Behavioral Cloning Project

The goals / steps of this project are the following:

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_lenet.py contains the logic to create and train the model
drive.py for driving the car in autonomous mode
model.h5 containing a trained convolution neural network
model.json contains a JSON representation of the neural net that can be used to predict steering angles in realtime
helper.py with some preprocessing helper functions
writeup_report.pdf summarizing the results
P3-Behavioral-cloning.ipynb for displaying some of the images and observations.

2. Submission includes functional code

Using the Udacity provided data (driving log and images) and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.json
```

Model Architecture and Training Strategy

1. I went through the NVIDIA and comma ai network architectures which are very dense and tend to overfit and are very memory intensive due to the humongous amount of hyper parameters involved.

I wanted to start with something simple and see if by means of some pre-processing, image augmentation and hyper parameter tuning techniques I could make the car stay on the road at all times.
For this exercise I used simple yet very effective LeNet architecture. I was able to train this on my modest CPU in 5-6 minutes time for 10 epochs.
LeNet has just 2 convolutional layers and 2 fully connected layers and surprisingly it did well without much of image augmentation. The car was able to stay on course for both the tracks (1 & 2) for the whole time.

Few subtle changes had to be made to the model architecture as listed below

1) Fed input image of size 64x64 to neural network.
2) Normalized the image using Keras lambda layer
3) Changed the size of the convolution kernels from 5x5 to 3x3 in order extract more finer features and that made a big difference.
4) Changed the activation layer from Relu to ELU(Exponential Linear Units) - Less prone to vanishing gradient problem
5) Changed the size of the pooling kernel from 2x2 to 4x4 - That is because the input image I am using is the size of 64x64 instead of the standard 32x32 that was used for LeNet.
6) Using dropout made the loss worse and the car would keep falling off the road, so got rid of it. Have found it to be not so effectively for smaller architectures.

Neural Network Architecture
Input
Accepts a 64x64x1 image as input. S-Channel of HSV color space

Normalization
Execute image normalization within the neural net. [-0.5, 0.5]

Layer 1: Convolution. Filter Size: 3x3. Filter Depth: 6. Output shape would be 62x62x6.
Elu Activation.
4x4 Max-Pooling. The output shape would be 15x15x6.
Layer 2: Convolution. Filter Size: 3x3. Filter Depth: 16. Output shape would be 13x13x16.
Elu Activation.
4x4 Max-Pooling. The output shape would be 3x3x16.
Flatten. Flatten the output of each convolution layer after max-pooling
Layer 4: Fully Connected. 120 outputs.
Elu Activation.
Layer 5: Fully Connected. 84 outputs.
Elu Activation.

2. Preprocessing steps and design strategy
a) Normalize the image [-.5,.5] (keras lambda layer line 21 in model_lenet.py)
b) Shuffle the data before every epoch. (line 71 in model_lenet.py)
c) Convert the image to converts it to HSV color space keeping only the S channel. While tuning the architecture, the different color spaces were tested by training the model and then evaluating their performance on the track. The S-channel in HSV was selected as the input color space for the neural net as it was found to be best suited to discern road boundaries even where a clear demarcation was missing. (line 46 in helper.py)
d) Crop the image to eliminate the horizon and the hood part of the car. (line 47 in helper.py)
e) Split the data into training and validation sets. (line 72 in model_lenet.py)

Used the standard Udacity provided dataset to train the model.

I did not have to use the keras generator since the amount of data required to train the network successfully was small and my CPU was decent enough to hold that. It could have been more useful for denser architectures.

The correction angle was empirically selected to be 0.22, and the corresponding steering angles are calculated as follows:
left image angle = steering angle + 0.22
right image angle = steering angle - 0.22

3. Model parameter tuning

The model used an adam optimizer, so the learning rate did not have to be tuned manually.
Learning Rate: 0.001 (default with adam optimizer)
Epochs: 10
Loss Function: Mean square error - since the output is a single numeric value
Batch size: 128