Behavioural Cloning Write Up

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

Image References:

[image1]: ./examples/left_2018_05_07_22_03_08_447.jpg "Left Image"
[image2]: ./examples/right_2018_05_07_22_03_08_447.jpg "Right Image"
[image3]: ./examples/center_2018_05_07_22_03_08_447.jpg "Center Image"

[image4]: ./examples/left_2018_05_14_22_58_02_995.jpg "Left Image"
[image5]: ./examples/right_2018_05_14_22_58_02_995.jpg "Right Image"
[image6]: ./examples/center_2018_05_14_22_58_02_995.jpg "Center Image"

Rubric Points:

Here I will consider the [rubric points] (https://review.udacity.com/#!/rubrics/432/view) individually and describe how I addressed each point in my implementation.

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 and model, model.new.h5 and model.new2.h5 contains a trained convolution neural network. model.new.h5 is the final one but all models are well performing.
* WriteUp.pdf summarizing the results

2. 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

```sh
python drive.py model.h5
```

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

1. An appropriate model architecture has been employed and presented below:

- Cropping the image
- Normalization of data using Keras lambda layer
- Dropout [0.2]
- Convolution [24, (5, 5)]
    - Subsample [2, 2]
    - Relu activation [*model includes RELU layers to introduce nonlinearity*]
- Convolution [36, (5, 5)]
    - Subsample [2, 2]
    - Relu activation [*model includes RELU layers to introduce nonlinearity*]
- Convolution [48, (5, 5)]
    - Subsample [2, 2]
    - Relu activation [*model includes RELU layers to introduce nonlinearity*]
- Dropout [0.2] [*attempt to reduce overfitting*]
- Convolution [64, (3, 3)]
    - Relu activation [*model includes RELU layers to introduce nonlinearity*]
- Conv2D [64, (3, 3)]
    - Relu activation *[model includes RELU layers to introduce nonlinearity*]
- Flatten
- Dense [100]
- Dense [50]
- Dense [10]
- Dense [1]

2. Attempts to reduce overfitting in the model

- Dropout layers were deployed to reduce overfitting

- Training and validation are performed on different data sets to ensure robustness

- Additionally, model was tested by running it through the simulator and ensuring that the vehicle can drive on the track.

3. Model parameter tuning

The model used an adam optimizer to tune the learning rate automatically.

4. Appropriate training data

- Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane

- I employed data from 2 laps in the normal direction, and 1 lap on the opposite direction.

**Refer to the next section for mode details.*

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to train with incremental findings.

- My first step was to use a convolution neural network model similar to the LeNet used in the tutorials. I was not able to complete full lap.

- To gauge how well the model was working, data was split into a training and validation set. Overfitting was observed (low mean squared error on the training set, but a high mean squared error on the validation set)

- To delete the hood and the sky and unnecessary elements in the image, cropping was performed. This helped with overfitting.

- There was still some issue, so I deployed the similar architecture provided by Nvidia. This architecture was powerful and at the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture is a convolution neural network similar to the Nvidia's self driving car architecture:

- Cropping the image
- Normalization of data using Keras lambda layer
- Dropout [0.2]
- Convolution [24, (5, 5)]
    - Subsample [2, 2]
    - Relu activation [*model includes RELU layers to introduce nonlinearity*]
- Convolution [36, (5, 5)]
    - Subsample [2, 2]
    - Relu activation [*model includes RELU layers to introduce nonlinearity*]
- Convolution [48, (5, 5)]
    - Subsample [2, 2]
    - Relu activation [*model includes RELU layers to introduce nonlinearity*]
- Dropout [0.2] [*attempt to reduce overfitting*]
- Convolution [64, (3, 3)]
    - Relu activation [*model includes RELU layers to introduce nonlinearity*]
- Conv2D [64, (3, 3)]
    - Relu activation *[model includes RELU layers to introduce nonlinearity*]
- Flatten
- Dense [100]
- Dense [50]
- Dense [10]
- Dense [1]

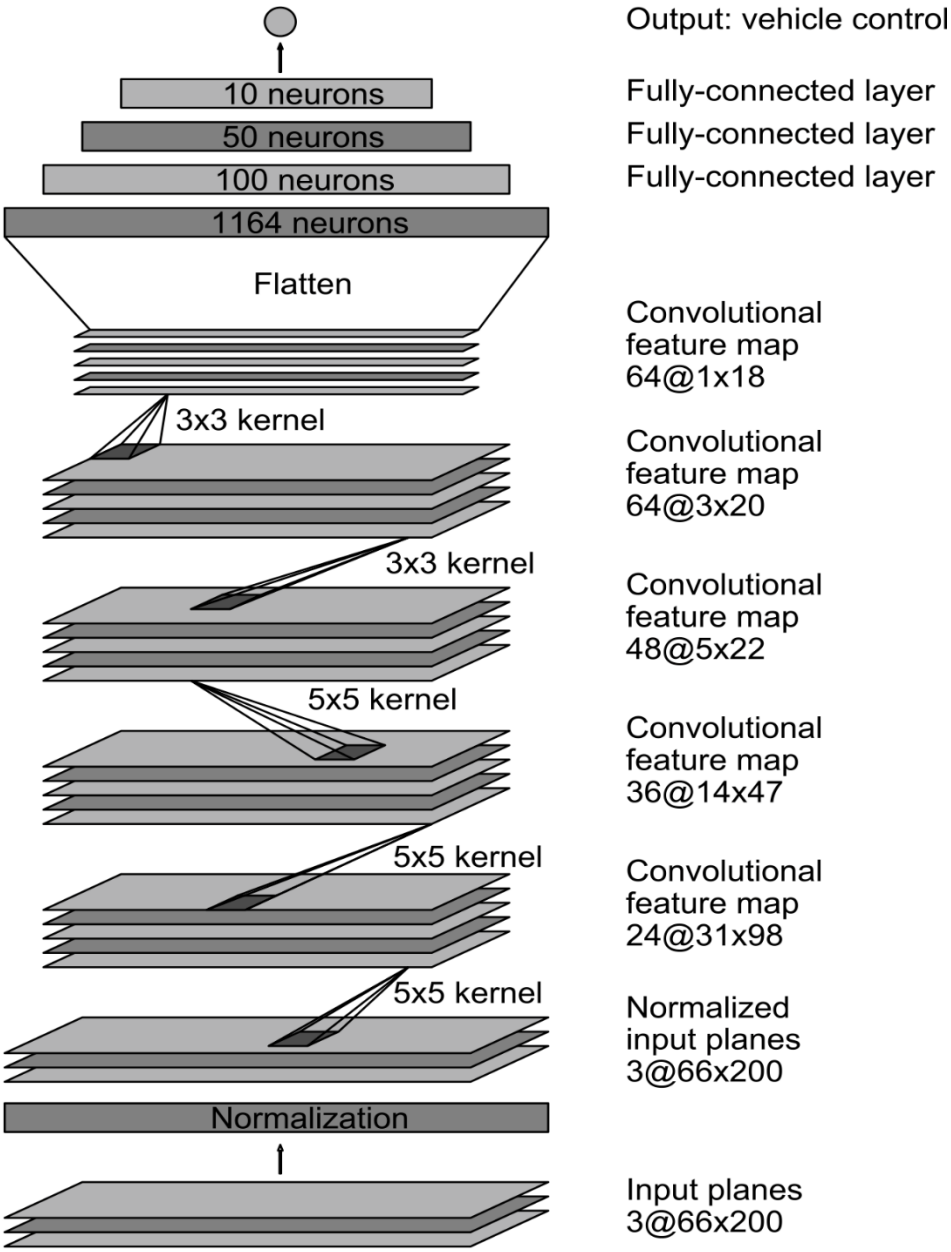

Figure 1: Architecture given by Nvidia

Output from the tensorflow log. Only 4 epochs were used the 2nd time, as the loss rate dropped rapidly.

3. Creation of the Training Set & Training Process

To capture good driving behaviour, I first recorded 2 laps on track one using centre lane driving. I then recorded the vehicle recovering from the left side and right sides of the road back to centre so that the vehicle would learn to recover when going close to the edge. Images were flipped images with further corresponding changes to augment the data. I then pre-processed this data by normalizing it and cropping it. Also, converted the data from BRG to RBG but it didn't help much. Training data used for training the model. The validation set was used to detect over or under fitting. The ideal number of epochs was 10 as evidenced by the loss rate stabilizing. In the final submission I only used 4 epochs, as the loss rate dropped really fast. Learning rate was adjusted by adam optimizer.