Behavioral Cloning Project

by: Brendan Kam

The goal of the behavioral cloning project was to create a neural network to teach a car to drive autonomously based on human driving behavior. For the training to be considered successful, the car must drive at least one lap around the track without leaving the driveable portion. The neural network created for this project met both criteria. This report describes in further detail the methods used to successful complete this project.

The first part of the project was to collect driving data through a driving simulator provided by Udacity. However, after several unsuccessful attempts at collecting decent driving data, a decision was made to use the driving data provided by Udacity. The Udacity's driving images consists of images from three cameras attached on the left, right and center of the car. Other data include the steering angle, the throttle and the speed of the car.

Upon further inspection of the data, it was found that the data was heavily skewed towards the car driving straight. Training the data with an unbalanced data set would lead to a bias where the car decides to drive straight even when it is supposed to turn. To overcome this problem a generator was used to randomly choose images from one of the three cameras. A small correction factor was added to the steering angles of the left and right camera images to account for the offset of the camera from the center. By using images from all three cameras, the bias to drive straight was reduced while also teaching the car to move towards the center of the road if it drifted off towards the sides. In the final model, a probability that an image from center camera to be chosen was 0.5 while the cameras at the sides were set to 0.25. The rationale behind doing this was that in the that if the probabilities were all equal, the car would then be biased towards driving towards turning. A function to randomly change the brightness of the images or flip them was also included to allow the car to learn to drive in different conditions.

Upon further inspection of the data, it was found that the data was heavily skewed towards the car driving straight. Training the data with an unbalanced data set would lead to a bias where the car decides to drive straight even when it is supposed to turn. To overcome this problem a generator was used to randomly choose images from one of the three cameras. A small correction factor was added to the steering angles of the left and right camera images to account for the offset of the

camera from the center. By using images from all three cameras, the bias to drive straight was reduced while also teaching the car to move towards the center of the road if it drifted off towards the sides. In the final model, a probability that an image from center camera to be chosen was 0.5 while the cameras at the sides were set to 0.25. The rationale behind doing this was that in the that if the probabilities were all equal, the car would then be biased towards driving towards turning. A function to randomly change the brightness of the images or flip them was also included to allow the car to learn to drive in different conditions.

Figure 1, 2 and 3 shows the views taken by the center, left and right cameras of the car.

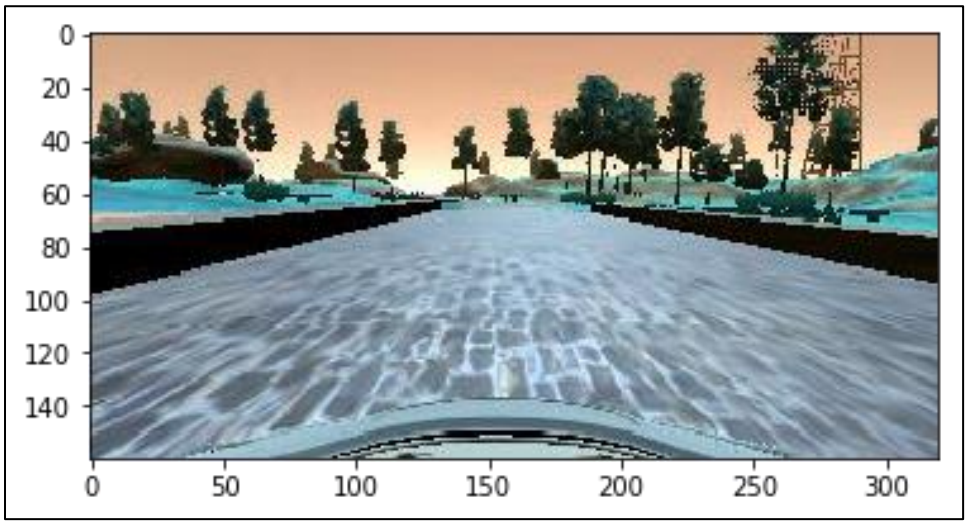

Figure 1: View from center camera

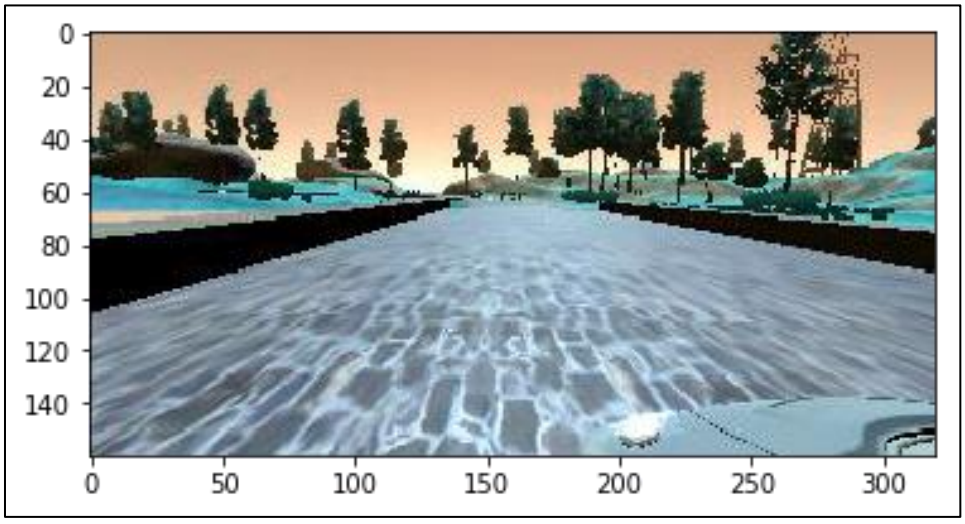

Figure 2: View from left camera

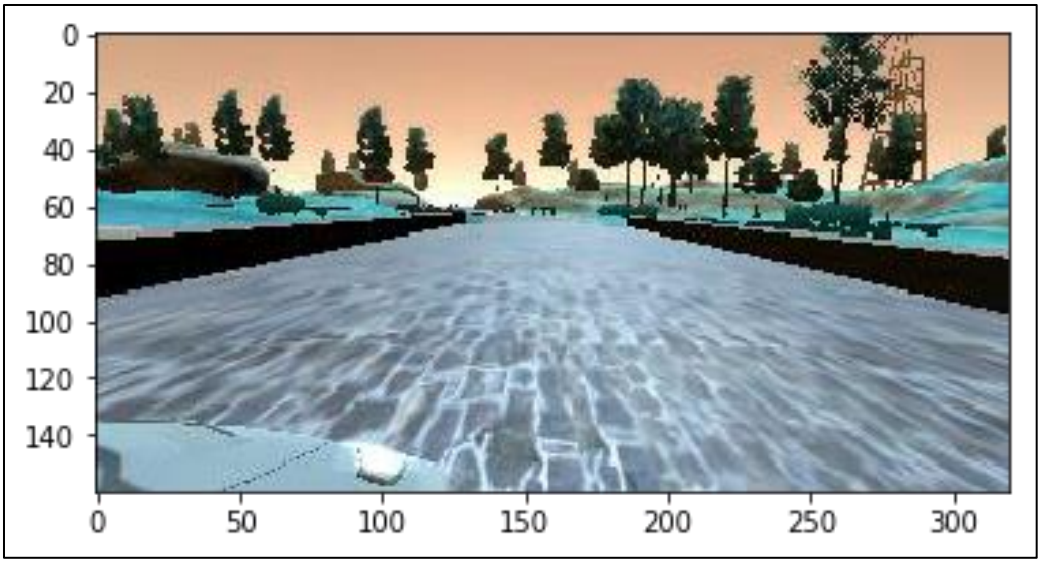

Figure 3: View from right camera

The neural network used in this project is a slight modification on the neural network used by NVIDIA.

The overall network consists of 8 layers. An input layer, 3 convolutional neural network (CNN) layers and 3 fully connected layers.

Table 1 describes in further detail the layers of the neural network used in the project.

Table 1: Model Architecture of Neural Network

Layers	Description
Input	• Lambda layer to normalize and center image data • Cropping to remove portions of image irrelevant to helping car make driving decisions
CNN 1	• 24 x 5 x 5 CNN • Strides of 2 • Relu activation function
CNN 2	• 36 x 5 x 5 CNN • Strides of 2 • Relu activation function

CNN 3	• 48 x 5 x 5 CNN • Strides of 2 • Max Pooling to reduce overfitting • Elu activation function
Fully Connected 4	• Input of 100 • Elu activation function • Dropout with 0.5 probability to reduce overfitting
Fully Connected 5	• Input of 50 • Elu activation function
Fully Connected 6	• Input of 10 • Elu activation function
Output layer	• Returns trained weights and biases

The original NVidia neural network had 5 CNN layers. However, upon further experimentation, for the data set used in this project, it was found that there was no difference in performance between 3 CNN layers and 5 CNN layers. To simplify the model, 2 CNN layers were removed. A max pooling and a dropout was also introduced to prevent overfitting of the data. After experimenting, adding more dropout layers, or pooling layers led to a decrease in performance.

Another modification made to the NVidia model architecture was the inclusion of a lambda layer to normalize and centralize the image data. A Keras cropping layer was also used to remove portions of the camera images that would be redundant in training the data.

Figure 4 is a cropped version of figure 1 shown above.

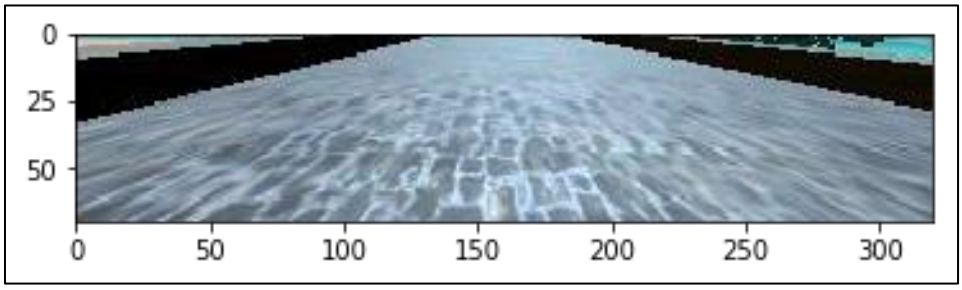

Figure 4: Cropped image of figure 1

As described earlier, a generator was used to randomly choose and modify the camera images. In training the model, the generator randomly picks and modifies 500 images every time it is called. The model was trained for 5 epochs with 20 000 different images used per epoch. The model was initially trained with 10 epochs. However, it was reduced to prevent overfitting the data and because the model performed worse after it was trained for more than 5 epochs. An Adam optimizer with an initial learn rate of 0.001 was chosen after several reiterative processes to find the best learn rate. It should be noted however that the Adam optimizer readjust the learn rates after every epoch.

During the simulation of the car, the speed in drive.py was set to 17.5 miles per hour. It was able to make a full lap around the track without leaving the track. However, when recording the simulation, the car failed to complete the track. Upon discussion with Udacity's forum mentors, the issue was isolated to hardware or Tensorflow limitations. It was suggested that the simulation speed be reduced. Upon reducing the speed to 12.5 miles per hour, the car drove successfully and smoothly around the track.

There were several lessons learn from this project. The most important lesson was that the training data needs to be balanced. An unbalanced data would cause the model to be biased, hence affecting the driving behavior of the car. Another lesson learn was that increasing the number of training epochs could cause the model to overfit the data. Finally, increasing the number of layers does not necessarily improve the training performance for simple cases.

References and acknowledgements

I was able to complete this project with the guidance of Udacity's forum mentors, Subodh Malgonde, Alex Cui, Gemma and Fernando Damasio who had previously completed this project from an earlier cohort.

Ideas used to complete this project were also taken from blog posts by several Udacity's Self-Driving Car Nanodegree students from an earlier cohort, namely,

Vivek Yadav's blog article, "An augmentation based deep neural network approach to learn human driving behavior"

https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9

An Nguyen's blog article, "You don't need lots of data! (Udacity Behavioral Cloning)"

https://medium.com/@fromtheast/you-dont-need-lots-of-data-udacity-behavioral-cloning-6d2d87316c52

Subodh Molgonde's blog article, "Teaching a car to mimic your driving behaviour"

https://medium.com/@subodh.malgonde/teaching-a-car-to-mimic-your-driving-behaviour-c1f0ae543686