Behavioral Cloning

Udacity Self-Driving Car Nanodegree Program

Sulabh Matele - April 17, 2017

Behavioral Cloning Project

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

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

My project includes the following files:

- model.py containing the script to create and train the model
- 'drive_fast.py' and 'drive_slow.py' for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- Writeup_BehavioralCloning.pdf summarizing the results

Submission includes functional code 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
```

The model.py file contains the code for training and saving the convolution neural network.

The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

My neural network architecture is a modified version of NVIDIA architecture, with customized parameters for this project.

My model consists of a 5 convolution neural network with 5x5 filter at the first and 3x3 at other 4 layers, sizes and depths between 32 and 64.

The model includes RELU layers to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer.

Lambda - Normalization layer
Convolution2D (32, 5x5) - Maxpool 2x2, Activation ‘relu’
Convolution2D (32, 3x3) - Maxpool 2x2, Dropout 0.2, Activation ‘relu’
Convolution2D (64, 3x3) - Maxpool 2x2, Dropout 0.2, Activation ‘relu’
Convolution2D (32, 3x3) - Activation ‘relu’
Convolution2D (64, 3x3) - Activation ‘relu’
Flatten()
Dense (80) - Activation ‘relu’
Dense (40) - Activation ‘relu’
Dense (10)
Dense (1)

Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting.

The model was trained and validated on different data sets to ensure that the model was not overfitting. The model was tested by running it through the simulator for multiple laps and ensured that the vehicle could stay on the track.

Model parameter tuning

The model used an ‘adam' optimizer, so the learning rate was not tuned manually.

Appropriate training data

Initially I started training with the data provided by the project resource from Udacity which contains data from the track for complete track.

After some time working with the data I realized that the data was not sufficient and I started recording my training data and combined the resource data from Udacity and also the data which I recorded.

My data collection was mainly focused on:
- Smooth drive on curves
- Smooth drive on bridge
- Driving in reverse direction on track to make data generalized.
- Recovering from drifting off to side and then coming back to center.

The above data was recorded for multiple times during data recording.

In model.py, during data reading from .csv I read data from “driving_log.csv” which is the standard data provided by Udacity, then I read data from “more_driving_log.csv” which contains the data which I have recoded.

After reading the data and collecting into single collection, I split the data into Train and Validation data, I use the 30% data for validation.

During training the images are read from the “ALL_IMG/” folder which contains the images from Udacity standard data and also the images from my recording.

There was different image augmentation techniques used to make more data for training and also to make the model generalize to be able to work in different road conditions and situations.

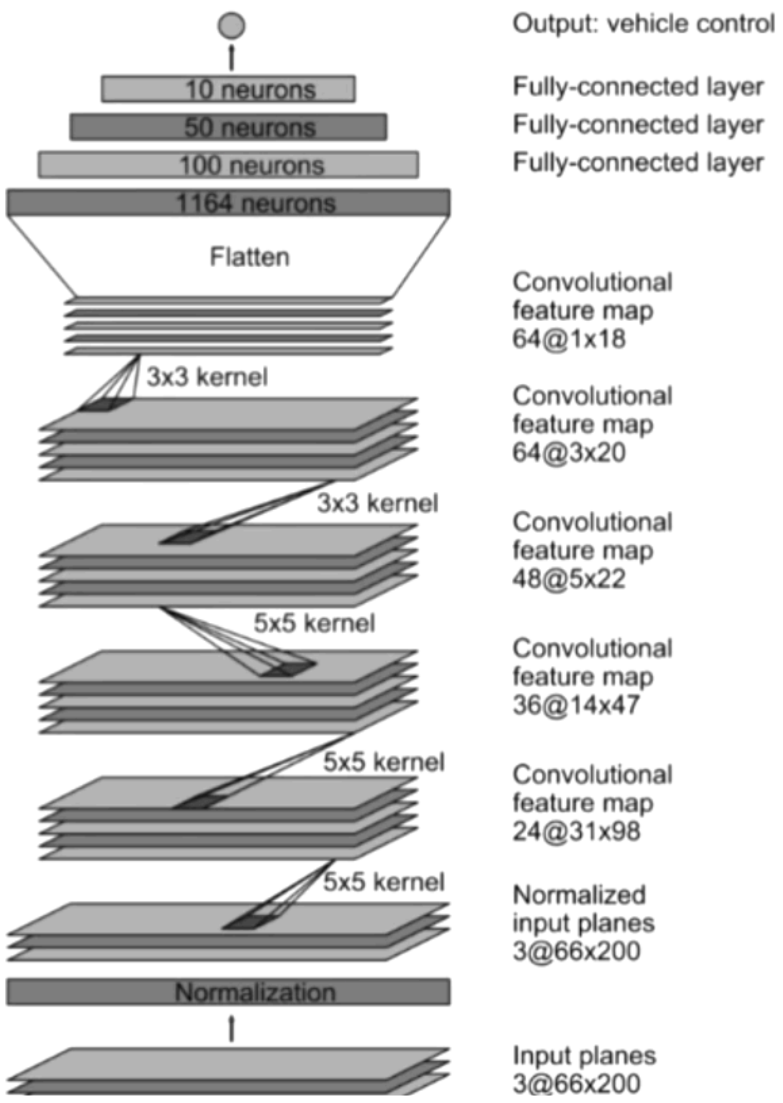
The image augmentation is discussed on different section further.

Solution Design Approach

The goal was very clear that the car should drive at the center of the road and should not drift off the road, just assume if there is some human inside then the car should be safe to ride on the track.

Initially with step by step I tried to make the model as simple as possible, and the car was just drifting off the road and was not able to handle curves.

Slowly I started increasing the convolution layers and also played some time with trying different combinations of 'relu', number of layers and filter sizes.



Then I referred the NVIDIA architecture and took that as base for this project and started training the same with my tried parameters, and the model started performing better.

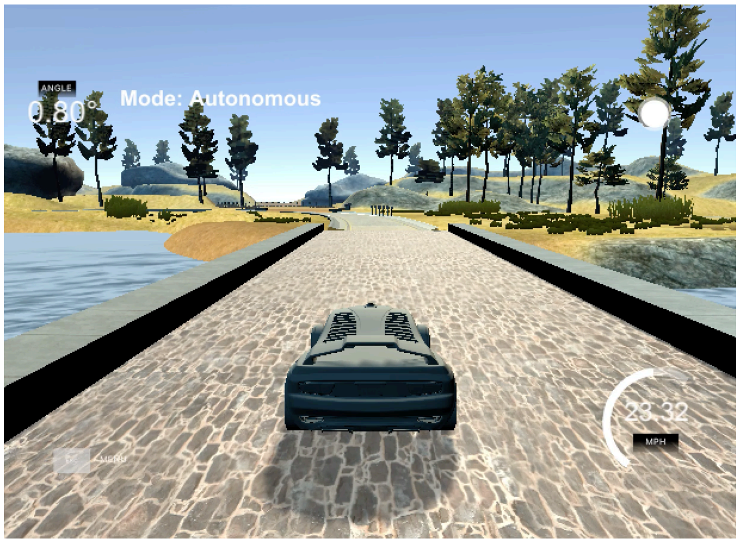
My final Neural network architecture is actually a modified version of the NVIDIA architecture with suited nodes and parameters for this project.

(Fig. Refers the original NVIDIA architecture)

At the end the modified model worked well for the track and car runs just fine on the track for multiple times without drifting off the road.

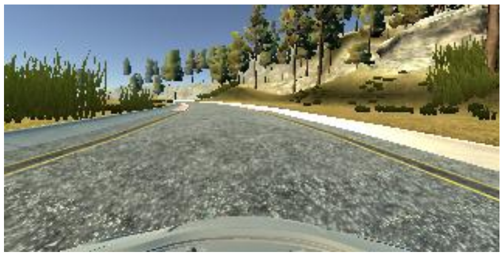
Creation of the Training Set & Training Process

To capture good driving behavior, I used the center lane driving standard material and also recorded my own training data with multiple laps and insist on curves, bridge and tricky behaviors. Here is an example image of center lane driving:

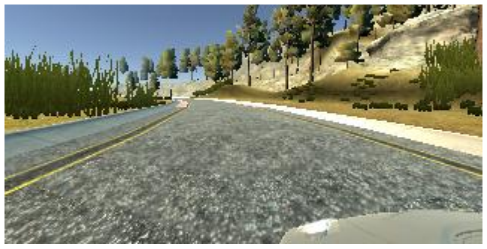


Training data contains **center, left and right** camera images, and each image of **size 160x320**

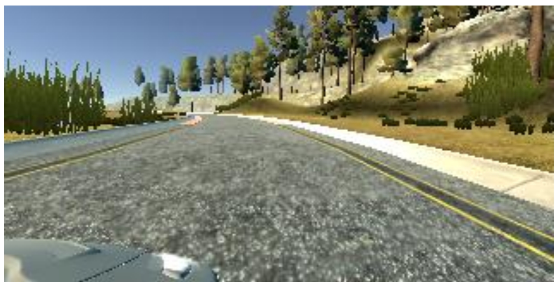
Center Camera Image



Left Camera Image



Right Camera Image

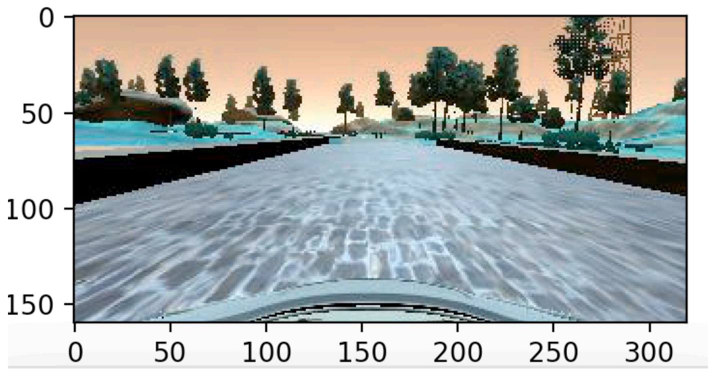


I have taken only center image for training my model.

In the center camera image above, it can easily be observed that,

- The upper portion of the image which contains mostly trees, and sky
- Bottom small portion contains the Car

These portion of the image are not really required for the model to learn drive on the track, so I used Keras “Cropping2D” to crop above 60 pixels and bottom 20 pixels from each image before starting the learning. Following images shows the the full size and cropped image view.



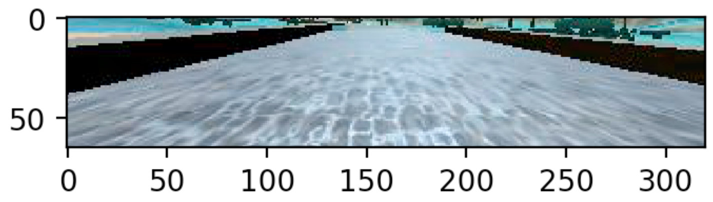


Image augmentation

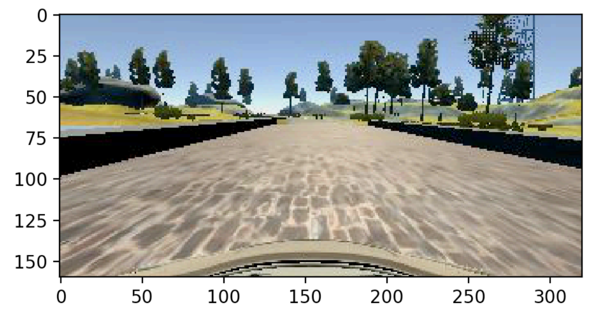
To make model more generalize and to create more data for training, I used different image augmentation which increased the training + validation data size by 4 times.

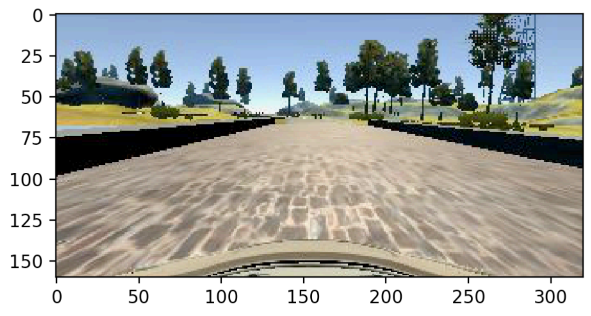
More data makes model to learn better, but I have also made sure that due to more data the model is not overfitting and hence used the dropout layers.

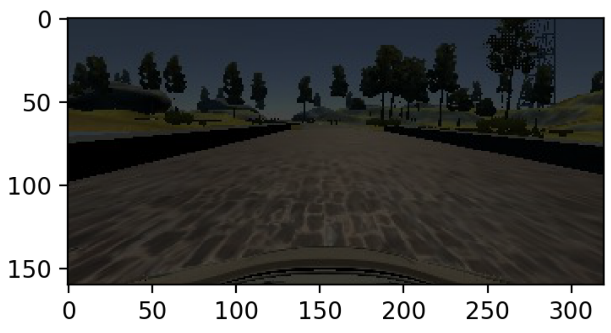
Following augmentation techniques were used, to generate more data.

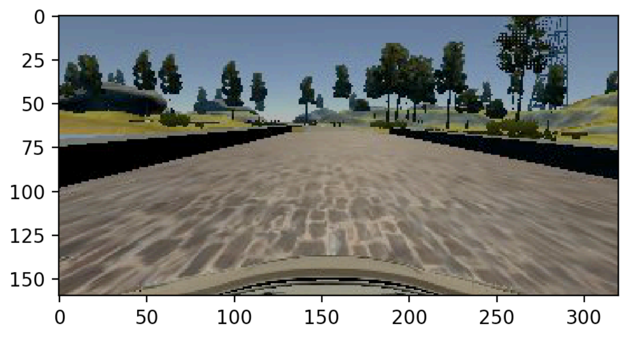
- Creating images with **different brightness** which could simulate the situation of driving in day and night on the same track and hence makes mode more generalized.

Following are the example images of different brightness.

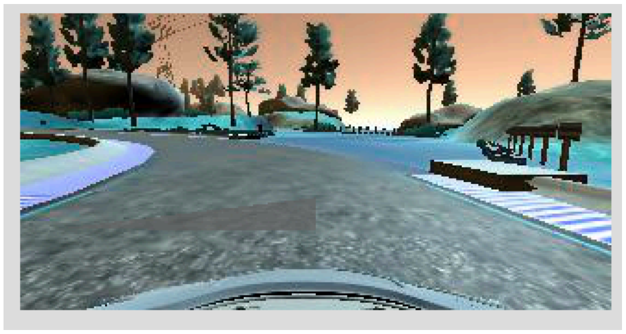
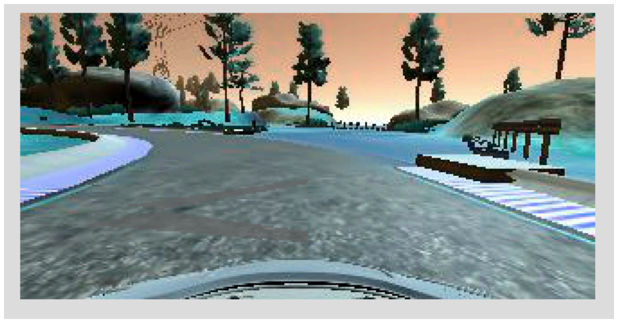
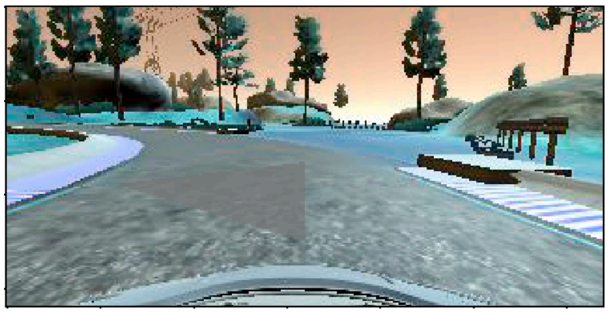
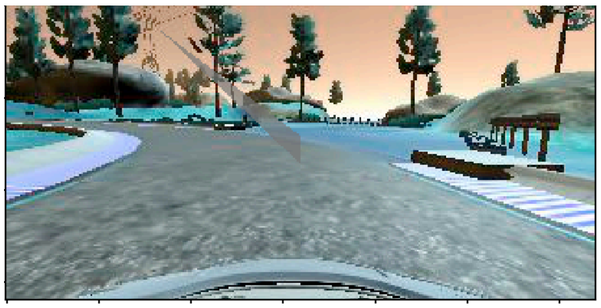








- Creation of different shadow shapes on the interested portion of the image, makes model to generalized with close to real road situation, with shows of different shape object.

- I also flipped the image horizontally to make model understand driving irrespective of the fact that the simulation track was left turn biased, so flipping the image makes mode understand that the car need to be at the center.

Since I flip the image so I also make the steering angle value negative ti make sure model understands that it was left turn made right turn so the steering should also turn right.

Keras generator for sampling

To train model with large amount of data we also need more and more memory to store and process data, but there are ways available to sample data and process sampled data to train the model, which makes the architecture more efficient.

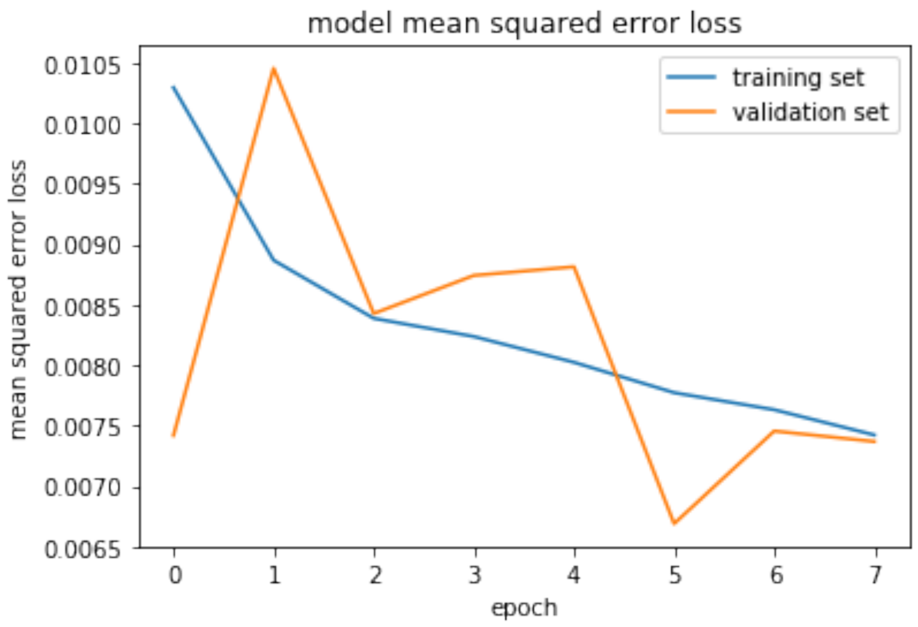
Using sampling method and training only small amount of data at a time, makes model slower to train, but it makes model very effective to run with less memory.

I used the Keras, “fit_generator” for generating the samples for training and validation data. I also multiply the samples_per_epoch by 4, since I generate 4 image per single recorded image, with the help of image augmentation discussed above.

So, for batch size 128, actually it process 128/4 = 32 recorded images.

Plotting the graph for loss

By plotting the graph for training and validation loss per epoch.


model mean squared error loss
mean squared error loss
epoch
training set
validation set