Behavior Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results in this report

Project Files

My project includes the following files:

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- nvidia.h5 containing a trained nvidia[1] convolution neural network
- lenet.h5 containing a trained LeNet[2] convolution neural network
- A recorded video of the autonomous driving (using nvidia model)
- A recorded video of the autonomous driving (using lenet model)

Code and its usage

The car can be driven autonomously around the track by executing

```
python drive.py nvidial.h5
```

Or (for LeNet model)

```
python drive.py lenet.h5
```

The model.py file has the code, which uses Keras to build the models. It has 3 functions each to build either a basic, LeNet or Nvida model. It can be used to generate and train new models. It has helpful comments explaining the broader logic.

Also, you can use the command below to pass the arguments:

```
python model.py -h
```

Typical usage:

```
python model.py ./recordings/data/ 3 #For Nvidia
```

```
python model.py ./recordings/data/ 2 #For LeNet
```

Model Architecture and Training Strategy

I started following the udacity given instructions to arrive at my final models, which worked. I started with a basic model of just one fully connected layer. Subsequently I tried LeNet based ans Nvidia models.

The following are common to both the models.

Learning rate:	Adam optimizer was used. Which takes care of reducing learning rate.
Overfitting prevention:	Relu activation as well as maxpool layers were used to prevent overfitting
Relu	Relu layers are used clubbed after the conv layers for introducing non-linearity

Training data

My game playing skills are below par. But thankfully my family members have good skills there. So I requested them to drive the car in the simulator for me. I reviewed their driving and ensured it was recorded only for good drives. I gave tolerance to occasional swerving within the lane, as that would be the case in real world too.
They all saved in separate folders. I made changes to model.py to incorporate loading from multiple folders. Described in more details below.

Architecture and Training Documentation

Solution Design Approach

The overall strategy for deriving a model architecture was to be able to test drive on at least track 1 without going off the road. For that first I wanted to bring validation loss below or around 1 or 2 %.

I quickly rejected the basic model, as it was was too small to learn anything. I tried with LeNet. And with some modifications and data, it could work reasonably well. As can be seen in the video_lenet.mp4.

I had to augment data using all the suggested strategies:

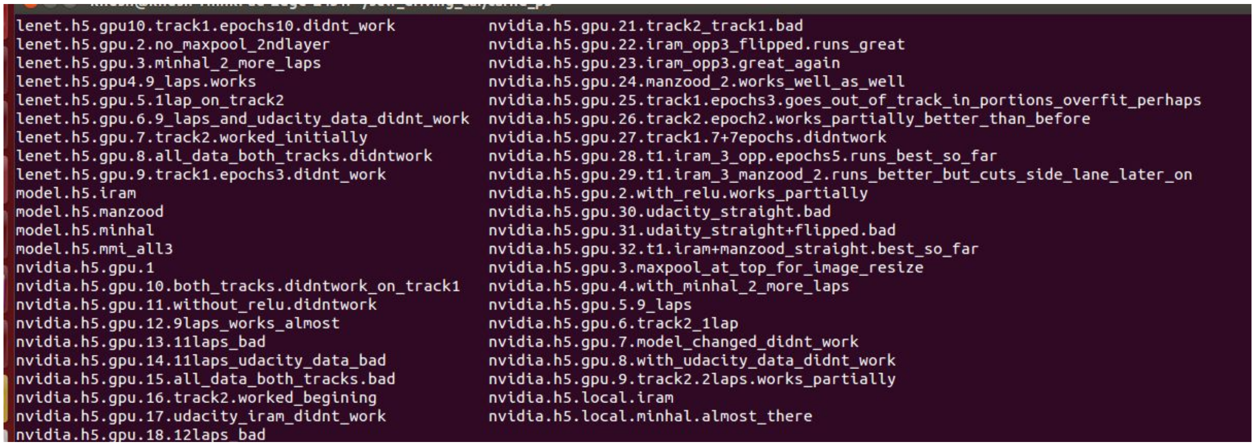
- Using left and right images with a correction(0.2)
- Flipping the images (TODO:see sample flipped image figure below)

Then I tried the same with Nvidia model. And that also worked. And worked better! The resultant automated video of it, is much better driving than LeNet. And cuts the lane only at one point. (I am sure I can further improve it, but can't do it for this project submission)

The final step was to run the simulator to see how well the car was driving around track one.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road for both LeNet and Nvidia models.

Of course to achieve that state, I had to run various combinations on the GPU for at least around 50 times. Please see below as snapshot of my local disk, for the saved models:



```
lenet.h5.gpu10.track1.epochs10.didnt_work
lenet.h5.gpu.2.no_maxpool_2ndlayer
lenet.h5.gpu.3.minhal_2_more_laps
lenet.h5.gpu4.9_laps.works
lenet.h5.gpu.5.1lap_on_track2
lenet.h5.gpu.6.9_laps_and_udacity_data_didnt_work
lenet.h5.gpu.7.track2.worked_initially
lenet.h5.gpu.8.all_data_both_tracks.didntwork
lenet.h5.gpu.9.track1.epochs3.didnt_work
model.h5.iram
model.h5.manzood
model.h5.minhal
model.h5.mmi_all3
nvidia.h5.gpu.1
nvidia.h5.gpu.10.both_tracks.didntwork_on_track1
nvidia.h5.gpu.11.without_relu.didntwork
nvidia.h5.gpu.12.9laps_works_almost
nvidia.h5.gpu.13.11laps_bad
nvidia.h5.gpu.14.11laps_udacity_data_bad
nvidia.h5.gpu.15.all_data_both_tracks.bad
nvidia.h5.gpu.16.track2.worked_begining
nvidia.h5.gpu.17.udacity_iram_didnt_work
nvidia.h5.gpu.18.12laps_bad
nvidia.h5.gpu.21.track2_track1.bad
nvidia.h5.gpu.22.iram_opp3_flipped.runs_great
nvidia.h5.gpu.23.iram_opp3.great_again
nvidia.h5.gpu.24.manzood_2.works_well_as_well
nvidia.h5.gpu.25.track1.epochs3.goes_out_of_track_in_portions_overfit_perhaps
nvidia.h5.gpu.26.track2.epoch2.works_partially_better_than_before
nvidia.h5.gpu.27.track1.7+7epochs.didntwork
nvidia.h5.gpu.28.t1.iram_3_opp.epochs5.runs_best_so_far
nvidia.h5.gpu.29.t1.iram_3_manzood_2.runs_better_but_cuts_side_lane_later_on
nvidia.h5.gpu.2.with_relu.works_partially
nvidia.h5.gpu.30.udacity_straight.bad
nvidia.h5.gpu.31.udaity_straight+flipped.bad
nvidia.h5.gpu.32.t1.iram+manzood_straight.best_so_far
nvidia.h5.gpu.3.maxpool_at_top_for_image_resize
nvidia.h5.gpu.4.with_minhal_2_more_laps
nvidia.h5.gpu.5.9_laps
nvidia.h5.gpu.6.track2_1lap
nvidia.h5.gpu.7.model_changed_didnt_work
nvidia.h5.gpu.8.with_udacity_data_didnt_work
nvidia.h5.gpu.9.track2.2laps.works_partially
nvidia.h5.local.iram
nvidia.h5.local.minhal.almost_there
```

Final Model Architecture

I tried successfully with two types of Models. One based on LeNet and other on Nvidia.

LeNet based model

Layer (type)	Output Shape	Param #	Details
lambda_1 (Lambda)	(None, 160, 320, 3)	0	None denotes variable no. of inputs. Also we use the layer to **normalize** the inputs (-0.5 to 0.5)
maxpooling2d_1 (MaxPooling2D)	(None, 80, 160, 3)	0	Maxpool later used upfront, as an alternative **resize**(scaling down) of the image by 2x2
cropping2d_1 (Cropping2D)	(None, 45, 160, 3)	0	Note the size change because of **cropping** out the top and bottom rows.
convolution2d_1 (Convolution2D)	(None, 43, 158, 6)	168	1st conv layer. 6 Filters of 3x3 Activation: Relu
maxpooling2d_2 (MaxPooling2D)	(None, 21, 79, 6)	0	Serves to reduce over fitting. Managing the size of the model.
convolution2d_2 (Convolution2D)	(None, 17, 75, 16)	2416	2nd conv layer. 16 Filters of 5x5

			Activation: Relu
maxpooling2d_3 (MaxPooling2D)	(None, 8, 37, 16)	0	Maxpool layer after conv layer
flatten_1 (Flatten)	flatten_1 (Flatten)	0	For flattening the data in a single row (to serve as inputs to FC layer ahead)
dense_1 (Dense)	(None, 300)	1421100	1st FC layer
dense_2 (Dense)	(None, 1)	301	2nd and final FC layer
Total params:	1,423,985		

Nvidia based model

Layer (type)	**Output Shape**	**Param #**	**Details**
Input layer	(None, 160, 320)	0	Images are of size 160x320
cropping2d_1 (Cropping2D)	(None, 90, 320, 3)	0	We improvise. At the outset I **crop** out the unneeded portions of the image i.e. top 50 and bottom 20 rows
lambda_1 (Lambda)	(None, 66, 235, 3)	0	We **resize** the image further to match the input height of Nvidia paper's architecture
lambda_2 (Lambda)	(None, 66, 235, 3)	0	**Normalization** layer To get the values between -0.5 to 0.5
convolution2d_1 (Convolution2D)	(None, 31, 116, 24)	1824	**1st conv** layer. 24 Filters of 5x5 Activation: Relu
convolution2d_2 (Convolution2D)	(None, 14, 56, 36)	21636	**2nd conv** layer. 36 Filters of 5x5 Activation: Relu
convolution2d_3 (Convolution2D)	(None, 5, 26, 48)	43248	**3rd conv** layer. 48 Filters of 5x5 Activation: Relu

convolution2d_4 (Convolution2D)	(None, 3, 24, 64)	27712	**4th conv** layer. 64 Filters of 3x3 Activation: Relu
convolution2d_5 (Convolution2D)	(None, 1, 22, 64)	36928	**5th conv** layer. 84 Filters of 3x3 Activation: Relu
flatten_1 (Flatten)	(None, 1408)	0	For flattening the data in a single row (to serve as inputs to FC layer ahead)
dense_1 (Dense)	(None, 100)	140900	**1st FC** layer
dense_2 (Dense)	(None, 50)	5050	**2nd FC** layer
dense_3(Dense)	(None, 10)	(None, 10)	**3rd FC** layer
dense_4 (Dense)	(None, 1)	11	**4th and final FC** layer
Total params:	277,819		

Creation of the Training Set & Training Process

I used the simulator's record option to generate images for each set. I augmented the data with using left, right images as well as flipped the data. So as to have 6 times the training data.

At first I made the mistake of running my program (model.py) via a script over various folders. But this was erroneous, as I learnt painfully. In this case it would just learn based on the last folder provided. I had assumed that a *transfer learning* kind of approach would work. But it didn't.

So I decided to load the data all at once to train, modified my model.py suitably. The code for that is in `load_all_sub_dirs()` function in model.py

Also I used the generator concept, as advised by udacity material, to prevent any possible out of memory errors.

After some trial and error, I arrived at 5 epochs as a no. which works well. Altough, sometimes the model/data combination worked fine for 3, and other times at 7 epochs.

The training set, validation set ration I settled for is 10% for validation.

My final validation accuracy, for good results was around 2% +/- .5%. But apparently the precise value had no direct correlation with how the car performed on the track. And I selected the results which looked good on the track, as this project is about behavior cloning.

Summary of Learnings

1. Some training data, which looked good, did not yield best results. So finally I had to pick the one which gave the best results. I realize that it raises questions on the training method. But the reason, I accept this, is that perhaps in real world we need much more amount of training data. The Nvidia paper[1] talks of 72 hours of training data. The test result would improve in a giving a better feel, with more data. Just that, it may cut the lane marker at few more times.

2. Track 2 is difficult, it has some very sharp turns. Its very hard to train the network for those kind of cases. And the car would normally get stuck there.

References

[1] Nvidia paper ‘End to end self driving cars’
https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf
[2] The (famous) LeNet model paper:
http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf