Behavioral Cloning Writeup

Matt Kontz

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

1. Are all required files submitted?

The submission includes a model.py file, drive.py, model.h5 a writeup_report.md/pdf and video.mp4.

My project includes the following files:

- model.py containing the script to create and train the model
  - model.py calls loadData.py which loads the data to create left, center and right image/measurement pairs and also mirrors each pair to remove left/right bias.
- drive.py for driving the car in autonomous mode which is unmodified
- model.h5 containing a trained convolution neural network created by model.py
- writeup_report.md and writeup_report.pdf that summarizing the results

The included files and can be used to run the simulator in autonomous mode.

Quality of Code

1. Is the code functional?

The model provided can be used to successfully operate the simulation.

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`

2. Is the code usable and readable?

The code in model.py uses a Python generator, if needed, to generate data for training rather than storing the training data in memory. The model.py code is clearly organized and comments are included where needed.

A python generator was not need even with the large dataset when running the large dataset that include the sample dataset provided because the CNN was trained in GPS mode.

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.

The loadData.py files is called by model.py and loads and prepares the data for the training pipeline in model.py.

Model Architecture and Training Strategy

1. Has an appropriate model architecture been employed for the task?

The neural network uses convolution layers with appropriate filter sizes. Layers exist to introduce nonlinearity into the model. The data is normalized in the model.

The model is a modified version of CNN presented in the class lecturers that originated from Nvidia. The version I implemented

- normalizes the data using a Lambda function
- crops the image
- Convolution layers 1: (lines 26 - 29)
  - 14 5x5 convolution kernels
  - 2x2 max pooling
  - 0.5 dropout
  - relu activation
- Convolution layers 2:
  - 36 5x5 convolution kernels
  - 2x2 max pooling
  - 0.5 dropout
  - relu activation
- Convolution layers 3:
  - 48 5x5 convolution kernels
  - 2x2 max pooling
  - 0.5 dropout
  - relu activation
- Convolution layers 4:
  - 64 3x3 convolution kernels
  - 0.5 dropout
  - relu activation
- Convolution layers 5:
  - 64 3x3 convolution kernels
  - 0.5 dropout
  - relu activation
- fully connected layers (lines 54 - 58)
  - size 100
  - size 50
  - size 10
  - size 1

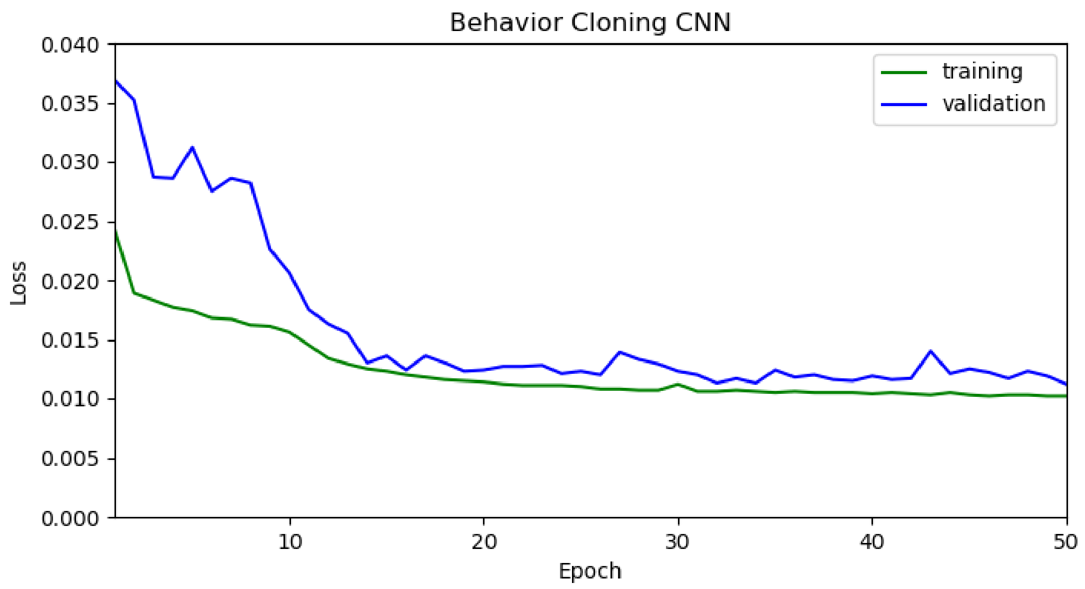
2. Has an attempt been made to reduce overfitting of the model?

Train/validation/test splits have been used, and the model uses dropout layers or other methods to reduce overfitting.

To reduce overfitting

- All 5 convolution layers are followed by a dropout layer.
- The first 3 convolutions layers also followed by a max pooling layer which can also reduce overfitting.
- The additional sample data was also added to the data I collected to increase the size of the dataset which also helps prevent overfitting.

The training plot below shows that overfitting minimized based on the difference between the training and validation loss from Epoch 15-50.



The model was trained (80%) and validated (20%) on different data sets to ensure that the model was not overfitting (code line 67). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Have the model parameters been tuned appropriately?

Learning rate parameters are chosen with explanation, or an Adam optimizer is used.

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 64).

4. Is the training data chosen appropriately?

Training data has been chosen to induce the desired behavior in the simulation (i.e. keeping the car on the track).

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. A steering correction of 0.2 and -0.2 was added steering angle for the left and right images respectively.

For details about how I created the training data, see the lines 60-60 in loadData.py. The function mirror*left*right() (lines 19-22 i n loadData.py) is used to mirror and duplicate all the images to eliminate left-right bias in the training. For all the mirror images, the steering angle is negated.

Architecture and Training Documentation

1. Is the solution design documented?

The README thoroughly discusses the approach taken for deriving and designing a model architecture fit for solving the given problem.

Approach to solving problem:

- Collect data by driving around the track multiple times.
- Prepare the data as described in the previous section.
    - Split the data into left, center right adding steering correction to left and right images
    - mirror all image steering measurement pair to eliminate left-right bias.
- Implement CNN based on LeNet architecture
    - The car didn't go very far before driving off the road
- Implemented CNN architecture from NVIDIA developed for behavioral cloning
    - This model has more CNN layers and more fully connected layers
    - Dropout was added to reduce overfitting
    - The car made it to the bridge before driving into the side of the bridge.
    - The steering seems to be to weak and would not steer aggressively enought towards the center of the lane.
- Added the sample data set to my training data
    - The sample data set includes almost 50,000 set of images and I was collecting only about 60 sets every time around the track -- **MORE**

DATA = GOOD!!
    - The the steering about was also increase from 0.2 to 0.5 to see if steering correction for the side images were not aggressive enough
    - **The result**: the car made it around the track without going over the edge, but the steering was really jerky and always seemed to over correct which led me to conclude that a steering correction of 0.2 was better than 0.5.
- Update steering correction (0.5 -> 0.2)
    - steering correction was reduced from 0.5 to 0.2
    - stride in the first three 5x5 convolution kernels were set to 1
    - 2x2 pooling layers replace the reduce in stride keeping the network close to the same size
    - **The result**: the car drove easily around the track as can be seen in **video.mp4**

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

2. Is the model architecture documented?

The README provides sufficient details of the characteristics and qualities of the architecture, such as the type of model used, the number of layers, the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged. Here is one such tool for visualization.

See section above Model Architecture (1st section in Model Architecture and Training Strategy).

3. Is the creation of the training dataset and training process documented?

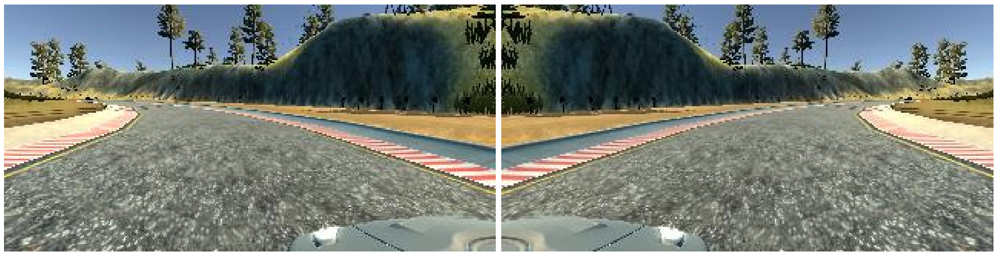
The README describes how the model was trained and what the characteristics of the dataset are. Information such as how the dataset was generated and examples of images from the dataset must be included.

See section above Training Strategy (1st section in Model Architecture and Training Strategy).

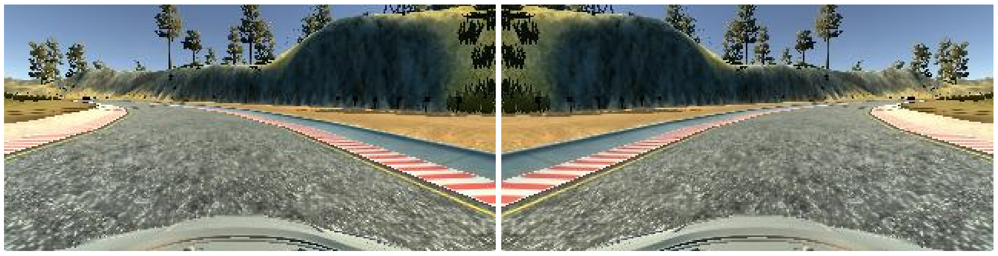
Example set of training images capture by driving around the track are shown in the following in the following 3x2 grid of images.

- The images on the left column are the original images
- The images on the right are what the original images would look like after they are mirrored in loadData.py.
- Notice that the front of the car moves in the image a lot compared to the trees in the background

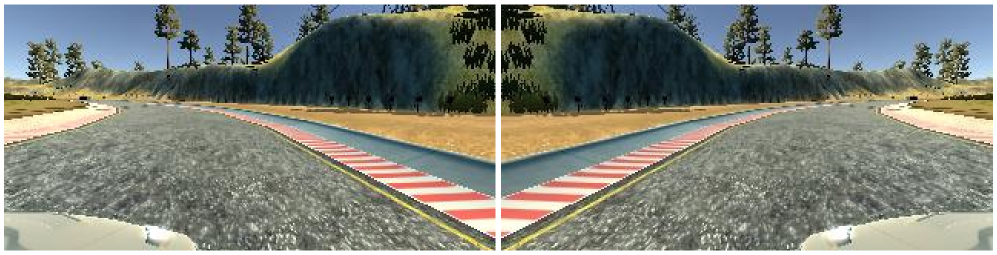
Left, left mirror



Center, center mirror



Right, right mirror



Simulation

1. Is the car able to navigate correctly on test data?

No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road as can be seen in **video.mp4**.