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

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
- writeup_report.pdf # summarizing the results
- video.mp4 # a video when testing the model

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 help_funcs contains some useful functions such as loading and pre-processing data. I didn't use generator in my code because my desktop has 32Gb memory with an 1080ti graphic card.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model for this task was Nvidia Self-Driving car CNN model. It didn't perform well at the first few times, but I believed it was a good model, so I thought I should spend my time adjusting the hyperparameters and training data instead of changing the model.

2. Attempts to reduce overfitting in the model

After adding some random transformation on the original training images, even though the validation set is split from the training set, none images in validation set will not be the same as one in the training set.

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

I also add some new transformed images so that there is more data.

The model contains dropout layers at a keep_prob level of 0.5.

3. Model parameter tuning

The model used an Adam optimizer with a 0.0001 learning rate.

4. Appropriate training data

To make sure the car stays in the center of the track, all three camera images are selected. Also, I randomly did some transformation on all images to avoid overfitting and can performs better when the car sees some roads that it has never seen before, e.g., bad weather, camera get blurred, etc.

The pre-process of the images were cropping and resizing the images. I cut the upper 60 pixels and bottom 25 pixels off, and then resized the image into (60,200,3), which is the input size that Nvidia CNN model wants.

Model Architecture and Training Strategy

1. Solution Design Approach

My first step was to use a convolution neural network model from NVIDIA. I thought this model might be appropriate because NVIDIA has tested it on a real self-driving car.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set by a scale of 0.15. The model works just terrible at first even though both the validation and training error was low. It was obviously an overfitting. Thus I considered make up some new images and added the dropout layer before the flatten.

At the end of the process, the vehicle can drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture consisted of a convolution neural network is same as the NVIDIA model except I added one more drop out layer.

Here is a visualization of the architecture

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 66, 200, 3)	0	lambda_input_1[0][0]
convolution2d_1 (Convolution2D)	(None, 31, 98, 24)	1824	lambda_1[0][0]
convolution2d_2 (Convolution2D)	(None, 14, 47, 36)	21636	convolution2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 5, 22, 48)	43248	convolution2d_2[0][0]
convolution2d_4 (Convolution2D)	(None, 3, 20, 64)	27712	convolution2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 1, 18, 64)	36928	convolution2d_4[0][0]
dropout_1 (Dropout)	(None, 1, 18, 64)	0	convolution2d_5[0][0]
flatten_1 (Flatten)	(None, 1152)	0	dropout_1[0][0]
dense_1 (Dense)	(None, 100)	115300	flatten_1[0][0]
dense_2 (Dense)	(None, 50)	5050	dense_1[0][0]
dense_3 (Dense)	(None, 10)	510	dense_2[0][0]
dense_4 (Dense)	(None, 1)	11	dense_3[0][0]

Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0

3. Creation of the Training Set & Training Process

I used the data set provide by Udacity. It contains 8036*3 = 24108 (3 cameras)

Here is one same scene from 3 cameras views.

The left and right views of the scene can help the car get back to the center of the road. However, we will have to adjust the steering angle before we train our model. For left and right views, I gave a offset steering angle of ± 0.20

I also wrote a function called *random_trans(images, steer)* to generate more training data. The idea was when taking an image, there's a 50% chance to flip the image. After flipping/not flipping the image, there is a 20% chance for the following transformation:

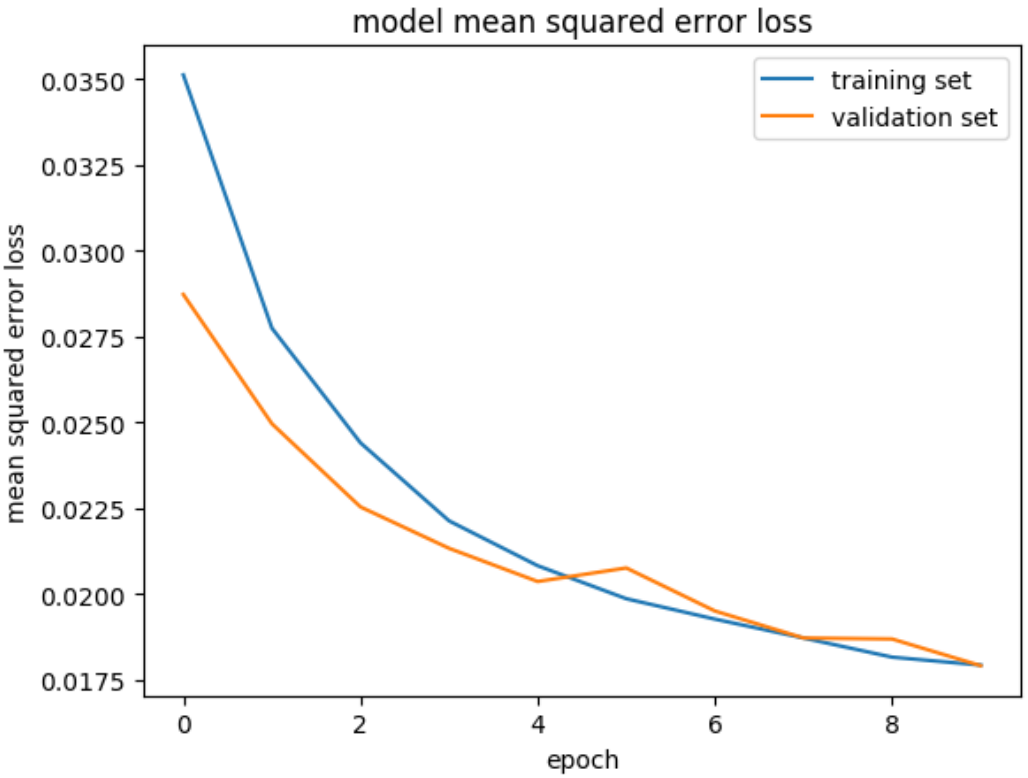
1. Do nothing
2. Adding some noise/gamma
3. Add some brightness
4. Shift horizontally and/or vertically
5. Dark part of the image

Each transformation will adjust the steering angle a little bit, and it will return a new sample of training image and steering angle.

Before generating new images, I first cropped and resized the images. I cut the upper 60 pixels and bottom 25 pixels off, and then resized the image into (60,200,3), which is the input size that Nvidia CNN model wants.

I generate each image twice, so the data set became 3 times bigger (original + 2 new samples). There was a total 61467 images in my training set, and 10848 images in my validation set. (total 72315)

I trained the model 10 epochs with a batch size of 100. Here is the training result for each epoch.



Other Comments:

1. The opencv library will read the image in BGR way, so in help_funcs.py, I wrote a function *read_rgb(file_name)* to read an image in RGB way.
2. I also test the model in the backward way, and it runs perfectly as well.
3. I spent over 10 hours on just a little bug: I copied the code from "left images selecting" to "right image selecting" without changing the index, so the model trained on same left images twice with different steering angles. ☹ But it works now! ☺
4. Here is an YouTube link for better quality. https://youtu.be/huxjtJJs1Dw