Behavioral Cloning

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.md or writeup_report.pdf summarizing the results
- utils.py include some image processing function to preprocessing and the image dataset

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 3x3 & 5x5 filter sizes and depths between 24, 36, 48 and 64 (model.py lines 34-38)

The model includes ELU layers to introduce nonlinearity (code line 34-38), and the data is normalized in the model using a Keras lambda layer (code line 30).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 39).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 21). 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 84).

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 .

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to refer the Nivida's paper.

My first step was to use a convolution neural network model similar to the nivida self-driving model I thought this model might be appropriate because there is paper to proof it useful to train a cloning model to drive a 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. 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 batch normalization and dropout layer.

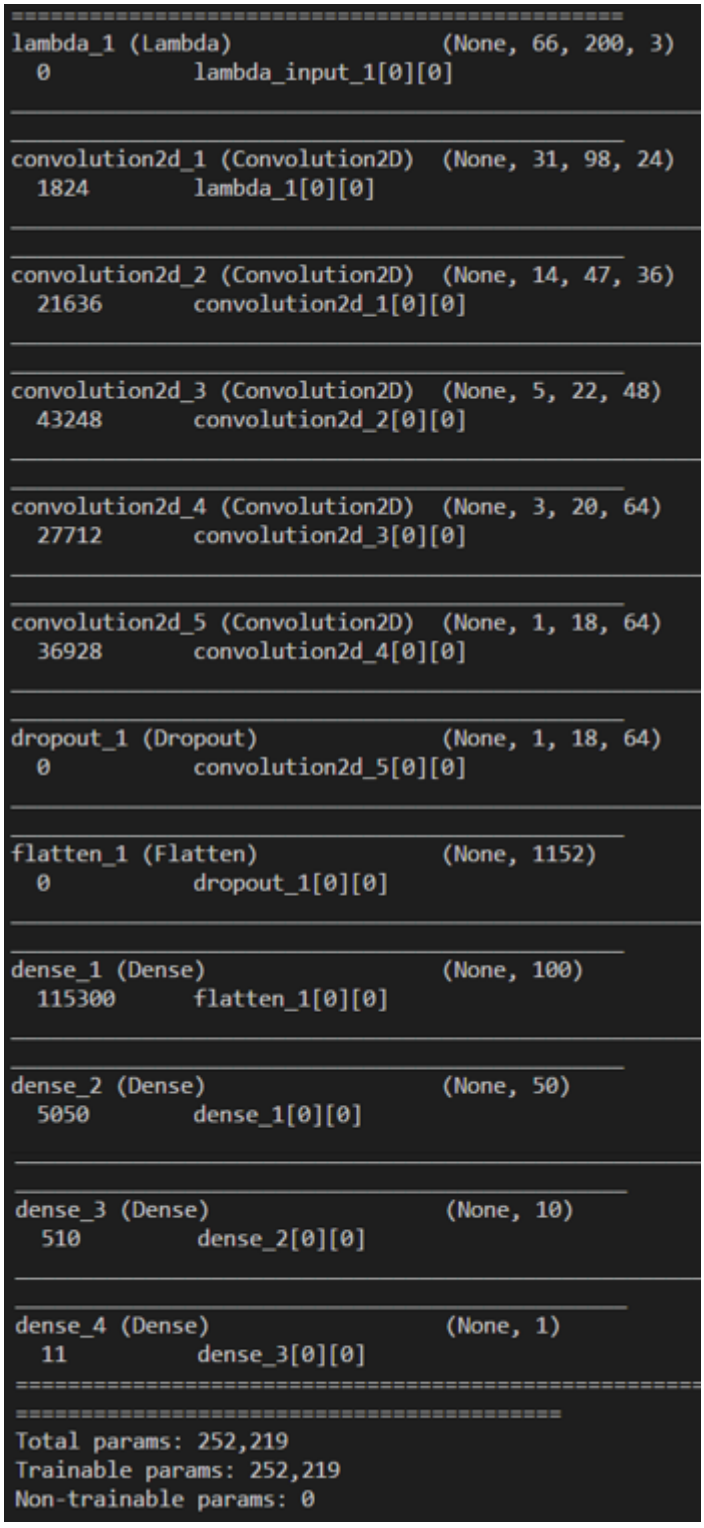
Then I train the model and plot the loss curve of each epoch again, and get smaller loss.

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 add some dataset when the large car steering angle and car is overlapping the road line.

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

2. Final Model Architecture

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)



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 more image data.

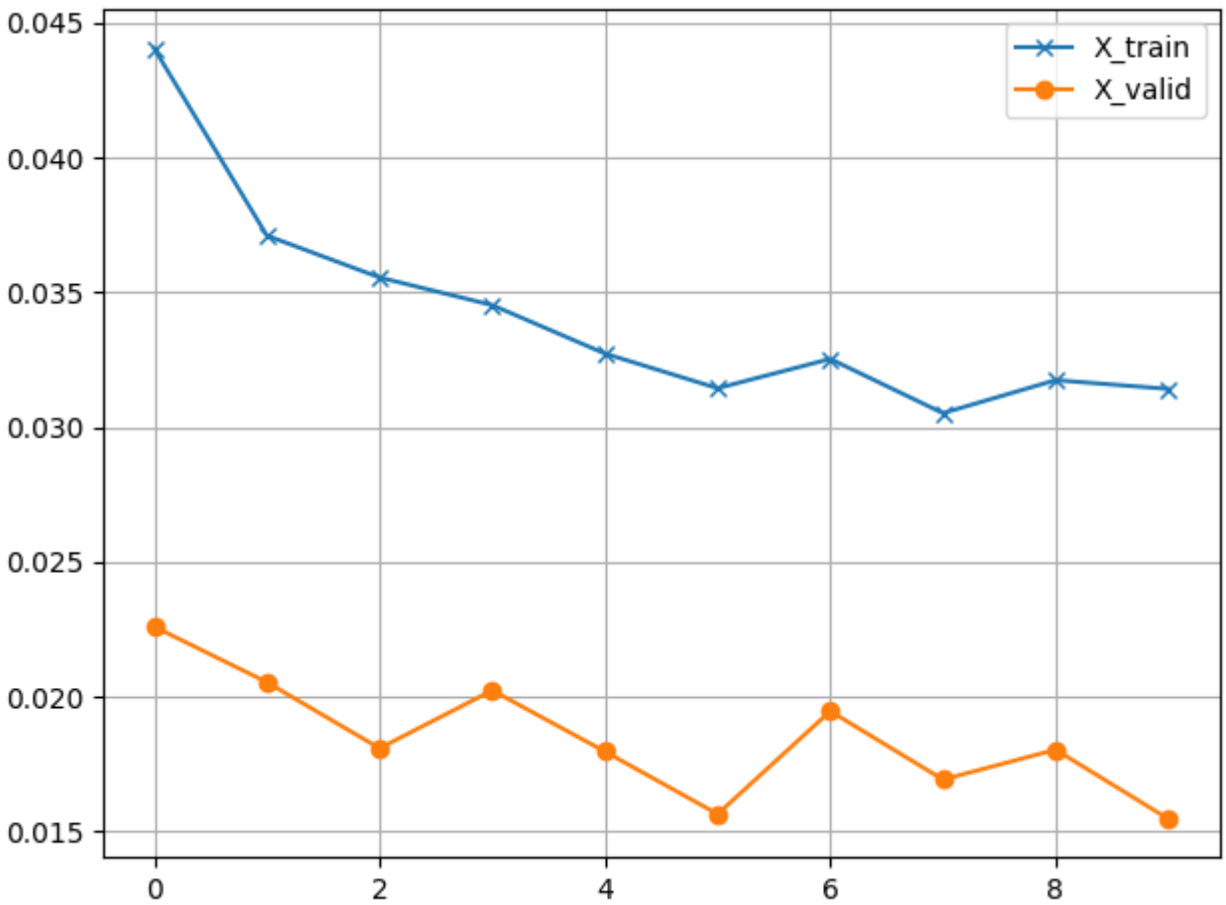
Then I repeated this process on track two in order to get more data points.

To augment the data sat, I also flipped images and angles thinking that this would make more data.

After the collection process, I had X number of data points. I then preprocessed this data by resize, crop and make rgb to yuv.

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 10 as evidenced by the loss curve.



I used an adam optimizer so that manually training the learning rate wasn't necessary.