Philippe Rougier

Behavioral Cloning

Goal:

Design and train a neural net model to drive an autonomous vehicle.

Set Up Overview:

Drive Simulator :

A provided simulator reproduces the visual environment surrounding a virtual vehicle in motion :

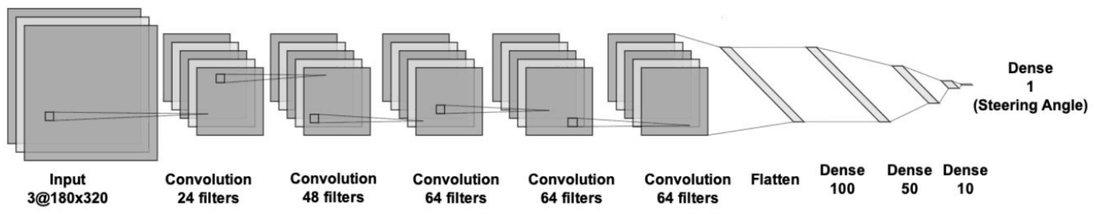
- data is collected (road track images and steering angles) while an operator drives the vehicle on a virtual track
- this data is used to train a convolutional neural network .
- in autonomous mode (inference) the simulator feeds images of the roadway to the trained model. For each image, the model predicts a steering angle, then the vehicle moves forward accordingly hence feeding the model with an other image. The repetition of these successive steps generates the self driving motion.

Data format:

Stream	training	inference
RGB images (320x160 pixels)	**input**	**input**
steering (unidimensional) float in range [-1,1]	**reference**	**hypothesis**

Neural network architecture:

The project uses the iconic “NVIDIA ” autonomous vehicle model designed for behavioral cloning:



- features are extracted from the images through a stack of convolution (CNN) layers
- the last CNN layer is flatten and fed to a stack of dense layers inferring the steering .

We implemented the following variation of the NVIDIA model :

(***C****onvolution_kernel:* ***s****ubsample)*/ (****D****ense)* * :

(320x160x3) -> **C**24_5 : **s**2 / **C**36_5 : **s**2 / **C**48_5 : **s**2 / **C**64_3 / **C**64_3 / **D**100 / **D**50 / **D**10 / **D**1

Programming environment :

Model implementation and training:

We'll use the high level API Keras (tensor flow backend). Training is done on the GPU set up for the Udacity workbook

Data processing/model enhancement:

while loading the images :

image size reduction :

In an attempt to shorten the GPU usage we reduced the images size to (160x160). (resize(image), (160,160)) (and adjusted cropping in the Keras layers). This reduction was applied :

- during training, before passing the images the to the Keras model (within model.py)
- during inference as the images are collected along the track to be sent to the model to infer the steering (within drive .py)

This resizing slightly **lowered accuracy** of the model. Consequently we stuck to the original image size **(160x320) .**

color space switch:
According to the Nvidia documentation the model is optimized for **YUV** images. Consequently we converted :

- during training : images read in are converted BGR to YUV (model.py) .
- during inference: images collected in RGB are converted to YUV (drive.py)

mono channel training and inference:
We tested the training on a small sets of 3 channel images (YUV) and on mono channel images (Y or U or V) to assess if one channel could increase the accuracy of the model . We used **all 3 channels** (YUV) as this solution induced the lowest validation loss.

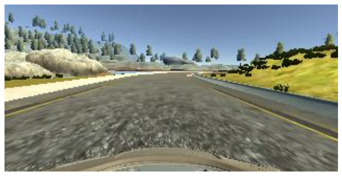
within the model (using Keras layers) :

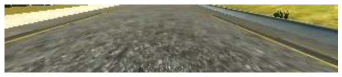
normalization:

To normalize each pixel within a [0.5, 0.5] range we added a Keras lambda layer on top Keras model *(Lambda (lambda x: x/255 -0.5)*)

noise reduction:

Top and bottom region of the images (landscaping /70 pixels, car hood /25 pixels)are cropped out of the image to focus on the roadway. This filters out unnecessary and misleading information hence raises the accuracy.


Loaded image (160x320)


Cropped Image(65x320)

We used the built-in Keras cropping layer *(Cropping2D(cropping=((70,25)).*

Resulting model characteristics :

(160x320x3)> (65x320x3)>> C24_5 : **s**2 / C36_5 : **s**2 / C48_5 : **s**2 / C64_3 / C64_3 / D100 / D50 / D10 / D1

Data location:

While the car is driven in manual mode, images of the driveway (a triptych taken by left, center, right cameras shooting simultaneously) are stored on disk. A list of these images addresses and **only the center image** steering angle is recorded in a csv file.

Training Pipeline:

generator motivation:

Training images are stored on disk. Since all images can't be uploaded simultaneously in memory, we'll use a generator to load them as needed for each batch.

Although the latest versions of Keras features a built-in " generator from data frame ", designing a similar generator from scratch is one of this project assignment.

custom generator :

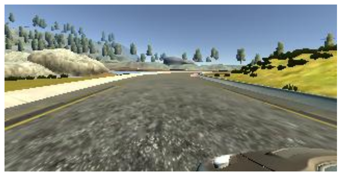
for each set (training and validation)

- goes sequentially through batch size sections of the set (stored in a csv file)
    - for each line of the csv file (*see ***images example*** below):
        - outputs images :
            - center, right and left images
            - flipped horizontally : right , center , left images
        - outputs steering :
            - for center image : from provided center steering
            - for left and right images : add (+/_ 0.2) to the center steering
            - the opposite steering for the flipped images (right , center, left)

- once the set is entirely processed, the set is reshuffled for the next epoch

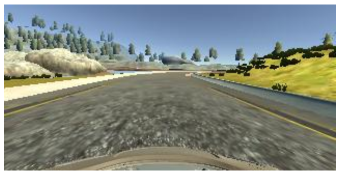
    - ***images and steering example*** (*) :

left camera



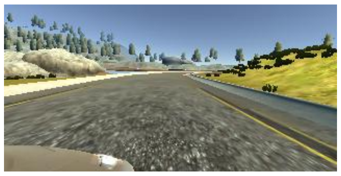
steering: 0.00127411 **+ 0.2**

center camera



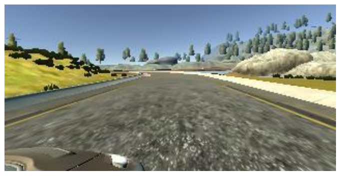
steering: 0.00127411

right camera



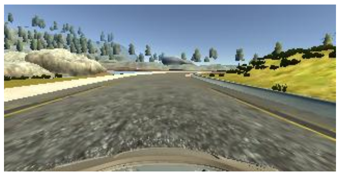
steering: 0.00127411 **+ 0.2**

flipped left



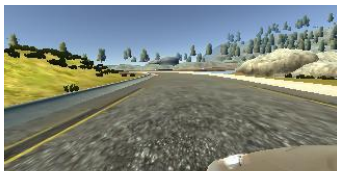
steering: **-**(0.00127411 **+ 0.2**)

flipped center



steering: **-**(0.00127411)

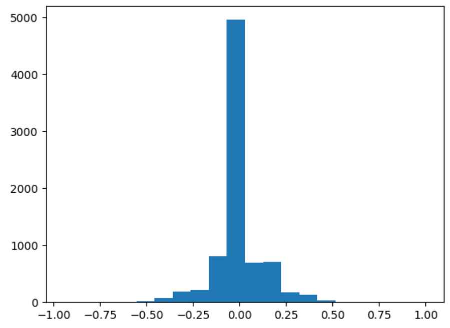
flipped right



steering: **-**(0.00127411 **+ 0.2**)

Training and results :

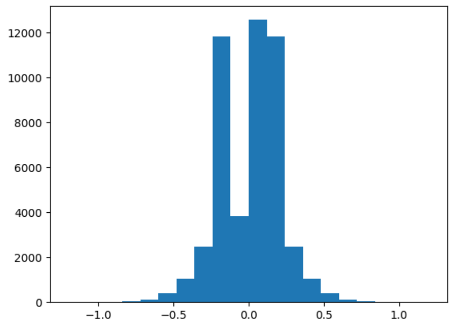
Provided data set training :



An histogram of the steering for the center images set (8,036 images) shows that the data is strongly skewed toward zero (i.e : no steering, straight lane driving)**.** A first training **try out** was done using a simple generator feeding only the center image to the model **(no data augmentation)**

As expected, due the lack of diversity in the center image training set, the model was not able to generalize (overfit) .Our model trained on this set failed to drive autonomously after a couple of yards.

Augmented data set training :



We then conducted the same training with a [custom generator]() that uses the 3 camera images (felt, center, right) and also uses the 3 cameras flip images.

This resulted in a more balanced 48,216 image training set, covering a wider steering spread [-0.55, +0.55] closer to a normal distribution.

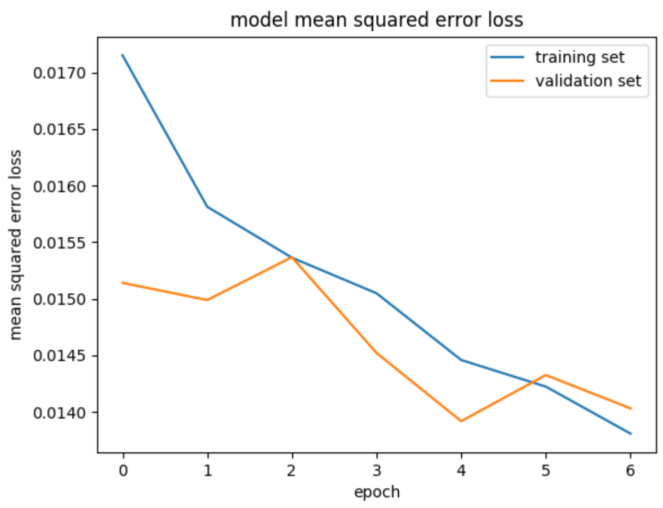
The diversity introduced by this augmented data set helped to address the over fitting problem : validation and training loss decreased similarly and significantly at each epoch and kept within the same value range . Still our model was overfitting as the vehicle will get dangerously close to the road boarder and ended up crashing in one of the last sharp curve.

residual overfit reduction : drop out

We added drop out after each dense layer (D100 / D50 / D10). We tested several drop out rate and retained 0.2 value. That further reduced the overfit.

Picking the best training :

We used the Keras function “checkpoint” to store the weights , the training and losses (training , validation) after each epoch in a dedicated folder (bestModelFolder) . For each training the losses history per epoch is visualized in a chart. (lossHistory.png)



lossHistory.png

We picked the best weight set from this loss tracker :

- to load weights to jumpstart new trainings
- to load for inference (drive.py model.07-Loss0.0138-valLoss0.0140.h5)

Once the over fitting issues addressed through data augmentation and drop out layers, our model training and validation losses converged to a low and stable value of 0.0140 after a 6 epoch training (batch size 256).

Autonomous Drive Simulator Results :

This was enough to ensure a smooth drive throughout the whole track loop : the vehicle drove steadily in the center of road the road throughout several whole loops.

(Page users : click on this pic to see the movie)

Conclusion:

Collecting recovery steerings sets (bringing the vehicle from the road side to the center of the road) was not necessary to achieve good results. Had this been necessary, this additional sets could also have been generated through data augmentation using image transforms and steering corrections within the generator.
This track's borders are crisp and contrasted from the roadway. This is not the case for the second track (out of this project scope). This other track would probably have required to pick a channel or a combination of channels within the (YUV) color space inducing more color separation .

Work Space Contents :

python files:

- **model.py** (contains the Nvidia model (using Keras API) , the generator, the loss visualization function …)
- **drive.py**

others:

model.07-Loss0.0138-valLoss0.0140.h5 : our best the weight set , used for driving.
lossHistory.png : loss history on the last training
run1.mp4 : video of two loops

bestModelFolder : an archive of all weights at each epochs for all training

examples (folder): a set o 7 images (Left, Center, Right, Flipped Left, Flipped Center, Flipped Right, Cropped center)