Behavioral Cloning Project

Writeup

The goals/steps of this project are the following:

1. Use the simulator to collect data on good driving behavior
2. Build, a convolution neural network in Keras that predicts steering angles from images
3. Train and validate the model with a training and validation set
4. Test that the model successfully drives around track one without leaving the road
5. 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.

Writeup / README

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 P4.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** command on the terminal.

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 3x3 filter sizes and depths between 32 and 128 (model.py lines 97 - 107)

The model includes RELU layers to introduce nonlinearity (code line 98), and the data is normalized in the model using a Keras lambda layer (code line 94).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 117).

The model was trained and validated on different data sets to ensure that the model was not overfitting. 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 127).

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, and then also performed data augmentation by flipping the data.

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 perform a behavioral cloning algorithm, which is a drive-by-wire concept wherein, the CNN-based image regression is performed which takes care of steering angle, acceleration, and braking.

My first step was to use a convolution neural network model similar to that of the Nvidia architecture. I thought this model might be appropriate because it uses various convolutional layers and dropout layers and was also developed by Nvidia and tested on a real vehicle.

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 so that it incorporates the dropout layer.

Then I used Adam optimizer which performs backpropagation and optimizes the weights of the network.

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 especially in the case of turns. To improve the driving behavior in these cases, I performed image augmentation by flipping the images and also collected data by driving on the track twice, each clockwise and anti-clockwise.

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

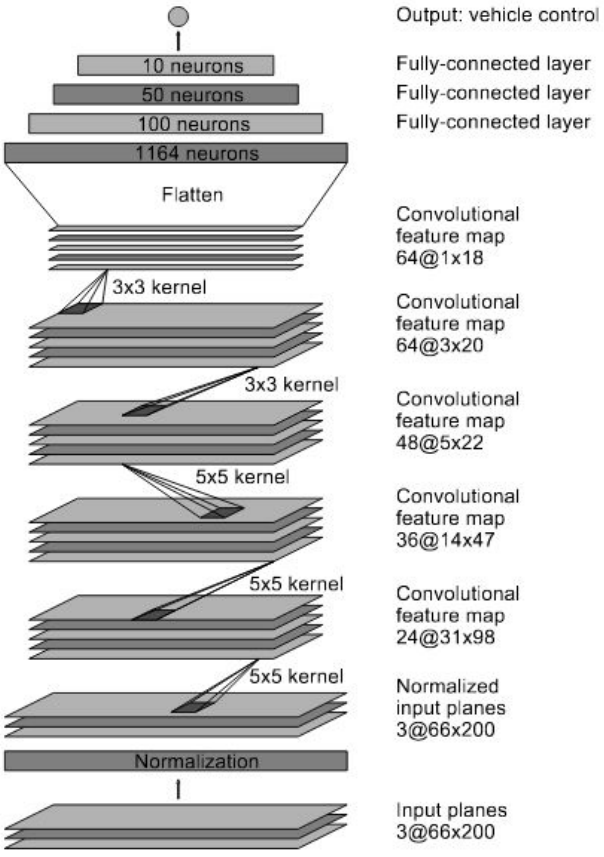
2. Final Model Architecture

The final model architecture (model.py lines 93-125) consisted of a convolution neural network with the following layers and layer sizes.

```
model=Sequential()

model.add(Lambda(lambda x: (x/255.0)-0.5, input_shape=(160,320,3)))

model.add(Cropping2D(cropping=((70,25),(0,0))))

model.add(Conv2D(24,(5,5), strides=(2,2)))

model.add(Activation('relu'))

model.add(Conv2D(36,(5,5), strides=(2,2)))

model.add(Activation('relu'))

model.add(Conv2D(48,(5,5), strides=(2,2)))

model.add(Activation('relu'))

model.add(Conv2D(64,(3,3), strides=(2,2)))

model.add(Activation('relu'))

model.add(Flatten())

model.add(Dense(1164))

model.add(Activation('relu'))
```

```
model.add(Dense(100))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(50))

model.add(Activation('relu'))

model.add(Dense(10))

model.add(Activation('relu'))

model.add(Dense(1))
```

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)

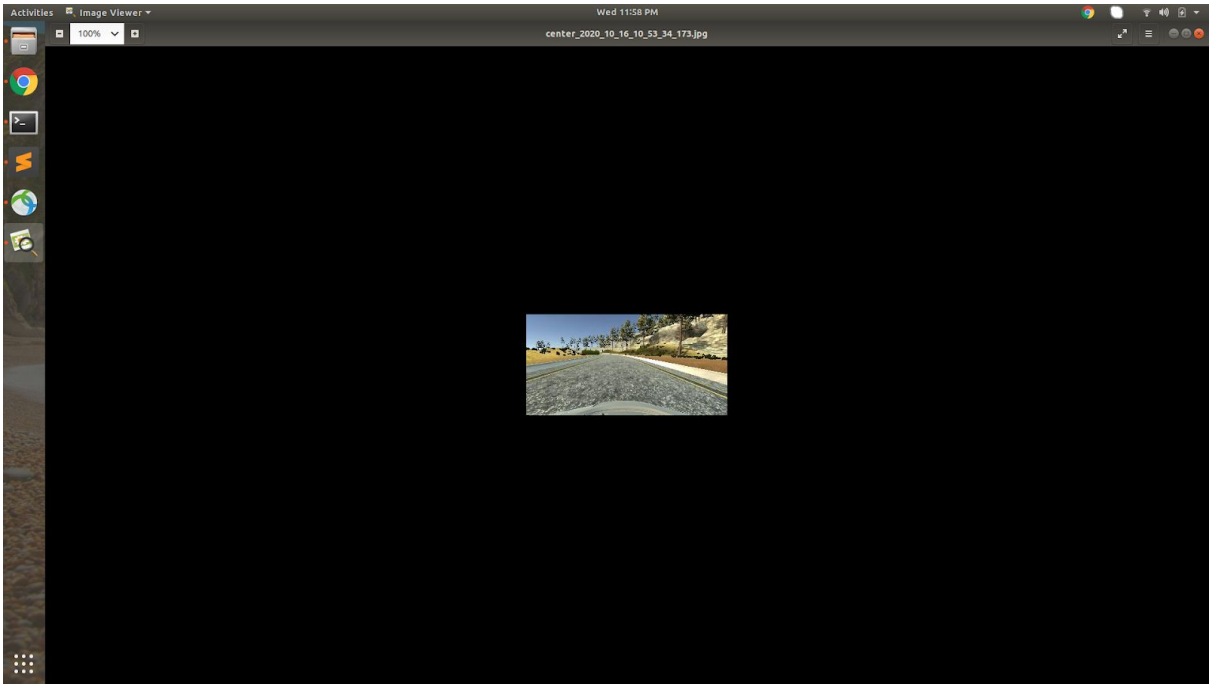


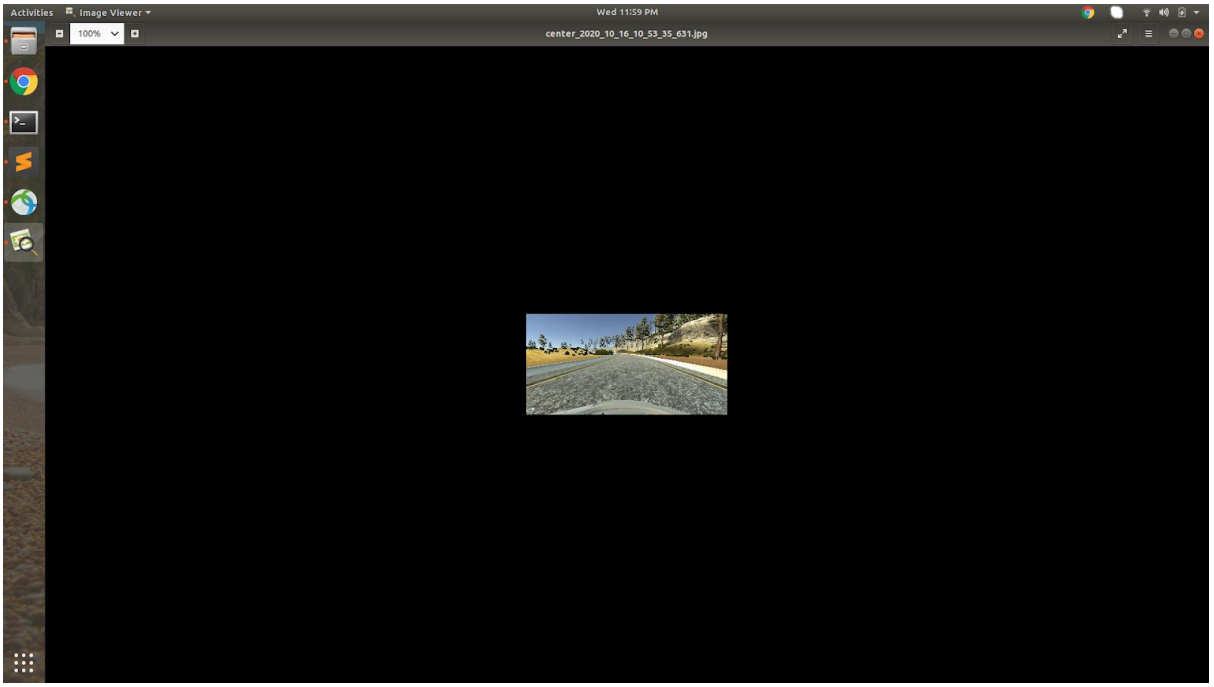
Note: I did not use the last convolutional layer.

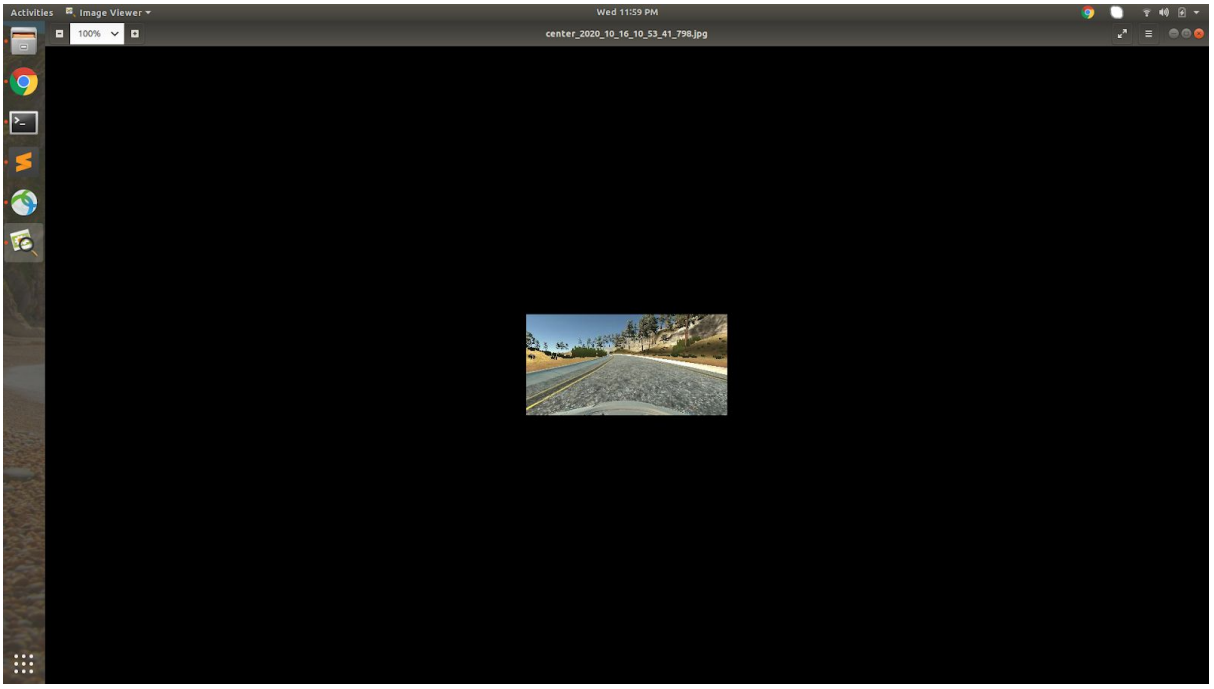
3. Creation of the Training Set & Training Process

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 the center so that the vehicle would learn various scenarios and how to align to the center of the road. These images show what a recovery looks like.







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

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

After the collection process, I had X number of data 17126 points. I then preprocessed this data by changing the angle by various factors and also flipping the images to create a larger dataset.

I finally randomly shuffled the data set and put 15% 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 underfitting. The ideal number of epochs was 5 as evidenced by the output of the neural network. I used an adam optimizer so that manually training the learning rate wasn't necessary.