

MODEL ARCHITECTURE

Architecture taken is based on the NVIDIA network ([paper](#)), that tries to minimize mean squared error between predicted steering angle output by the network and the given one by the simulator (or adjusted in case of not centered cameras and flipped images).

The input image is a cropped section of 66x200x3 taking the most relevant area of information, in YUV color space.

The network starts with a normalization layer, followed by 5 convolutional layers, followed by a flatten one and 3 fully connected layers before the output.

After each convolutional and fully-connected layer, I have added an activation layer to introduce nonlinearity. I have tried with *tanh* and *ELU* (Exponential Linear Unit), taking the last one in the final model.

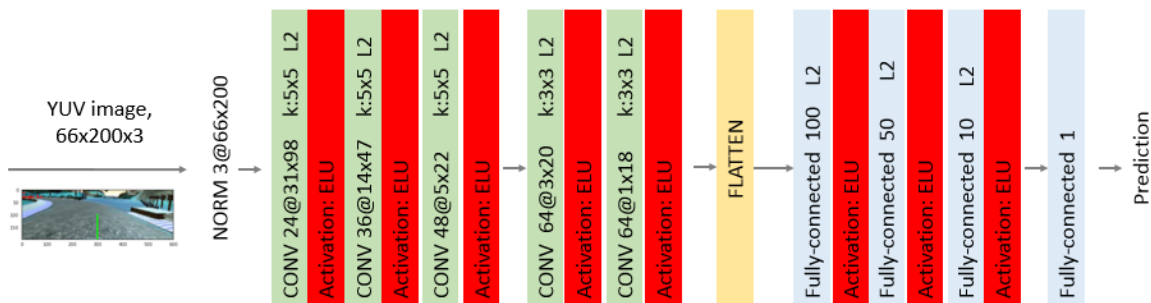
To reduce overfitting I have tried two things:

- Dropout layers after convolutional ones
- Performing L2 regularization for weights matrix on convolutions and dense (fully-connected) layers.

Although I didn't manage to improve the model with dropouts, so I just applied weights L2-regularization.

The model uses a generator to feed images although the preprocessing process is done before so I could have more control on the changes done. Some of the process done like adding flipping images, YUV conversions, or image distortion could have done in the generator function.

That's the schema model implemented



TRAINING STRATEGY

I have done lots of iterations to come up with a stable solution working with track 1. To get such a working solution I have followed 3 directions, and while the model was not working was hard to guess what direction was the root cause of poor results obtained, and what exactly to improve:

- **Data collection used to train the model.** The quality of data recorded has proven to be crucial to get a good driving model, and in my experience the most critical thing is to drive as much as possible in the center of the

lane. In my case I have discarded some tracks where I was not driving centered enough (for example, 3 rounds in rapid conduction) and just kept:

- Slow tracks where I was able to drive carefully on the center
- Some recovering tracks
- A counter-clockwise lap

- **Data augmentation, balancing and preprocessing.** (Further explained in the next point) Main things done here:

- Add left and right images with a steering angle adjusted
- Flip images in the training set in order to reduce the left unbalancing issue
- Reduce the amount of images with steering angle close to 0. (reduction of -36%, before splitting in training and test samples)

Main preprocessing steps (that are not done in the model generator, but previous. In order to improve program memory efficiency, it could be added there):

For both, training and validation sets, we follow these steps. Notice that similar steps are implemented in the preprocessing method in *drive.py* :

- Cropping the image, keeping just more relevant part, 20 px. from bottom and 50 from top.
- Applying Gaussian blur transformation
- Resizing the image to match network input size needs (66x200x3)
- Changing to YUV space color

And just for training data:

- Following that student work ([blog post](#)), I have tried with adding random distortions (in a similar `random_distortion` function) for training data, including random brightness adjustments, useful for changes in lighting/textures in some parts, and add vertical shift of the horizon position. It has improved the stability of the model in some parts.

- **Model parameters tuning.** Different parameters were needed to fine tune, with a manual grid search approach:

- Epochs used. I realized that were not much difference among using 5 or up to 10 epochs, since the validation error didn't decrease much after 5. I finally kept 10 for last iterations, but could have been 5 and have a faster training
- Learning rates or adam optimizer. I tried different learning rates although at the end I took the adam optimizer, so the learning rate was not touched manually
- Threshold for weights regularization for convolutions and dense layers. I tried with 0.01, 0.001, and 0.0001, taking finally 0.001.
- Dropout probabilities. I tried with 0.5 and 0.8 as `keep_probability` in drop layers, although I didn't see much improvements in the stage I tried out, so I chose L2 weight regularization.
- Batch size. I was trying almost all the process with 128 batch size, and when I tried 32, the solution obtained improved.

Other parameters not strictly related to the model that needed adjustment were:

- Angle adjustment for left and right images. After reading different comments and trying out offset parameters around 0.1 to 0.25, I took that higher.

Tactics followed to get a working model:

- First of all, to get a working model with basic udacity data (and just center images, and flipped ones), able to pass the first curve and crashing just after the bridge,
- Once gotten that first model, I tried adding more conduction data to train better the model. I added 3 laps of my own conduction and few recovering tracks. But it ended with really worse outcomes. I added some extra laps with really slow conduction and better control of the center driving. Even with that, the car crashed every time before reaching the first curve, during it or just before the bridge.
- Then, I worked on Improving data augmentation and preprocessing:
 - o adding left and right images,
 - o reducing the proportion of small steering angle pictures
 - o adding random distortions of the training set (inspired by the post mentioned before)
- Still results were not good, and the car crashed in different points, or drove in a very unstable way. Sometimes, it got to achieve the right curve, or get off the side just after the bridge...
- I tweaked further the model in order to increase regularization penalties, and increase the learning rate. No improvements.
- I went back to remove some recovering data and my laps with fast conduction, and I realized that simulation conduction was something more stable and that both training and validation error were higher than before. I may have passed from overfitting the model to underfitting it.
- I decided to record another slow track taking care of driving in the center and a counter-clockwise slow lap, and it made the difference!
- And during all that process, to look for ideas and comments in class forums has been really helpful. Some ideas taken from there: how to adjust left and right steering angle, to distort images in preprocessing phase, how to reduce the amount of near 0-steering angle images,...

DATA COLLECTION, TREATMENT AND EXAMPLES

Data Collection

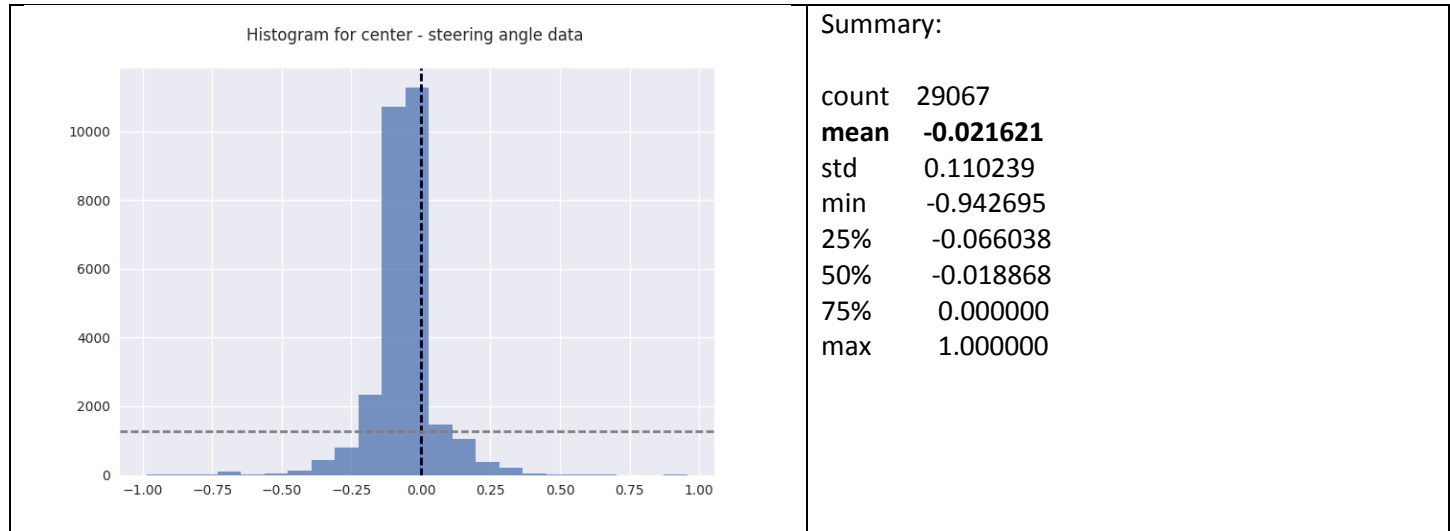
After a lot of experimentation, I decided to discard some of my data collected, since not getting off the lane was not enough to train a working model, but also ensure a good center driving.

Final data I used was composed by:

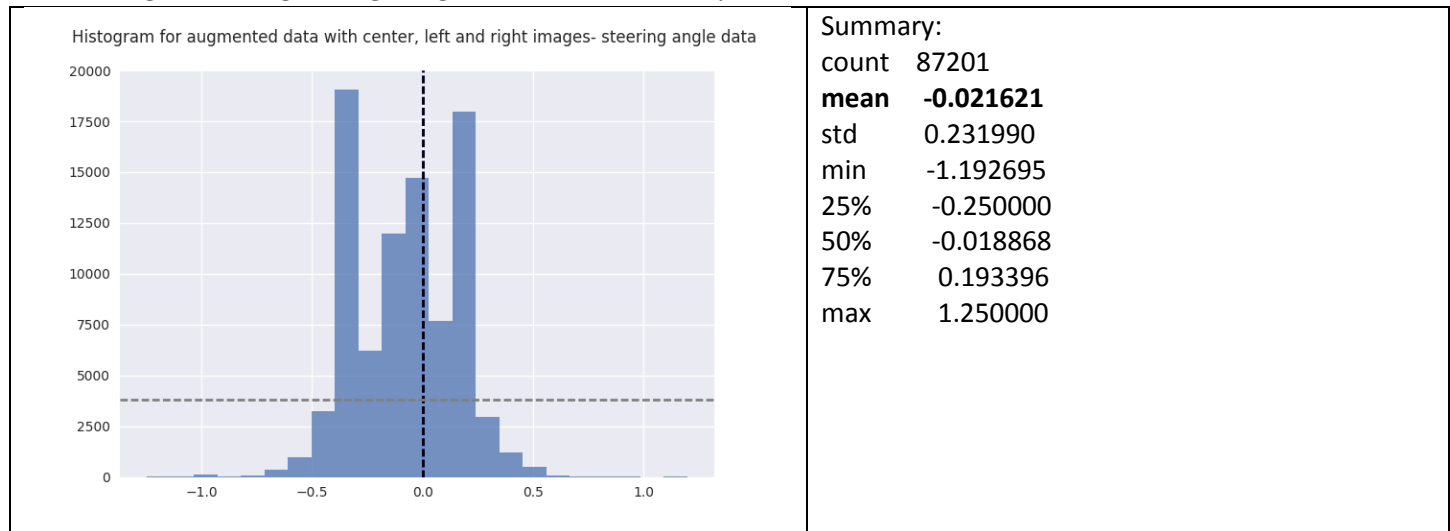
	center image	center + right +left images	perc over total
udacity data	8036	24108	28%
extra track	2804	8412	10%
recoverings	1020	3060	4%
slow driving	12568	37704	43%
counter clockwise	3769	11307	13%
curve track	870	2610	3%
Total	29067	87201	

Data cleaning and augmentation

The total amount of 29.067 center-images showed a steering angle histogram:



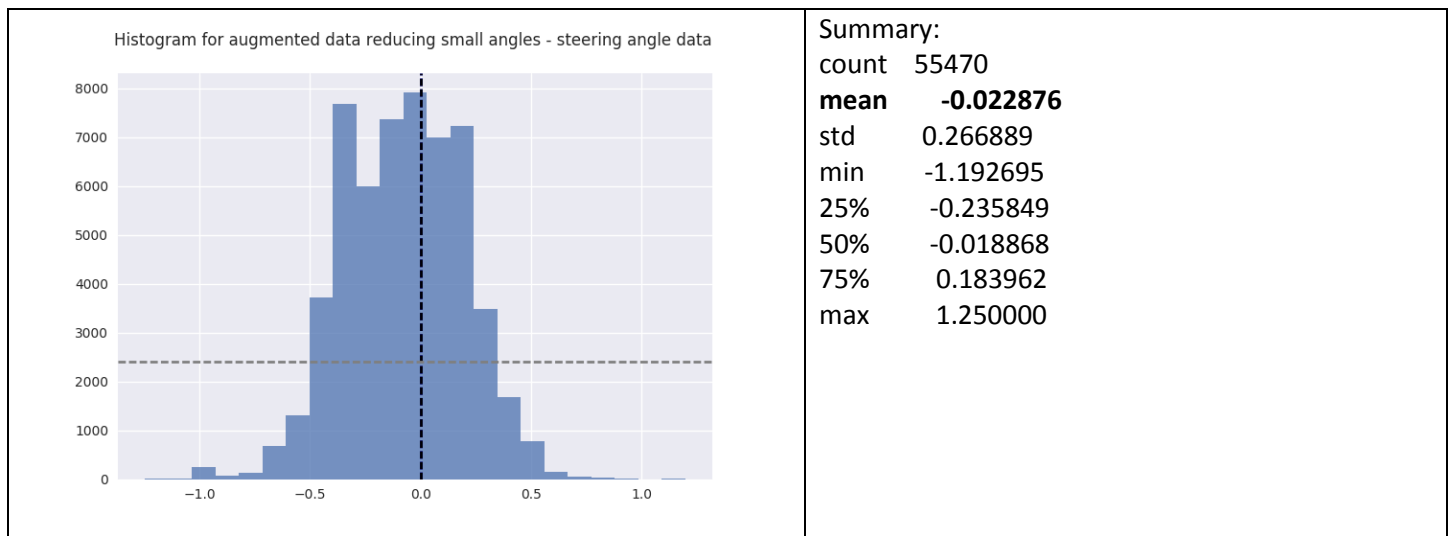
After adding left and right images, I got 87.201 with the shape:



Once I got that enriched data set, I did some reductions:

- Remove with a certain probability images in histogram bins that had more images than the average image that should have each bin in a uniform distribution (number of total images / number of bins)
Each bin that has less than that average (showed in a horizontal dashed line) keeps all its images, whether the central ones, with steering angle > -0.34 and < 0.40 , will just keep its images with a certain probability depending on the size of the bin.

After these changes, I got:



As it can be spotted here, with that changes we only balanced a bit the difference among curve and straight direction in the images, but not the “towards to left” bias coming from having a track with almost all curves with left directions. Mention here that the problem is somehow reduced with adding a lap driving counter-clockwise, that accounts for the 13% of data, but it’s not enough. The problem will be tackled adding flipping images for each that has steering-angle in absolute value higher than 0.33, for the training set.

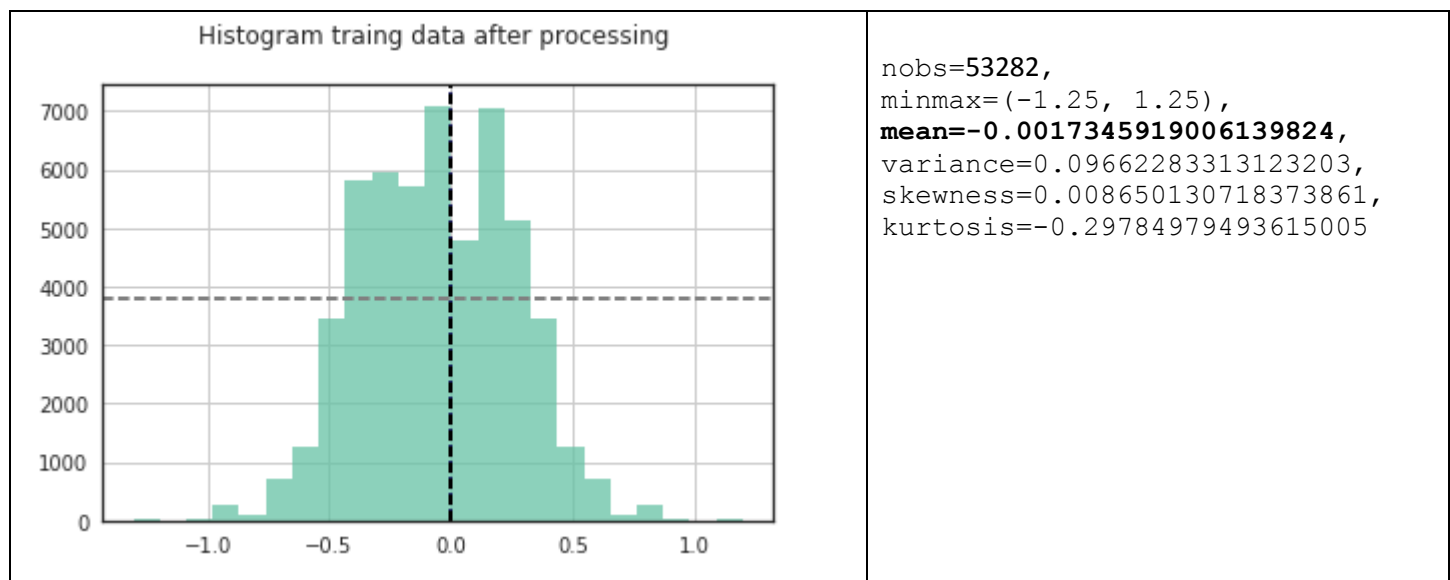
Now, I split data into training and validation set (20%), and for training set:

- Images are randomly distorted: random brightness changes, randomized shadow rectangles, and random horizon shift
- Added flipping images as just described below

Final data volumes after each process:

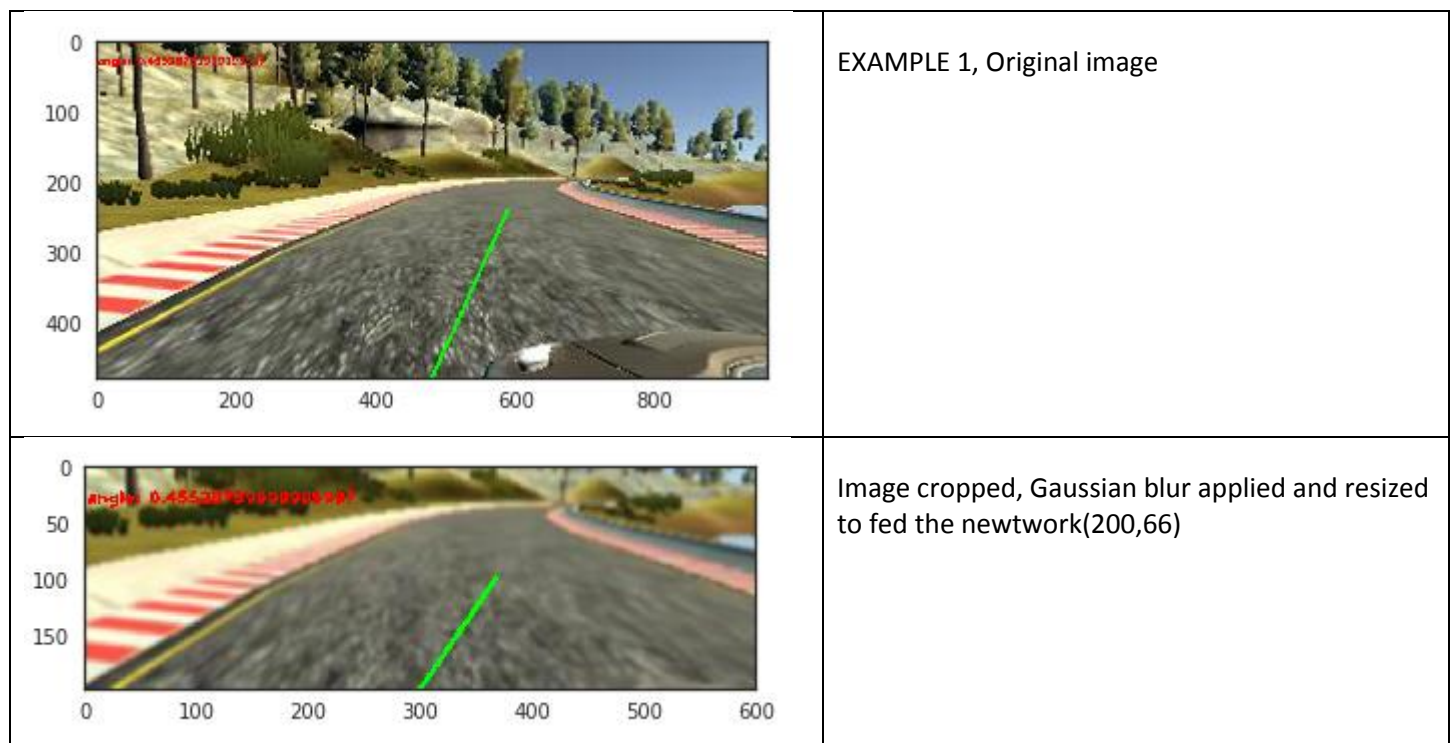
	Random splitting	% over total	After augmentation of flipping images	Increase after flipping	% over total
Training data	44376	80%	53282	20,1%	83%
Validation data	11094	20%	11094	No changes	17%
Total	55470		64376		

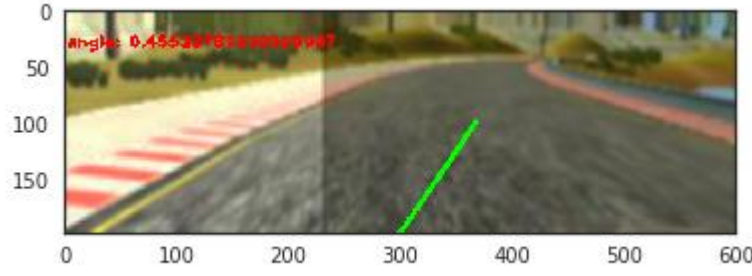
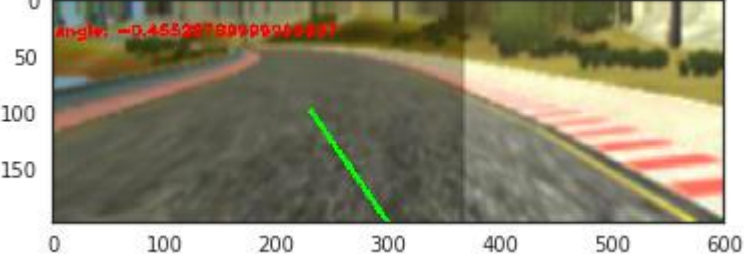
So, we end with 83% of training data, with a more balanced shape (left-right), and mean much closer to 0 than before:



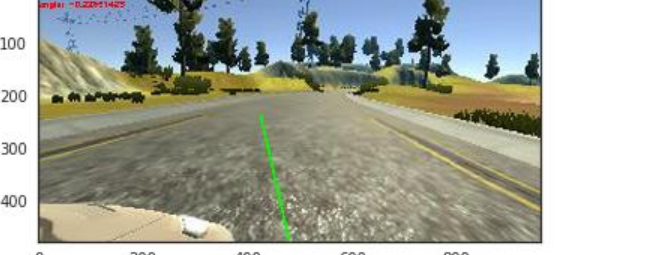





Data Examples

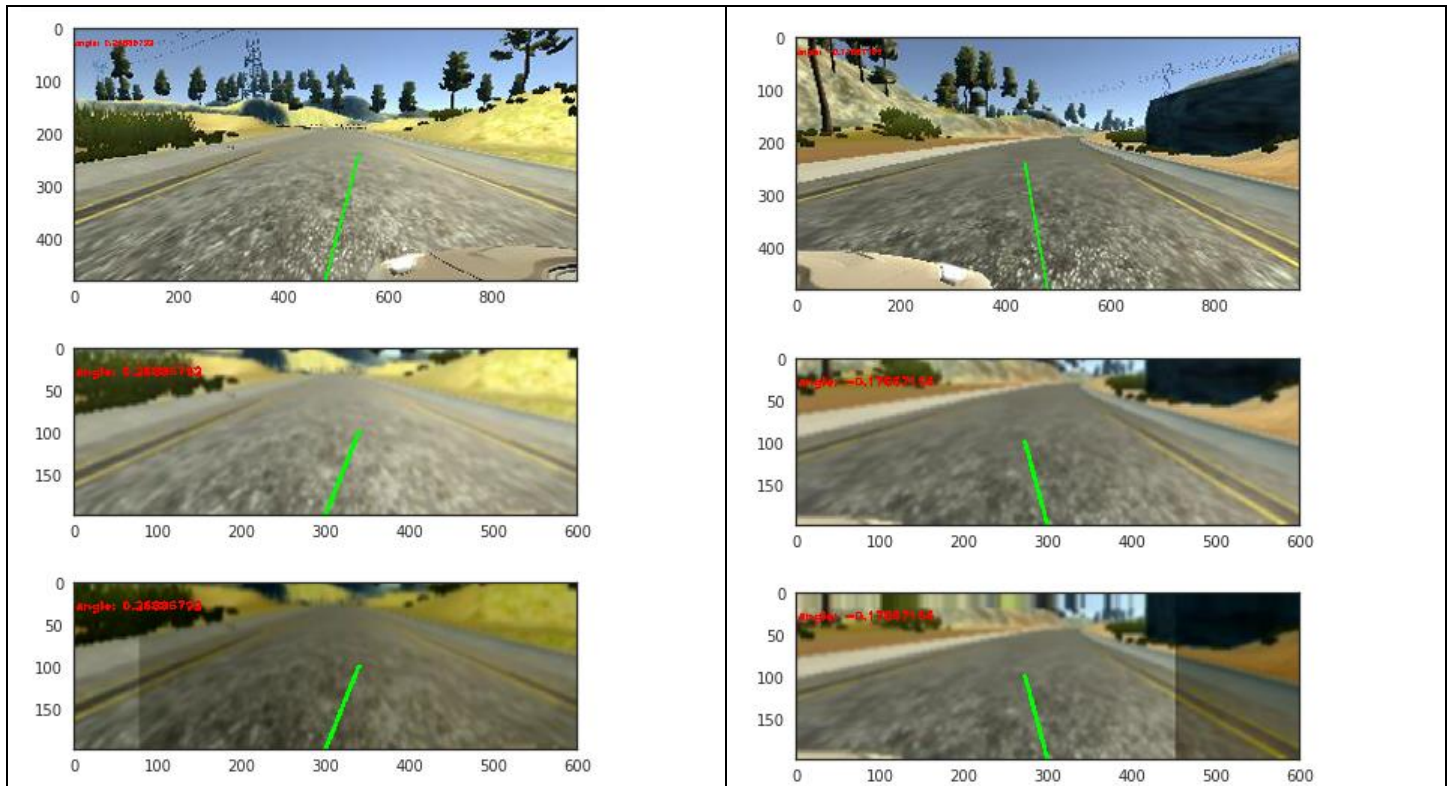
Examples of data processing:



	<p>Distortions applied (*)</p>
	<p>Flipping the image</p>

(*) distortions are found useful, and the origin of the idea and methods used come from that blog post ([link](#))



RESULTS

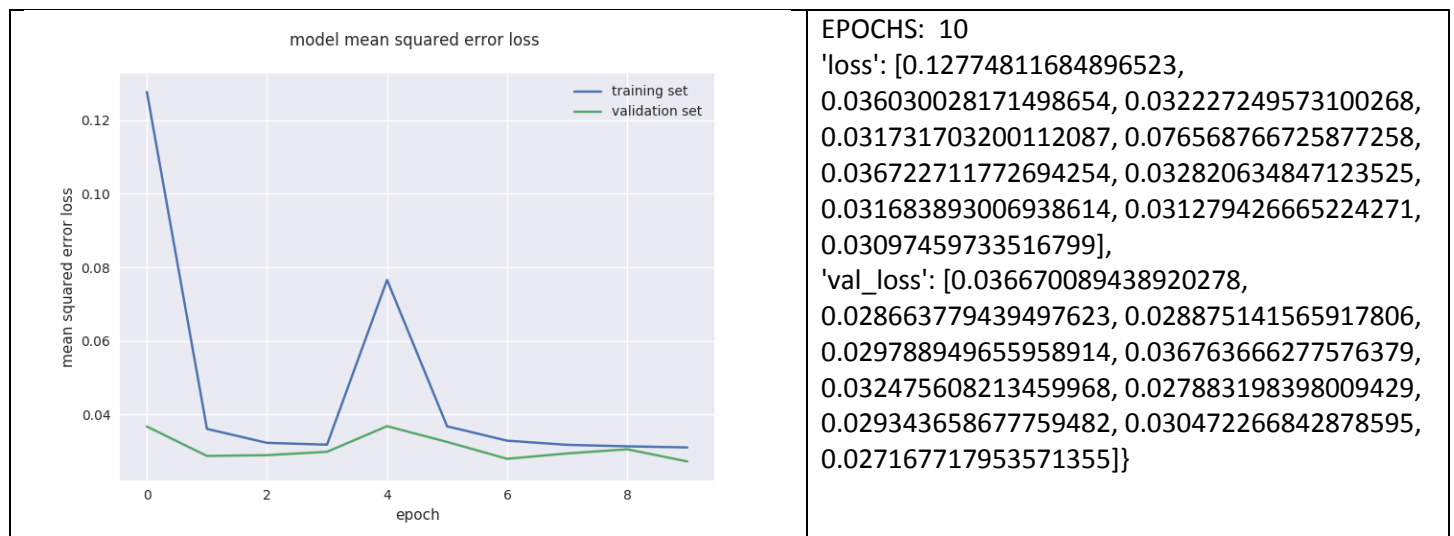
Final parameters chosen:

EPOCHS: 10

batch_size: 32

L2-regul: 0.001

With that, the final validation error obtained is 0.027 and we can see how it doesn't improve since epoch 5. Training error obtained is 0.03



Final model summary:

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 66, 200, 3)	0	lambda_input_1[0][0]
convolution2d_1 (Convolution2D)	(None, 31, 98, 24)	1824	lambda_1[0][0]
elu_1 (ELU)	(None, 31, 98, 24)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 14, 47, 36)	21636	elu_1[0][0]
elu_2 (ELU)	(None, 14, 47, 36)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 5, 22, 48)	43248	elu_2[0][0]
elu_3 (ELU)	(None, 5, 22, 48)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 3, 20, 64)	27712	elu_3[0][0]
elu_4 (ELU)	(None, 3, 20, 64)	0	convolution2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 1, 18, 64)	36928	elu_4[0][0]
elu_5 (ELU)	(None, 1, 18, 64)	0	convolution2d_5[0][0]

flatten_1 (Flatten)	(None, 1152)	0	elu_5[0][0]
dense_1 (Dense)	(None, 100)	115300	flatten_1[0][0]
elu_6 (ELU)	(None, 100)	0	dense_1[0][0]
dense_2 (Dense)	(None, 50)	5050	elu_6[0][0]
elu_7 (ELU)	(None, 50)	0	dense_2[0][0]
dense_3 (Dense)	(None, 10)	510	elu_7[0][0]
elu_8 (ELU)	(None, 10)	0	dense_3[0][0]
dense_4 (Dense)	(None, 1)	11	elu_8[0][0]
=====			
Total params: 252,219			
Trainable params: 252,219			
Non-trainable params: 0			

FEW FINAL COMMENTS

- That project was really fun and challenge for me, it took me a lot to realize about the importance of having as good as possible data or at least, not having bad data, and since I have not much experience with racing videogames, it's been a bit challenge to get some working tracks, and just remove all the noisy records
- I have used AWS GPU in order to train the model
- The model works for track 1 and gets off the lane for track 2 since I have not included any training track from there, in order to let the model generalize better.