CarND-Behavioral-Cloning-P3

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

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.

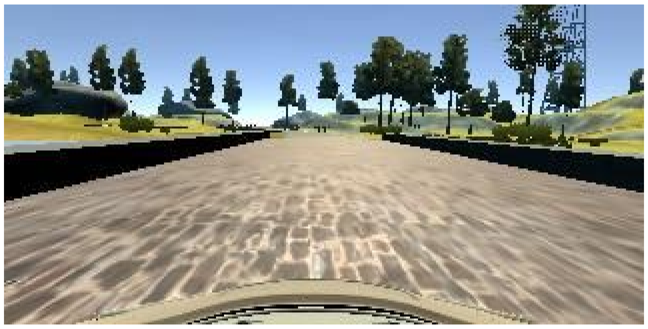
Preprocess and Data Augmentation

- Data package provided by Udacity have been used to train/validate this behavior cloning project.
- Both center image, left-side image, and right image have been used to train /validate model.
- A horizontal correction of 0.2 has been used for left and right side image.
- Flipping image has also been added to get even more data.
- Data have been normalized by lambda from Keras.
- Using cropping function to erase the useless information like the hood of vehicle and sky, tree. I believe this help the model to focus more on the road which will have a better effect on driving behavior cloning.

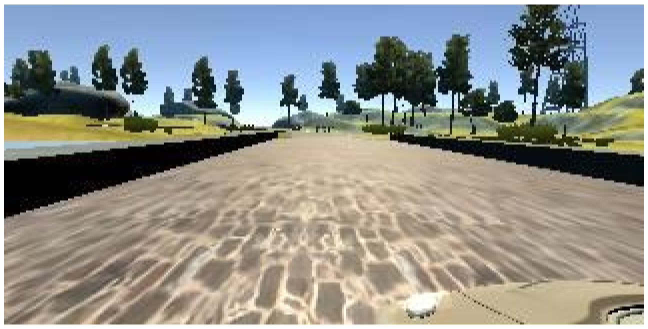
Training Data

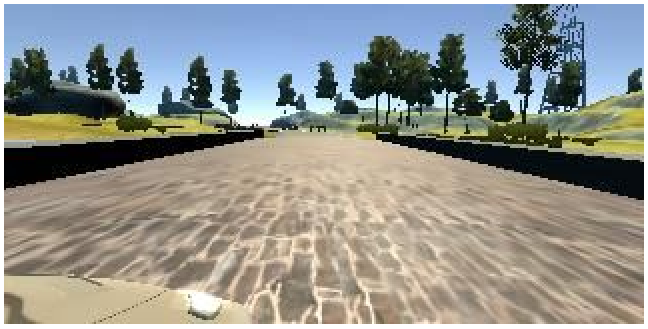
Based on installation position, left side camera's view and right side camera' view are different from center camera view. So a correction of 0.2 for steering angle has been used when adding left side camera image and right side camera image into the training dataset. Therefore more data is available for neural network training.

Following images are Center Camera View, Left-side Camera View and Right-side Camera View at the same time.(Center steering angle is 0).



Center Camera View(Steering angle is 0)


Left-side Camera View(steering angle is 0.2)


Right-side Camera View(steering angle is -0.2)

Model Architecture and Training Strategy

1.LeNet-5

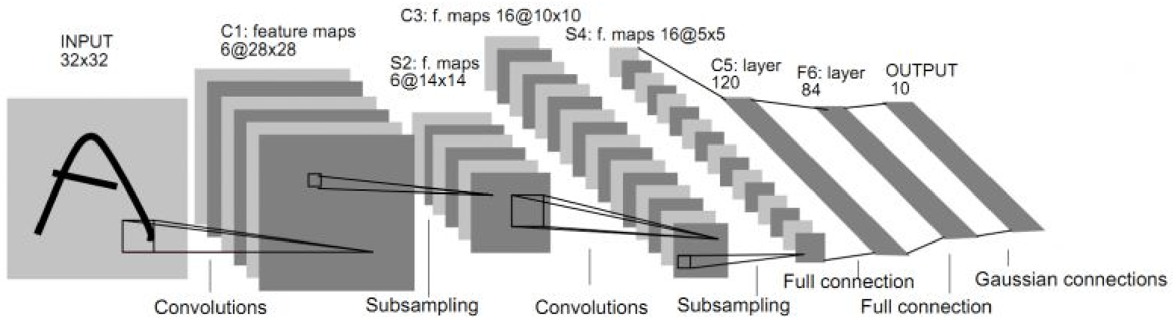
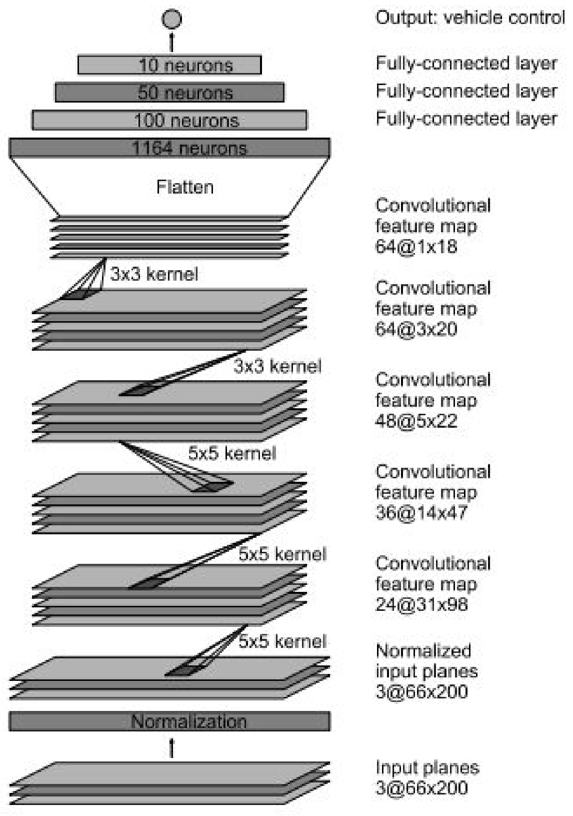


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Inspired by Udacity Self-Driving Behavioral Cloning project instruction, LeNet-5 solution model architecture from Traffic-Sign-Classifier project have been applied in the first place. During simulation test, the vehicle could not finish one lamp at first track. Need more advanced architecture.

2.Nvidia Autonomous Car



Based on the actual neural network model applied on NVIDIA autonomous car, a modified NVIDIA neural network model has been applied to this behavior cloning project.
Adding a dropout layer of 0.5 after five convolutional layers is the method of overfit combating.
The whole structure is:

- Input Image normalization
- Cropping image, 70 pixel rows at the top and 25 pixel rows at the bottom been erased.
- Convolution: 5x5, filter: 24, strides: 2x2, activation: RELU
- Convolution: 5x5, filter: 36, strides: 2x2, activation: RELU
- Convolution: 5x5, filter: 48, strides: 2x2, activation: RELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: RELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: RELU
- Dropout(0.5)
- Flatten()

- Fully connected: neurons: 100
- Fully connected: neurons: 50
- Fully connected: neurons: 10
- Fully connected: neurons: 1 (output)

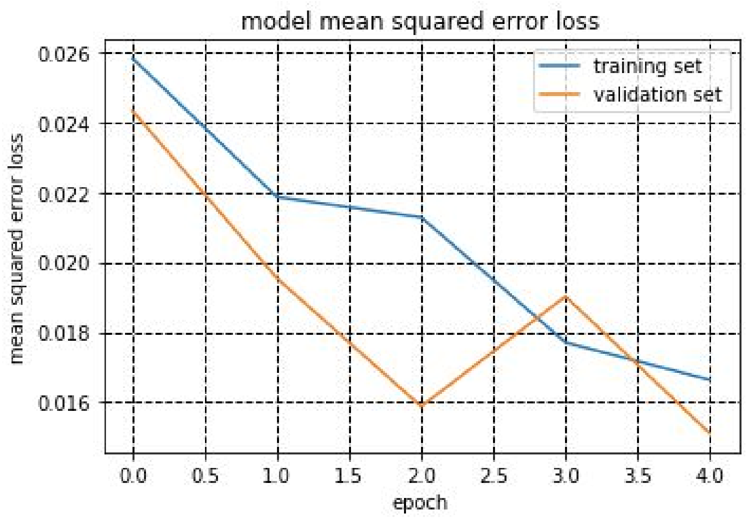
3. Training and Validation

Adam optimizer and mean-squared-error have been used for optimization and loss function.
20% of the whole data have been used for validation.

In Keras, the model.fit() and model.fit_generator() methods have a verbose parameter that tells Keras to output loss metrics as the model trains. The verbose parameter can optionally be set to verbose = 1 or verbose = 2.
Setting model.fit(verbose = 1) will

- output a progress bar in the terminal as the model trains.
- output the loss metric on the training set as the model trains.
- output the loss on the training and validation sets after each epoch.

After epoch = 5 times training, the mean squared error loss of training set and validation set are:



After this training, the vehicle could run autonomously on the first track.