Behavioral Cloning

Writeup Template

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 the model * train.py containing the script to train the model * generator.py defined generator to feed processed data to model * preprocess.py for preprocessing image data to the generator * drive.py for driving the car in autonomous mode * model.h5 containing a trained convolution neural network * writeup_report.md summarizing the results * para.py external parameters for development(please ignore this file)

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 train.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.py contains the convolution neural network model structure definition. generator.py defines the data process pipline and preprocess.py contains data preprocessing functions.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model is a simplified version of the Nvidia Model, courtesy to Alexander Lavin.

It consists of a convolution neural network with 8x8, 5x5 filter sizes and depths between 16 and 64 (model.py lines 31-37). For better accuracy and smooth transition, the model includes ELU layers to introduce nonlinearity (model.py line 32, 35, 42, 47), and the data is normalized in the model using a Keras lambda layer (model.py line 23).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 41, 46).

The model was trained and validated on different data sets to ensure that the model was not overfitting (train.py line 51). 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 53).

4. Training data

For training data, Udacity's Sampe training data is used, which can be retirieved from here. In order to help the vehicle to keep driving on the road, images from left and right cameras with adjusted steering angle are used to augment the training data.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to abstract road information for model to predict steering angle.

My first step was to use a convolution neural network model similar to the Nvidia model. I thought this model might be appropriate because it is a published method with proven records. From Alexander Lavin's github repository I implemented his simplified version of the end to end self driving model.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. The model works well out of the box, no overfitting issues were observed.

Since the traning data is too large to fit into memory, two generators are implemented to generate training and validation data on the fly.

Furthermore, to help my model generalize better, images are preprocessed and augmented for the training generator, these include crop, translate, flip, augment brightness and casting random shadows. For images from side cameras, an empirical +-0.25 steering angle value is add on top of left and right training labels respectively, this helps the model to turn back on the center of the track in case it deviates from ideal path. All images have been slightly translated before being fed into the generator, their steering angle adjustment is also adjusted according to $\alpha = \alpha + 0.4\Delta x/\Delta X$ (preprocess.py line 44-45), in which α is the recorded steering angle, Δx is the number of horizontal shifting pixels and ΔX is the maximum horizontal shifting range, in this project it is a constant value of 150.

Training is conducted with Adam optimizer minimizing mean squared error. Resample rate is set to 3, meaning for every 1 training picture, 3 syntheic images based on the original image are created for the model to learn. After 1000 training epochs, my model is then tested on the track.

The final step was to run the simulator to see how well the car was driving around track one. 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

3. Creation of the Training Set & Training Process

The Udacity sample training set includes 2-3 laps on track one using center lane driving. Here is an example image of center camera:

Using side cameras with estimated steering angle helps the vehicle learn to adjust steering if itself deviates from the ideal track. An example of recordings of the 3 cameras is shown below, images from left, center, right camera respectively.

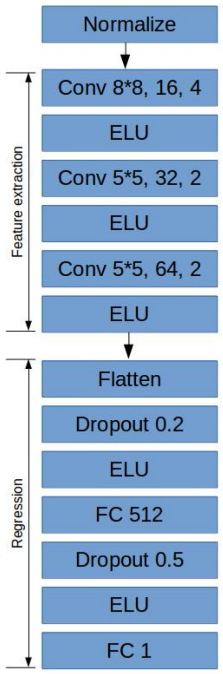


Figure 1: alt text

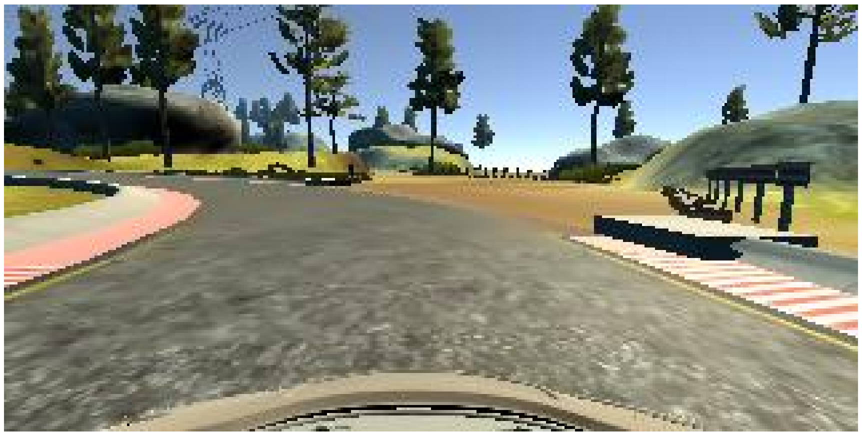
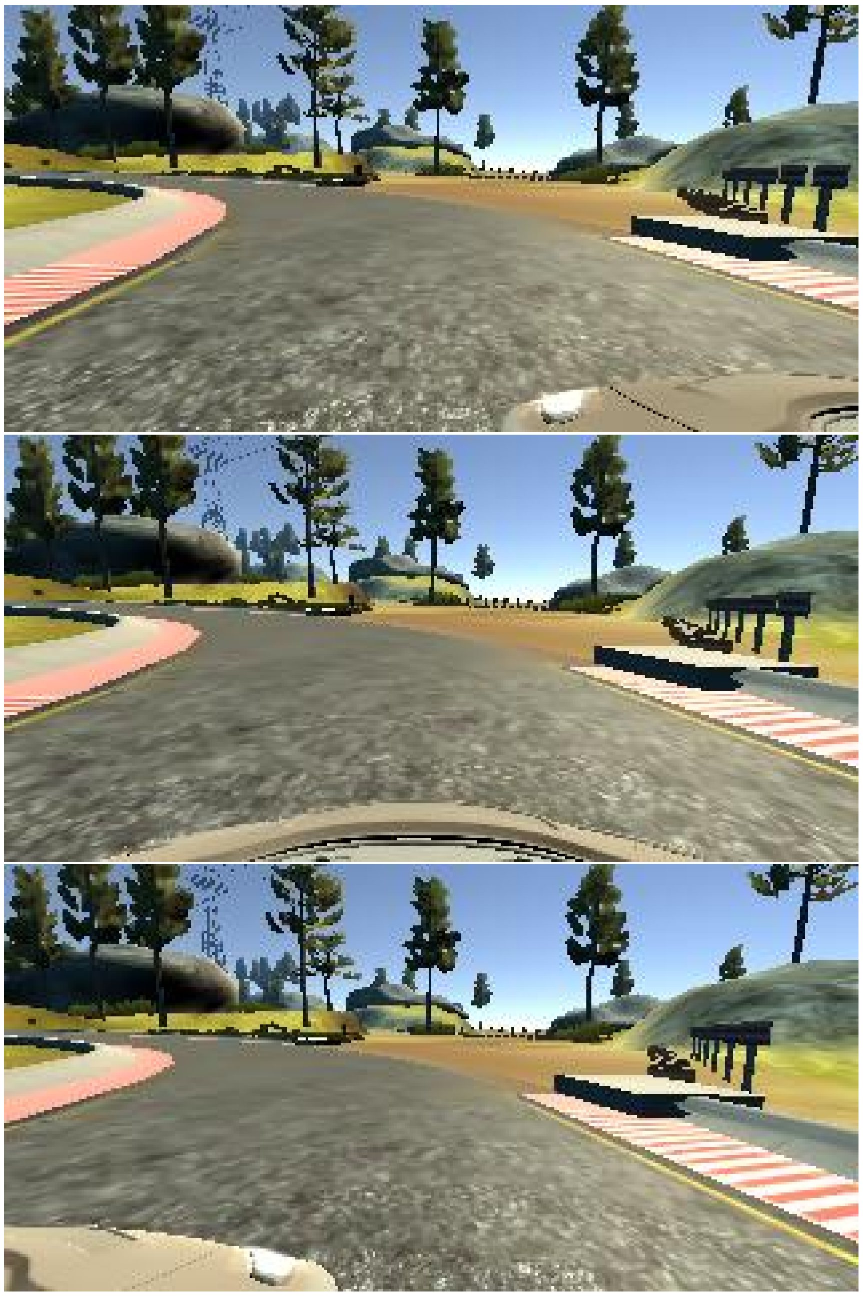
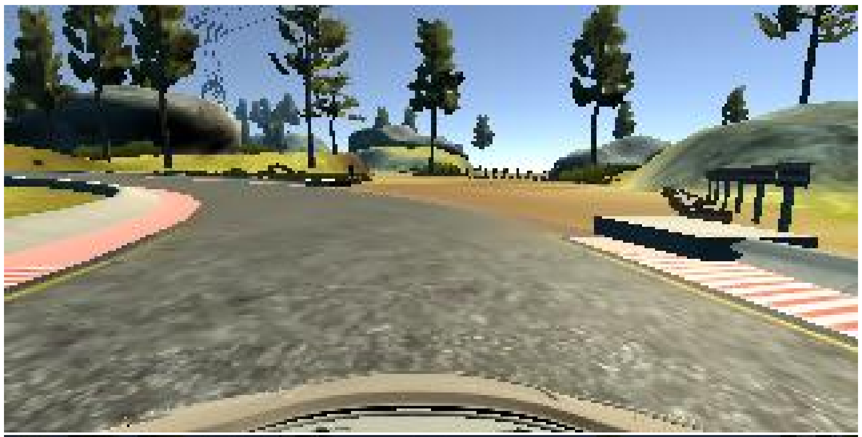


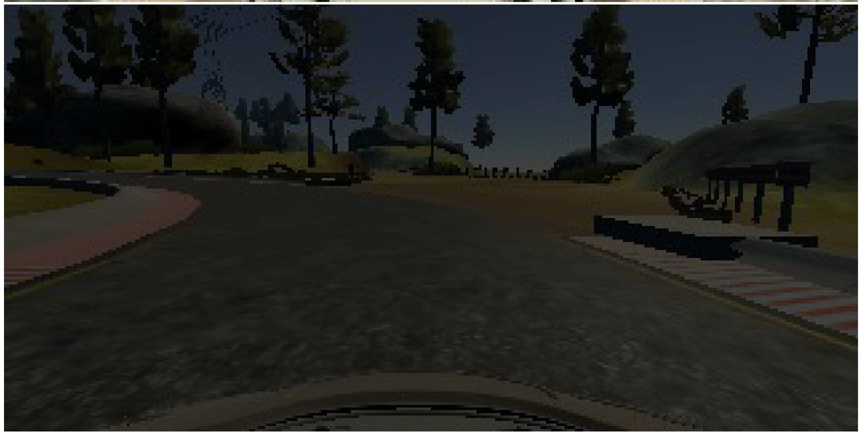
Figure 2: alt text



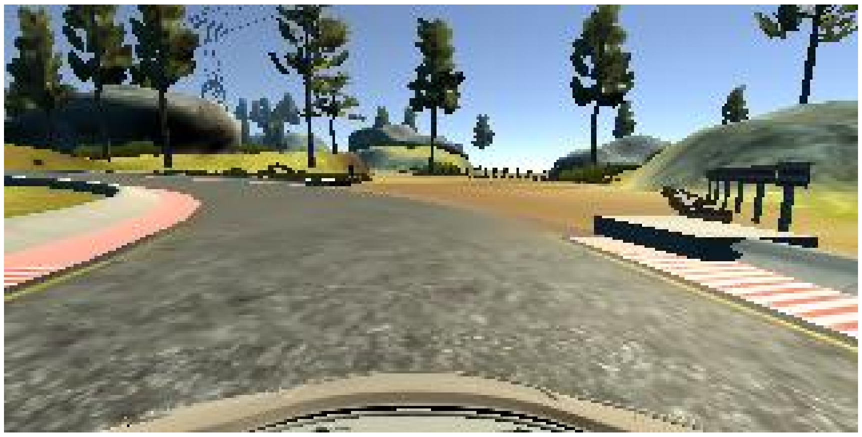
To augment the data set, multiple techniques have been implement. The following sample images illustrate the effects on training data.

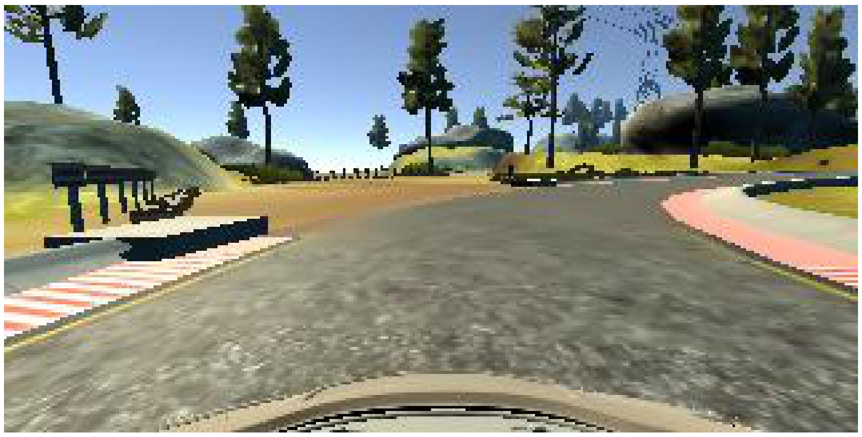
- Augment brightness



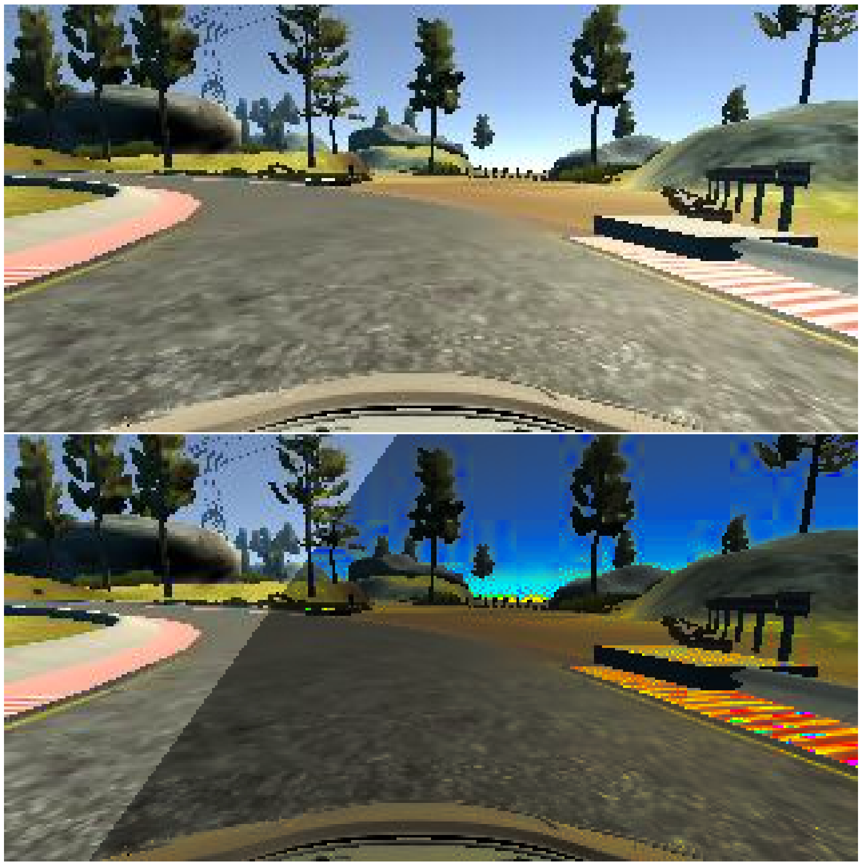


- Flip

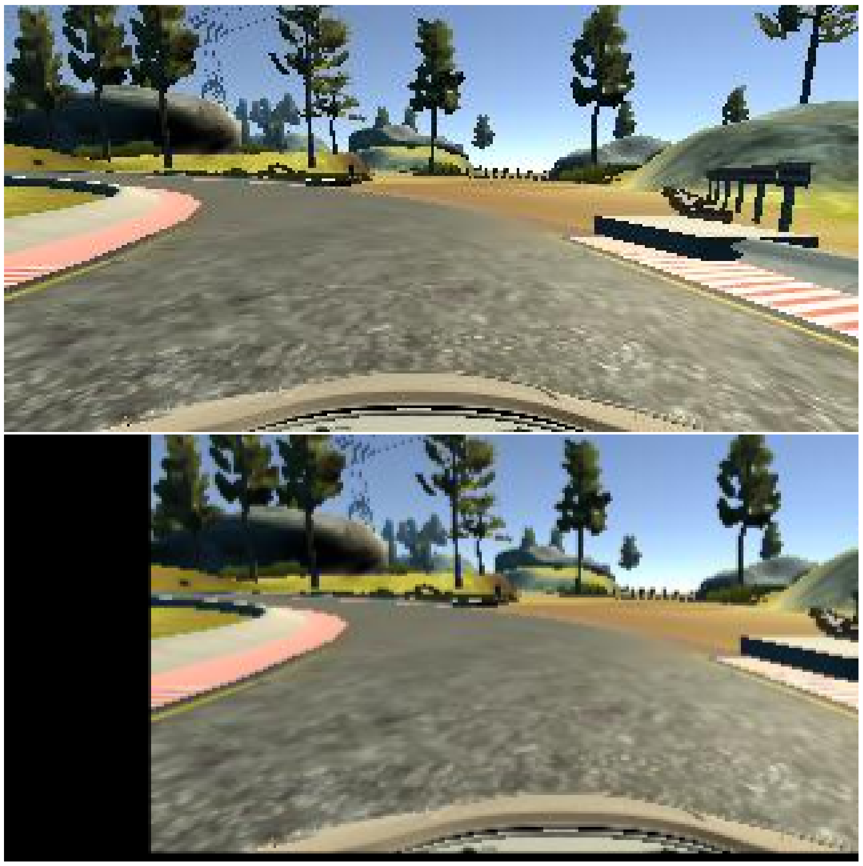




- Random shadow



- Random translate



From the sample dataset, I had 8036 number of data points. These data points were then randomly shuffled and put 25% of the data into a validation set, the validation set helped determine if the model was over or under fitting.. I then preprocessed this data by center cropping the image into 200 * 66, then fed into the network.

During training, every training image is augmented into 3 syntheic images to help model to generalize. The model was trained for 100 epochs and retained the weights with least validation loss. Adam optimizer is used so that manually training the learning rate wasn't necessary.

Model Test

The trained model is tested on track 1, it has demonstated stable performance on both clock-wise and conter-clock-wise driving with target speed of 30mph. The vehicle is able to recover to desired path even manual disruptions were introduced. Video files illustrate auto steering are provided in this repository, video.mp4 recorded during counter-clock-wise driving, video_reverse.mp4 recorded during clock-wise driving, and disrup.mp4 demonstrates recovery from disruptions.