Behavioral Cloning Project

Paul Comitz 8/9/2017

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

Files Submitted and Code Quality

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
- Video (https://vimeo.com/229389712) of one lap around the track

Functional Code

Using the Udacity provided simulator and the drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.h5
```

Submission Code

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

Model Architecture

The model was developed iteratively following the steps described in class. The initial architecture was a flattend image connected to a single output node. The network output predicted the steering angle. This initial network, and several of the early networks, drove in circles.

The basic approach was to successvely refine the network using different techniques such as normalization augmentaion, cropping, adding convolutional layers, and max pooling. The first partially succesful approach used the following model, also described in class:

```
model.add(Lambda(lambda x: (x/255.0)-0.5, input_shape=(160,320,3)))
model.add(Convolution2D(6,5,5, activation = "relu"))
model.add(MaxPooling2D())
model.add(Convolution2D(6,5,5, activation = "relu"))
model.add(MaxPooling2D())
model.add(Dense(120))
model.add(Dense(84))
model.add(Flatten())
model.add(Dense(1))
```

This model above was able to drive to the dirt exit after the bridge. The failure at the bridge was difficult to overcome. The above model was subsequently refined to with augmentation, and cropping without success. Side camera images were then added with a correction factor of +/- 0.2 applied to the left and right images. The network was trained with 3 to 7 epochs, but would always fail at the dirt road exit after the bridge.

The first successful transversal of the entire track occurred after realizing that OpenCV was transforming the RGB images to BGR. After transforming the images back to RGB (after augmentation etc), the car successfully made it around the track. This first successful model used the architecture above, the left and right images with corrections applied, flipping with the measurement of the flipped image multiplied by -1, and 5 training epochs. The loss on this model was 0.0145, with a validation loss of 0.0187.

After this initial success, several other models were implemented and evaluated for comparison. A steering model (https://github.com/commaai/research/blob/master/train_steering_model.py) from coomai was implemented but was not successful. A modified version of the nvidia model, presented in class produced the best redults. The model is described below.

The basic steps for developing the final model were:

1. Read Udacity supplied training data center, left, and right
2. Add correction to left measurement, subtract correction from right measurement
3. Apply bgr to rgb on all images
4. Perform augmentation by flipping all images
5. Normalize all images
6. Crop all images, top and bottom
7. Apply the nvidia architecture (many other archictuctures were attempted)
8. Optimize with adam

The final model consists of a convolutional neural network and several fully connected layers. A number of variants were attempted, with varying results. The final model is based on the nvidia architecture, with a few modifications. The network was shown in class and is described in the paper End to End Learning for Self-Driving Cars (https://arxiv.org/pdf/1604.07316v1.pdf).

The nvidia archiecture is designed to minimize the MSE bewteen the steering commands that are output by the network and the driver or the measurements associated with the training data from the images. As described in the paper, the configuration of the layers was experimentally derived.

The basic details for the model are:

- The keras Sequential (https://keras.io/getting-started/sequential-model-guide/) model is used.
- The first layer is a Lambda layer. The input shape is specified as (160,320,3). The image pixels are normalized in this layer.
- In the next layer the images are copped. 70 pixels are removed from the images. the top 70 pixels are mostly blue sky. the bottom 25 pixels are the hood of the car. This information is not useful for training the model.
- The cropping is followed by 5 Convolutional Layers as specified in the nvida paper. The details of the Convolution layers are:
  - Convolution Layer 1: 24 filters with a 5 x 5 kernel., A 2x2 stride is used with a RELU activation.
  - Convolution Layer 2: 36 filters with a 5x5 kernel. A 2 x 2 stride is used with a RELU activation.
  - Convolution Layer 3: 48 filters with a 5x5 kernel. A 2 x 2 stride is used with a RELU activation.
  - Convolution Layers 4 and 5: 64 filters with a 5x5 kernel. A RELU activation is used.

The network is flattened and followed by 4 fully connected layers. The fully connected layers provides an implementation of the familiar neural net mechanization of (xW +b). As indicated above the configuration of the layers was empically derived.

The final nvidia archiecture was trained for 4 epochs. On an i7 with 16GB of RAM the training approximatley two hours. The loss of the final model was:

- loss : 0.0126
- val_loss: 0.0177

When attempting more than 4 epochs the validation loss would increase in later epochs, suggesting overfitting. The model with 4 epochs was chosen to minimze this overfitting. The driving performance of that network can be seen in the follwoing video:

- Final Model around track video (https://vimeo.com/229389712)

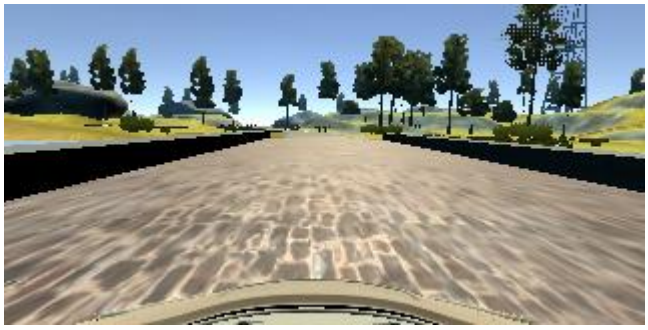
Training Data

After using all of the techniques described above, augmentation, and left and right camera images etc, there were 38572 training samples amd 9644 validation samples (20% split).

Center Camera Images

The training data supplied by Udacity was used. There were several attempts to record training data. The recorded data was used with several of the architectures that were used. Better results were obtained with the Udacity training data. The decision was made to use this Udacity supplied data and concentrate on the model.

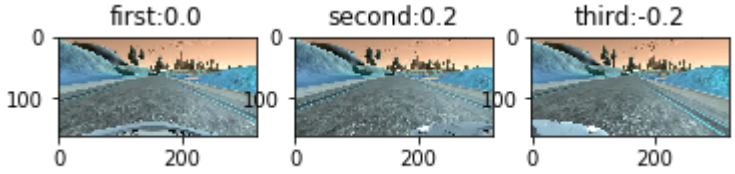
The initial set consists of 8036 measurements with center, left and right images. The images are each 320 x 160 RGB images. Initially, only the center images were used. An example center image is shown below.



Training Data BGR Images

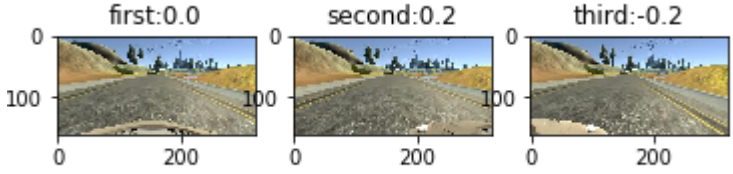
When reading the data with OpenCV the images are read in Blue, Green, Red format. As discussed later, this had a major impact on performance.

Examples of Center , Left, and Right are shown below.



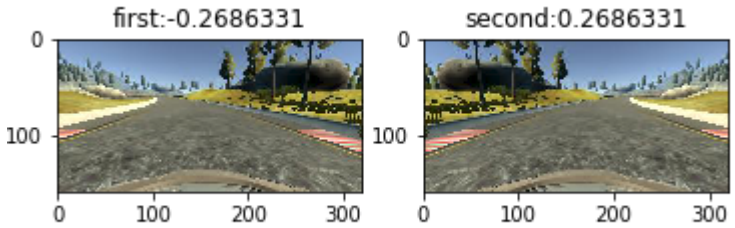
Training Data - Multiple Cameras and RGB Images

The initial models used the center images.The left and right images were subsequently added to the training data. The images were also converted from BGR to RGB. The examples below show Center, Left, and Right images after conversion to RGB. This turned out to be a key step. The car could not succesfully navigate the dirt road offramp after the second curve until after this conversion.



Training Data - Augmentation

Flipped Images



All images are flipped using `cv2.flip(image,1)`. For each flipped image the associated masurement is also "flipped" by multiplying by -1. This technique is used to overcome the left turn bias associated with the base training data.

In [ ]: