Project 3: Behavioural Cloning

The goals / steps of this project are the following:

Use the simulator to collect data of good driving behaviour

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

Criteria	Description	Status
Submission	Model.py	✓
	Drive.py	✓
	Model.h5	✓
	Video.mp4	✓
	A write-up report (PDF)	✓
Quality of Code	Code functional	✓
	Code useable and readable	✓
Model Architecture	Appropriate model architecture	✓
	Overfitting reduction	✓
	Tuning of model parameters	✓
	Selection of training data	✓
Architecture and Training Documentation	README – Solution design	✓
	README – Details of the architecture	✓
	README – Usage of dataset to train, validate and test	✓
Simulation	Acceptable driving behaviour	✓

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:

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

writeup_report.md or writeup_report.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

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

The model is inspired from NVIDIA's paper: End-to-End Learning for Self-Driving Cars.

This model uses one normalization layer, followed by 3 5x5 convolutional layer, followed by 2 3x3 convolutional layer and 3 fully connected layer.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 123).

More data is collected and augmented (flip image) 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 (model.py line 134).

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.

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

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to see

My first step was to use a convolution neural network model similar to the NVIDIA End-to-End Deep Learning paper. I thought this model might be appropriate because of their successful implementation on the public road.

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.

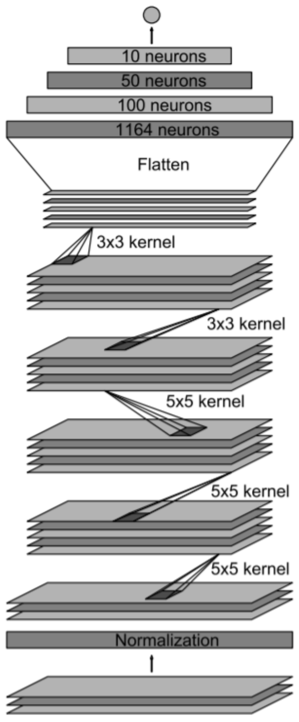
Then I trained the model again with one more epoch.

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. To improve the driving behavior in these cases, I collected more data, especially the location the vehicle underperformed (for example, poor steering control when negotiating curved road).

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

2. Final Model Architecture

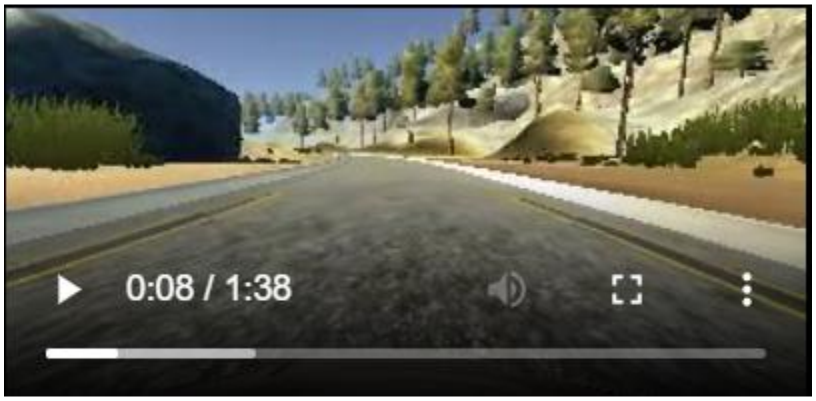
The final model architecture (model.py lines 110-129) consisted of 5 convolution neural network (3 5x5 convolutional layer, 2 3x3 convolutional layer), 3 fully connected layer. Refer below for an example of the architecture.



3. Creation of the Training Set & Training Process

1. Collect data by driving in the simulator
2. Check for the model for overfitting or under fitting
3. Analyse the possible cause from the dataset, and add more training data to improve the accuracy or to reduce loss
4. Run the deep learning model again
5. Implement in the simulations and check for safe behaviour

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:



Then I repeated this process on track two in order to get more data points.

To augment the data sat, I also flipped images and angles thinking that this would reduce overfitting and promote generalization.

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. The ideal number of epochs was 2 as evidenced. I used an adam optimizer so that manually training the learning rate wasn't necessary.