Behavioral Cloning

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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:

- **behavioral_cloning.ipynb** containing the whole procedure training and playback video
- **drive.py** for driving the car in autonomous mode
- **Nvidia_model-0.008571.h5** containing a trained convolution neural network
- **run_result.mp4** front camera view for auto pilot
- **behavioral_cloning.pdf** it is a readable copy file relative to behavioral_cloning.ipynb
- **writeup_report.md** or 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

```
CUDA_VISIBLE_DEVICES=1 python3 drive.py Nvidia_model-0.008571.h5 run_result
```

3. Submission code is usable and readable

The behavioral_cloning.ipynb 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.

Main Procedure

- In this project, I used a deep neural network (built with Keras) to clone car driving behavior.
- The dataset used to sample data, which nums are 8036*3.
- Sample data consists of images taken from three different camera angles (Center - Left - Right), in addition to the steering angle, throttle, brake, and speed during each frame.
- The network is based on NVIDIA's paper End to End Learning for Self-Driving Cars, which has been proven to work in this problem domain.

There are several steps, more details are shown in behavioral_cloning.pdf

- **Data Loading.**
- **Data Augmentation.**
- **Data Preprocessing.**
- **Model Architecture.**
- **Model Training and Evaluation.**
- **Model Testing on the simulator.**

About ENV setting

- Ubuntu 16.04
- CUDA 9.0
- CuDNN 7.0.5
- Python 3.5
- Keras 1.2.1
- TensorFlow 1.10

For data loading, i use dataset from here. This dataset contains more than 8,000 frame images taken from the 3 cameras (3 images for each frame), in addition to a `csv` file with the steering angle, throttle, brake, and speed during each frame. For data set, i use function of `train_test_split` and test_size is 0.2 to spliting train set and valid set.

For data preprocessing

- Cropping the image to cut off the sky scene and the car front.
- Resizing the image to (66 * 200), the image size that the selected model expects.
- Converting the image to the YUV color space that paper required.

For data augmentation

- Adjusting the steering angle of random images.
- Flipping random images horizontaly, with steering angle adjustment.
- Shifting (Translating) random images, with steering angle adjustment.
- Adding shadows to random images.
- Altering the brightness of random images.

For model architecture In this step, we will design and implement a deep learning model that can clone the vehicle's behavior. We'll use a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. We'll use the ConvNet from NVIDIA's paper End to End Learning for Self-Driving Cars, which has been proven to work in this problem domain. According to the paper: "Network Architecture We train the weights of our network to minimize the mean squared error between the steering command output by the network and the command of either the human driver, or the adjusted steering command for off-center and rotated images. Our network architecture is shown in Figure 4. The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network. The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing. The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. We use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers. We follow the five convolutional layers with three fully connected layers leading to an output control value which is the inverse turning radius. The fully connected layers are designed to function as a controller for steering, but we note that by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractor and which serve as controller."

For model training and evaluation

- I've splitted the data into 80% training set and 20% validation set to measure the performance after each epoch.
- I used Mean Squared Error (MSE) as a loss function to measure how close the model predicts to the given steering angle for each input frame.
- I used the Adaptive Moment Estimation (Adam) Algorithm minimize to the loss function. Adam is an optimization algorithm introduced by D. Kingma and J. Lei Ba in a 2015 paper named Adam: A Method for Stochastic Optimization. Adam algorithm computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop algorithms, Adam also keeps an exponentially decaying average of past gradients mtmt, similar to momentum algorithm, which in turn produce better results.
- I used `ModelCheckpoint` from Keras to check the validation loss after each epoch and save the model only if the validation loss reduced.

Model Training:

Layer (type)	Output Shape	Params	Connected to
lambda_1 (Lambda)	(None, 66, 200, 3)	0	lambda_input_1

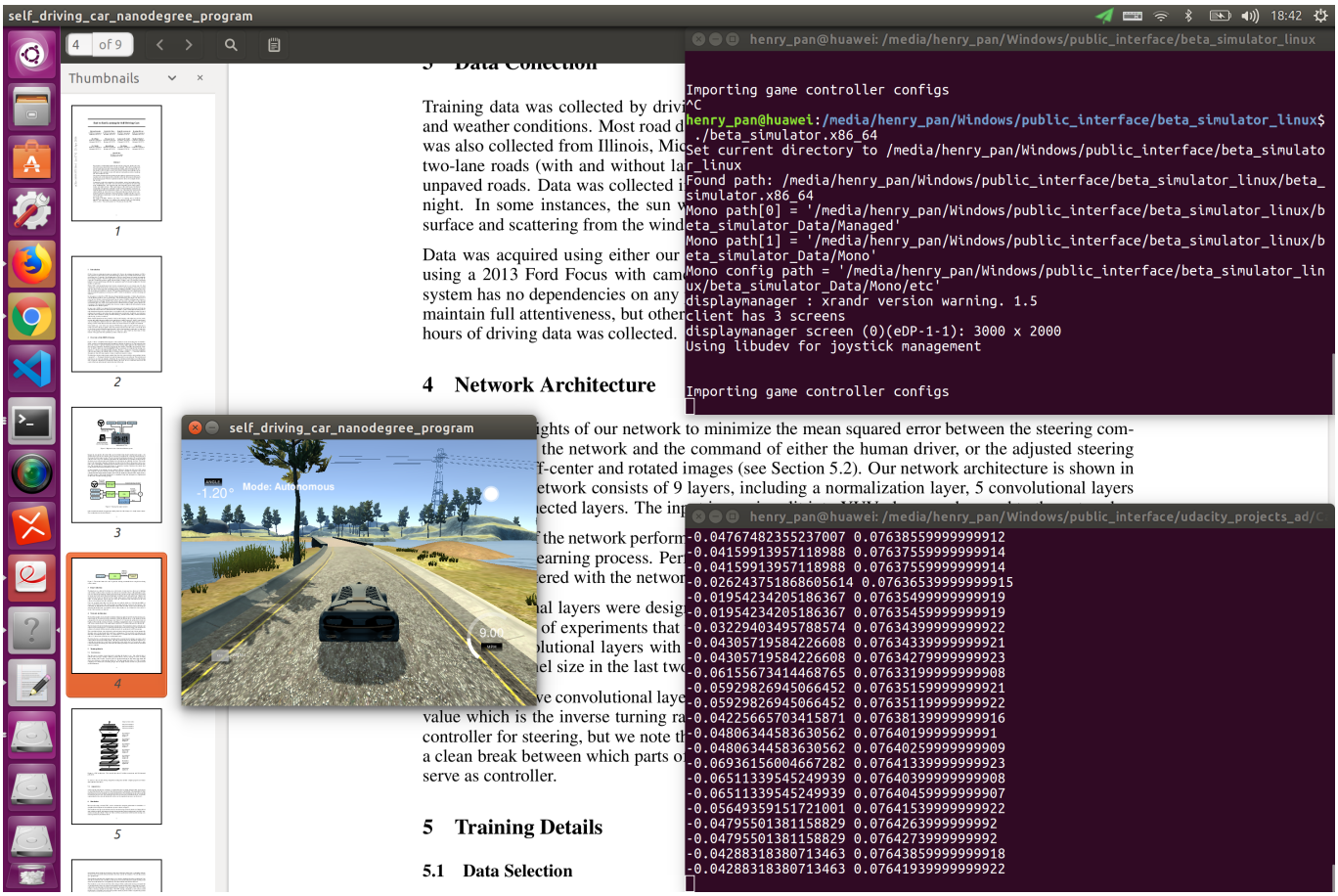
Layer (type)	Output Shape	Params	Connected to
convolution2d_1 (Convolution2D)	(None, 31, 98, 24)	1824	lambda_1
convolution2d_2 (Convolution2D)	(None, 14, 47, 36)	21636	convolution2d_1
convolution2d_3 (Convolution2D)	(None, 5, 22, 48)	43248	convolution2d_2
convolution2d_4 (Convolution2D)	(None, 3, 20, 64)	27712	convolution2d_3
convolution2d_5 (Convolution2D)	(None, 1, 18, 64)	36928	convolution2d_4
dropout_1 (Dropout)	(None, 1, 18, 64)	0	convolution2d_5
flatten_1 (Flatten)	(None, 1152)	0	dropout_1
dense_1 (Dense)	(None, 100)	115300	flatten_1
dense_2 (Dense)	(None, 50)	5050	dense_1
dense_3 (Dense)	(None, 10)	510	dense_2
dense_4 (Dense)	(None, 1)	11	dense_3
	Total params	252,219	

Model Evaluation:

Epoch	Loss	Validation Loss
1/10	0.0287	0.0122
2/10	0.0234	0.0110
3/10	0.0210	0.0115
4/10	0.0187	0.0114
5/10	0.0175	0.0093
6/10	0.0167	0.0093
7/10	0.0166	0.0104
8/10	0.0159	0.0100
9/10	0.0160	0.0086
10/10	0.0154	0.0101

For Model Testing on the simulator

The model was able to drive the car safely through the track without leaving the drivable portion of the track surface. Here is example when i simulating the model with simulator



More details refer to behavioral_cloning.pdf