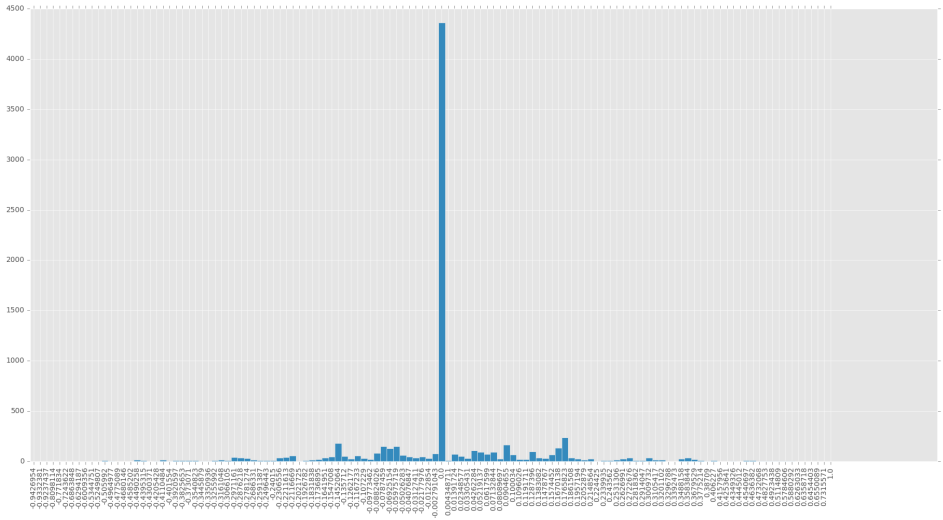
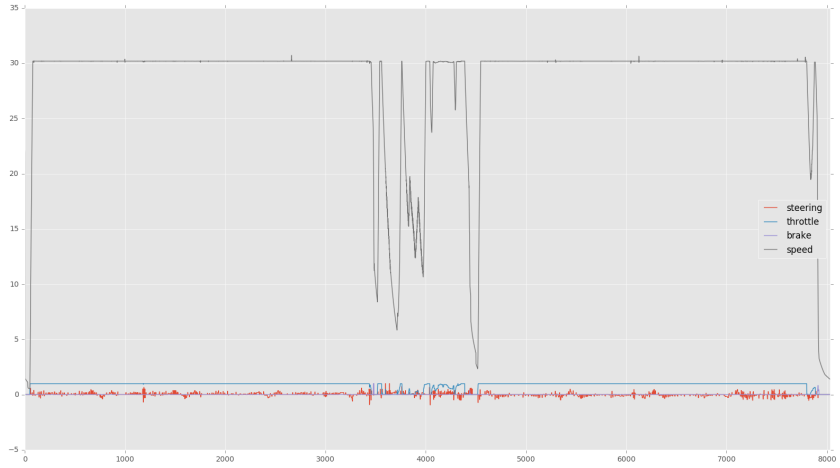
Data set Visualization

The data set used is the one supplied by Udacity.
Before starting to create a model is important to visualize the data set
in order to understand what kind of data we will be using.
It's important to see if the data are "balanced" and if there is the need
to get additional data (by augmenting existing ones or by finding new data)
Plotting the steering angle I noticed that we have a huge amount of 0.0 steering
angle data (approx 4300 sample / 7222 (total)), compared to the other steering angle and
so the data set (udacity data set) is very unbalanced.



Original Dataset: Steering angle distribution

There are also many data where the speed is very low...so I will get rid of them.



Original Dataset data insight

Augmenting and balancing the data set

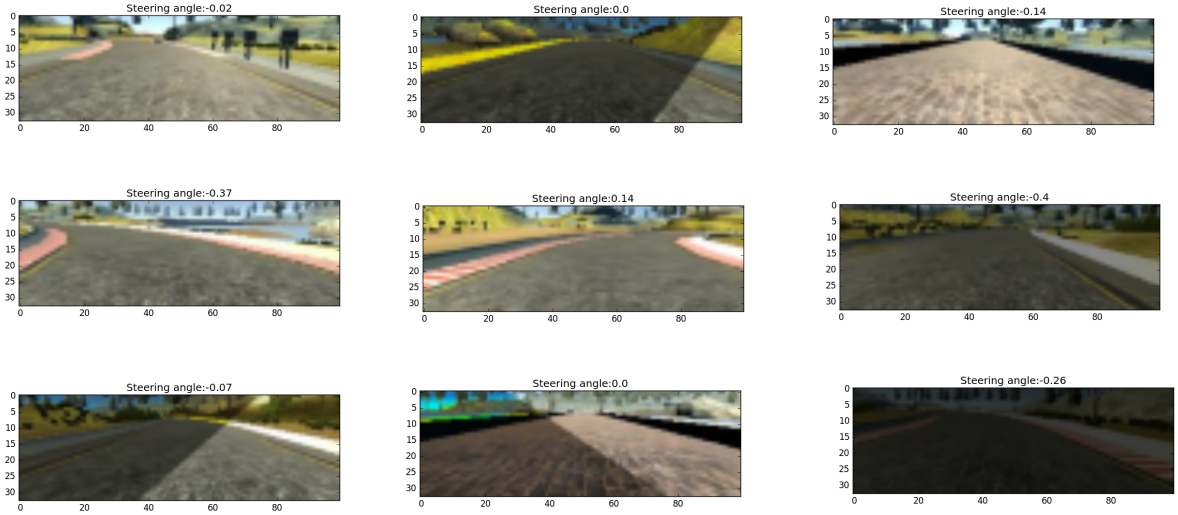
I used multiples approaches:
- flipping vertically left and right images when steering angle is not 0 and reversing
  steering angle (multiply by -1)
- Probably something can be done also with left and right images...use them

to generate recover situation and make a (static) correction on the steering angle (by trial and error)
- Run first clockwise and then anticlockwise on the track in order to have almost even turn left and right (and then duplicate them by flipping images)
- Changing the brightness of the images in order to make the model more robust (due to change of light and pavement on the track)
- The image is then cropped in order not to keep details that are not useful (i.e. tree or sky) and then rescaled in order to adhere to the Nvidia paper.

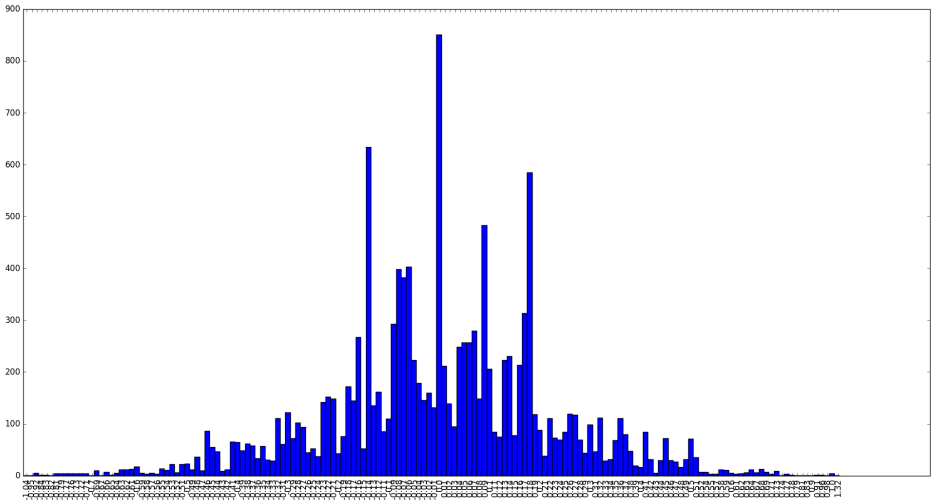
After some test the image was rescaled more to a size equal to half of the size of the Nvidia model so (100 x 33 x 3). It will speed up the training phase and gives good results too.

The image data is normalized between -0.5 and 0.5 and this is done inside the network as first layer.

Starting from an image read from disk randomly generate in memory other images. It will be done at run time with a generator. Doing so the number of samples per epoch is multiplied by a factor of 5 in the model (but could be multipied more, as the brightness and the shadow are randomly created on runtime phase.)
Regarding the zero data steering sample I decided to use them all, but limiting the number of samples that are used every epoch. So I can keep all the different images, but prevent the high number of zero steering angle to imbalance my dataset.
This is done inside the generator, where all file names are passed, but only a fraction of zero data will be kept per every epoch, they are just removed from the list in that particular epoch, but could be reused in the next one. So actually no zero data are really removed from the dataset, they are just chosen at random in order not to imbalance the data set.
The steering angle was also rounded to the 4th decimal digit.
Here's some images used as input to the neural network:



And here's the distribution of the steering angles generated by the generator at run time (this is one epoch only)



Data set distribution: dataset generated by generator function

Model description

For the model I started from the model described in the Nvidia paper,
and added some dropout layer in order to reduce overfitting.
Then I reduce number of neurnons in the fully connected layers and changed
some convolutional kernel, strides and paddin type (valid instead of same).
The drop probability is .7 for the first layer, abd .5 for the other layers.
I mantained the Nvidia activation function (ELU) and the Adam optimizer with learning rate 0.0001.
Due to this small learning rate the submitted model has been obtained after
110 epoch of training. Training has been done on a standard pc leveraging with
Nvidia graphic card.

Every epoch the model is saved by a callback and so many model have been tested
on the simulator before making a choice.
For validation set, the data set has been splitted with the scikit learn
train_test_split function.
The test set is produced by the simulator when the
model runs on the track.

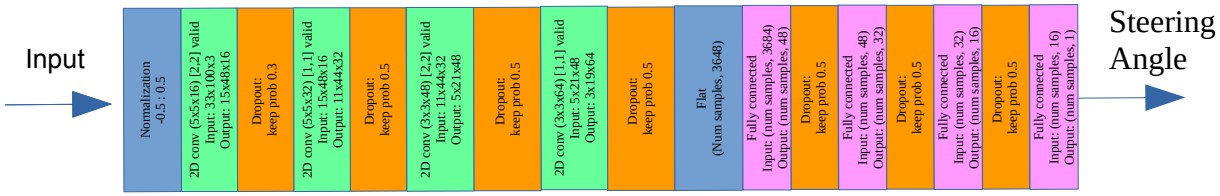
The training was done for 250 epochs and the model at epoch 110 was chosen.
It's nice to see how the fist models suffer of under fitting (they cannot
go almost anywhere and tends to follow the direction they have already had),
being easyly confused by sudden changes on the road. On the opposite side
if I use model over 115 epochs they tended to be very precise on the training
trak but not so good on the second track (sometimes the cliff are almost touched)
so they loose generalization power due to overfitting.

The final model is as the following:

Layer	Input_shape	Notes
lambda_1 (Lambda)	(None, 33, 100, 3)	Normalization layer
Convolution2D	(None, 15, 48, 16)	(filter 5x5x16) strides=[2,2] padding=valid
Dropout		(drop probability .7)
Convolution2D	(None, 11, 44, 32)	(filter 5x5x32) strides=[1,1] padding=valid
Dropout		(drop probability .5)
Convolution2D	(None, 5, 21, 48)	(filter 3x3x48) strides=[2,2] padding=valid
Dropout		(drop probability .5)
Convolution2D	(None, 3, 19, 64)	(filter 3x3x64) strides=[1,1] padding=valid

Dropout (drop probability .5)
Flatten (None, 3648)
Fully Connected (None, 48)
Dropout (drop probability .5)
Fully Connected (None, 32)
Dropout (drop probability .5)
Fully Connected (None, 16)
Fully Connected (None, 1)

Total params: 232,897



Conclusions

The model performs quite good also on track two where can finish an entire lap.
(no samples from track 2 were used for training or cross validation).
The model does not perform good if all the graphics detail are enabled. Only
up to "Fast" graphic level a full lap can be done (no problem
if I increase the resolution).
On the simulator the car can complete both tracks only if graphic details
is set at the lowest level (up to the called Fast level).
It seems to be due to the pavement texture and the shadows as the model has some problem.
Probably adding more data are needed (adding artificial shadow for example)
to solve this issue too (training will be longer as we added other samples).
A big thank to Vivek Yadav for his good ideas
https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9#.74m9mw2e9

On the first track it can perform at all graphics levels and at all resolutions.

drive.py was a bit changed to in order to preprocess the image (rescaling and
crop), and to play a bit with the speed parameter because there is a very
high slope in the track2 that couldn't succeed to overtake with stock drive.py
file. (I added some command line parameters in order to have more flexibility
during testing phase so --model parameter now is mandatory
(i.e. python3 drive.py --model model.json)