Behavioral Cloning

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
- report.pdf summarizing the results
- data_generator.py for generating training and testing data

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 5x5 and 3x3 filter sizes and depths between 24 and 64 (model.py lines 35-51)

The model includes ELU layers to introduce nonlinearity (code line 39), and the data is normalized in the model using a Keras lambda layer (code line 37). I choose ELU rather than RELU because ELU becomes smooth slowly and it can generate negative output.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 45). Also, data augmentation is applied in data_generator.py file.

The model was trained and validated on different camera image randomly selected from left, center, and right camera 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 (model.py line 54).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used the dataset provided by Udacity and applied data augmentation to make the model better.

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 predict the steering angle given a camera image.

My first step was to use a convolution neural network model introduced by Nvidia. I thought this model might be appropriate because it has enough convolutional layer to learn the feaure of images.

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 so that each fully connected layer is followed by a dropout layer except for the output layer.

Then I applied data augmentation to input dataset, including random flip, random brightness, and random rotation.

The final step was to run the simulator to see how well the car was driving around track one. The vehicle seldom fell off the track, it keep inside the lane in the most of time . However, the vehicle will drive off track when it meets sudden turn or drives to the edge of traffice lane. Hence, for each sample in dataset, I randomly choose left, center, or right camera image as inputs to train the model.

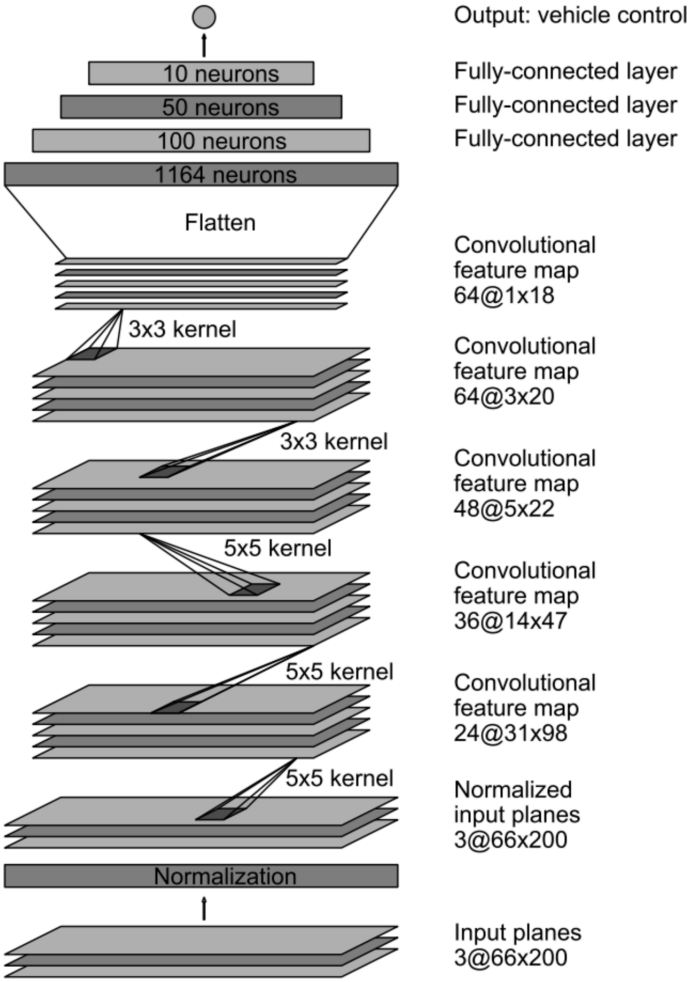
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 35-52) consisted of a convolution neural network with the following layers and layer sizes.

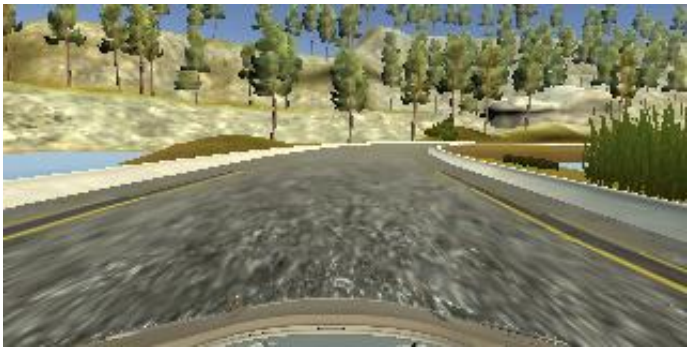
Nvidia Network						
Layer No.	*Layer Type*	*Kernel Size*	*Input Dim*	*Output Dim*	*Input Channels*	*Output Channels*
1	Normalization	-	66x200	66x200	3	3
2	Conv2D	5	66x200	31x98	3	24
3	Conv2D	5	31x98	14x47	24	36
4	Conv2D	5	14x47	5x22	36	48
5	Conv2D	3	5x22	3x20	48	64
6	Conv2D	3	3x20	1x18	64	64
7	Flatten	-	1x18	1164	64	1
8	Fully connected	2	1164	100	1	1
9	Dropout	5	100	100	1	1
10	Fully connected	-	100	50	1	1
11	Dropout	-	50	50	1	1
12	Fully connected	-	50	10	1	1
13	Dropout	-	10	10	1	1
14	Output	-	10	1	1	1

Here is a visualization of the architecture.



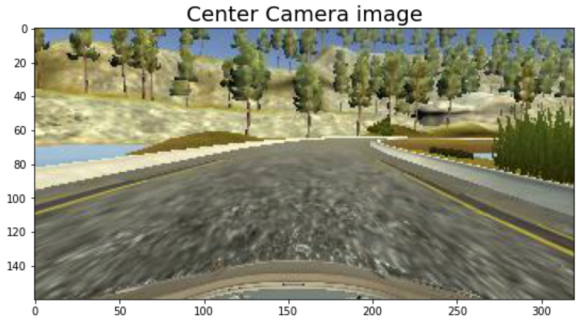
3. Creation of the Training Set & Training Process

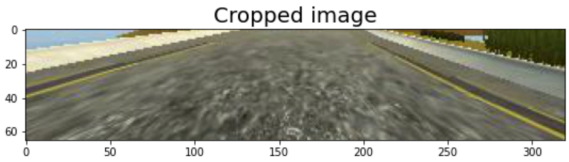
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:



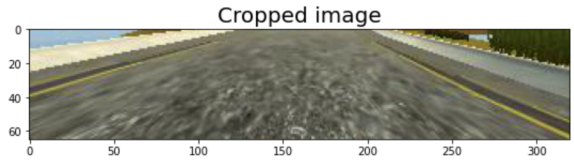
I randomly selected left, right, or center camera image as the input data.

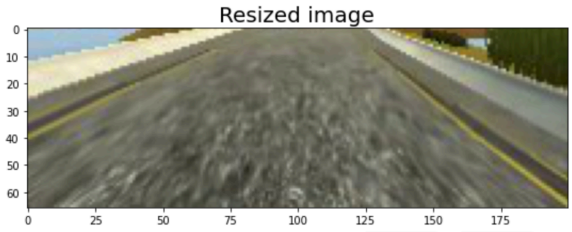
After the collection process, I had 8036 of data points. I then preprocessed this data by crop and resize the image. Because the top of image is sky and tree, the bottom of image is the front of car, I selected center area as region of interest.



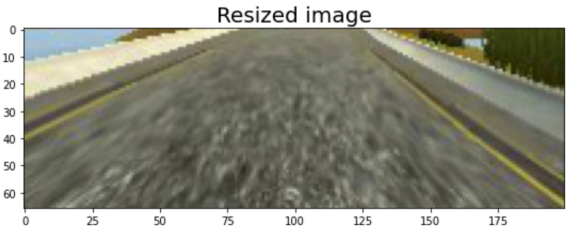


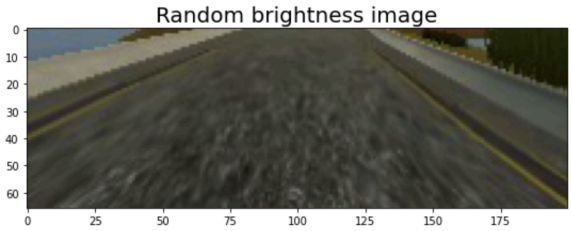
I resized the image to [66,200,3] to fit the input size of Nvidia model.



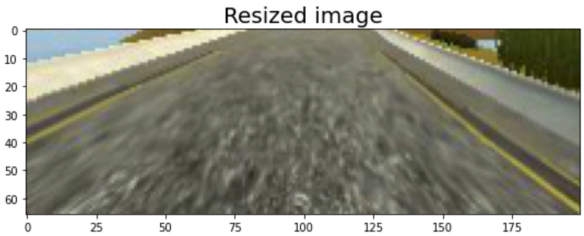


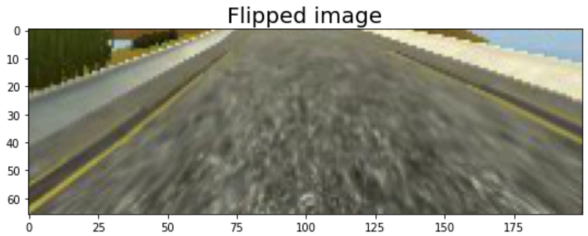
To augment the data sat, I first change the brightness of image randomly. The track1 is under sunny day, but sometimes it is dark, so the model can performe better if we provide input images with various darkness.



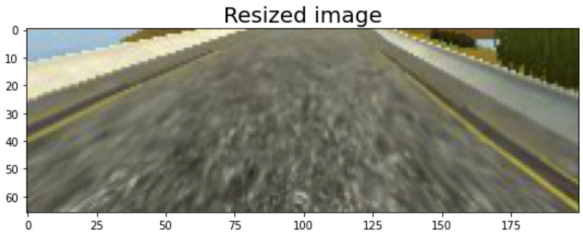


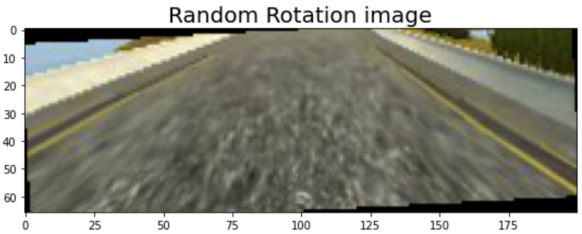
Also, I randomly flip the image to get more data.





Finally, I applied random rotation to the image in order to make the model handle more scenes which are uncommon.





After data augmentation process, I 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. I set the training epoch to 5, because there is no obvious improvement with more epochs. I used an adam optimizer so that manually training the learning rate wasn't necessary.

The final video can be watched here https://www.youtube.com/watch?v=KtJYCwgLia0.