Behavioral Cloning

Longxing Tan

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
- 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 5 x 5 and 3x3 filter sizes and depths between 24 and 64 (model.py lines 128-146)

The model includes ELU layers to introduce nonlinearity (code line 131-135), and the data is normalized in the model using a Keras lambda layer (code line 130).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 140).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 27). 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 145).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road with correctness coefficient 0.213.

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 simulate the steering angle according to the pictures from camera.

My first step was to use a convolution neural network model similar to the Nvidia model. I thought this model might be appropriate because the training error and testing error is in an acceptable domain, and the car could drive successfully in autonomous mode.

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 the 'mes' error of training and testing data is both all right.

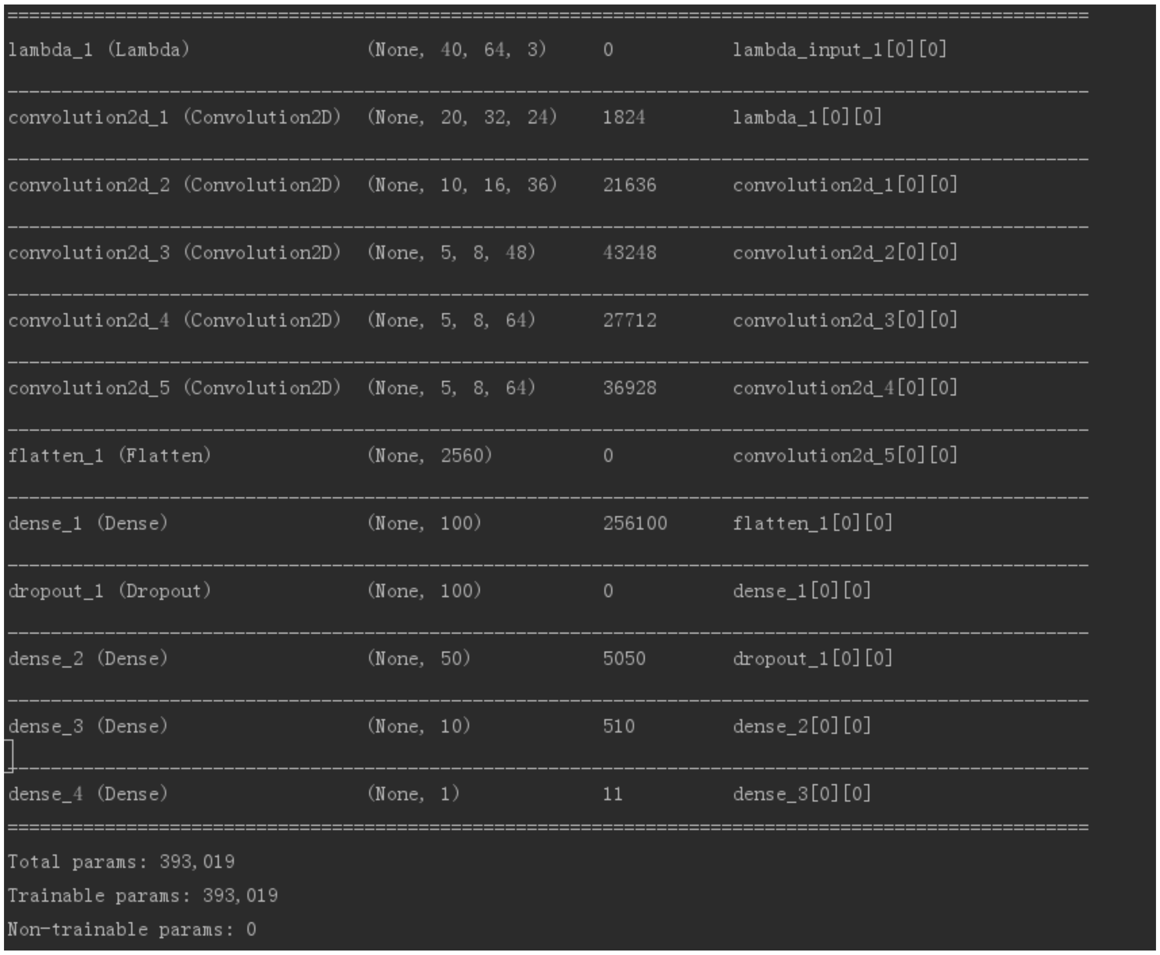
Then I argument the data to improve the regularization. Because all the training data is only anti-clock-wise, so the flip is necessary, and also other argument method, like the random_brightness, random_shear.

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track... to improve the driving behavior in these cases, I adjust the argument methods and the left/right lane to make the vehicle kept in the track.

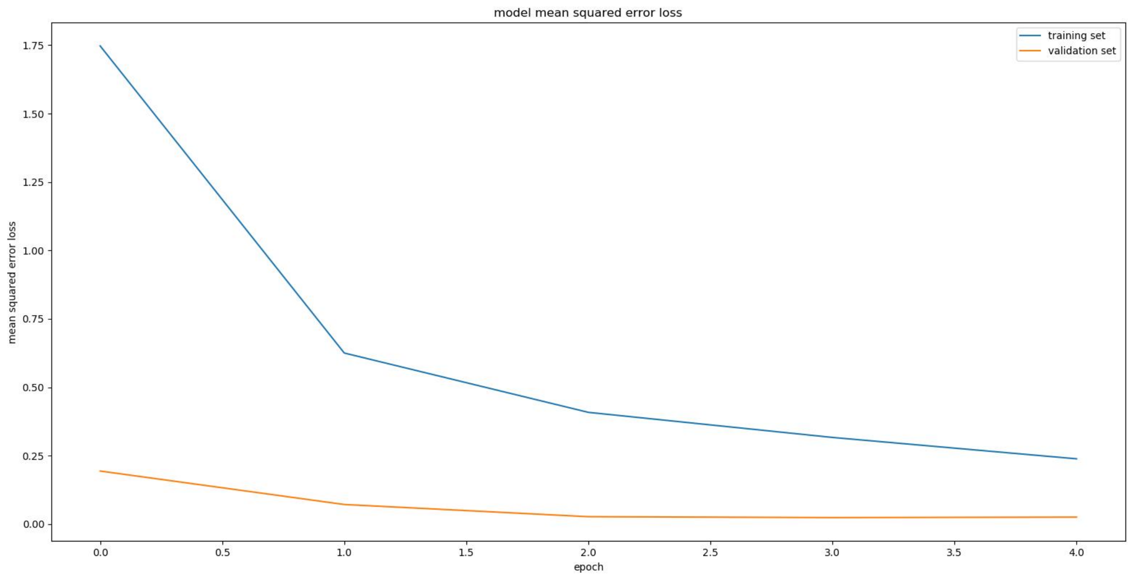
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 128-146) consisted of a convolution neural network with the following layers and layer sizes ...

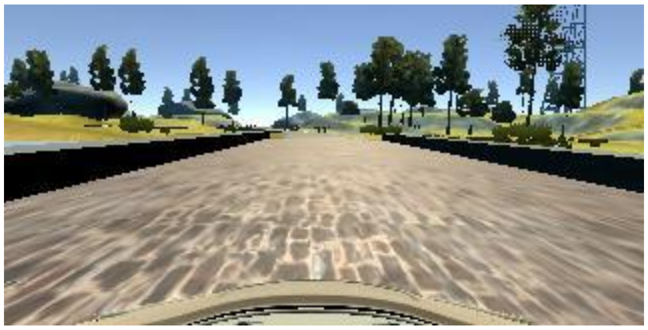


```
====================================================================================================
lambda_1 (Lambda)                (None, 40, 64, 3)     0           lambda_input_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 20, 32, 24)    1824        lambda_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 10, 16, 36)    21636       convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 5, 8, 48)      43248       convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 5, 8, 64)      27712       convolution2d_3[0][0]
____________________________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 5, 8, 64)      36928       convolution2d_4[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 2560)          0           convolution2d_5[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 100)           256100      flatten_1[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 100)           0           dense_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        dropout_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: 393,019
Trainable params: 393,019
Non-trainable params: 0
```

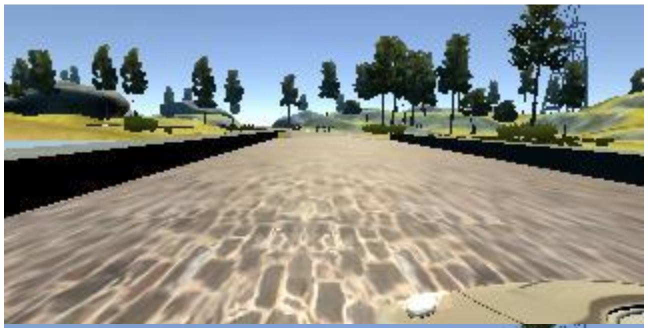
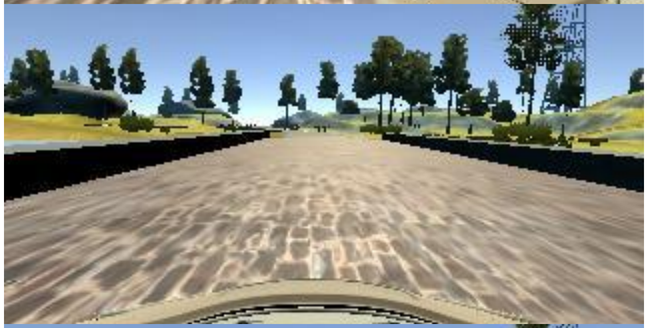
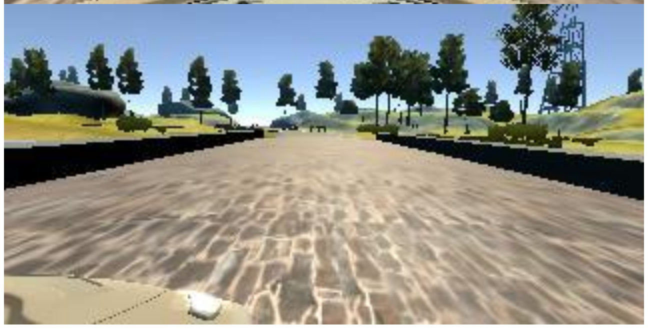


3. Creation of the Training Set & Training Process

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:

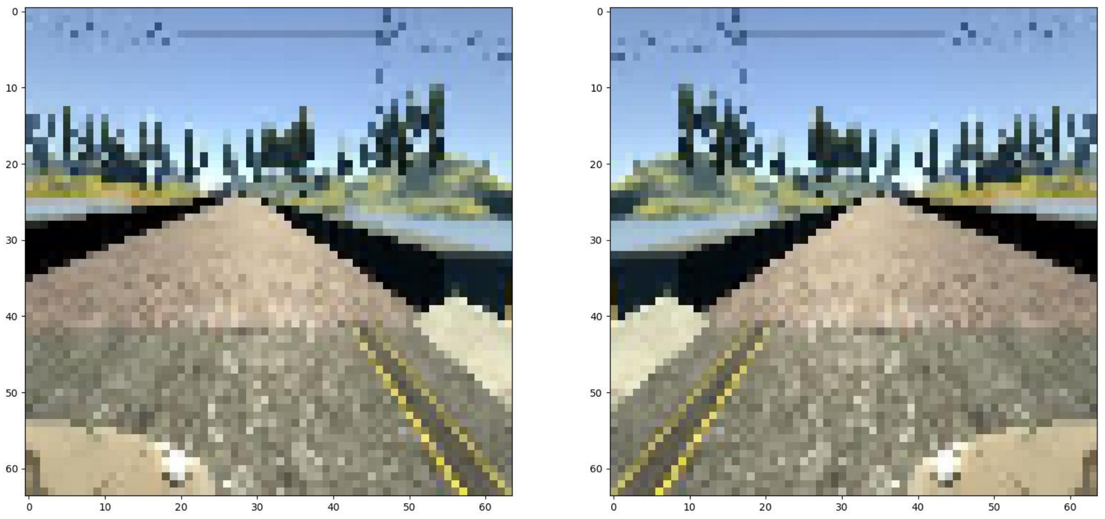


I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to .... These images show what a recovery looks like starting from ... :

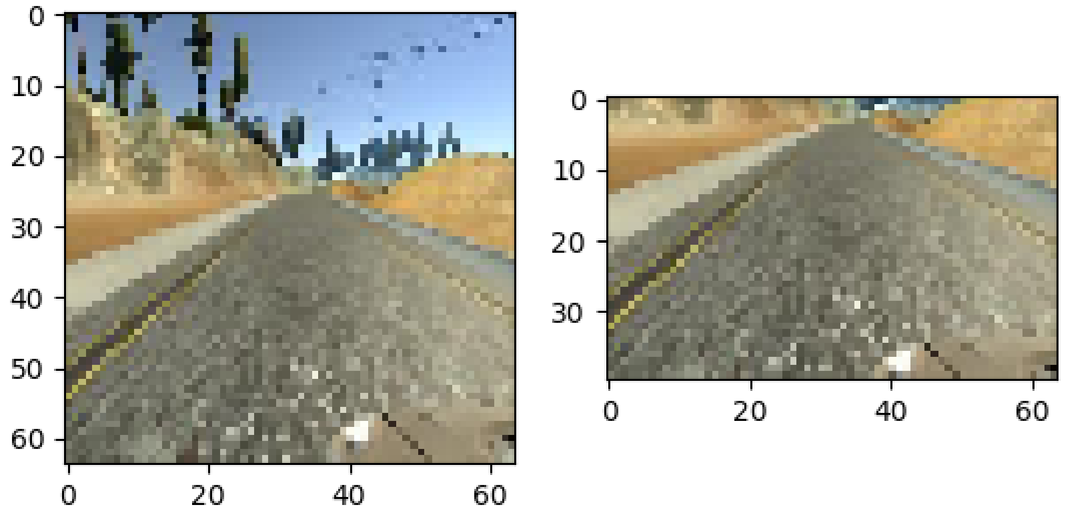




Then I repeated this process on track two in order to get more data points.

To augment the data sat, I also flipped images and angles thinking that this would ... For example, here is an image that has then been flipped:



The crop example:



After the collection process, I had (40,64,3) number of data points. I then preprocessed this data by normalization x/127.5-1

I finally 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. The ideal number of epochs was 5. I used an adam optimizer so that manually training the learning rate wasn't necessary.