Behavioral Cloning

Report based on the standard template.

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 with a written report

Files Submitted & Code Quality

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, based on the original with minimal changes.
- model.h5 containing a trained convolution neural network.
- weights.hdf5: containing the best weight from training phase.
- Project3_LucianoSilveira.pdf summarizing the results (this file).

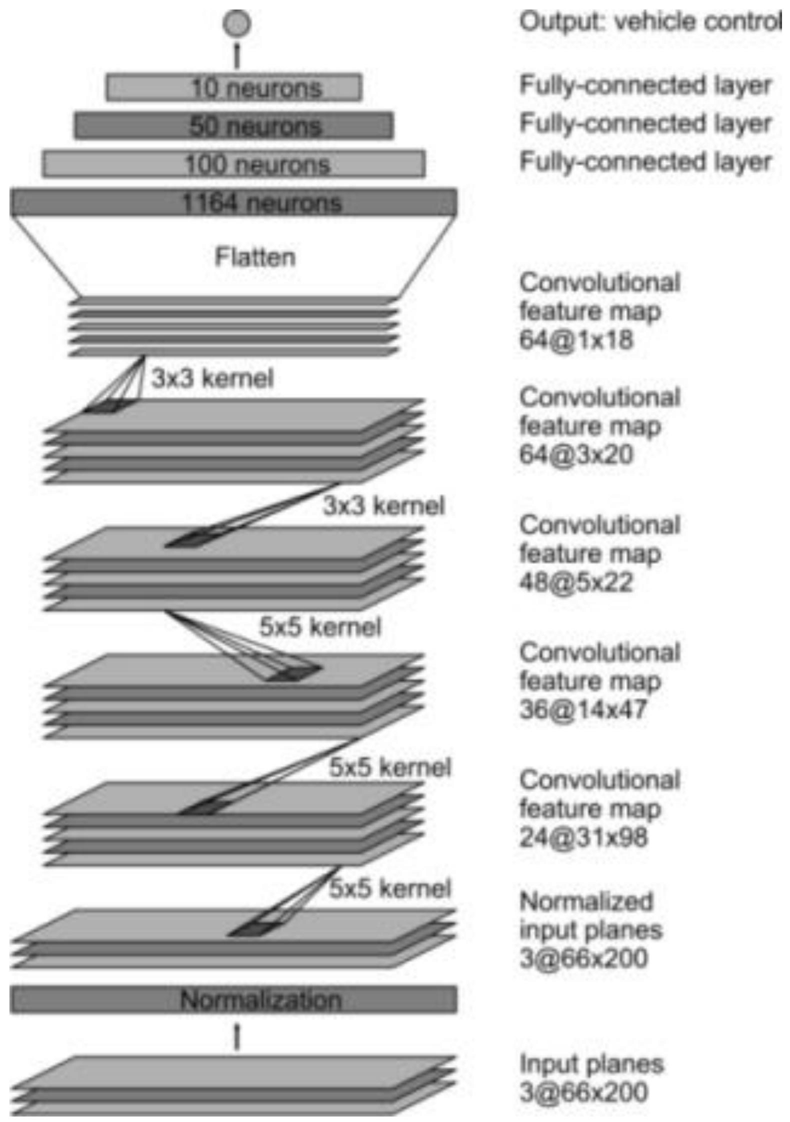
Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

python drive.py model.h5

Model Architecture and Training Strategy

1. Model Architecture

The model uses the proposed NVidia Neural Network from here (paper).



I did some modifications:

- Using ELU activation instead of RELU.
- Adding MaxPooling
- Adding several dropout layers to lower overfitting

The first layer of the network crops the image to the relevant section and the following one does normalization.

The next layers are composed of several convolutions with different stride and kernel sizes.

Then a MaxPooling operating was done to consolidate the extracted features and continues with 4 fully connected layers leading to the desired steering output value. The core of the model is:

```
model.add(Cropping2D(cropping=((top_crop,bottom_crop), (0,0)), input_shape=(row, col,
channel)))
model.add(Lambda(lambda x:x/255.0 - 0.5)) # Normalize data
# Architecture
model.add(Conv2D(24,5,5,subsample=(2,2),activation=activation))
model.add(Conv2D(36,5,5,subsample=(2,2),activation=activation))
model.add(Conv2D(48,5,5,subsample=(2,2),activation=activation))
model.add(Conv2D(64,3,3,activation=activation))
model.add(Conv2D(64,3,3,activation=activation))
model.add(Dropout(dropout))
model.add(MaxPooling2D())
model.add(Dropout(dropout))
model.add(Flatten())
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
return model
```

2. Attempts to reduce overfitting
The model contains dropout layers in different locations to reduce overfitting.

3. Model parameter tuning
The model used an adam optimizer, so the learning rate was not tuned manually; the selected value is 0.001. I tested the following values: 0.0001, 0.01.

4. Appropriate training data
The training data consists of:

- Sample data provided.
- One forward lap on the primary track.
- One backward lap on the primary track.
- On forward lap in the second track.
- Corner cases, specific locations were the steering angle was wrong to recover from the sides of the road.

Model Architecture and Training Strategy

1. Solution Design Approach
I started with a simple Neural Network as proposed on the class just to make sure the whole pipeline worked.

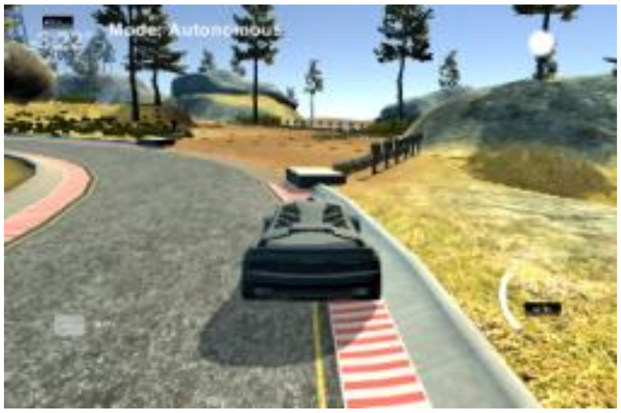
Then I reviewed the NVidia paper and replicated it on my infrastructure using Keras.

In parallel I added more data as detailed in the section above.

The model at the beginning worked bad it never completed a lap and continued steering to the same side.

I split my image and steering angle data into a training and validation set (80%-20%). The first execution got a high mean squared error on the validation set. This implied that the model was overfitting. I reviewed the pipeline and added a few epochs to train.

After these modifications the car could drive for some time but got some problem in specific places such as the following:



I generated more data; specifically I selected that location and drove so the car can recover from the side of the road.

I got minor improvements so tested different approaches:

- Change the correction from 0.2 to 0.3 on the steering angle for left and right camera.
- Change epochs from 5 to 20.
- Change activations from RELU to exponential relu (ELU).
- Change batch_sizes 16, 32, 64.

I got marginal improvements.

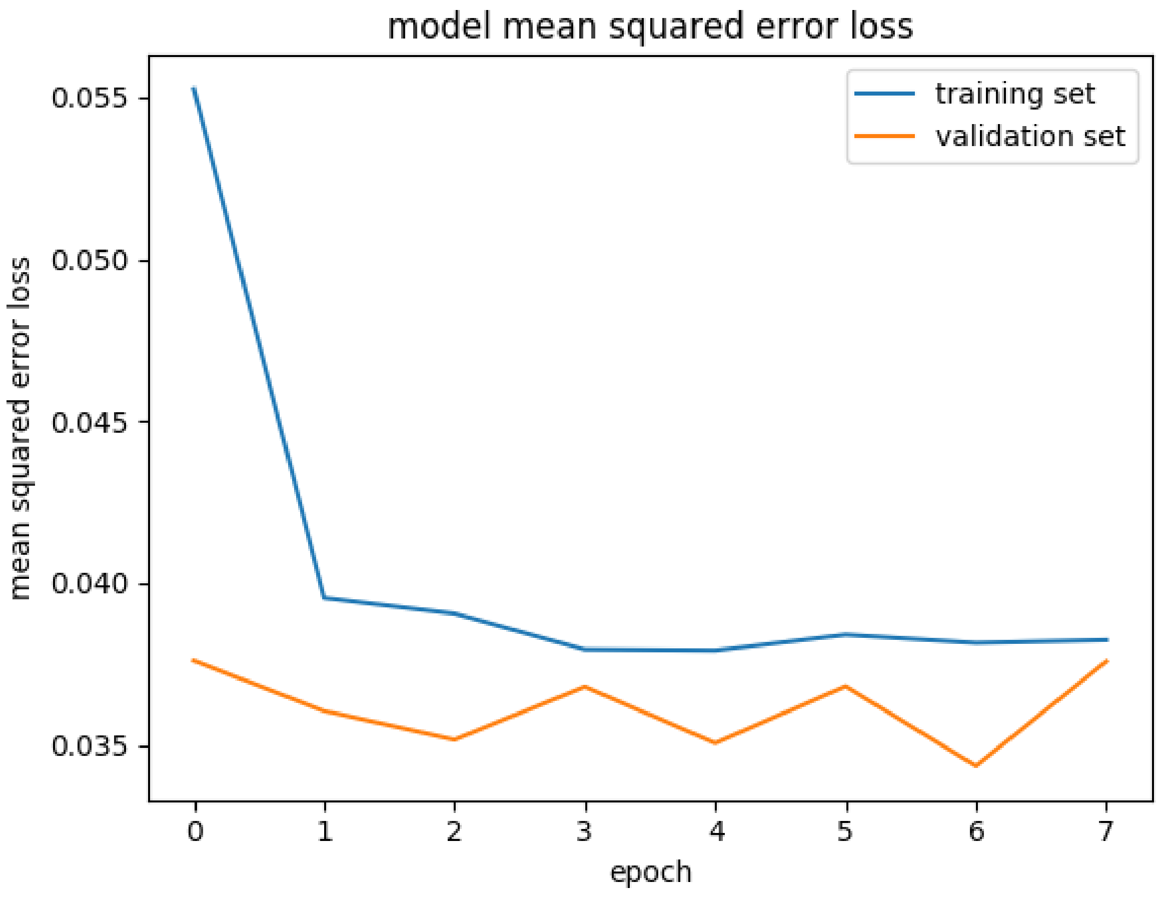
I reviewed the preprocessing pipeline, and decided to try changing the color space to HSV and use only the S channel as detailed in the discussion forum.

I needed to change accordingly the model.py and drive.py

- Change the model to receive a 1 channel image.
- Change the generator to yield the same image shape.
- Duplicate the same pre-processing on the drive.py file.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road. I doubled the throttle and worked fine but only for the first track.

After training, I was saved the best weights and got the following report:



And detail training:

```
Epoch 1/8

9728/9747 [============================>.] - ETA: 0s - loss: 0.0553Epoch 00000: val_loss
improved from inf to 0.03760, saving model to weights.hdf5

9747/9747 [==============================] - 23s - loss: 0.0553 - val_loss: 0.0376

Epoch 2/8

9728/9747 [============================>.] - ETA: 0s - loss: 0.0395Epoch 00001: val_loss
improved from 0.03760 to 0.03604, saving model to weights.hdf5
```

```
9747/9747 [==============================] - 19s - loss: 0.0395 - val_loss: 0.0360

Epoch 3/8

9728/9747 [============================>.] - ETA: 0s - loss: 0.0391Epoch 00002: val_loss improved from 0.03604 to 0.03516, saving model to weights.hdf5

9747/9747 [==============================] - 19s - loss: 0.0391 - val_loss: 0.0352

Epoch 4/8

9728/9747 [============================>.] - ETA: 0s - loss: 0.0380Epoch 00003: val_loss did not improve

9747/9747 [==============================] - 19s - loss: 0.0379 - val_loss: 0.0368

Epoch 5/8

9728/9747 [============================>.] - ETA: 0s - loss: 0.0379Epoch 00004: val_loss improved from 0.03516 to 0.03505, saving model to weights.hdf5

9747/9747 [==============================] - 19s - loss: 0.0379 - val_loss: 0.0351

Epoch 6/8

9728/9747 [============================>.] - ETA: 0s - loss: 0.0384Epoch 00005: val_loss did not improve

9747/9747 [==============================] - 19s - loss: 0.0384 - val_loss: 0.0368

Epoch 7/8

9728/9747 [============================>.] - ETA: 0s - loss: 0.0382Epoch 00006: val_loss improved from 0.03505 to 0.03435, saving model to weights.hdf5

9747/9747 [==============================] - 19s - loss: 0.0381 - val_loss: 0.0343

Epoch 8/8

9728/9747 [============================>.] - ETA: 0s - loss: 0.0382Epoch 00007: val_loss did not improve

9747/9747 [==============================] - 19s - loss: 0.0382 - val_loss: 0.0376
```

2. Final Model Architecture

The final model architecture is detailed in section Model Architecture.

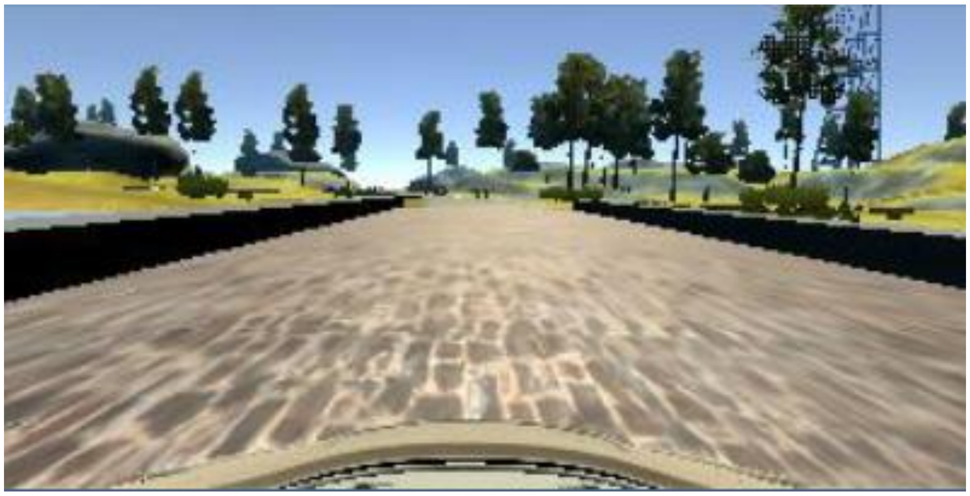
3. Creation of the Training Set & Training Process

I used the initial data set provided plus:

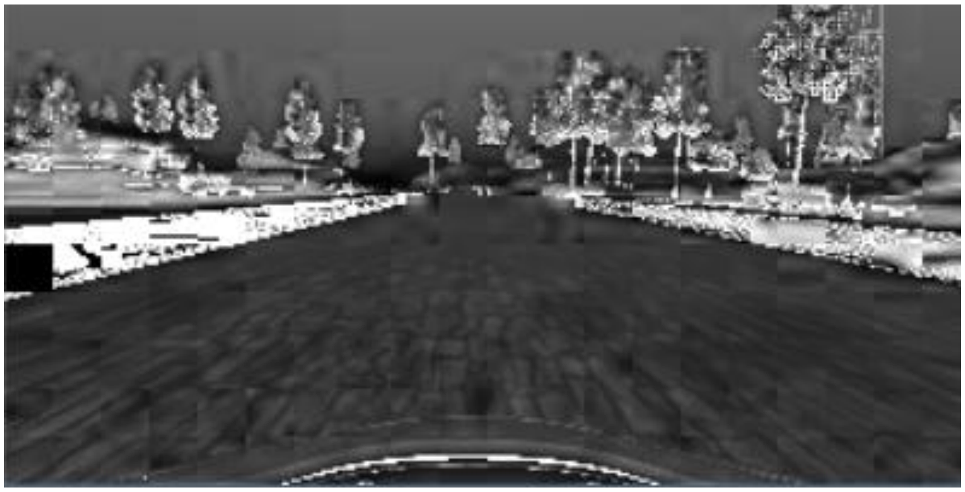
- One forward lap on the primary track.
- One backward lap on the primary track.
- On forward lap in the second track.
- Corner cases, specific locations were the steering angle was wrong to recover from the sides of the road.

To augment the data set, I flipped the images 50 % of the time and make sure to use the 3 images provided (using the random function within the generator).

A sample image is the following

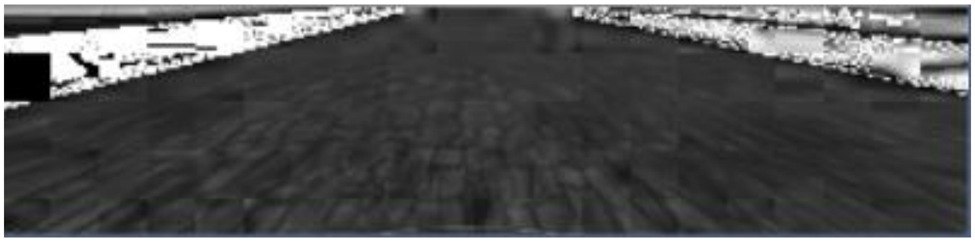


Within the generator the image is converted to HSV and projected the S channel:



This is the only pre-processing done.

Then, within the model pipeline the first step is to crop it:



And then normalize it

Considerations

I recorded data on "corner cases" but could not figure out how it impacted the training so kept the idea of using left and right images with a correction value (as suggested).

I decided to switch to exponential relu (ELU) as activation function as it seems that it helps to generate smoother angles for this case (discussed on the forum).

Other hyper-parameters from the Neural Network were modified based on try and error approach.