Behavioral Cloning

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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.

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:

Filename	Description
model.py	define and train the neual network
model.h5	saved model by keras
drive.py	communicate with simulator and use saved model to predict steering angle
video.py	converting video from images captured from Simulator
video.mp4	track 1 video record

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

```
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

My model consists of a consists of a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 36,48 and 64.

The model includes ELU layers to introduce nonlinearity, the images are cropped and the data is normalized in the model using a Keras lambda layer.

2. Attempts to reduce overfitting in the model

In order to avoid overfitting multi-camera images from left,centre,right were recorded. As looking at the track it has left turn bias because of that more samples were observed for left turns and samples are little skewed towards left angle data. In order to avoid overfitting each camera image (left,centra,right) were flipped and corresponding flipped angle is calculated in the training set. Hence for each steering angle for each record in the training data 6 steering angles were calculated for left,left_flipped,right,right_flipped,centre,centre_flipped images.

The model contains dropout layers in order to reduce overfitting.

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

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

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road, counter-clockwise driving and driving smoothly arround curves.

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

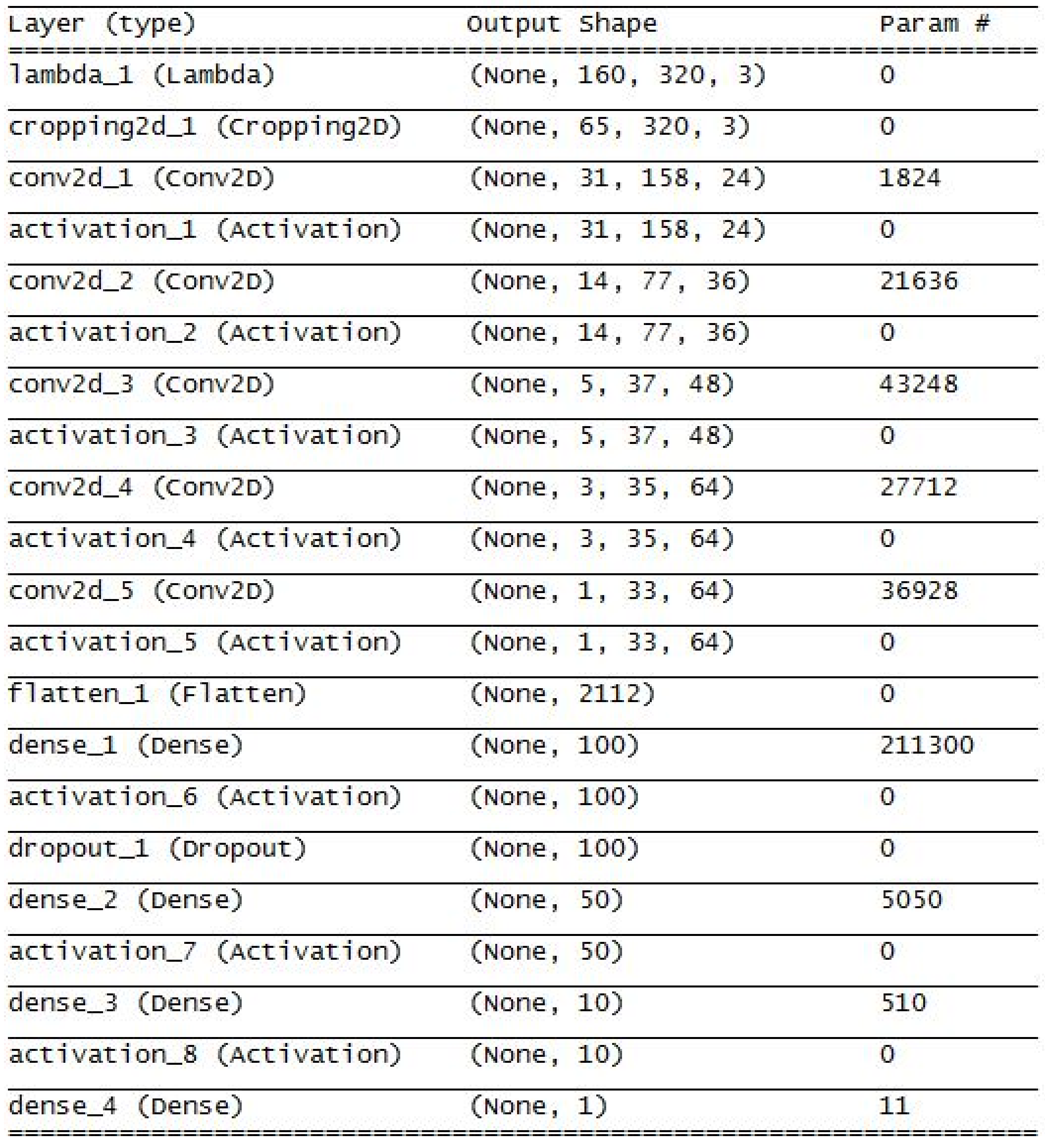
I tested the model provided by NVIDIA as discussed in Udacity Student hub. The model architecture is described here. As an input this model takes in image of the shape (60,266,3) but our dashboard images/training images are of size (160,320,3). Architecture of the remaining model was same only input dimensions have changed accordingly.

In the end, the model looks like as follows:

- Image normalization
- Convolution: 5x5, filter: 24, strides: 2x2, activation: ELU
- Convolution: 5x5, filter: 36, strides: 2x2, activation: ELU
- Convolution: 5x5, filter: 48, strides: 2x2, activation: ELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: ELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: ELU
- Drop out (0.25)
- Fully connected: neurons: 100, activation: ELU
- Fully connected: neurons: 50, activation: ELU
- Fully connected: neurons: 10, activation: ELU

- Fully connected: neurons: 1 (output)

As per the NVIDIA model, the convolution layers are meant to handle feature engineering and the fully connected layer for predicting the steering angle. The below is a model summary output from the Keras:



```
Layer (type)                     Output Shape              Param #
=================================================================
lambda_1 (Lambda)                (None, 160, 320, 3)       0
_________________________________________________________________
cropping2d_1 (Cropping2D)        (None, 65, 320, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)                (None, 31, 158, 24)       1824
_________________________________________________________________
activation_1 (Activation)        (None, 31, 158, 24)       0
_________________________________________________________________
conv2d_2 (Conv2D)                (None, 14, 77, 36)        21636
_________________________________________________________________
activation_2 (Activation)        (None, 14, 77, 36)        0
_________________________________________________________________
conv2d_3 (Conv2D)                (None, 5, 37, 48)         43248
_________________________________________________________________
activation_3 (Activation)        (None, 5, 37, 48)         0
_________________________________________________________________
conv2d_4 (Conv2D)                (None, 3, 35, 64)         27712
_________________________________________________________________
activation_4 (Activation)        (None, 3, 35, 64)         0
_________________________________________________________________
conv2d_5 (Conv2D)                (None, 1, 33, 64)         36928
_________________________________________________________________
activation_5 (Activation)        (None, 1, 33, 64)         0
_________________________________________________________________
flatten_1 (Flatten)              (None, 2112)              0
_________________________________________________________________
dense_1 (Dense)                  (None, 100)               211300
_________________________________________________________________
activation_6 (Activation)        (None, 100)               0
_________________________________________________________________
dropout_1 (Dropout)              (None, 100)               0
_________________________________________________________________
dense_2 (Dense)                  (None, 50)                5050
_________________________________________________________________
activation_7 (Activation)        (None, 50)                0
_________________________________________________________________
dense_3 (Dense)                  (None, 10)                510
_________________________________________________________________
activation_8 (Activation)        (None, 10)                0
_________________________________________________________________
dense_4 (Dense)                  (None, 1)                 11
=================================================================
```

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

To combat the overfitting, I modified the model with the dropout of 0.25 with a decent training samples.

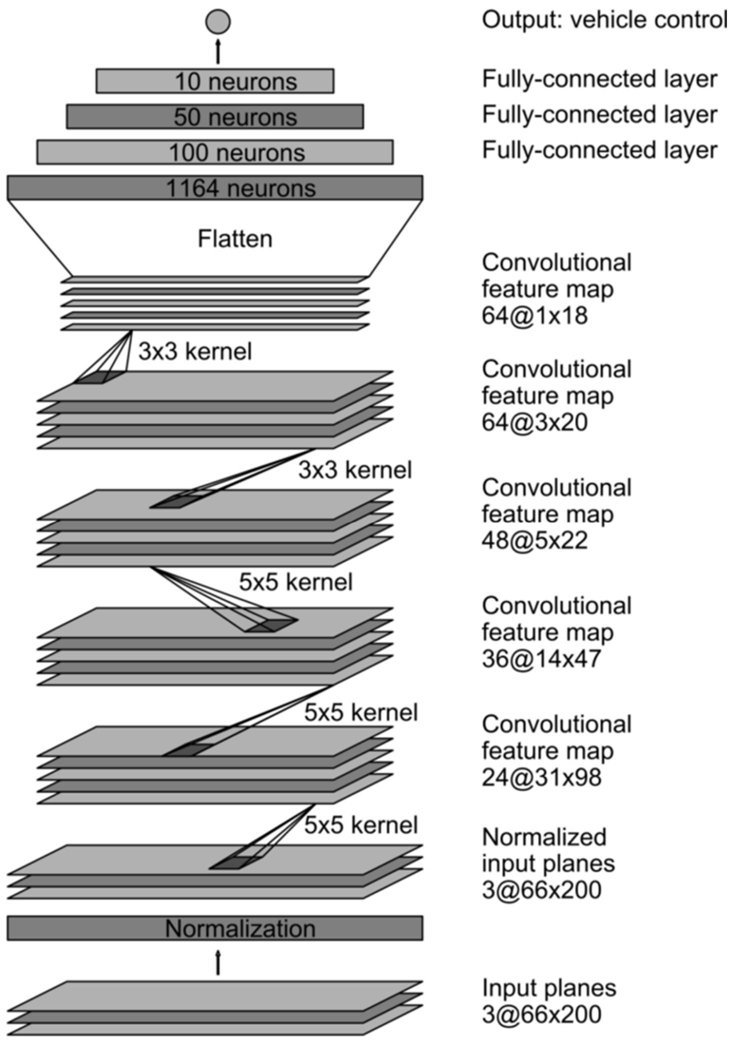
The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle almost touched the track-lines but model is trained well with recovery driving also. So model is able to get back to middle of the road itsel.

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 consisted of a convolution neural network with a normalization layer, three 5x5 convolution layers, two 3x3 convolution layers and 3 Full-Connected Layers + Output Layer.

Here is a visualization of the architecture (taken from the NVIDIA website):



3. Creation of the Training Set & Training Process

I splitted the images into train and validation set in order to measure the performance at every epoch. Testing was done using the simulator.

Model was trained for 5 epochs on the entire dataset and accracy was increasin in each epoch but very marginal.

I used mean squared error for the loss function to measure how close the model predicts to the given steering angle for each image. I used Adam optimizer for optimization with default value of learning rate. I used ModelCheckpoint from Keras to save the model only if the validation loss is improved which is checked for every epoch.