Behavioral Cloning

Self Driving Car Nanodegree

Alberto Rivera
April 10, 2017

Domain Background

This writeup and the accompanying files are part of Project 3 from Term 1 of the Udacity Self Driving Nanodegree program. This projects aims to train a simulated car to complete a lap around a course. The simulation environment is made using Unity and was provided by Udacity. The model is trained by capturing data while driving the car manually.

The template for this project can be found on the following Github repository:
https://github.com/udacity/CarND-Behavioral-Cloning-P3

My implementation of this project can be found on the following Github repository:
https://github.com/alberto139/CarND_BehavioralCloning
Note that the 'data' folder and 'run1.mp4' have been removed from the repository because of size constraints.

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.

Required Files

The files for this project include:

- model.py - The script used to create and train the model.
- drive.py - The script to drive the car. (Unmodified)
- model.h5 - The saved model. Here is the documentation explaining how the file was created.
- writeup_report.pdf - (This file) Report explain the structure of your network and training approach.
- video.mp4 - A video recording of your vehicle driving autonomously at least one lap around the track

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
```

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

The model used is very similar to the model described in this paper by NVIDIA, with some minor modifications. The NVIDIA model is illustrated in Figure 1.

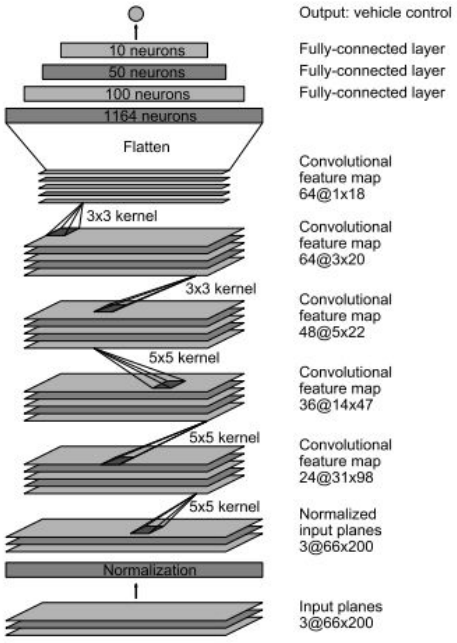


Figure 1: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

The final model used for this project, as implemented using Keras, is the following:

```
# Convolutional Neural Network
# NVIDIA Architecture with modifications
model = Sequential()
# Preprocess (Normalize and Mean Center)
model.add(Lambda(lambda x: x / 255.0 - 0.5, input_shape = (160,320,3)))
model.add(Cropping2D(cropping=((50,20), (0,0)), input_shape=(160,320,3)))
model.add(Convolution2D(16, 3, 3, subsample=(2, 2), activation="relu"))
model.add(Convolution2D(32, 3, 3, subsample=(2, 2), activation="relu"))
model.add(Convolution2D(64, 5, 5, subsample=(2, 2), activation="relu"))
model.add(Convolution2D(64, 3, 3, activation="relu"))
model.add(Convolution2D(64, 3, 3, activation="relu"))
model.add(Flatten())
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
```

Figure 2. Modified NVIDIA CNN

The model takes in 160x320x3 images as input and normalizes them in the lambda layer. The images are cropped by 50 pixels on top and 20 pixels on the bottom. This is because The top 50 pixels on the data contains information that is not relevant to how the car is being driven, such as the skyline and trees. The bottom 20 pixels are mostly the hood of the car which is also not relevant to the car's steering angles. Figure 3 shows an example of the full image captured by the camera and the cropped version of the image.

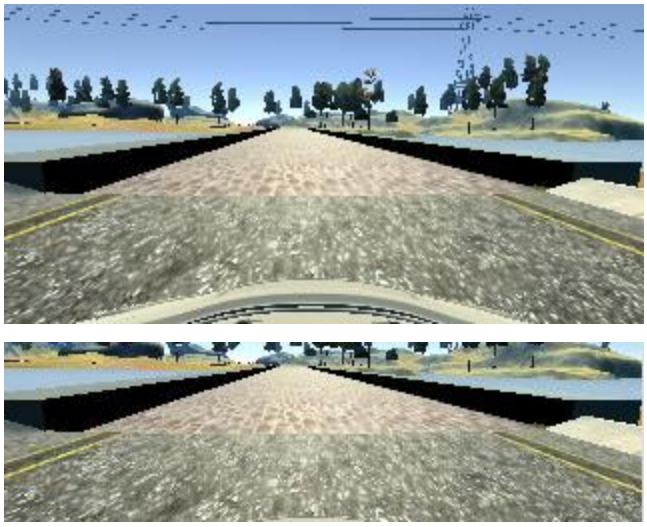


Figure 3. The top image is the original uncropped image (160x320x3). The bottom image is the result of cropping (90x320x3).

After the image is cropped the output is fed through 5 convolutional layers with 'relu' activation functions. The first convolutional layer has a 3x3 filter size with a dimensionality of 16 and a 2x2 stride. The following 3 layers are very similar but with different dimensionalities and strides. The last two layers do not have strides. After the convolutional layers the output is flattened and then transformed to a 100 neuron fully connected layer. The following layers continue to compress the fully connected layer until there is a single output which is the angle measurement for controlling the car.

The model does not include a dropout layer because the performance was satisfactory without one. However the the model was trained and validated on data gathered from 5 clockwise and another 5 counterclockwise laps to reduce overfitting. The data was split using the `train_test_split` function provided by scikit-learn in the following code line:
`train_samples, validation_samples = train_test_split(samples, test_size=0.2)`
The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. video.mp4 is a recording of the first lap done autonomously with the model created by the training model described.

The model used an adam optimizer.
`model.compile(loss = 'mse', optimizer = 'adam')`
The Adam optimizer is an algorithm for first-order gradient-based optimization, so the learning rate was not tuned manually. Adam optimization was introduced in this paper by Kingma and Lei Ba from OpenAI and University of Toronto respectively.

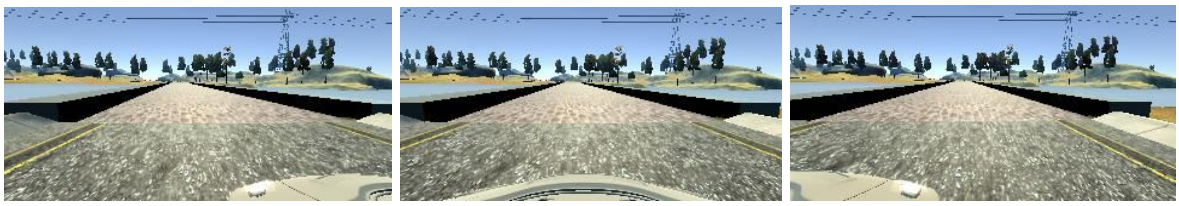
Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road. The data was gathered over 10 training laps, 5 clockwise and 5 counterclockwise laps. For further details about how I created the training data, see the next section.

Architecture and Training Documentation

The deep learning model is largely inspired by the NVIDIA model described in the previous section and Figure 1. The main differences from that model to the model used are some of the dimensionalities and strides of the first 4 convolutional layers, and one 5th convolutional layer which is a copy of the 4th convolutional layer. The modifications were made largely on a trial and error basis and not because of scientific and thought out reasons. The model was tested locally on a laptop with very few images for training. With

little data the model seem to overfit which in this case is indicative that the model is working properly.

The overall strategy for deriving a model architecture was to acquire steering angle data relating to images of the track as viewed from the car at that certain point in time. The car is outfitted with 3 cameras, aiming towards the center, left, and right of the road. Figure 3 shows images recorded from the 3 cameras.



Left Center Right

Figure 4. Left, Center, and Right images captured by the cameras mounted on the car at the start of training.

The side images proved very useful in training the model. The side images were fed to the training model with a corrected steering angle taking into account the angle and placement of the camera. The purpose of doing this is to teach the model how to react when the center camera encounters an image similar to that of the side images, meaning that the car is not centered on the road. The a correction constant is added to the original angle and then flipped. This makes the car steer toward the middle of the road as is particularly helpful during turns. The training data was also augmented by flipping the images and corresponding steering angles. This was done in a python generator function to avoid memory resource problems. Using the generator the model could be trained with all of the data on a normal laptop without a GPU, however this would take days! The final model was trained on an AWS instance and it took around 5 minutes. Figure 5 shows the code for the generator function.

```
def generator(samples, batch_size=128):
    num_samples = len(samples)
    while 1: # Loop forever so the generator never terminates
        shuffle(samples)
        for offset in range(0, num_samples, batch_size):
            batch_samples = samples[offset:offset+batch_size]

            images = []
            angles = []
            correction = 0.5
            for batch_sample in batch_samples:
             # Center Image
                name = "./data/IMG/IMG/"+batch_sample[0].split('/')[-1]
                center_image = cv2.imread(name)
```

```
                center_angle = float(batch_sample[3])
                images.append(center_image)
                images.append(cv2.flip(center_image,1))
                angles.append(center_angle)
                angles.append(center_angle*-1.0)
                # Left Image
                name = "./data/IMG/IMG/"+batch_sample[1].split('/')[-1]
                left_image = cv2.imread(name)
                images.append(left_image)
                images.append(cv2.flip(left_image,1))
                angles.append(center_angle + correction)
                angles.append((center_angle + correction) *-1.0)
                # Right Image
                name = "./data/IMG/IMG/"+batch_sample[2].split('/')[-1]
                right_image = cv2.imread(name)
                images.append(right_image)
                images.append(cv2.flip(right_image,1))
                angles.append(center_angle - correction)
                angles.append((center_angle - correction) *-1.0)

            # trim image to only see section with road
            X_train = np.array(images)
            y_train = np.array(angles)
            yield (X_train, y_train)
```

Figure 5. Python Generator Function

Overall the model performed very well and the car is able to traverse the track smoothly and indefinitely. Further work could include speed as another feature to consider, which could make quantifying the performance of the car much more interesting. Lap times could be used as a metric. Another idea that might be worth looking into using a PID controller for the angle correction rather than a constant.