**Validating Your Network**

In order to validate your network, you'll want to compare model performance on the training set and a validation set. The validation set should contain image and steering data that was not used for training. A rule of thumb could be to use 80% of your data for training and 20% for validation or 70% and 30%. Be sure to randomly shuffle the data before splitting into training and validation sets.

If model predictions are poor on both the training and validation set (for example, mean squared error is high on both), then this is evidence of underfitting. Possible solutions could be to

* increase the number of epochs
* add more convolutions to the network.

When the model predicts well on the training set but poorly on the validation set (for example, low mean squared error for training set, high mean squared error for validation set), this is evidence of overfitting. If the model is overfitting, a few ideas could be to

* use dropout or pooling layers
* use fewer convolution or fewer fully connected layers
* collect more data or further augment the data set

Ideally, the model will make good predictions on both the training and validation sets. The implication is that when the network sees an image, it can successfully predict what angle was being driven at that moment.

**Testing Your Network**

Once you're satisfied that the model is making good predictions on the training and validation sets, you can test your model by launching the simulator and entering autonomous mode.

The car will just sit there until your Python server connects to it and provides it steering angles. Here’s how you start your Python server:

python drive.py model.h5

Once the model is up and running in drive.py, you should see the car move around (and hopefully not off) the track! If your model has low mean squared error on the training and validation sets but is driving off the track, this could be because of the data collection process. It's important to feed the network examples of good driving behavior so that the vehicle stays in the center and recovers when getting too close to the sides of the road.

## Lambda Layers

In Keras, [**lambda layers**](https://keras.io/layers/core/#lambda) can be used to create arbitrary functions that operate on each image as it passes through the layer.

In this project, a lambda layer is a convenient way to parallelize image normalization. The lambda layer will also ensure that the model will normalize input images when making predictions in drive.py.

That lambda layer could take each pixel in an image and run it through the formulas:

pixel\_normalized = pixel / 255

pixel\_mean\_centered = pixel\_normalized - 0.5

A lambda layer will look something like:

Lambda(lambda x: (x / 255.0) - 0.5)

Below is some example code for how a lambda layer can be used.

from keras.models import Sequential, Model

from keras.layers import Lambda

*# set up lambda layer*

model = Sequential()

model.add(Lambda(lambda x: (x / 255.0) - 0.5, input\_shape=(160,320,3)))

...

WEITER

### Flipping Images And Steering Measurements

A effective technique for helping with the left turn bias involves flipping images and taking the opposite sign of the steering measurement. For example:

import numpy as np

image\_flipped = np.fliplr(image)

measurement\_flipped = -measurement

The cv2 library also has similar functionality with the [**flip method**](http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#flip).