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 V
- Build, a convolution neural network in Keras that predicts steering angles from images V
- Train and validate the model with a training and validation set V
- Test that the model successfully drives around track one without leaving the road V
- Summarize the results with a written report V

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 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

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 5x5 & 3x3 filter sizes and depths between 32 and 128 (model.py lines 86-99)

The model includes RELU layers to introduce nonlinearity (code lines 88-112), and the data is normalized in the model using a Keras lambda layer (code line 77).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 90-113).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 14-15). 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 122).

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 . Move left and right and then press the record button. The process of returning to the center was repeated. After passing the bridge, I left when I went on the left curve. So, the same section was recorded several times. After going through the above process and training, I didn't deviate.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to modify the layer from the lecture appropriately.

My first step was to use a convolution neural network model similar to the model from the lecture [Even More Powerful Network] I thought this model might be appropriate because I think the data provided by Udacity is the most reliable.

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 using dropout. 'Model.add(Dense 75) 'was placed between layers. And I've collected more data.

Then I increase the number of epochs. At first, when epoche was 5, the loss was high, but when epoche was increased to 7, the loss was lowered.

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. When passing over the bridge, there was a phenomenon of continuous departure. To improve the driving behavior in these cases, I Data from repeatedly failing sections were collected. However, despite collecting a lot of data, it continued to fail. Eventually, I found the cause. This was due to the color channel. I change the color space BGR to RGB using 'image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)'.After adding this function, it did not leave the lane.

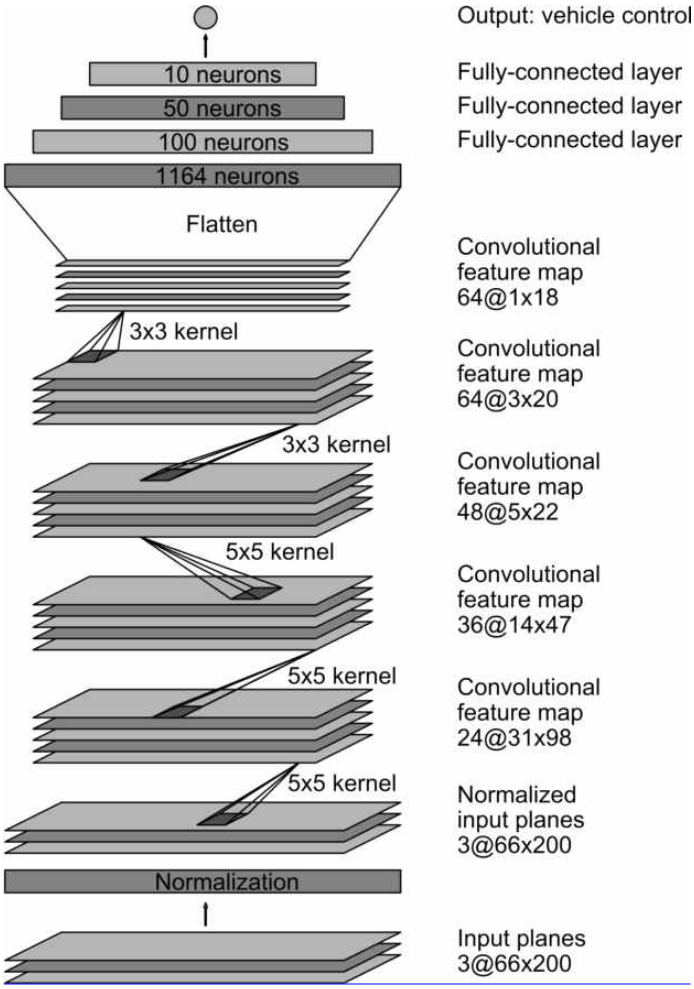
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 18-24) consisted of a convolution neural network with the following layers .

- Normalized input planes 3@66x200
- Convolutional feature map 24@31x98 (5x5 kernel)
- Convolutional feature map 36@14x47 (5x5 kernel)
- Convolutional feature map 48@5x22 (5x5 kernel)
- Convolutional feature map 64@3x20 (3x3 kernel)
- Convolutional feature map 64@1x18 (3x3 kernel)
- Flatten
- Fully connected layer 100neurons
- Fully connected layer 50neurons
- Fully connected layer 10neurons
- Output : vehicle control

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 how to come back when you're about to get out of the lane.

These images show what a recovery looks like. The car comes back to the center from the left. :



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

To augment the data set, I also flipped images and angles thinking that this would double the amount of data. For example, here is an image that has then been flipped:


We inverted the image and multiplied the steering angle value by -1. In this way, the data obtained when bending to the left could also be used when bending to the right.

After the collection process, I had 10000 number of data points. I then preprocessed this data by making generation function. There are a total of three cameras. Each camera has a different position. Think of each location as a center and adjust the angle. The data was classified into the left, middle, and right. And the constant was added to the angle to make it have the same effect as driving three times. And each data went through a filp process. Then, it was put in the images and steering_anges list.

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