Behavioral Cloning

Writeup Summary

Hong Cai,

May 14, 2017

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

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_best.h5 containing a trained convolution neural network
- writeup.pdf summarizing the results

•video.mp4 showing result with autonomous driving using the trained model

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

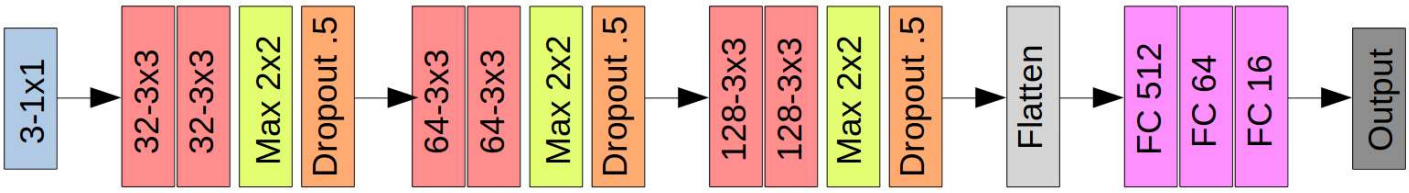
```
python drive.py model_best.h5
```

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



I implemented the model architecture above for training the data. The first layer is 3 1X1 filters, this has the effect of transforming the color space of the images. Research has shown that different color spaces are better suited for different applications. As we do not know the best color space apriori, using 3 1X1 filters allows the model to choose its best color space. This is followed by 3 convolutional blocks each comprised of 32, 64 and 128 filters of size 3X3. These convolution layers were followed by 3 fully connected layers. All the convolution blocks and the 2 following fully connected layers had leaky relu as activation function. I chose leaky relu to make transition between angles smoother.
The model includes ELU layers to introduce nonlinearity , and the data is normalized in the model using preprocessing.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting with dropout prob of 0.5.

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 (using a learning rate of 0.0001).

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 .

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 using 3 blocks of CNN and then 3 layes of Fully Connected layers.

My first step was to use a convolution neural network model similar to VGG model. I thought this model might be appropriate because the road scene is a typical view of real world to apply CNN models. 3 blocks of CNN should be good enough to identify key textures on the road/background.

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 trained the model 10 times and select the best one as final model for autonomous driving.

The final step was to run the simulator to see how well the car was driving around track one. The vehicle could self-drive pretty good and almost always kept in the middle of the road no matter how the road change (like curve to the left/right, passing a bridge, etc.)

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 consisted of a convolution neural network with the following layers and layer sizes.

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 64, 64, 3)	0	lambda_input_1[0][0]
conv0 (Convolution2D)	(None, 64, 64, 3)	12	lambda_1[0][0]
elu_1 (ELU)	(None, 64, 64, 3)	0	conv0[0][0]
conv1 (Convolution2D)	(None, 62, 62, 32)	896	elu_1[0][0]
elu_2 (ELU)	(None, 62, 62, 32)	0	conv1[0][0]
conv2 (Convolution2D)	(None, 60, 60, 32)	9248	elu_2[0][0]
elu_3 (ELU)	(None, 60, 60, 32)	0	conv2[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0	elu_3[0][0]
dropout_1 (Dropout)	(None, 30, 30, 32)	0	maxpooling2d_1[0][0]
conv3 (Convolution2D)	(None, 28, 28, 64)	18496	dropout_1[0][0]
elu_4 (ELU)	(None, 28, 28, 64)	0	conv3[0][0]
conv4 (Convolution2D)	(None, 26, 26, 64)	36928	elu_4[0][0]
elu_5 (ELU)	(None, 26, 26, 64)	0	conv4[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 13, 13, 64)	0	elu_5[0][0]
dropout_2 (Dropout)	(None, 13, 13, 64)	0	maxpooling2d_2[0][0]
conv5 (Convolution2D)	(None, 11, 11, 128)	73856	dropout_2[0][0]
elu_6 (ELU)	(None, 11, 11, 128)	0	conv5[0][0]
conv6 (Convolution2D)	(None, 9, 9, 128)	147584	elu_6[0][0]
elu_7 (ELU)	(None, 9, 9, 128)	0	conv6[0][0]
maxpooling2d_3 (MaxPooling2D)	(None, 4, 4, 128)	0	elu_7[0][0]
dropout_3 (Dropout)	(None, 4, 4, 128)	0	maxpooling2d_3[0][0]
flatten_1 (Flatten)	(None, 2048)	0	dropout_3[0][0]
hidden1 (Dense)	(None, 512)	1049088	flatten_1[0][0]
elu_8 (ELU)	(None, 512)	0	hidden1[0][0]
dropout_4 (Dropout)	(None, 512)	0	elu_8[0][0]
hidden2 (Dense)	(None, 64)	32832	dropout_4[0][0]
elu_9 (ELU)	(None, 64)	0	hidden2[0][0]
dropout_5 (Dropout)	(None, 64)	0	elu_9[0][0]
hidden3 (Dense)	(None, 16)	1040	dropout_5[0][0]
elu_10 (ELU)	(None, 16)	0	hidden3[0][0]
dropout_6 (Dropout)	(None, 16)	0	elu_10[0][0]
output (Dense)	(None, 1)	17	dropout_6[0][0]

Total params: 1,369,997
Trainable params: 1,369,997
Non-trainable params: 0