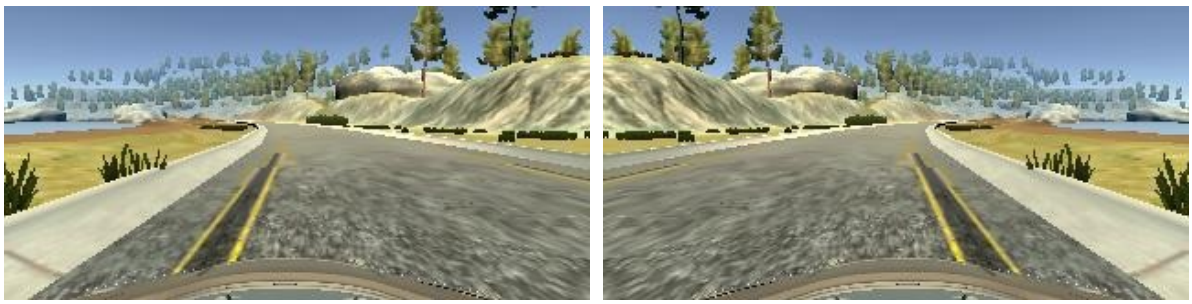


[Left:Original, Angle:0.31 / Right:Move to right 35px, Angle 0.66]

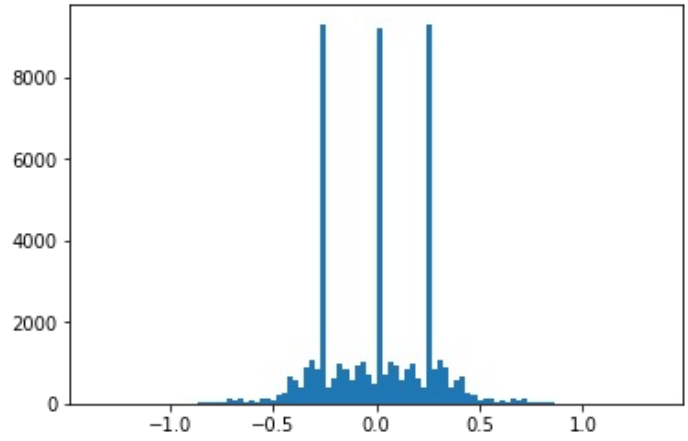


I also flipped images and angles to make my imageset more be balanced. For example, here is an image that is flipped:

[Left:Original, Angle:0.5 / Right:Flipped image, Angle -0.5]



After flip the imageset, the histogram of angle value became to below chart.



After the collection process, I had 52,442 number of data points. When I trained with preprocessed images by mean/std or min/max normalization, my car off the lane easily. So I trained without any image preprocessing, and it worked. Therefore, my model doesn't have image preprocessing stage.

However, I used Keras Batch Normalization layer after each convolutional layers to make my model find proper values early. Before I applied Batch Normalization, 11 epoch was required to get good values. However, after using this normalization, I can reach to the answer after 2 epoch.

I finally randomly shuffled the data set and put 5% of the data into a validation set. After I sure that my model is stable, I trained again with 0.1% of validation set, to improve model accuracy.

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 2 since more than or less than 2 makes driving fail. To know a proper number of the epoch, I set epoch value as 11, and used Keras check point function, to save dataset of each epoch. Also, I used an Adam optimizer so that manually training the learning rate wasn't necessary.