Behavioral Cloning

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Pre-process the collected data to improve the accuracy of the model
- 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.

1 Files Submitted & Code Quality

1.1 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 convolution neural network
- this file: writeup_report.pdf summarizing the results
- video.mp4: video capture of a successful simulator run around track1

1.2 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
```

1.3 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.

2 Model Architecture and Training Strategy

2.1 An appropriate model architecture has been employed

My model consists of a convolution neural network with 2 convolution layers with 5x5 filter sizes and 1 convolution layer with a 3x3 filter size. The depths of the layers go from 24 to 36 to 48 with these three convolution layers. (model.py lines 94-108)

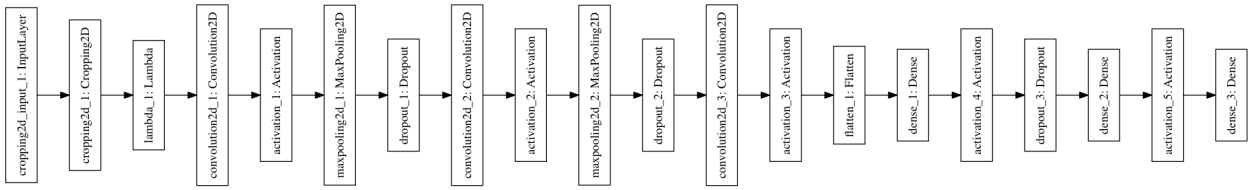
The model includes RELU activation layers to introduce nonlinearity.

The three convolution layers are followed by three fully connected layers with output dimensions reducing from 1024 to 100 to 1. The last fully connected layer is the output layer.

The data is normalized in the model using a Keras lambda layer (code line 92).

There is a cropping layer added at the beginning to eliminate 80 pixels from the top to eliminate the horizon and 20 pixels from the bottom to eliminate the car.

The full model is shown below:



2.2 Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 98, 104, 116). Dropout layers were added in two of the three convolution layers and in one of the three fully connected layers; the intent was to add DropOut layers in earlier layers so as to not drop higher level features.

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 214). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

2.3 MaxPool layers to reduce dimensionality

The model contains MaxPool layers to reduce the dimensionality of the output and to reduce the parameters. There are MaxPool layers added after 2 of the three convolution layers (model.py lines 97, 103).

2.4 Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 210).

2.5 Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of the following data:

- data provided with the project
- simulation data primarily for recovery scenarios from the left and right sides of the road.
- augmenting data; details of augmentation techniques I used are in the next section.

3 Model Architecture and Training Strategy

3.1 Solution Design Approach

The overall strategy for deriving a model architecture was to approach the problem as follows:

- Data collection - I collected data using the following mechanisms:
    - data provided with the project
    - simulation data primarily for recovery scenarios from the left and right sides of the road.
- Data Augmentation – I augmented the collected data using the following mechanisms:
    - Flip the images and use the negative of the steering angle
    - Translate the image by a small margin. This technique added a fair amount of time to the training and did not seem to improve the rate of loss significantly; so, I did not include this in the final model.
    - I considered adding shadows to random images as suggested by this paper. However, this technique also did not improve the accuracy on track1 which is well lit. So, I did not want to add the additional cost to training. I believe that this technique will be helpful for less lit tracks
- Pre-processing of data
    - Cropping – I added a cropping layer to my model to eliminate 80 px from the top (for the horizon) and 20 px from the bottom (for the car).
    - I added a resizing lambda layer to the model to resize the cropped

image but that did not seem to improve the model accuracy. So, I removed this layer from the final model.
    - I normalized the data using a lambda layer to scale the values to be in a range of [-1,1]
    - I used the left and right images collected by the simulator by adjusting the steering angle. I started with a correction of 0.2 and arrived at a final value of 0.25 after a few iterations.
- Iteratively arrive at a model architecture
    - My first step was to use a convolution neural network model similar to the model used by NVIDIA as described in this paper.
    - The training time using the NVIDI model was fairly long. So, I pruned some of the convolution layers and some of the fully connected layers from the model to arrive at the final model
- Train and test the accuracy of the model on validation data
    - 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 reduce the overfitting, I added DropOut layers to the model after 2 of the 3 convolution layers and after 1 of the 3 fully connected layers. This reduced the loss on the validation data set.

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 fell off the track. These happened in the following spots:

- The right lane line disappeared into a muddy patch.
- After going over the bridge the car continued to veer to the left. I suspect this was due to the transition from the bridge to the normal road.
- After the lane lines with red borders transitioned to normal lane lines, the car tended to veer towards the lanes.

To improve the driving behavior in these cases, I collected more training data using the simulator. I primarily collected data for recovery scenarios.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

3.2 Final Model Architecture

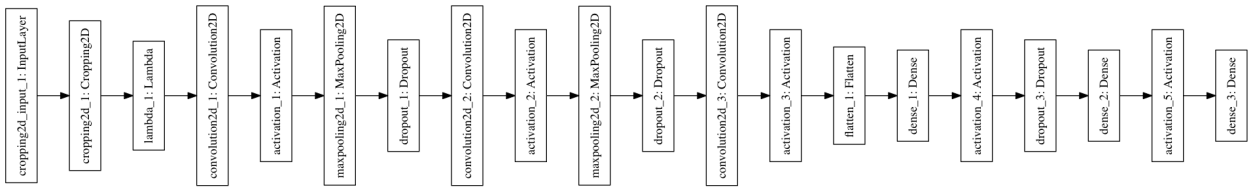
The final model consists of a convolution neural network with 2 convolution layers with 5x5 filter sizes and 1 convolution layer with a 3x3 filter size. The

depths of the layers go from 24 to 36 to 48 with these three convolution layers. (model.py lines 94-108)

The model includes RELU activation layers to introduce nonlinearity.

The three convolution layers are followed by three fully connected layers with output dimensions reducing from 1024 to 100 to 1. The last fully connected layer is the output layer.

Here is a visualization of the architecture:



4 Creation of the Training Set & Training Process

To capture good driving behavior, I started with the sample data set provided by the project. I them added to this data set by:

- Driving a lap on sections of track 1
- Adding recovery images for sections of the track where my model was causing the car to veer closer to the lane lines.

After the collection process, I had about 15000 data points where each data point captured a center, left, and right image. I used all three images from each data point by adding a steering angle correction for the left and right images. I also used flipped versions of the 3 images with a negative of the steering angle.

I finally randomly shuffled the data set and put 20% of the data into a validation set. I used this training data for training the model. The validation set helped determine if the model was over or under fitting. I used a generator to ensure that I did not load the large data set into memory. I trained the entire model on my non-GPU MacBook and the generator pattern was critical to be able to complete the training.

The ideal number of epochs was 5 as evidenced by the rate of loss which plateaued after 5 epochs.

I used an adam optimizer so that manually training the learning rate wasn't

necessary.