Behavioral Cloning

Data Sets

During the process of training the model the following 4 data sets were used:

1st Collection

- Size: ~1GB
- Car Controller: Keyboard Arrows
- Car Speed: Fastest

First Track	Second Track
• 3 Direct Center Driving Laps • ~5 Direct Curves Recordings • ~5 Inverse Curves Recordings • 1 Inverse Center Driving Lap	• 2 Direct Center Drivin Laps • ~5 Direct Curves Recordings • ~5 Inverse Curves Recordings • 1 Inverse Center Driving Lap

2nd Default Collection

This is the default provided data set.

- Size: ~350 MB

3rd Collection

- Size: ~700 MB
- Car Controller: Mouse
- Car Speed: Slow

First Track	Second Track
• 2 Direct Center Driving Laps • ~5 Direct Curves Recordings • ~5 Inverse Curves Recordings • 1 Inverse Center Driving Lap	• 2 Direct Center Driving Laps • ~5 Direct Curves Recordings • ~5 Inverse Curves Recordings • 1 Inverse Center Driving Lap

4th Collection

- Size: ~750 MB
- Car Controller: Mouse
- Car Speed: Very Slow
- Focused on parts that needed improvements.

First Track	Second Track
• 2 Direct Center Driving Laps • ~10 Direct Curves Recordings • Records of: After Turn Sections, Bridge Entrance	• 2 Direct Center Driving Laps • ~10 Direct Curves Recordings • Records of: Dangerous Turns

As it was to be expected, the mouse was superior in recording precise steering angles, because of it's easy to control dynamic, and constant angle curving.

5th Collection

- Size: ~500 MB
- Car Controller: Mouse
- Car Speed: Very Slow
- Focused on just the 2 turns from Second Track

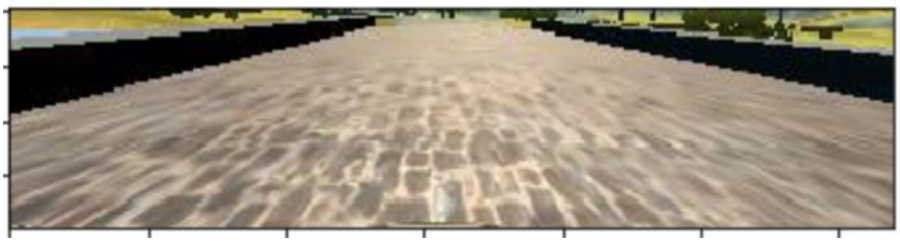
Data Processing

I used all 3 provided camera images from the the beginning - **center, right, left** - with an steering adjustment of +/- 0.2 for the right and left images.

The data was **normalized** and **mean centered** using the following formula:

image / 255.0 - 0.5

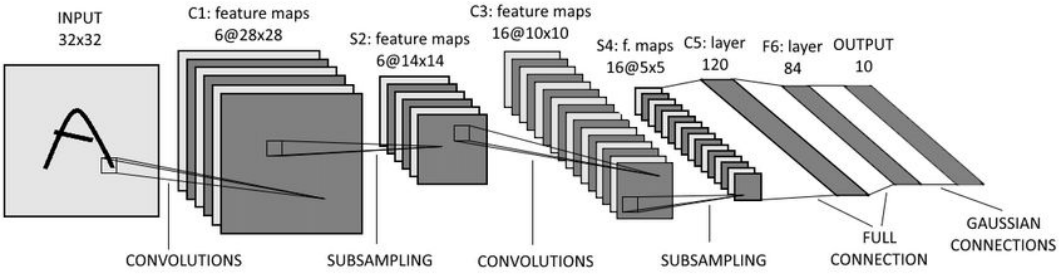
Also, for the final model the images were **cropped** and **resized to 66x200**. This is an example of a final preprocessed image:



Training

I have started to train the model on an AWS EC2 p2.2xlarge instance with the simplest possible architecture - 1 Convolution and 1 Fully Connected Layer, and the 1st Collection, just for curiosity. The image normalization and mean centering was added from the start in a lambda layer to the beginning of the network. By this the model achieved a validation loss of **0.7649.**

Then I decided to try out **LeNet**, which is a good starting point for any Convolutional Neural Network:



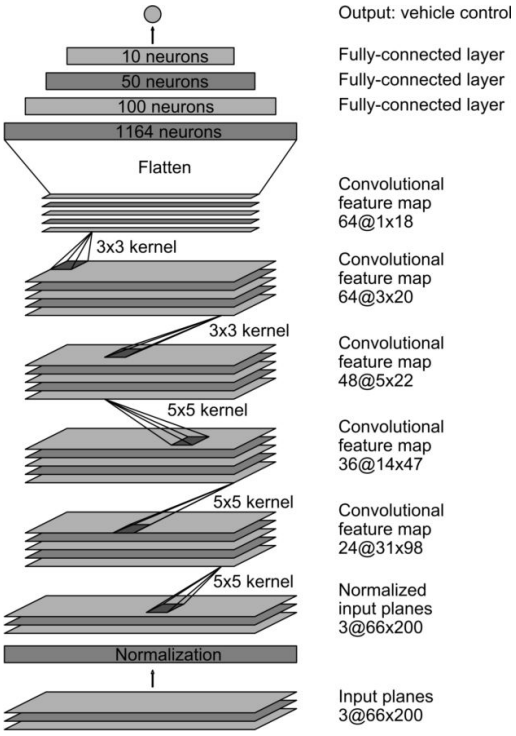
Using **LeNet** with normalization, made the validation loss decrease at **0.0157**:

loss: 0.0153 - acc: 0.1798 - val_loss: 0.0157 - val_acc: 0.1851

After trying the car on the first track it seemed that it still needed improvements.

I decided to try **image augmentation**, so I have flipped the images horizontally and inversed their steering angle measurements. Also to eliminate outliers and noise, the images were cropped to keep only the road in focus, using a keras cropping layer. Even with these changes, the model didn't seem to perform really well.

The **Nvidia End to End** model was the next architecture tried:



At this stage, I decided to add **Dropout** to prevent overfitting with a keep probability of 0.2, and I have also switched to an **p2.8xlarge** instance for faster training. Also the images were **normalized**, **cropped** and **resized** to the default nvidia input size of *66x200*.

I used the **Adam Optimizer** for **17 epochs**, with a starting **learning rate** of **0.005**, **beta_1** of **0.9** and **beta_2** of **0.999**.

With this changes, the **validation loss** achieved was **0.213**, but the model still wasn't performing really well.

The next step was to retrain the model from scratch, using both the **1st Collection** and the **2nd Collection** - the default dataset provided. But when trying to augment and process all the data, *OutOfMemory* was easily encountered.

To solve the error, I delegated the processing task to a **generator** which was in charge of the following: reading a batch of images, cropping images, change colour space (if needed), augment images and resize images. By training the model using the *fit_generator* keras method and the generator function, the memory error was eliminated.

Model checkpoints were also used to save at each epoch the model that had the best score.

After 17 epochs of training with the two sets of data concatenated, these were the results:

loss: 0.0179 - acc: 0.0066 - val_loss: 0.0148 - val_acc: 0.0069

At this stage I decided to also train the model from scratch only with the default data - the 2nd Collection, but it's results on the first track were not so good. The results achieved are:

loss: 0.0166 - acc: 0.1808 - val_loss: 0.0157 - val_acc: 0.1814

The **transfer learning model** was performing well on the first track, but it still had a lot to learn for the second one. Therefore, I gathered another set of examples, and made a second round of transfer learning with the previous model by tuning the previous trained weights using the 3rd Collection:

loss: 0.0187 - acc: 0.0069 - val_loss: 0.0156 - val_acc: 0.0057

This gave significant improvements for the second track, but it also affected a little the first track driving - in several parts of the roads the car was not driving always right in the middle of the lane, and in one place it was approaching close to the edge of the line. But I decided that this is acceptable. Still, there were two large turns in the second track which were not handled properly by the car.

I gave another chance to the previous model's weights trained from scratch with the default data - that performed poorly on the track - by training it through transfer learning on the 3rd Collection. Although its results improved, the **transfer learning model** was way superior.

At this stage, after training the custom nvidia model with different data sets and transfer learning, the car was able to drive the first track well - with only one place close to the edge, and for the second there were just 2 turns that needed to be corrected.

Hence I decided to train the model on a 4th dataset that I gathered, the 4th Collection. Collecting OVERALL data for this dataset, instead of sticking just to the problematic parts, proved to be a mistake - it made some parts that were previously working to fail now, and sometimes the model was not driving along the center even on a straight road.

I also started to realize that at this points it is not so much about lowering the accuracy further - which was already small, but more about tuning the weights of the network in a new distribution in which to solve just the specific problems encountered on the track. But for this to happen, the new data set used needs to be large enough to be able to challenge the traits already nurtured by the previous data. This can also be adapted using the hyper-parameters.

In the end I collected 500MBs of just the 2 problematic curves - especially actions of driving towards center with maximum angle when the car was on the right or left side of the road. I have trained the transfer learning model again, and guess what - all the 2nd tracks' curves were taken really smoothly, but now it failed on simple parts of the relatively straight road. Because of time constraints, I decided to end the game here right now, and come back later in the next period to solve this annoying conundrum.

You can check the accompanying *'Model Creation'* notebook for more practical insights into the model creation and training process.

Final Model Architecture

The final model used the Nvidia End to End one, trained for 4 rounds of transfer learning. Starting with the 1st Collection, then adding the 2nd, 3rd and 4th ones. Each round was run for **17 epochs**, with a generator **batch size** of **128,** and using the default **Adam Optimizer. Checkpoints** were used to save the model at each epoch if its loss score was improved. The architecture of the final model is summarized below:

- 2D Convolution - **2DC**
- Fully Connected - **FC**

Layer	Component	Input	Filters	Kernel Size	Strides	Padding	Activation
Normalization: x / 255.0 - 0.5							
L1	2DC	(60, 200)	24	(5, 5)	(2, 2)	VALID	Relu
	Dropout - 0.8						
L2	2DC	**L1**	36	(5, 5)	(2, 2)	VALID	Relu
	Dropout - 0.8						
L3	2DC	**L2**	48	(5, 5)	(2, 2)	VALID	Relu
	Dropout - 0.8						
L4	2DC	**L3**	64	(3, 3)	(1, 1)	VALID	Relu
	Dropout - 0.8						
L5	2DC	**L4**	64	(3, 3)	(1, 1)	VALID	Relu
	Dropout - 0.8						
FLATTEN							
L6	FC	**L5**	100	-			

	Dropout - 0.8			
L7	FC	**L6**	50	-
	Dropout - 0.8			
L8	FC	**L7**	10	-
	Dropout - 0.8			
L9	FC	**L8**	1	-

The models provided in the root of the repository(mode.hdf5 and model_1st_track.hdf5) are checkpoints from different epochs from the last training session on the 3rd Collection for 1st and 2nd tracks, and 5th Collection for the second track. Each one manifests different traits in successfully navigating the 1st track, and both have one different "maxima" point in which they get really close to the edge of the track, although they don't pass it. You can use the *model.h5* for the evaluation on the 1st Track as reference.

Discussions

This project reminded me about the reasoning black box in which ML & AI model take decision. After arriving at several good performing models, it would had been awesome if I could have taken the best parts of each of them, and put them together. One model was really good at keeping the center for a given turn, while another was better at doing the same thing for a different one. Unfortunately, this is not totally possible given that we don't have the necessary insight in the present to attribute model successes to specific parts of the network.

For the second track it seems that having the white line center helped a lot the car in keeping the center even in complicated parts of the road - this makes sense because the network can easily translate the curvature of the center lines into steering angles.

As improvements, I was thinking that maybe using the advanced lane recognition pipeline from the next project, could help in pre-processing the images. One of the problems at this stage is that images taken from different source look different (different lane colours, pavement texture etc.). Also there is not one single precies feature that the model can use to learn the steering angle for any road scene. By using the above pipeline, we could try to set the perspective transformed fitted lines as a main feature for steering angle recognition in a grayscale image, and hence more generalize the feature characteristics (just some curved lines), and help the mother take more precise decisions.

Files Included

The following files were included in the repository:

- model.py
  - I have included only the relevant **model** and **generator** code here. For the training process I have used a notebook - ModelCreation.ipynb - where you can find more detailed information.
- drive.py
  - I have added cropping and resizing in this file
- model.hdf5
  - This is the model that can be evaluated - it works well on the 1st Track.
- writeup_report
- video.mp4
  - First track video
  - Recorded using model.hdf5
- video_1st_track.mp4
  - Second first track video
  - Recorded using model_1st_track.hdf5
- video_2nd_track.mp4
  - Second track video
  - Recorded using model.hdf5
  - At two large turns I need to shortly enter in manual mode
- model_tryouts
  - Several other models kept for reference