Udacity self driving car behavior cloning project3 in a video game like simulator

Project has following python package dependencies

argparse base64 json numpy socketio eventlet PIL flask pandas matplotlib scipy keras tensorflow

Project Codes

I have added my own helper utility:

- hDrvUtils.p which has file IO and image resizing script, as well as randomized camera angle fetcher with steering coefficient compensator.

The above script is provided along with project required:

1. model.py - The script used to create and train the model.
2. drive.py - The script to drive the car.
   Slightly modified to take in 64x64 image
3. model.json - The model architecture.
4. model_up1_0.h5 - The model weights.

   The json & h5 are trained model saved from model.py The trained model managed to go through the train track w/o being off course. The data used were 300MB or so provided by udacity & combined with 100MB of my own twice around the track. No particular part of track were emphasized (which was recommended by some other students in the FAQ). This & other techniques like change in lighting, sheering, rotating, etc... suggested might improve the poor showing on the validation track, where it just hits a portion of dark wall & do NOT go anywhere after first 20 sec.

   This will be worked on for future reference.

•Based on the 1st reviewer's response, I have created 3 different csv files

- driving_mod.csv, driving_tst.csv, & driving_val.csv

•Now each of those files are completely separated

•Also, dropout line were added back in

- model.add(Dropout(.5))

•When tested with above changes

- Epoch 7/7
- 20736/20083 [==============================] - 37s - loss: 0.0176 - val_loss: 0.0276
- ----------
- In [6]: evaluate = model.evaluate_generator(test_gen, 1008)
- In [8]: print("model.evaluate", evaluate)
- model.evaluate 0.028620492667

•The test data evaluation came up 0.029 which is very similar to the validation loss.

The ML model followed nvidia End to End Learning for Self-Driving Cars https://arxiv.org/pdf/1604.07316v1.pdf

Train the weights of our network to min(MSE) the steering command output by the network & the command of either the human driver, or the adjusted steering command for off-center and rotated images 9 layers, including a normalization layer, 5 convolutional layers & 3 fully connected layers.

1.L Image normalization hardcoded using x/127.5 -1 (64x64x3)

2. ~4L 2x2 stride & 5x5 kernel CNN - starts out with Filters [24, 36, 48, 64, 64]

5. ,6L non-strided CNN 3x3 Kernel

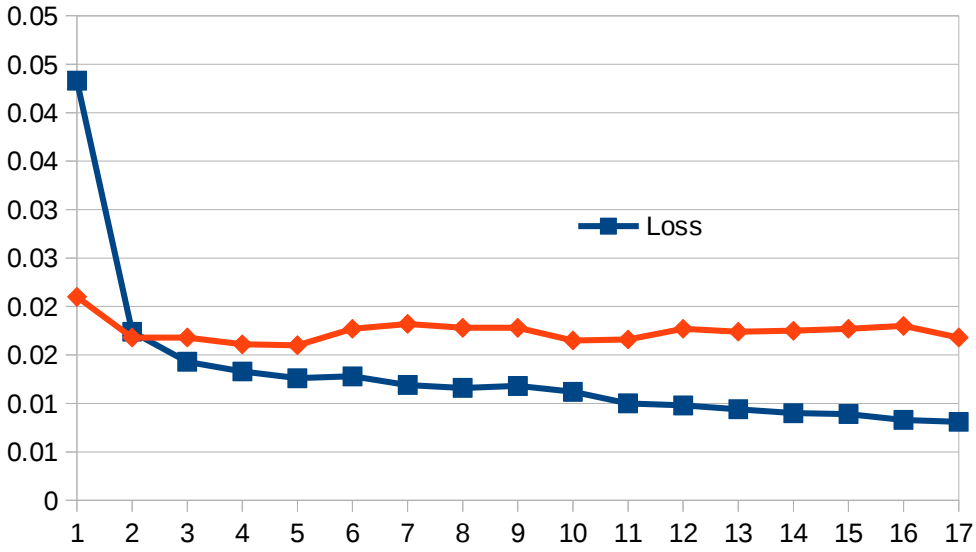
7. ~9L fully conn = output inverse turning radius. [1164, 100, 40, 10]

And, optimized with Adam & compiled

I have used 7 epochs with image sized reduced to 64x64 using a rather powerful nVidia GT 1060 graphics card. Each iteration took about 5 minutes to train ** other Notables ** The data collected using keyboard is erratic as mentioned by others, but data provided is too smooth. Combining both had definite desired effect. Code itself is pretty much self documenting. Not PEP8, but similar to technique I use at work.

- The suggested L1 & L2 regularization will be tested out a later date
- Epoch of 7 was chosen as validation did not improve beyond that point. 2nd ~3rd epoch actually achieves same validation results as beyond, which suggested that without more fundamental changes to the model adding more epochs is unlikely to improve the results and I may be just over fitting the data. Still, test set having higher than validation set loss was encouraging.

    - 20736/20083 [==============================] - 281s - loss: 0.0433 - val_loss: 0.0210
    - Epoch 2/17
    - 20736/20083 [==============================] - 84s - loss: 0.0174 - val_loss: 0.0168
    - Epoch 3/17
    - 20736/20083 [==============================] - 51s - loss: 0.0143 - val_loss: 0.0168
    - Epoch 4/17
    - 20736/20083 [==============================] - 42s - loss: 0.0133 - val_loss: 0.0161
    - Epoch 5/17
    - 20736/20083 [==============================] - 39s - loss: 0.0126 - val_loss: 0.0160
    - Epoch 6/17
    - 20736/20083 [==============================] - 38s - loss: 0.0128 - val_loss: 0.0177
    - Epoch 7/17
    - 20736/20083 [==============================] - 37s - loss: 0.0119 - val_loss: 0.0182
    - Epoch 8/17
    - 20736/20083 [==============================] - 37s - loss: 0.0116 - val_loss: 0.0178
    - Epoch 9/17
    - 20736/20083 [==============================] - 37s - loss: 0.0118 - val_loss: 0.0178
    - Epoch 10/17
    - 20736/20083 [==============================] - 37s - loss: 0.0112 - val_loss: 0.0165
    - Epoch 11/17
    - 20736/20083 [==============================] - 37s - loss: 0.0100 - val_loss: 0.0166
    - Epoch 12/17
    - 20736/20083 [==============================] - 37s - loss: 0.0098 - val_loss: 0.0177
    - Epoch 13/17
    - 20736/20083 [==============================] - 37s - loss: 0.0094 - val_loss: 0.0174
    - Epoch 14/17
    - 20736/20083 [==============================] - 37s - loss: 0.0090 - val_loss: 0.0175
    - Epoch 15/17
    - 20736/20083 [==============================] - 37s - loss: 0.0089 - val_loss: 0.0177
    - Epoch 16/17
    - 20736/20083 [==============================] - 37s - loss: 0.0083 - val_loss: 0.0180
    - Epoch 17/17
    - 20736/20083 [==============================] - 37s - loss: 0.0081 - val_loss: 0.0168
    - model.evaluate 0.015556178987



- I have used both center, left, & right images with steering coefficient to after much trial and error. Center image alone failed to produce ran around the track miserably, suggesting that additional perspective improves learning rate.
- The reduced smaller image size does an excellent job of reducing the training time cycle. And even Nvidia team used similar approach and reduced the image size. I started with original images, but quickly found the task daunting, not to mention lower image quality meant that I could use various image sizes for recording.
- This probably means taking sky and hood out for the training should improve learning rate as well. Will attempt this after below issue is solved.

- **Unfortunately, rerunning the whole video second time (post review) ran into a critical problem.**
  - I get Trace back:

```
$ python drive.py combDir/model_works.h5
Using TensorFlow backend.
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcublas.so.8.0 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcudnn.so.5 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcufft.so.8.0 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcurand.so.8.0 locally
Traceback (most recent call last):
  File "drive.py", line 74, in <module>
    model = model_from_json(json.load(jfile,encoding="ISO-8859-1"))
  File "/opt/miniconda3/envs/Py3/lib/python3.5/json/__init__.py", line 265, in load
    return loads(fp.read(),
  File "/opt/miniconda3/envs/Py3/lib/python3.5/codecs.py", line 321, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0x89 in position 0: invalid start byte
```

- This is even with previously working json. This is likely related to upgrading the python packages since the last run. I will post this issue at the forum and see if I could get answers.
- The above changes (hopefully) are enough to address the issues raised during the 1st review.
- I will address the encoding issue and reattempt, but this is becoming a time consuming issue that seems to evade efforts to resolve it.