Behavioral Cloning Project

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

Fig. 1 to Fig. 4 show five randomly selected images in each category with measurement angles as title of each image.

Fig. 1 Normal Images

Fig. 2 Recovery Images

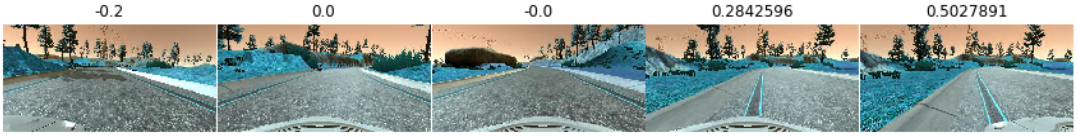

Fig. 3 Recovery Reverse Images

Fig. 4 Reverse Images

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.pdf summarizing the results (this report)
* test_model.ipynb is a notebook that I created to help with the project

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
```

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

I primarily followed the Nvidia End to End Learning for Self-Driving Cars https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf. Fig. 5 shows the CNN architecture.

The model includes activation='relu' to introduce nonlinearity for both convolution layers and fully connected layers, and the data is normalized in the model using a Keras lambda layer.

```
    # process images, normalization
    model.add(Lambda(lambda x: x/255.0 - 0.5, input_shape=(160,320,3) ))
    # process images, crop out top 50 pixels and bottom 20 pixels
    model.add(Cropping2D(cropping=((50,20),(0,0))))
```

Keras APIs might have some performance advantage over OpenCV APIs to leverage GPU. I used AWS GPU instance for development and testing.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting in get_model().
I used MaxPooling2D and Dropout with 0.4 parameter in the convolution layers.
```
    model.add(MaxPooling2D((2,2), border_mode='valid'))
    model.add(Dropout(0.4))
```
I also added another Dropyout in the fully connected layer.
```
    model.add(Dropout(0.3))
```

Before I added MaxPooling and Dropout, self-driving car was smooth from the CNN model but was over side lines couple of times although it was able to recover. After adding them, it's much improved as overfitting was improved.

The model was trained and validated on different data sets to ensure that the model was not 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.

4. Appropriate training data

Training data were 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, and reverse.

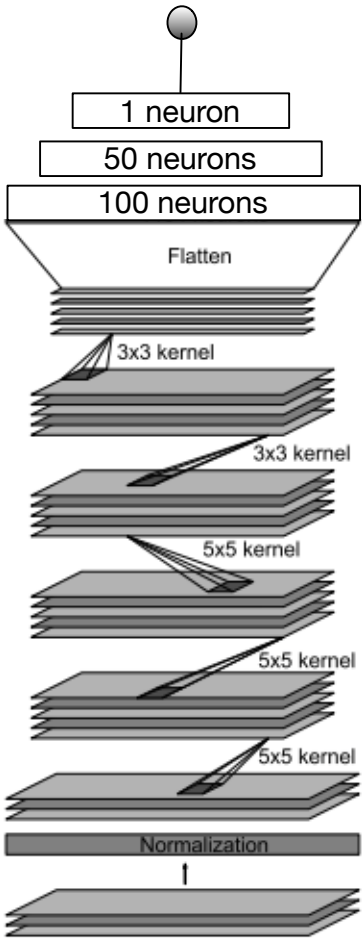

Fig. 5 CNN Architecture

5. Creation of the Training Set & Training Process

	Training Data
Normal	11968
Recovery	3156
Recovery Reverse	3192
Reverse	12336

Table 1 Number of training data from different ways of collection

I finally randomly shuffled the data set and put 20% of the data into a validation set by using model.fit validation_split option. Train on 24521 samples, validate on 6131 samples

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. With 10 epochs, the results are acceptable. I used an Adam optimizer so that manually training the learning rate wasn't necessary. Fig. 6 shows the history model mean square loss comparing between training set and validation sets.

As I used notebook in development, it was able to handle it well. For large dataset, we can use generator as lecture mentioned if necessary to overcome out of memory problem.

```
dict_keys(['val_acc', 'val_loss', 'loss', 'acc'])
```

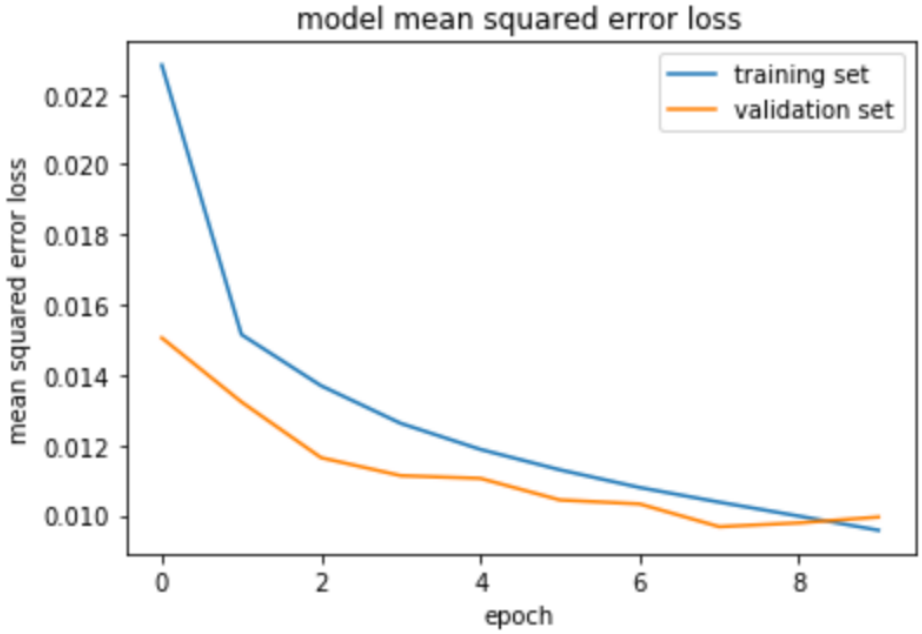

Fig. 6 History Model Mean Squared Error Loss