writeup_report.md

Behavioral Cloning

Behavioral Cloning Project Writeup

The goals of this Behavioral Cloning 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

The rubric points as specified dictated how each point of the project was implemented.-

Files Submitted & Code Quality

1. Required files that can be used to run the simulator in autonomous mode

This project submissions includes the following files:

- **model.py** contains the script used to create and train the model
- **drive.py** contains the script used for driving the car in autonomous mode in the simulator
- **model.h5** contains the trained convolution neural network as a result of running **model.py**
- **writeup_report.md** and **writeup_report.pdf** summarize the project results
- **video.mp4** contains a video of the vehicle autonomously driving for more than one lap around the track

2. Functional code

Using the Udacity provided simulator and my **drive.py** file, the car can be driven autonomously around the track by executing the following Terminal command:

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

Note that the simulator must be running in autonomous mode while the above command is executed.

3. 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 used for training and validating the model, and also contains comments to explain how the code works.

Model Architecture and Training Strategy

1. Model architecture

The model architecture begins on line 86 of **model.py**, where a `Sequential` model (i.e., a linear stack of layers) has been been implemented in Keras.

The first layer in the model is a `Lambda` layer. This inputs images recorded from driving the simulator in training mode and normalizes them (**model.py** line 87).

The next layer in the model is a `Cropping2D` layer. This layer takes the normalized images and crops them 65 pixels from the top and 20 pixels from the bottom, which saves on processing by removing image pixels that are unimportant to the driving scene (**model.py** line 88).

The model then uses five sequential `Conv2D` layers, making this model a convolutional neural network. The first three `Conv2D` layers use `filters` that increase from 24 to 48. All of the first three `Conv2D` layers use a `kernel_size` of 5x5, `strides` of 2x2, and ReLU `activation` to introduce nonlinearity (**model.py** lines 89-91). The final two `Conv2D` layers are exactly the same, where both use a `kernel_size` of 3x3, no specified `strides` (which use a default of 1x1), and ReLU `activation` (**model.py** lines 92-93). No other `Conv2D` function parameters were specified, leaving them at their defaults (e.g., `padding = 'valid'`).

After the convolutional layers, the model then uses a `MaxPooling2D` layer to down sample the feature maps and summarize features present in the spatial data (**model.py** line 94).

The output of the max pooling layer was flattened using a `Flatten` layer (**model.py** line 95).

Five `Dense` densely connected neural network layers were added with decreasing unit sizes from 500 to 1 (**model.py** lines 96-100). No additional parameters of the `Dense` function, such as activation or bias, were used.

2. Attempts to reduce overfitting in the model

The effort to reduce overfitting was performed primarily by data augmentation. Several laps of driving were recorded on both tracks of the driving simulator. Additionally, multiple laps of *recovery driving* were recorded to train the model on driving toward to the center of the road if found near the road's edge. In sum, 88,845 images were collected (1.3 GB) for training.

For each captured data point, the center, left, and right camera images were used. All of these images were flipped horizontally to further augment the data set. To compensate for left and right camera offsets, a `steering_correction` factor of 0.065 was used to keep the vehicle in the center of the road.

The model architecture does not contain additional layers such as `Dropout` to reduce overfitting. Models trained that included a `Dropout` layer did not result in observed autonomous driving performance.

When training the model, it was observed that training over a greater number of epochs often resulted in an increased amount of loss in the final model. The final version of the model trained over only five epochs (**model.py** line 102).

3. Model parameter tuning

During model compilation, the adam optimizer was used. Therefore, it was unnecessary to manually tune the model learning rate **model.py** line 101). The step of compiling the model also utilized the mean squared error loss function (`loss = 'mse'`), as it is well suited for a regression problem (as opposed to a classification problem).

4. Appropriate training data

As previously mentioned, several laps of driving were recorded on both tracks of the driving simulator to collect the appropriate training data. Additionally, multiple laps of *recovery driving* were recorded to train the model on driving toward to the center of the road if found near the road's edge. Images from all three cameras collected by the simulator recording were utilized. In sum, 88,845 images were collected (1.3 GB) for training.

Model Architecture and Training Strategy

1. Solution Design Approach

The strategy for creating the model architecture was to follow the lessons learned in the Behavioral Cloning section of the Udacity Self-Driving Car Engineer Nanodegree program.

The first step was to use a convolutional neural network model similar to the one implemented in NVIDIA's End to End Learning for Self-Driving Cars. Being that this model proved to be effective for one of the leaders in deep learning and self-driving car technologies, it would be, at the very least, a good place to start. The convolutional layers of the final model nearly matched that of NVIDIA's model. However, the referenced paper by NVIDIA makes no mention of Rectified Linear Units (ReLU) in their convolutional layers.

In order to test the model, image and steering angle data was split into a training set (80%) and validation set (20%). The initial model tested had a low mean squared error on the training set relative to a much higher mean squared error on the validation set. As learned in the Behavioral Cloning section of the Udacity Self-Driving Car Engineer Nanodegree program, this implied that the model was overfitting.

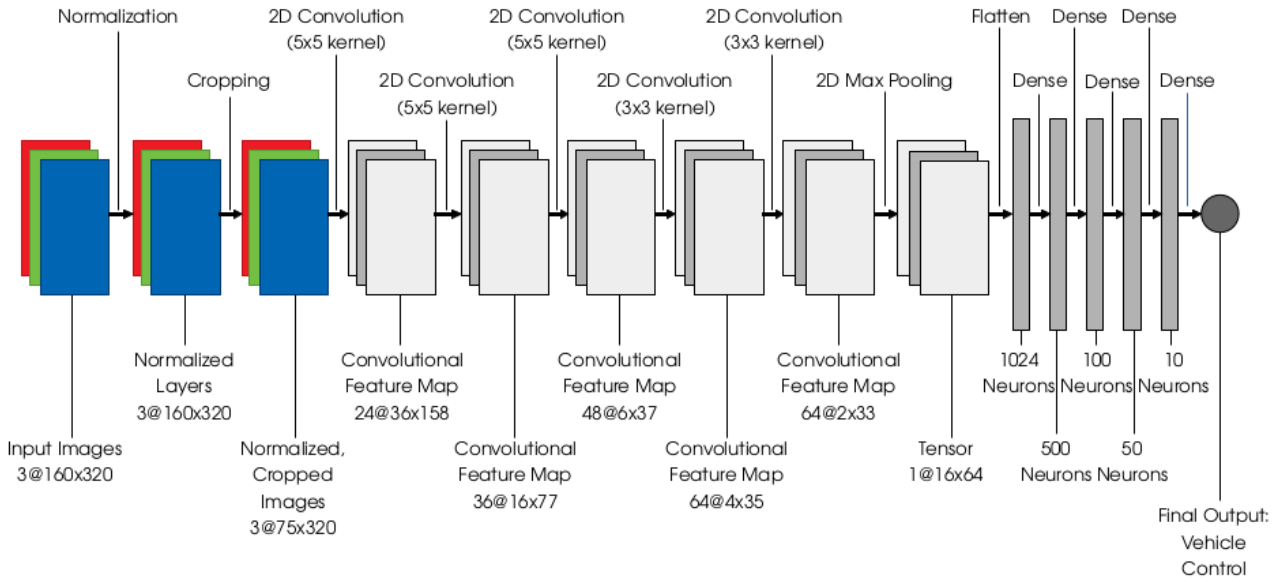
To reduce overfitting, more driving data was collected, including driving on both tracks, recovery driving, and driving slowly in the center of the road. Furthermore, the model was trained using five epochs, as it was observed that validation loss increased over a higher number of epochs. The models produced after a high number of training epochs also resulted in decreased autonomous driving performance. In other words, overtrained models drove off of the road in the simulator.

After making these adjustments, the vehicle was able to autonomously drive around the track repeatedly without driving off of the road.

2. Final Model Architecture

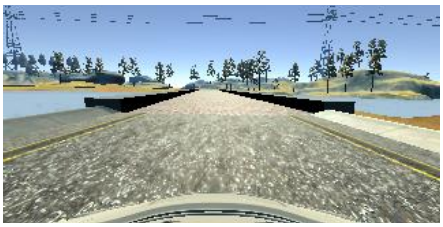
The final model architecture (model.py lines 18-24) consisted of a convolution neural network with the following layers and layer sizes ...

The below image is a visualization of the model architecture.



3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded three laps on track one by driving in the center of the track. Below is an example image of driving in the center of the track:

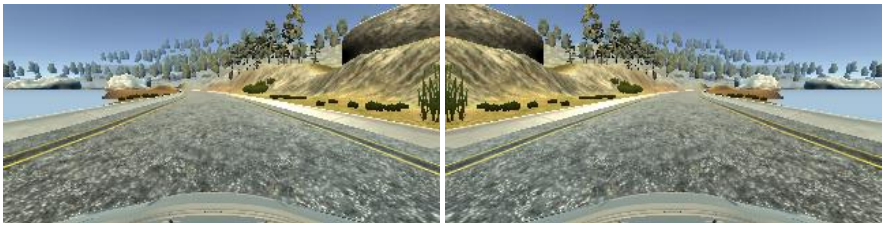


Over one lap of recovery driving was recorded in an effort to make the car learn how to adhere to the center of the track, and correct if it was to find itself at the track's edge. To do so, the car would be driven to the edge of the track, and recording would be activated prior to driving the car to the center of the track. Recording would then be deactivated once the car reached the center of the track. This process was repeated throughout the track, recovering from both right and left edges of the track. An example of recovery driving is displayed in the five images below.

Two laps of driving were recorded on track two. The image below is an example image from track two as part of the collected data set.

Two additional laps of slow driving on track one were recorded to augment the data set. More attention was paid to staying in the center of the track than previously recorded driving. The following image is another example of driving on the center of the track.

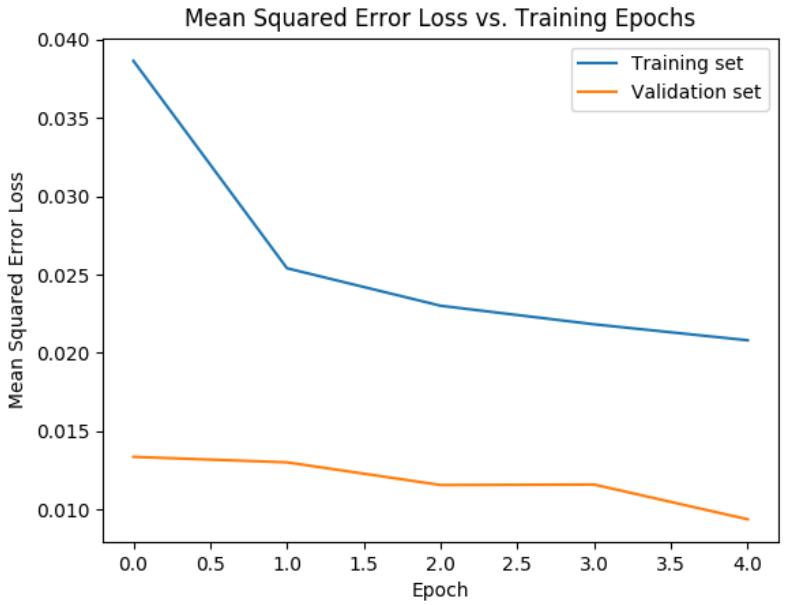
In **model.py**, each processed driving image was additionally flipped horizontally to further augment the data set. Hence, the data set size was doubled. A factor of negative one was also applied to the steering angles corresponding to the flipped images so that the model would train to drive in the correct direction. Below is an image recorded from driving the car in the simulator, followed by an image of its horizontally flipped counterpart.



As a result of recording data in the simulator, there were 88,845 images (1.3 GB) available for the model to train on. The image data was preprocessed using Keras to normalize and crop the images.

To prepare the images neural network training, the image data was randomly shuffled, in which 20% of the shuffled images were reserved for the validation data set. Loss reported by model training could vary, which appeared to be a result of shuffling.

The image training data set was used to train the model (80% of the total image data). The validation set was used to determine if the model was over or under fitting. The ideal number of epochs was approximately five, as evidenced by the increased mean squared error loss observed when training over a larger number of epochs (not shown). The adam optimizer was used to prevent the need to manually adjust the model's learning rate. Below is a graph of mean squared error loss versus training epochs from the final model, trained on 5 epochs. Note that the x-axis of the graph is zero-based, making the first epoch report at x = 0.



Results of Training the Model

Recorded video of the car driving autonomously on track one can be seen in the file video.mp4.