UDACITY

PROJECT

Use Deep Learning to Clone Driving Behavior

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Congratulations 👏👏👏
Nice job with this submission. Well documented code, modular structure, a very detailed and to the point writeup. It's amazing.
Keep up the good work !!!
I leave you with a very good read which can help you to improve your model: http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html
Well Done and Good luck for next projects. :)

Required Files

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

Thanks for providing all the required files and naming the files as expected. Good job for demonstrating the performance of your model through the video.

Quality of Code

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

Well done for implementing the neural network that controls the car in the simulator. Overall, the drive through the track 1 is relatively smooth. Car never touched the lanes while driving.

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.

Awesome

- The code is well formatted and appropriately commented which makes it easy to read the code and understand the algorithm.
- Good job using the generators to generate data at training time instead of storing it in memory. Right and left camera images and flipped images are appended through generators.

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.

Awesome

- Great job trying NVIDIA end-to-end model, and improving it further.
- The model uses RELU activation functions to introduce nonlinearity into the model. Leaky RELU is another good candidate to experiment with.
- The data is normalized in the model using a Lambda layer
- Images are cropped within the model. Well done.

Suggestion

A suggestion would be to crop the images before normalization to save normalization operations.

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

- An awesome job to monitor and avoid overfitting. The training of the model includes also validation set that allows monitoring the performance of the model and avoids overfitting.
- Dropout technique is applied to avoid overfitting.

Readings/Suggestions

- L2 Regularization is proven to be helpful as a regularizer to avoid overfitting, in this project.
- A suggestion would be to try spatial dropout with or without dropouts between fully connected layers. It would be interesting to see the experiment results.
- In case you're interested the following paper suggested by a fellow reviewer- https://arxiv.org/abs/1412.6806 proposes to discard the pooling layer in favor of architecture that only consists of repeated CONV layers. To reduce the size of the representation they suggest using larger stride in CONV layer once in a while. Discarding pooling layers has also been found to be important in training good generative models. It seems likely that future architectures will feature very few to no pooling layers.
- Longer or shorter training might cause overfitting/underfitting of the model to the training set. Therefore, it is important to find the right balance Instead. You can do that quite easily by identifying the number of the epoch that provides the maximum performance (or convergence) on the validation set. It will be useful to implement a condition, that when met the training iteration stops. That can save processing time and will promise optimal network for the following analysis. You can also use the function ModelCheckpoint that will save the model after every epoch and in turn the best model from the training. Those models can be tried on track and a further comparison about the error and model performance on track can be made.
- Another call back method which does the early stopping can be used with model checkpoint. You can read about it and other cool callback methods in the link above.

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

Awesome

Adam Optimizer is used. You might be interested in reading some other reasons why it is a better choice. The tf.train.AdamOptimizer uses Kingma and Ba's Adam algorithm to control the learning rate. Adam offers several advantages over the simple tf.train.GradientDescentOptimizer. Foremost is that it uses moving averages of the parameters (momentum). Simply put, this enables Adam to use a larger effective step size, and the algorithm will converge to this step size without fine tuning. The main down side of the algorithm is that Adam requires more computation to be performed for each parameter in each training step (to maintain the moving averages and variance, and calculate the scaled gradient); and more state to be retained for each parameter (approximately tripling the size of the model to store the average and variance for each parameter). A simple tf.train.GradientDescentOptimizer could equally be used, but would require more hyperparameter tuning before it would converge as quickly

Suggestion

- There is another version of Adam optimizer called nadam optimizer. You can read the paper about it here: You can read the paper here : http://cs229.stanford.edu/proj2015/054_report.pdf
- You can read about other gradient descent based optimizers here: http://sebastianruder.com/optimizing-gradient-descent/index.html

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

Training data was properly chosen to induce the desired driving behavior in the simulation. The training set includes maneuvers that keep the car away from the lane line and make sure that the car stays on the road throughout the simulation.

Architecture and Training Documentation

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

Very good job explaining your approach for solving the given problem. Well Done !!!

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.

The architecture of the implemented model is perfectly detailed and illustrated in the README. Great job here.

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.

- Great job once again. The training and dataset preprocessing are properly described in the README.
- Well done for sharing the images of your dataset and results of augmentation techniques here.

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).

Awesome work. The drive through the track is smooth.

DOWNLOAD PROJECT

RETURN TO PATH