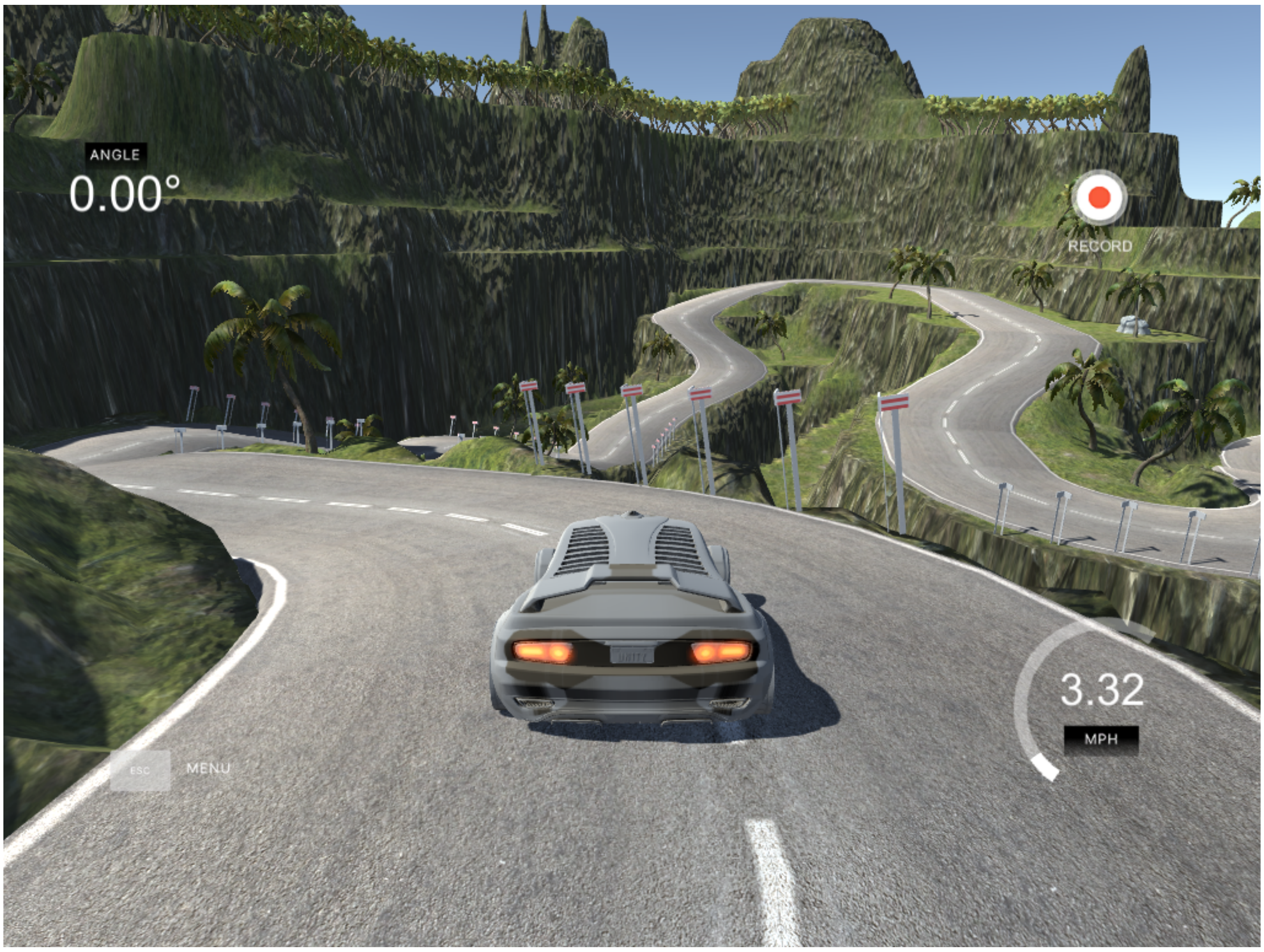
Resources For Completing the Project

This project will be completed and submitted via workspaces. The required files and resources, described below, are already present in Behavioral Cloning Project workspace.

- The GitHub repository has the following files:
  - drive.py: a Python script that you can use to drive the car autonomously, once your deep neural network model is trained
  - writeup_template.md: a writeup templates
  - video.py: a script that can be used to make a video of the vehicle when it is driving autonomously
- Sample driving data (optional) - if you choose to use a workspace, this is already included in your files. You can find it in /opt/carnd_p3/data/ (/opt is in the directory above /home, where your workspace is contained) when using GPU mode only. Note that if you choose to only use your own training data, you'll want to save it to a different directory to make sure they are not accidentally combined.
- a simulator, containing two tracks

We encourage you to drive the vehicle in training mode and collect your own training data, but we have also included sample driving data for the first track, which you can optionally use to train your network. You may need to collect additional data in order to get the vehicle to stay on the road. To review project completion requirements, please see the Project Rubric.

Running the Simulator



Here are the latest updates to the simulator:

1. Steering is controlled via position mouse instead of keyboard. This creates better angles for training. Note the angle is based on the mouse distance. To steer hold the left mouse button and move left or right. To reset the angle to 0 simply lift your finger off the left mouse button.

2. You can toggle record by pressing R, previously you had to click the record button (you can still do that).
3. When recording is finished, saves all the captured images to disk at the same time instead of trying to save them while the car is still driving periodically. You can see a save status and play back of the captured data.
4. You can takeover in autonomous mode. While W or S are held down you can control the car the same way you would in training mode. This can be helpful for debugging. As soon as W or S are let go autonomous takes over again.
5. Pressing the spacebar in training mode toggles on and off cruise control (effectively presses W for you).
6. Added a Control screen
7. Track 2 was replaced from a mountain theme to Jungle with free assets , Note the track is challenging
8. You can use brake input in drive.py by issuing negative throttle values

If you are interested here is the source code for the simulator repository

When you first run the simulator, you'll see a configuration screen asking what size and graphical quality you would like. We suggest running at the smallest size and the fastest graphical quality. We also suggest closing most other applications (especially graphically intensive applications) on your computer, so that your machine can devote its resources to running the simulator.

Training Mode

The next screen gives you two options: Training Mode and Autonomous Mode.

Select Training Mode.

The simulator will load and you will be able to drive the car like it's a video game. Try it!

You'll use autonomous mode in a later step, after you've used the data you collect here to train your neural network.

Collecting Training Data In order to start collecting training data, you'll need to do the following:

Enter Training Mode in the simulator.

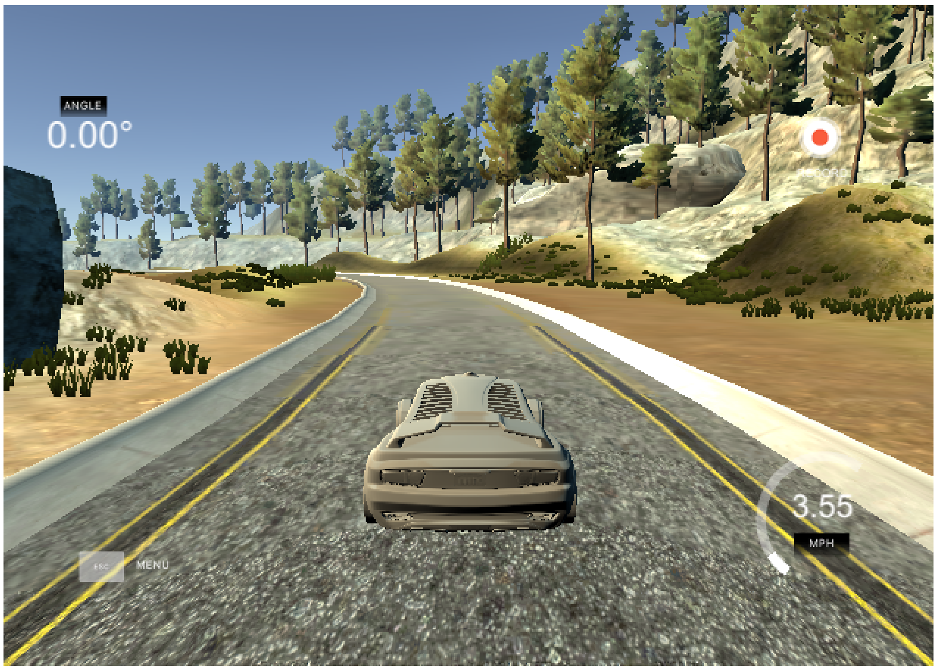
1. Start driving the car to get a feel for the controls.
2. When you are ready, hit the record button in the top right to start recording.
3. Continue driving for a few laps or till you feel like you have enough data.
4. Hit the record button in the top right again to stop recording.

Center Driving

So that the car drives down the center of the road, it's essential to capture center lane driving. Try driving around the track various times while staying as close to the middle of the track as possible even when making turns.

In the real world, the car would need to stay in a lane rather than driving down the center. But for the purposes of this project, aim for center of the road driving.

Example of Center Lane Driving



Strategies for Collecting Data

Now that you have driven the simulator and know how to record data, it's time to think about collecting data that will ensure a successful model. There are a few general concepts to think about that we will later discuss in more detail:

- the car should stay in the center of the road as much as possible
- if the car veers off to the side, it should recover back to center
- driving counter-clockwise can help the model generalize
- flipping the images is a quick way to augment the data
- collecting data from the second track can also help generalize the model
- we want to avoid overfitting or underfitting when training the model
- knowing when to stop collecting more data

If everything went correctly for recording data, you should see the following in the directory you selected:

1. IMG folder - this folder contains all the frames of your driving.
2. driving_log.csv - each row in this sheet correlates your image with the steering angle, throttle, brake, and speed of your car. You'll mainly be using the steering angle.

Center Image	Left Image	Right Image	Steering Angle	Throttle	Break	Speed
IMG/center_2016_10_21_	IMG/left_2016_10_21_17_	IMG/right_2016_10_21_1	0	0.09803098	0	0
IMG/center_2016_10_21_	IMG/left_2016_10_21_17_	IMG/right_2016_10_21_1	0	0.09803098	0	0.06057268

NOTE: cv2.imread will get images in BGR format, while drive.py uses RGB. In the video above one way you could keep the same image formatting is to do "image = ndimage.imread(current_path)" with "from scipy import ndimage" instead.

Note that the workspace for this project is equipped with a GPU which can be enabled for training your network. So, rather than working in a shell on AWS you can enable GPU mode and work in a workspace shell.

Training Your Network

Now that you have training data, it’s time to build and train your network!

Use Keras to train a network to do the following:

1. Take in an image from the center camera of the car. This is the input to your neural network.
2. Output a new steering angle for the car.

You don’t have to worry about the throttle for this project, that will be set for you.

Save your trained model architecture as model.h5 using model.save('model.h5').

Note that this video describes the process for running your network on a local machine with a model built using an AWS GPU instance. Since the project is in a GPU enabled workspace there is no need to download a GitHub repo (we have included the repo in the workspace) or transfer the model (the model should be trained in the workspace with GPU enabled).

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 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)))
...
```

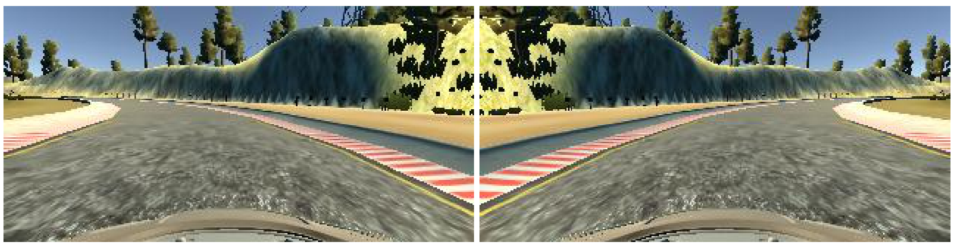
Note: The above video uses outdated Keras syntax - make sure you use the syntax applicable to Keras v2 in your own code!

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.



The simulator captures images from three cameras mounted on the car: a center, right and left camera. That's because of the issue of recovering from being off-center.

In the simulator, you can weave all over the road and turn recording on and off to record recovery driving. In a real car, however, that's not really possible. At least not legally.

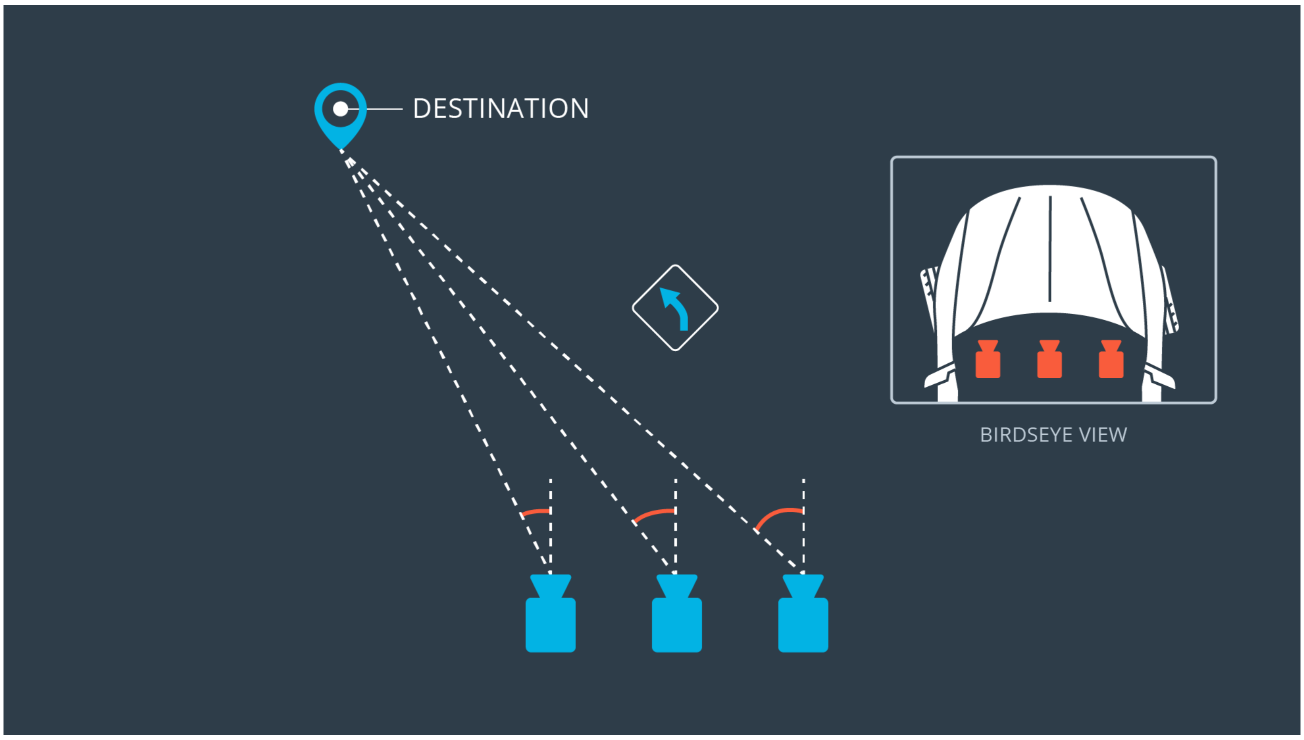
So in a real car, we'll have multiple cameras on the vehicle, and we'll map recovery paths from each camera. For example, if you train the model to associate a given image from the center camera with a left turn, then you could also train the model to associate the corresponding image from the left camera with a somewhat softer left turn. And you could train the model to associate the corresponding image from the right camera with an even harder left turn.

In that way, you can simulate your vehicle being in different positions, somewhat further off the center line. To read more about this approach, see this paper by our friends at NVIDIA that makes use of this technique.

Explanation of How Multiple Cameras Work

The image below gives a sense for how multiple cameras are used to train a self-driving car. This image shows a bird's-eye perspective of the car. The driver is moving forward but wants to turn towards a destination on the left.

From the perspective of the left camera, the steering angle would be less than the steering angle from the center camera. From the right camera's perspective, the steering angle would be larger than the angle from the center camera. The next section will discuss how this can be implemented in your project although there is no requirement to use the left and right camera images.



Multiple Cameras in This Project

For this project, recording recoveries from the sides of the road back to center is effective. But it is also possible to use all three camera images to train the model. When recording, the simulator will simultaneously save an image for the left, center and right cameras. Each row of the csv log file, driving_log.csv, contains the file path for each camera as well as information about the steering measurement, throttle, brake and speed of the vehicle.

Here is some example code to give an idea of how all three images can be used:

```
with open(csv_file, 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        steering_center = float(row[3])

        # create adjusted steering measurements for the side camera images
        correction = 0.2 # this is a parameter to tune
        steering_left = steering_center + correction
        steering_right = steering_center - correction

        # read in images from center, left and right cameras
        path = "..." # fill in the path to your training IMG directory
        img_center = process_image(np.asarray(Image.open(path + row[0])))
        img_left = process_image(np.asarray(Image.open(path + row[1])))
        img_right = process_image(np.asarray(Image.open(path + row[2])))

        # add images and angles to data set
        car_images.extend(img_center, img_left, img_right)
        steering_angles.extend(steering_center, steering_left, steering_right)
```

During training, you want to feed the left and right camera images to your model as if they were coming from the center camera. This way, you can teach your model how to steer if the car drifts off to the left or the right.

Figuring out how much to add or subtract from the center angle will involve some experimentation.

During prediction (i.e. "autonomous mode"), you only need to predict with the center camera image.

It is not necessary to use the left and right images to derive a successful model. Recording recovery driving from the sides of the road is also effective.

The cameras in the simulator capture 160 pixel by 320 pixel images.

Not all of these pixels contain useful information, however. In the image above, the top portion of the image captures trees and hills and sky, and the bottom portion of the image captures the hood of the car.

Your model might train faster if you crop each image to focus on only the portion of the image that is useful for predicting a steering angle.

Cropping2D Layer

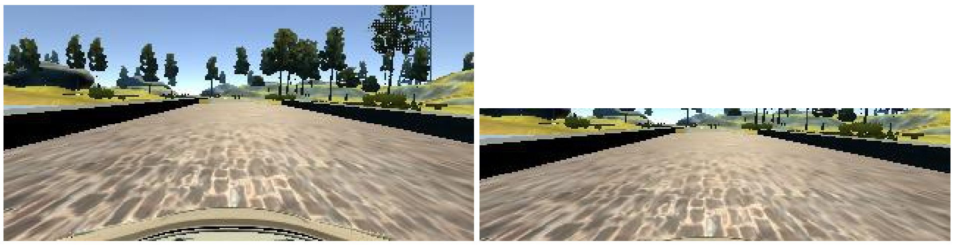
Keras provides the Cropping2D layer for image cropping within the model. This is relatively fast, because the model is parallelized on the GPU, so many images are cropped simultaneously.

By contrast, image cropping outside the model on the CPU is relatively slow.

Also, by adding the cropping layer, the model will automatically crop the input images when making predictions in drive.py.

The Cropping2D layer might be useful for choosing an area of interest that excludes the sky and/or the hood of the car.

Here is an example of an input image and its cropped version after passing through a Cropping2D layer:



Cropping Layer Code Example

```python
from keras.models import Sequential, Model
from keras.layers import Cropping2D
import cv2

# set up cropping2D layer
model = Sequential()
model.add(Cropping2D(cropping=((50,20), (0,0)), input_shape=(160,320,3)))
...
```

The example above crops:

- 50 rows pixels from the top of the image
- 20 rows pixels from the bottom of the image
- 0 columns of pixels from the left of the image
- 0 columns of pixels from the right of the image

Recovery Laps

If you drive and record normal laps around the track, even if you record a lot of them, it might not be enough to train your model to drive properly.

Here's the problem: if your training data is all focused on driving down the middle of the road, your model won't ever learn what to do if it gets off to the side of the road. And probably when you run your model to predict steering measurements, things won't go perfectly and the car will wander off to the side of the road at some point.

So you need to teach the car what to do when it's off on the side of the road.

One approach might be to constantly wander off to the side of the road and then steer back to the middle.

A better approach is to only record data when the car is driving from the side of the road back toward the center line.

So as the human driver, you're still weaving back and forth between the middle of the road and the shoulder, but you need to turn off data recording when you weave out to the side, and turn it back on when you steer back to the middle.

Driving Counter-Clockwise

Track one has a left turn bias. If you only drive around the first track in a clock-wise direction, the data will be biased towards left turns. One way to combat the bias is to turn the car around and record counter-clockwise laps around the track. Driving counter-clockwise is also like giving the model a new track to learn from, so the model will generalize better.

Using Both Tracks

If you end up using data from only track one, the convolutional neural network could essentially memorize the track. Consider using data from both track one and track two to make a more generalized model.

Collecting Enough Data

How do you know when you have collected enough data? Machine learning involves trying out ideas and testing them to see if they work. If the model is over or underfitting, then try to figure out why and adjust accordingly.

Since this model outputs a single continuous numeric value, one appropriate error metric would be mean squared error. If the mean squared error is high on both a training and validation set, the model is underfitting. If the mean squared error is low on a training set but high on a validation set, the model is overfitting. Collecting more data can help improve a model when the model is overfitting.

What if the model has a low mean squared error on both the training and validation sets, but the car is falling off the track?

Try to figure out the cases where the vehicle is falling off the track. Does it occur only on turns? Then maybe it's important to collect more turning data. The vehicle's driving behavior is only as good as the behavior of the driver who provided the data.

Here are some general guidelines for data collection:

- two or three laps of center lane driving
- one lap of recovery driving from the sides
- one lap focusing on driving smoothly around curves

Outputting Training and Validation Loss Metrics

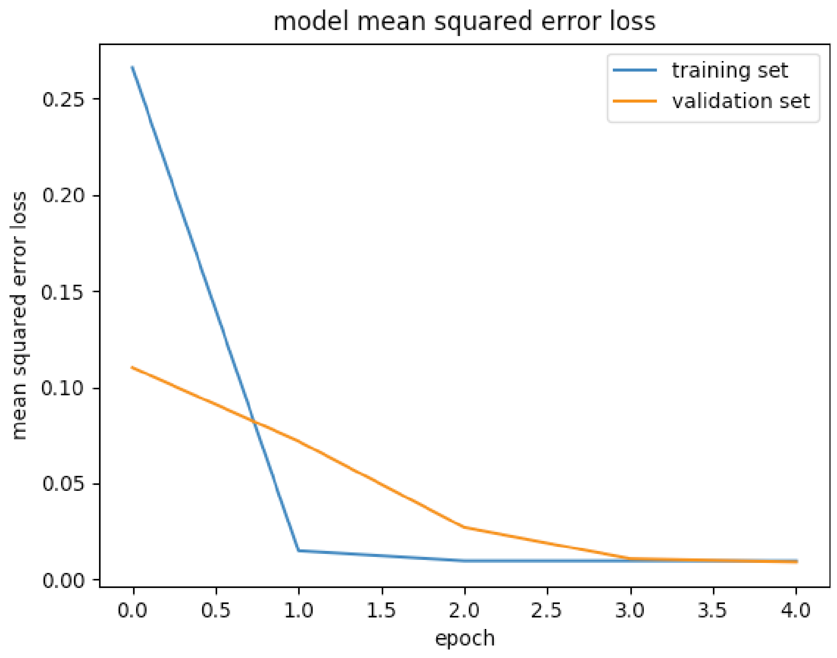
In Keras, the model.fit() and model.fit_generator() methods have a verbose parameter that tells Keras to output loss metrics as the model trains. The verbose parameter can optionally be set to verbose = 1 or verbose = 2.

Setting model.fit(verbose = 1) will

output a progress bar in the terminal as the model trains. output the loss metric on the training set as the model trains. output the loss on the training and validation sets after each epoch. With model.fit(verbose = 2), Keras will only output the loss on the training set and validation set after each epoch.

Model History Object

When calling model.fit() or model.fit_generator(), Keras outputs a history object that contains the training and validation loss for each epoch. Here is an example of how you can use the history object to visualize the loss:



The following code shows how to use the model.fit() history object to produce the visualization.

```
from keras.models import Model
import matplotlib.pyplot as plt

history_object = model.fit_generator(train_generator, samples_per_epoch =
    len(train_samples), validation_data =
    validation_generator,
    nb_val_samples = len(validation_samples),
    nb_epoch=5, verbose=1)

### print the keys contained in the history object
print(history_object.history.keys())

### plot the training and validation loss for each epoch
plt.plot(history_object.history['loss'])
plt.plot(history_object.history['val_loss'])
plt.title('model mean squared error loss')
plt.ylabel('mean squared error loss')
plt.xlabel('epoch')
plt.legend(['training set', 'validation set'], loc='upper right')
plt.show()
```

How to Use Generators

The images captured in the car simulator are much larger than the images encountered in the Traffic Sign Classifier Project, a size of 160 x 320 x 3 compared to 32 x 32 x 3. Storing 10,000 traffic sign images would take about 30 MB but storing 10,000 simulator images would take over 1.5 GB. That's a lot of memory! Not to mention that preprocessing data can change data types from an int to a float, which can increase the size of the data by a factor of 4.

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.

A generator is like a coroutine, a process that can run separately from another main routine, which makes it a useful Python function. Instead of using return, the generator uses yield, which still returns the desired output values but saves the current values of all the generator's variables. When the generator is called a second time it re-starts right after the yield statement, with all its variables set to the same values as before.

Below is a short quiz using a generator. This generator appends a new Fibonacci number to its list every time it is called. To pass, simply modify the generator's yield so it returns a list instead of 1. The result will be we can get the first 10 Fibonacci numbers simply by calling our generator 10 times. If we need to go do something else besides generate Fibonacci numbers for a while we can do that and then always just call the generator again whenever we need more Fibonacci numbers.

```
def fibonacci():
    numbers_list = []
    while 1:
        if(len(numbers_list) < 2):
            numbers_list.append(1)
        else:
            numbers_list.append(numbers_list[-1] + numbers_list[-2])
        yield numbers_list

our_generator = fibonacci()
my_output = []

for i in range(10):
    my_output = (next(our_generator))

print(my_output)
```

Here is an example of how you could use a generator to load data and preprocess it on the fly, in batch size portions to feed into your Behavioral Cloning model .

```

import os
import csv

samples = []
with open('./driving_log.csv') as csvfile:
    reader = csv.reader(csvfile)
    for line in reader:
        samples.append(line)

from sklearn.model_selection import train_test_split
train_samples, validation_samples = train_test_split(samples, test_size=0.2)

import cv2
import numpy as np
import sklearn

def generator(samples, batch_size=32):
    num_samples = len(samples)
    while 1: # Loop forever so the generator never terminates
        shuffle(samples)
        for offset in range(0, num_samples, batch_size):
            batch_samples = samples[offset:offset+batch_size]

            images = []
            angles = []
            for batch_sample in batch_samples:
                name = './IMG/'+batch_sample[0].split('/')[-1]
                center_image = cv2.imread(name)
                center_angle = float(batch_sample[3])
                images.append(center_image)
                angles.append(center_angle)

            # trim image to only see section with road
            X_train = np.array(images)
            y_train = np.array(angles)
            yield sklearn.utils.shuffle(X_train, y_train)

# Set our batch size
batch_size=32

# compile and train the model using the generator function
train_generator = generator(train_samples, batch_size=batch_size)
validation_generator = generator(validation_samples, batch_size=batch_size)

ch, row, col = 3, 80, 320  # Trimmed image format

model = Sequential()
# Preprocess incoming data, centered around zero with small standard deviation
model.add(Lambda(lambda x: x/127.5 - 1.,
        input_shape=(ch, row, col),
        output_shape=(ch, row, col)))
model.add(... finish defining the rest of your model architecture here ...)

model.compile(loss='mse', optimizer='adam')
model.fit_generator(train_generator, /
            steps_per_epoch=ceil(len(train_samples)/batch_size), /
            validation_data=validation_generator, /
            validation_steps=ceil(len(validation_samples)/batch_size), /
            epochs=5, verbose=1)
```

Recording Video in Autonomous Mode

Because your hardware setup might be different from a reviewer's hardware setup, driving behavior could be different on your machine than on the reviewer's. To help with reviewing your submission, we require that you submit a video recording of your vehicle driving autonomously around the track. The video should include at least one full lap around the track. Keep in mind the rubric specifications:

"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)."

In the GitHub repo, we have included a file called video.py, which can be used to create the video recording when in autonomous mode.

The README file in the GitHub repo contains instructions about how to make the video recording. Here are the instructions as well:

```
python drive.py model.h5 run1
```

The fourth argument, run1, is the directory in which to save the images seen by the agent. If the directory already exists, it'll be overwritten.

```
ls run1

[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_424.jpg
[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_451.jpg
[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_477.jpg
[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_528.jpg
[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_573.jpg
[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_618.jpg
[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_697.jpg
[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_723.jpg
[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_749.jpg
[2017-01-09 16:10:23 EST]  12KiB 2017_01_09_21_10_23_817.jpg
...
```

The image file name is a timestamp of when the image was seen.. This information is used by video.py to create a chronological video of the agent driving.

Using video.py

```
python video.py run1
```

Creates a video based on images found in the run1 directory. The name of the video will be the name of the directory followed by '.mp4', so, in this case the video will be run1.mp4.

Optionally, one can specify the FPS (frames per second) of the video:

```
python video.py run1 --fps 48
```

The video will run at 48 FPS. The default FPS is 60.

Running and submitting the Behavioral Cloning Project in workspaces

Behavioral Cloning project workspace

This workspace is designed to be a simple, easy to use environment in which you can code and run the Behavioral Cloning project. The project repo is already included so there is no need of downloading the project repo. For tips on workspaces use, please review the earlier Workspaces lesson.

Accessing and using the workspace

Go to the workspace in the next lesson. You will be asked if you want to use a GPU-enabled workspace. Click YES to run in GPU-enabled mode only if you:

- Want to collect training data.
- Want to test your solution.
- Want to train your neural network using GPU (network training will be challenging without using a GPU).

Click NO to run in GPU-enabled mode only if you:

- Want to code your solution on the text editor.

Simulator

To run the simulator click on the button Simulator on the bottom right, then go to the newly opened tab in your browser and double-click on the simulator icon in the desktop. This will open the simulator's configuration window, finally, run and test your code or gather data. Note: If you get an error when clicking on the Simulator button, please check that you have enabled the GPU.

Simulator

Project submission when using the workspace

To submit your project, just click the button SUBMIT PROJECT and follow the instructions to submit!

Make sure that the following files are present in the workspace and follow the naming conventions below to make it easy for reviewers to find the right files:

- model.py - The script used to create and train the model.
- drive.py - The script to drive the car. You can feel free to resubmit the original drive.py or make modifications and submit your modified version.
- model.h5 - The saved model. Here is the documentation explaining how to create this file.
- writeup_report as a markdown or pdf file. It should explain the structure of your network and training approach. The write-up must also include examples of images from the dataset in the discussion of the characteristics of the dataset. While we recommend using English for good practice, writing in any language is acceptable (reviewers will translate). There is no minimum word count so long as there are complete descriptions of the problems and the strategies. See the rubric and the writeup_template.md for more details about the expectations.
- video.mp4 - A video recording of your vehicle driving autonomously at least one lap around the track.
- *IMPORTANT**: Make sure that your directory /home/workspace/ doesn't include your training images since the Reviews system is limited to 10,000 files and 500MB. If you have more than that you will get the following error too many files when trying to submit. To fix this error just move your images to directory below~/opt/.

Things to keep in mind

- If you leave your workspace unattended, it will time out and need to be refreshed. Your most recent work will be restored, but the list of open files or any running shell sessions will not be restored. Also, the data collected by you will not be stored, and you will have to drive the car in the simulator again to collect the data.
- Try to not save data directly within /home/workspace/ because this directly only have the storage capacity of 3 GB, and storing a large amount of data within this directory, can freeze your workspace. If you need to save a large amount of data use /opt/ but all the data saved there will be lost once you leave the classroom.
- If you want to avoid collecting data every time you refresh workspace, the data can be stored in your local environment (Dropbox, GitHub, Google Drive, etc.) and can be pulled over to the workspace, using the command 'wget' command.

  ## Common Issues

- "No VNC" or "Serv ice is not running" error when launching simulator - this is related to either A) not having the workspace GPU enabled (the simulator needs a GPU to run), or B) the web browser being used. Safari is likely to produce this error, while Chrome should run the simulator fine.
- "No session for PID" error when launching simulator - when the desktop simulator is opened, sometimes a "PID error" window will appear. This error does not impact the simulator itself and can be safely ignored or click OK, it is harmless.
- Missing simulator icon - the simulator icon may fail to appear after a short wait within the Linux Desktop. If this is the case, click on the Terminal icon in the Desktop, and the simulator icon will typically appear. Please note that you still will use the actual Terminal within the primary workspace, and not the one in the Desktop.

Commit to GitHub

Students are highly encouraged to commit their project to a GitHub repo. To do this, you must change the upstream of the current repository and add your credentials. We have supplied a bash script to help you do this. Please open up a terminal, navigate to the project repository, and enter: ./set_git.sh, then follow the prompts. This will set the upstream remote to your own repository and add your email and username to the git configuration. At this time we are not configuring passwords, so you will need to enter your username and password for each push. Since credentials are not persistent, it will be necessary to run this script each time you open, refresh, or reset the workspace.

Running and submitting the Behavioral Cloning Project in your local machine

Project Submission

For this project, a reviewer will be testing the model that you generated on the first test track (the one to the left in the track selection options). Whether you decide to zip up your submission or submit a GitHub repo, please follow the naming conventions below to make it easy for reviewers to find the right files:

- model.py - The script used to create and train the model.
- drive.py- The script to drive the car. You can feel free to submit the original drive.py or make modifications and submit your modified version.
- model.h5 - The saved model. Here is the documentation explaining how to create this file.
- writeup_report as a markdown or pdf file. It should explain the structure of your network and training approach. The writeup must also include examples of images from the dataset in the discussion of the characteristics of the dataset. While we recommend using English for good practice, writing in any language is acceptable (reviewers will translate). There is no minimum word count so long as there are complete descriptions of the problems and the strategies. See the rubric and the writeup_template.md for more details about the expectations.
- video.mp4 - A video recording of your vehicle driving autonomously at least one lap around the track. As a reminder all of the files we provide for this project (rubric, simulator, GitHub repository, track 1 sample data) can be found in the lecture slide titled "Project Resources".

  ### Further Help

- Use a generator (such as the fit_generator function provided by Keras). Here is some documentation that will help.
- Paul Heraty, a student in the October cohort, has written a helpful guide for those of you looking for some hints and advice.
- You can use our sample data for track 1 (see the "Project Resources" lecture for the link)
- Keep in mind that training images are loaded in BGR colorspace using cv2 while drive.py load images in RGB to predict the steering angles.

Using GitHub and Creating Effective READMEs

If you are unfamiliar with GitHub , Udacity has a brief GitHub tutorial to get you started. Udacity also provides a more detailed free course on git and GitHub. To learn about README files and Markdown, Udacity provides a free course on READMEs, as well. GitHub also provides a tutorial about creating Markdown files.

Project Support

If you are stuck or having difficulties with the project, don't lose hope! Ask (and answer!) questions on Knowledge tagged with the project name or in the Behavioral Cloning channel in your Student Hub. We also have a previously recorded project Q&A that you can watch here!