**Meets Specifications**

Dear Excellent Student,  
Congratulations You nailed it.  
The car drives around extremely well on the track without any tire leaving the drivable portion of the track during the entire simulation.  
You are already using a very good set of techniques in your model and this shows a good understanding of the project. I see that the model architecture designed is based on the NVIDIA model. The approach used is well discussed in the write-up file and it was very explicit! This is great, as doing this provided you with a successful model. I did have a nice time reviewing your work. You can create a blog to share your work with the world and help others who might want to continue on this for further research with a real-world self-driving car. Continue working hard, you're doing a great job. Good luck with the next project.

I also wanted to ask if in theory it's useful to add the additional layer I did to the model's architecture or probably insignificant.

Depending upon how large your dataset is, the CNN architecture is implemented. Adding layers unnecessarily to any CNN will increase your number of parameters only for the smaller dataset, say in 1000s (total 1000). It’s true for some reasons that on adding more hidden layers, it will give more accuracy. This is true for larger datasets, as more layers with less stride factor will extract more features for your input data. In CNN, how you play with your architecture is completely dependent on what your requirement is and how your data is. There is a term called “Overfitting” and “Underfitting”. Increasing Unnecessary parameters will only overfit your network.

**More In-depth Knowledge**

If you have some time, please make sure to go through these various medium posts from students who succeeded in having the good solutions. You can read about the approach they took and the challenges faced.

* [Udacity Self-Driving Car Nanodegree Project 3 — Behavioral Cloning](https://medium.com/udacity/udacity-self-driving-car-nanodegree-project-3-behavioral-cloning-446461b7c7f9).
* [You don’t need lots of data! (Udacity Behavioral Cloning](https://medium.com/@fromtheast/you-dont-need-lots-of-data-udacity-behavioral-cloning-6d2d87316c52).
* [Make a car drive like yourself](https://medium.com/@ksakmann/behavioral-cloning-make-a-car-drive-like-yourself-dc6021152713).
* [A convolutional neural network for end-to-end driving in a simulator, using TensorFlow and Keras](https://medium.com/@mithi/cloning-driving-behavior-with-videogame-like-simulator-and-keras-e31129a8e3b6).

**Required Files**

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

Well done submitting all the required files as this permitted us to clearly evaluate your submission.

**Quality of Code**

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

Fantastic! The implemented model successfully operates the simulation.

**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.**

A good job was done in generating data needed during training. However, you could include comments in the code explaining your intended strategy where necessary.

**More In-depth Knowledge**

Generators can be a great way to work with large amounts of data. Instead of storing the preprocessed data in memory all at once, using a generator you can pull pieces of the data and process them on the fly only when you need them, which is much more memory-efficient. I suggest you use keras’ [fit\_generator method](https://keras.io/models/model/#fit_generator) that helps one to maximise memory usage by fitting the model with data yielded on a batch by batch basis  
Also make sure to check out [this Medium post](https://medium.com/@fromtheast/implement-fit-generator-in-keras-61aa2786ce98) on Python generators to get an idea of how they work.

More information on python generators can be gotten from the following links:

* [2 great benefits of Python generators (and how they changed me forever)](https://www.oreilly.com/ideas/2-great-benefits-of-python-generators-and-how-they-changed-me-forever).
* [Generators](https://wiki.python.org/moin/Generators).
* [How   and why   you should use Python Generators](https://medium.freecodecamp.org/how-and-why-you-should-use-python-generators-f6fb56650888).
* [Iterators & Generators](https://anandology.com/python-practice-book/iterators.html) .

**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.**

Indeed the neural network uses [Convolutional Neural Networks](http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/) with 6 convolutional layer, 5x5 and 3X3 filter sizes was deployed in this solution. The properties of this model are appropriate for this solution and lambda normalization was done on the training data at the beginning.

**More In-depth Knowledge**

Here are some other layers you can use to introduce nonlinearity into the model.

* [Max Pool](https://keras.io/layers/pooling/#maxpooling2d).
* [Average Pool](https://keras.io/layers/pooling/#averagepooling2d).
* [Batch Normalization](https://keras.io/layers/normalization/#batchnormalization).
* [Kernel, bias and activity regularizers](https://keras.io/regularizers/).
* [Dropout](https://keras.io/layers/core/#dropout).
* Weight initialization is also a good way of improving the accuracy of the model. Keras has a good [documentation](https://keras.io/initializers/) on **Usage of initializers** and I encourage you to learn it.  
  If you are wondering why we introduce nonlinearities, here is a good Quora discussion on [Why should we introduce nonlinearities in CNN models](https://www.quora.com/What-is-the-point-of-introducing-nonlinearity-like-ReLU-in-convolutional-NNs).

**Pro Tips**

Please, the following links may also be of great importance for further research:

* [Neural Network Architectures](https://towardsdatascience.com/neural-network-architectures-156e5bad51ba).
* [The 8 Neural Network Architectures Machine Learning Researchers Need to Learn](https://medium.com/@james_aka_yale/the-8-neural-network-architectures-machine-learning-researchers-need-to-learn-2f5c4e61aeeb).
* [Explore Network Architecture, Deep Learning, and more!](https://www.pinterest.com/pin/796292777834087333/).

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

Fantastic! The Train/validation/test splits have been logically used and dropout layers implemented to decrease overfitting.

**More In-depth Knowledge**

Apart from using dropouts, there are few other things you can try to reduce overfitting. I am listing them below.

* **Increasing training data**: It is not possible to increase training data for many of the applications but if you can then try increasing it at first place before trying any other solution.
* **Reduce network size**: Reducing network size will decrease the number of parameters and hence the network won’t be able to fit the training data arbitrarily well. The problem is that the optimal network size is not known and hence you can easily skip this step.
* **Regularization**:This is commonly used for overfitting as it does not require changing data or the network. Commonly used methods are L2 regularization, L1- regularization and dropout. The intuition is to make the network prefer to learn small weights keeping other parameters the same. More detail can be found [here](http://neuralnetworksanddeeplearning.com/chap3.html#overfitting_and_regularization). You should try this out first place.
* **Artificially expanding the training data**: your training data is too small and hence regularization may also not be useful. You can add more variations of the training data in the existing dataset.  
  I hope this helps if not you can have a full summary of these from this [Quora Discussions](https://www.quora.com/Convolutional-Neural-Networks-My-CNN-starts-overfitting-at-around-40-iterations-What-am-I-doing-wrong).

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

This solution made good use of Adam Optimizer. This is a good choice as it's configuration is way easier and it always tends to give superior results. You can read more about it from this site [Gentle Introduction to the Adam Optimization Algorithm for Deep Learning](https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/)

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

Good job collecting the training data by driving on the track. The training data was well chosen for this solution to keep the vehicle driving on the road. 

**Architecture and Training Documentation**

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

The approach used in this solution was well discussed in the write-up file and it was very explicit!

**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.**

Well done providing insightful details of the characteristics and qualities of the implemented architecture in the writeup file.

[A close up of a device

Description automatically generated](https://udacity-reviews-uploads.s3.us-west-2.amazonaws.com/_attachments/58227/1605966181/1.PNG)

**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.**

A good description of how the model was trained was given in the write-up file. The description includes how the dataset was generated and example images of the dataset were also included in the write-up file. Good job!

**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 simulation was very stable and the car didn't leave the track during the simulation which indicated excellence!