UDACITY

PROJECT

Use Deep Learning to Clone Driving Behavior

A part of the Self-Driving Car Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

🏆 Excellent job with the project! You seem to have grasped all the main concepts from this section of the term, while also adding many extras that go above and beyond the specs. In addition, your report discussion and documentation is very comprehensive and well-written, and I'm nomination your project for excellence. 😎

You can also watch this video of George Hotz from Comma.AI to learn about some of the drawbacks of behavioral cloning, or check out these links for more on steering techniques:

- http://selfdrivingcars.mit.edu/
- https://medium.com/udacity/teaching-a-machine-to-steer-a-car-d73217f2492c

Required Files

✓

The submission includes a model.py file, drive.py, model.h5 a writeup report and video.mp4.

Good work including all the required files.

Quality of Code

✓

The model provided can be used to successfully operate the simulation.

Nice job creating a model that operates the simulator and drives the car autonomously. 😄

✓

The code in `model.py` uses a Python generator, if needed, to generate data for training rather than storing the training data in memory. The `model.py` code is clearly organized and comments are included where needed.

Great work implementing the generator to train the model and including extensive comments and documentation in the notebooks to help readers understand the implementation.

I particularly like how you outputted the images from the generator to see exactly what kind of images the network is being trained on. 😎

Model Architecture and Training Strategy

✓

The neural network uses convolution layers with appropriate filter sizes. Layers exist to introduce nonlinearity into the model. The data is normalized in the model.

Excellent job constructing the CNN layers and including ELU activations to introduce non-linearity.

- It can also help a model's training to use Batch Normalization, which maintains the mean activation ~0 and standard deviation ~1...

https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

✓

Train/validation/test splits have been used, and the model uses dropout layers or other methods to reduce overfitting.

Nice work using training and validation splits and adding L2 reg to reduce overfitting, after you found that using dropout wasn't helpful. 😉

✓

Learning rate parameters are chosen with explanation, or an Adam optimizer is used.

Good choice here to use an Adam optimizer — you can read more on gradient descent optimization algorithms here: http://sebastianruder.com/optimizing-gradient-descent/

✓

Training data has been chosen to induce the desired behavior in the simulation (i.e. keeping the car on the track).

Nice work choosing the training data to clone the car's driving behavior, and including extensive processing steps such as cropping, smoothing, and brightness shifts. This really seems to have helped the model generalize well. 😄

Note that the sample data provided by Udacity (see here) can also be sufficient on its own to train a robust model for Track 1 — simply training with fewer `0` steering angle samples can help a model to generalize enough.

(e.g., create a generator that randomly skips training 50% (or more) of the images with `0` steering angles)

Architecture and Training Documentation

✓

The README thoroughly discusses the approach taken for deriving and designing a model architecture fit for solving the given problem.

Great job explaining your approach to the model architecture by trying out different models before settling on the NVIDIA network. 😎

For more on the model architectures already built into keras, you can also check out this post: https://medium.com/towards-data-science/an-intuitive-guide-to-deep-network-architectures-65fdc477db41

✓

The README provides sufficient details of the characteristics and qualities of the architecture, such as the type of model used, the number of layers, the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged.

Good work discussing the network architecture and showing the number and size of layers of the model with both the diagram and `model.summary()` output.

- I also liked seeing your extra efforts to visualize the model filters; this is the first time I've seen a student attempt that. 😃
- For more ideas on visualizing CNN models, you can check out the python package Picasso.

✓

The README describes how the model was trained and what the characteristics of the dataset are. Information such as how the dataset was generated and examples of images from the dataset must be included.

Nice work describing how the model was trained and some of the characteristics of the dataset, such as using reverse driving and both center-lane & recovery driving from the sides of the road.

It's great to see how you also plotted the histograms of steering angles to reveal the prevalence of `0` or very low angles. 😉

Simulation

✓

No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

The automated car successfully navigates the track and stays on the drivable portion of the track surface. Well done!

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ