Behavioral Cloning Project

Goals of the project:

- 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

1. Submission includes all required files and can be used to run the simulator in autonomous mode

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
- model.h5 containing a trained convolutional neural network
- P3_Behavioral_Cloning_writeup_report.pdf summarizing the results.

2. Submission includes functional code and a youtube link to video recorded on track 1

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 youtube link for one lap of driving on Track 1 is at :

https://youtu.be/qT-bBjxrLTk

3. Submission code is usable and readable

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

File : model.py

1. An appropriate model architecture has been employed

My model consists of a convolution neural network that is implemented in the famous nvidia end-to-end steering model.

https://arxiv.org/abs/1604.07316

(code lines 18-80)

The model includes ELU (Exponential Linear Unit) layers to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer (code line 20).

He normalization is used for initialization of weights

The only change was to add a 1x1x3 convolution filter that helps in choosing the ideal color space automatically after normalization. This made a huge difference in how my model ran. This idea and preprocessing techniques were inspired from posts by Vivek Yadav. https://chatbotslife.com/@vivek.yadav

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (code lines 18-80)).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 237). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. Youtube link provide above.

3. Model parameter tuning

The model used an adam optimizer with a learning rate of 0.0001 (code line 77).

4. Appropriate training data

I used only the training data provided by Udacity. I used all three center, left and right camera images to train adding a steering offset of 0.25 for left/right images (code lines 229,230).

Model Architecture and Training Strategy

1. Solution Design Approach

Arriving at the model was a long iterative process.

I had a fixed goal of learning outcomes from the project when compared to Project 2 (traffic sign classification) apart from just finishing the project.

The goals were:

- Learn Keras generator to generate random values from a list or iterate through the list sequentially and reinitialize when it comes to end. I implemented both of these strategies one for training and one for validation. (code lines 156-215).
- Master preprocessing techniques and augmentation of a dataset because not always we have data that is huge for our training. I used horizontal translation, creating random shadows , changing brightness and horizontal/vertical flipping of images. (code lines 83-135).
- Train and validate iteratively saving only the model with best validation accuracy no matter the epoch it occurred in. (code lines 265-276)
- The biggest goal was to solve the problem with **CNN + LSTM** . Although I wasn't successful in driving through the track completely I could get an architecture working. You may check my work at https://github.com/ramsrigouthamg/Udacity_SelfDrivingCar_Term1/tree/master/Behavioral_Cloning_Keras/CNN_RNN_LSTM
- I later reverted back to my CNN architecture and finished the problem.

Iteration process:

I had a basic nvidia model with just brightness augmentation and flipping of left/right images. The model was going off the road on curves. Also the validation loss was increasing after a few epochs.

I added aggressive dropout to the nvidia model which improved the training. I added random shadow augmentation but it didn’t help improving the model on curved tracks. I later added random horizontal translations which helped the car to navigate through sharp turns. The only problem was that the car was confusing when there are shadows and driving on the curb. So I understood that although I added random shadows the model is not performing well here so I decided to add 3 1x1 filters right after the normalization layer of the model in the beginning. This helped the car navigate through the terrain very effectively as it learnt the ideal color space to operate in by itself. This was a game changer for me after which the model consistently drove well.

2. Final Model Architecture

The final model architecture (model.py lines 18-80) consisted of a convolution neural network with the following layers and layer sizes ...

- Lamba layer for normalization of input.
- 3 filters of size 1x1 followed by ELU.
- 24 filters of size 5x5 followed by subsampling (2x2), a dropout of 0.2 and ELU.
- 36 filters of size 5x5 followed by subsampling (2x2), a dropout of 0.2 and ELU.
- 48 filters of size 5x5 followed by subsampling (2x2), a dropout of 0.2 and ELU.
- 64 filters of size 3x3 followed by a dropout of 0.2 ELU.
- 64 filters of size 3x3 followed by a dropout of 0.2 ELU.
- A fully connected layer of 1164 followed by dropout of 0.4 and ELU
- A fully connected layer of 100 followed by dropout of 0.2 and ELU
- A fully connected layer of 50 followed by dropout of 0.2 and ELU
- A fully connected layer of 10 followed by an ELU
- A fully connected layer to output a single steering angle.

3. Creation of the Training Set & Training Process

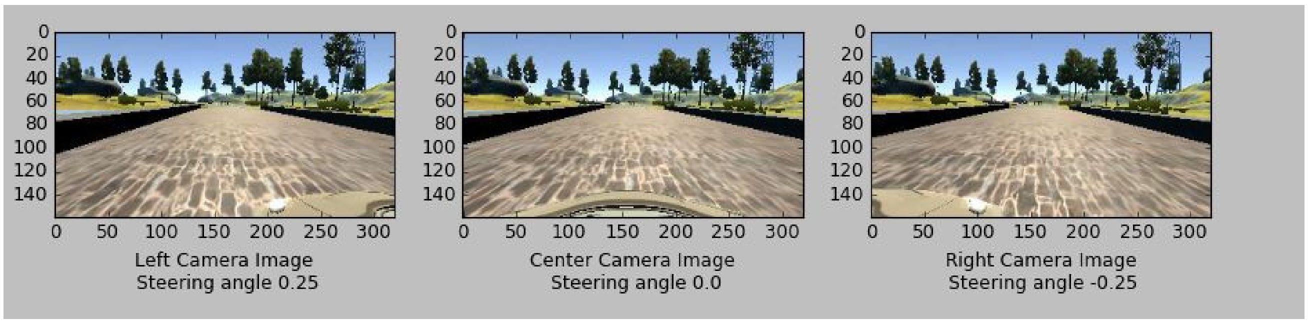
I used the dataset provided by udacity for my training and validation. The images are of the size 160x320. I added a steering angle of 0.25 for left camera images and subtracted a steering angle of 0.25 for right camera images. We only have steering angle for the center image. For example if the car sees from its center dash cam the same image it sees in the left camera image then it has to steer harder to he right in order to stay on the track. Although we don't have steering angle for the left image it can be approximated by adding a fixed bias (0.25) to the center image under this assumption.

Similarly if the car sees in its center dashcam what it sees in the right camera image then it has to steer harder to the left in order to stay on the track. Hence we subtract a steering angle of 0.25 from the center steering angle to approximate the steering angle for the right camera.

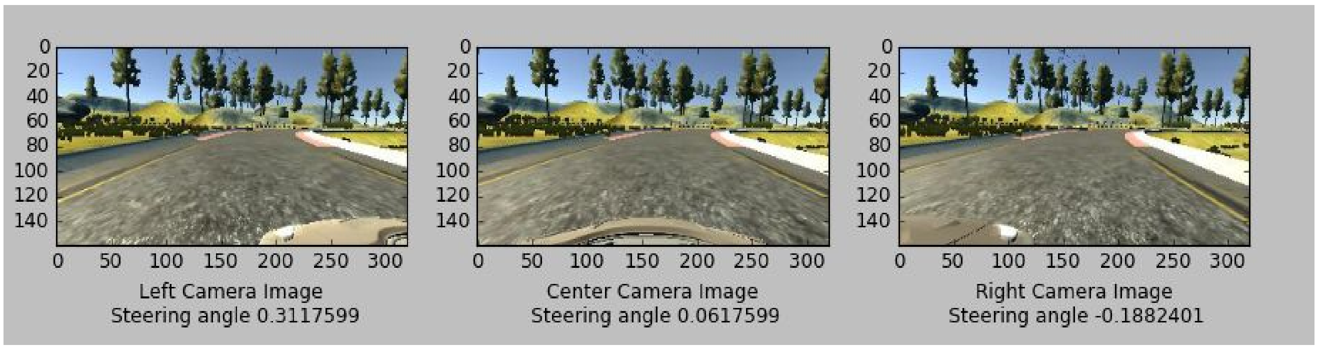
A few samples showing left, center and right camera images from udacity provided dataset before applying any preprocessing. Note the 0.25 offset for left and right camera when compared to center images.

Before preprocessing images:

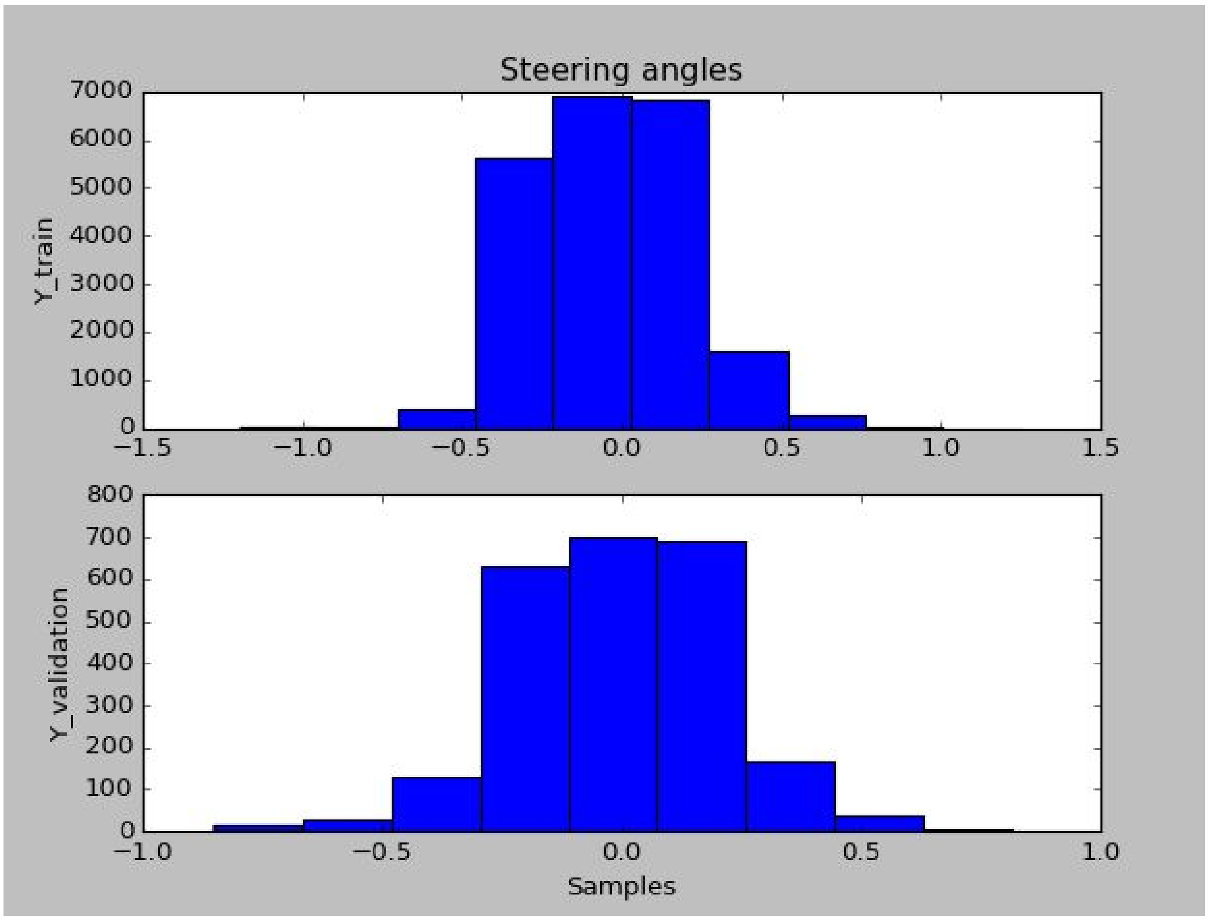
Sample 1:



Sample 2:



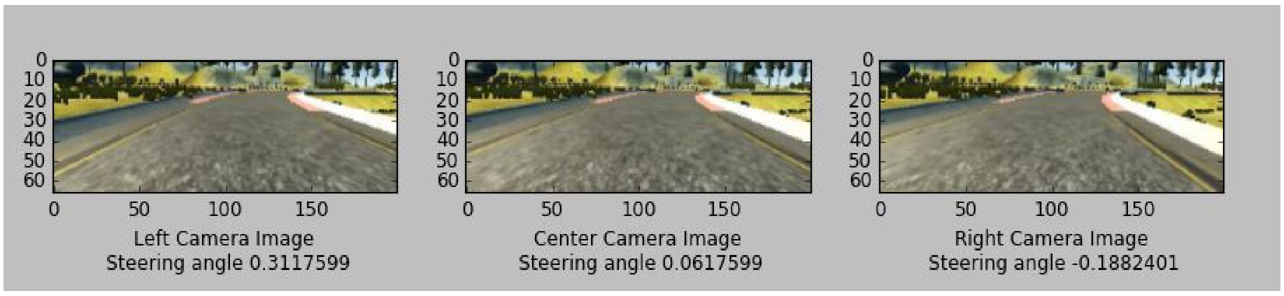
I split the udacity dataset into a **90:10** split for training and validation.After splitting I had about 22k images including left,right and center images for training and about 2.4k images for the validation set. Below you can see the steering angle distribution for training and validation set after using left, right and center images.



Preprocessing :

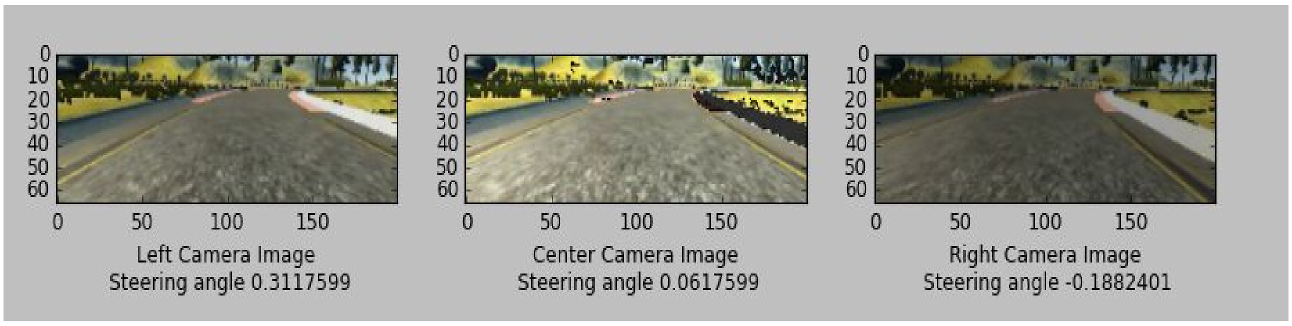
I preprocessed the images by **cropping** (code lines 122,123) and **resizing** to 200x66 to fit the dimensions of input to the keras model.

Images after **cropping** and **resizing**:

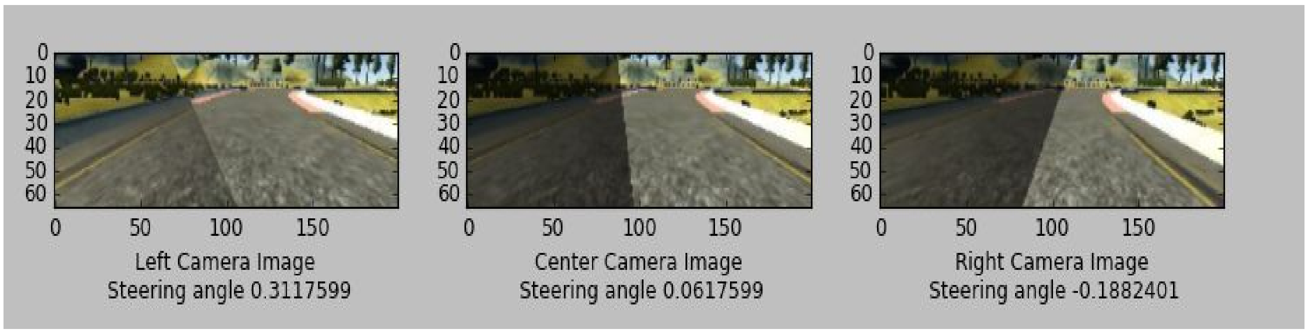


In the generator used for training I apply random brightness/shadows or flipping/translation to an image when it is called for preprocessing (code lines 138-150) from the generator during training.

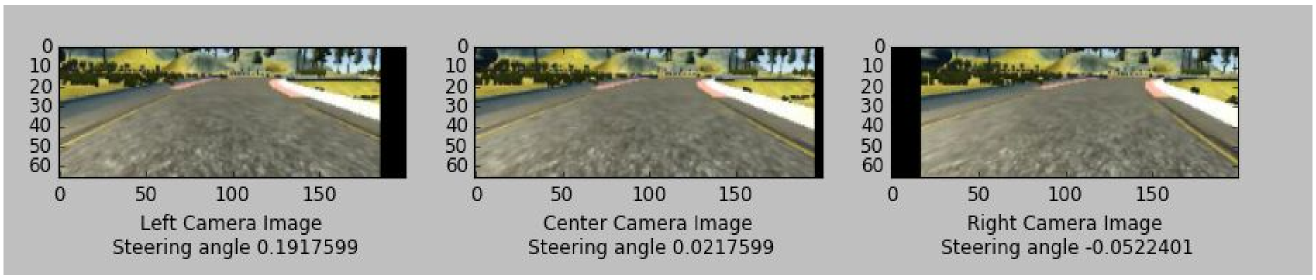
Brightness augmented images:



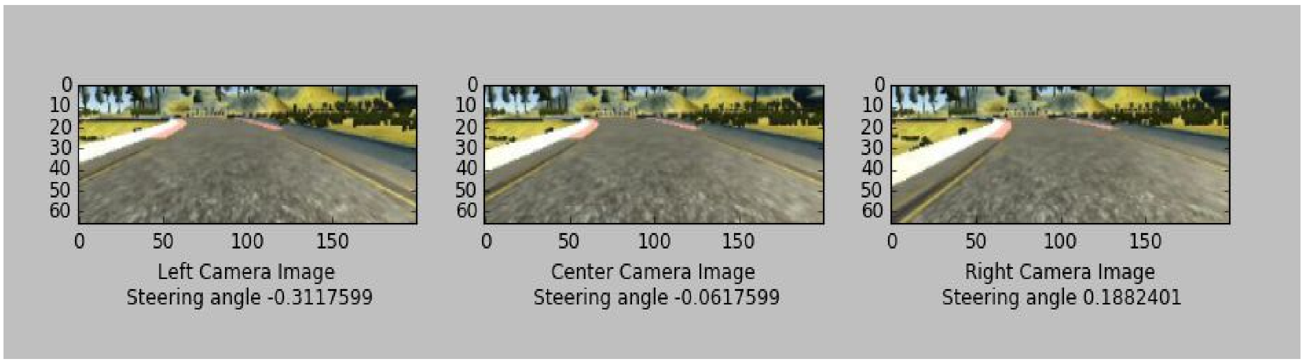
Random **Shadows:**



Random **Horizontal Translations (**modifying steering angle by 0.008 per pixel of translation. **)**



Flipping (Reversing the sign of steering angle**):**



I used the above mentioned augmentation techniques on the images and trained the model. Each epoch was trained with 19,200 images generated by the python generator in batches of 256. In each batch, 256 images are randomly picked from the whole training set of 22k images(including left, right and center). Each image from this batch of 256 again has random augmentations (discussed above) applied to it (code lines 140-151 in model.py).The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 12 and I saved the model with least validation set error. I used an adam optimizer so that manually training the learning rate wasn't necessary.