Behavioral Cloning

Behavioral Cloning Project

The goals / steps of this project are the following:

- Prepare a training and validation set for the model
- Build a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with the 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:

- 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_WX.md or writeup_WX.pdf summarizing the results
- Video.mp4 demonstrating a successful run of one lap using the model trained

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 convolution neural network with 5x5 or 3x3 filter sizes and depths between 24 and 64 (model.py lines 85-99), which is connected to three fully-connected (FC) layers with the ultimate output being a scalar (vehicle steering angle).

The input training data is normalized in the model using a Keras Lambda layer and cropped using a Keras Cropping2D layer (model.py lines 86-87) to 65x320 size images. This is to ensure that the training process is more efficient and only the relevant image contents (i.e., the track and its associated elements) are kept for the training.

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 103). 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 101).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I have expanded the training/validation set with augmented data which includes images from all of the three cameras (with enhanced steering angles for the left and right cameras) and images flipped from all existing ones.

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

Model Architecture and Training Documentation

1. Solution Design Approach

The overall strategy for deriving a model architecture was shown in the following steps:

1. Setup training/validation set and training target
2. Build an initial model quickly
3. Tune the model with performance improvement strategies

My first step was to read in all training data and setup an initial model pipeline quickly. The model includes two Convolution2D layers with 5x5 filter sizes and the depth of 6 before applying MaxPooling2D, which is then linked to three fully-connected (FC) layers with 120, 84 and 1 neuron(s), respectively. The training target is set as "mse" which stands for `mean square error` and the `adam` algorithm is chosen as the optimizer. Then I trained the model with a training/validation split of 80:20.

After training the initial model, I have tested it running in the simulator in `autonomous mode`. The car could be driven automatically however it was easily fell off the track. This did not surprised me since the `mse` loss showed huge numbers for both training and validation sets. Therefore, the next step was to Tune the model with various performance improvement strategies.

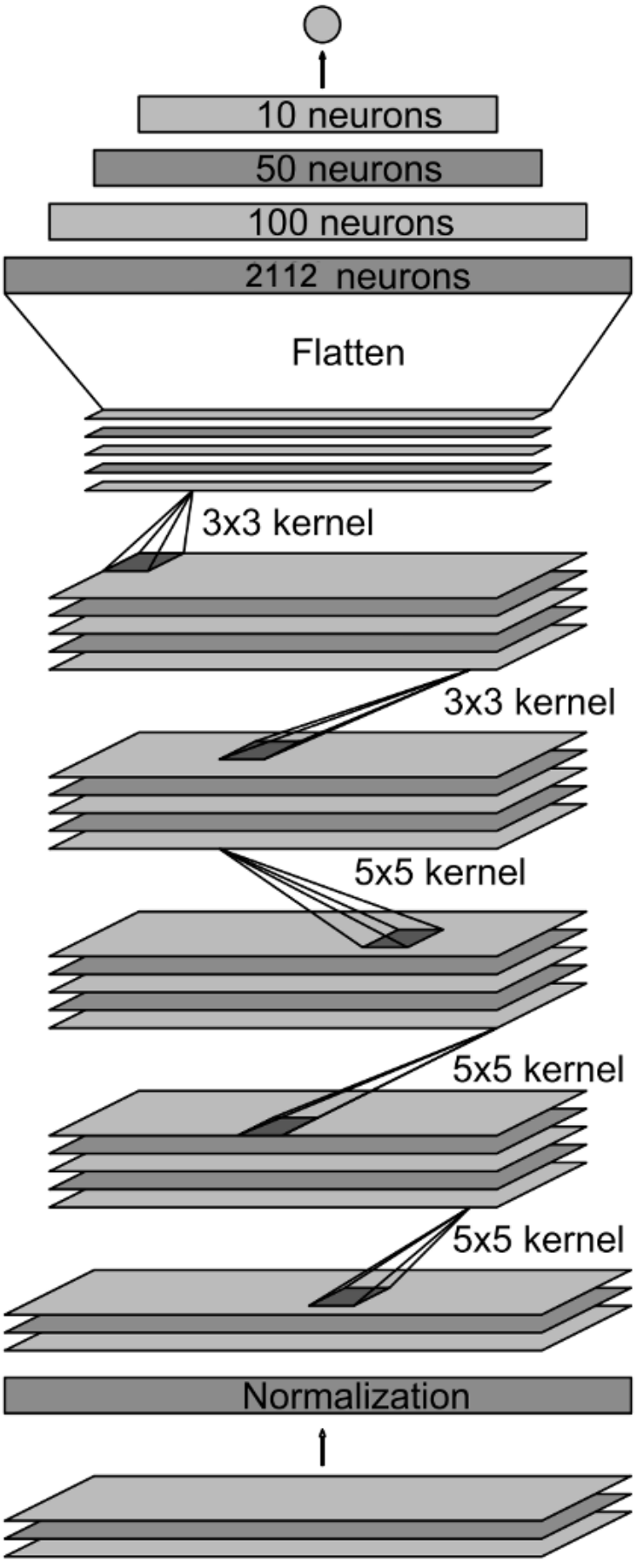
The improvement strategies I have tried include:

- Normalization for each pixel value: x = (x / 255.0) - 0.5
- Data augmentation by adding side camera images and image flipping
- Adopting a more complicated model architecture which referenced to the model in this paper.
- Cropping of training images to focus on the relevant contents (i.e., the track and its associated elements).

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

2. Final Model Architecture

The final model architecture (model.py lines 85-99) consisted of a convolution neural network with the layers and layer sizes as shown in the figure below:

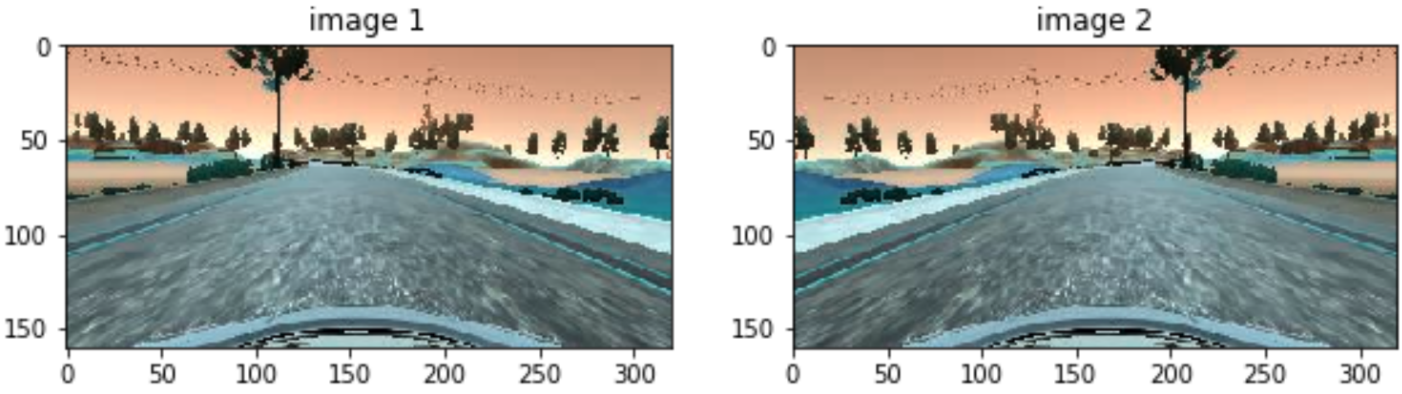

Output: vehicle control
10 neurons
Fully-connected layer
50 neurons
Fully-connected layer
100 neurons
Fully-connected layer
2112 neurons
Flatten
Convolutional feature map
64@1x33
3x3 kernel
Convolutional feature map
64@3x35
3x3 kernel
Convolutional feature map
48@5x37
5x5 kernel
Convolutional feature map
36@14x77
5x5 kernel
Convolutional feature map
24@31x158
5x5 kernel
Normalized input planes
3@65x320
Normalization
Input planes
3@65x320

3. Creation of the Training Set & Training Process

For this project, I have chosen to use the provided sample driving data as the base to train my model. (this is mainly due to the reason that when I run the simulator in the workspace, the animation was too lagging to collect a full set of training data. Please refer to the `Discussion` Section for more details).

To augment the data set, I have added camera images at left and right sides of the car with adjusted steering angles (plus or minus 0.2). This enables the model to learn how to steer if the car drifts off to the left or the right. This has enlarged the size of the training set by two times.

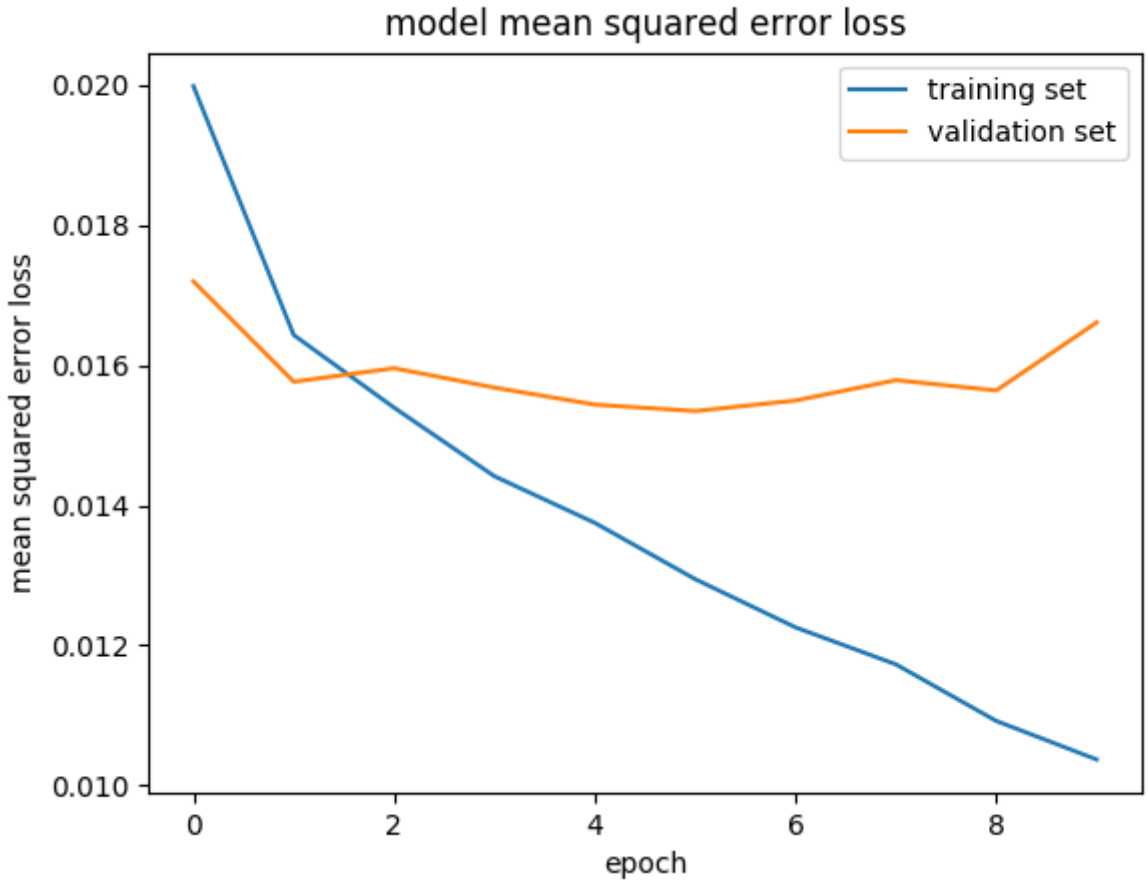
Other than that, I also flipped images and angles to double the size of the training set. For example, here shows how an image is flipped:



After the data augmentation process, I had 48,216 number of data points. I then preprocessed this data by normalization and cropping the image to show only the lower portion of it.

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 5 as evidenced by the figure shown below:



Simulation

Simulation results (video)

Using the Udacity provided simulator and the drive.py file, a video file `Video.mp4` in the project folder is created to demonstrate a successful run of one lap for the first track using the model trained aforementioned.

Discussion

For this project, I have chosen to use the provided sample driving data as the base to train my model. This is mainly due to the reason that when I run the simulator in the workspace, the animation was too lagging to collect a full set of training data.

However, additional improvement strategies could be implemented in future, including:

- To spend some time to practice running the simulator, at least to successfully record one to two laps of driving data.

- To record additional driving data for the vehicle recovering from the left side and right sides of the road back to center.
- To collect more data from track two.