Files Submitted Explanation

My project inclused the following files:

(1) model.ipynb containing the script to create and train the model

(2) model.generator.ipynb containing the script to create and train the model by generator

(3) drive.py for driving the car in autonomous mode

(4) model.h5 containing a trained convolution neural network

(5) writeup_report.pdf summarizing the results

(6) run1.mp4 and run2.mp4 recording complete laps of the track1 and track2 respectively

Using the Udacity provided simulator and the drive,py file, the car can be driven autonomously around the track by executing:

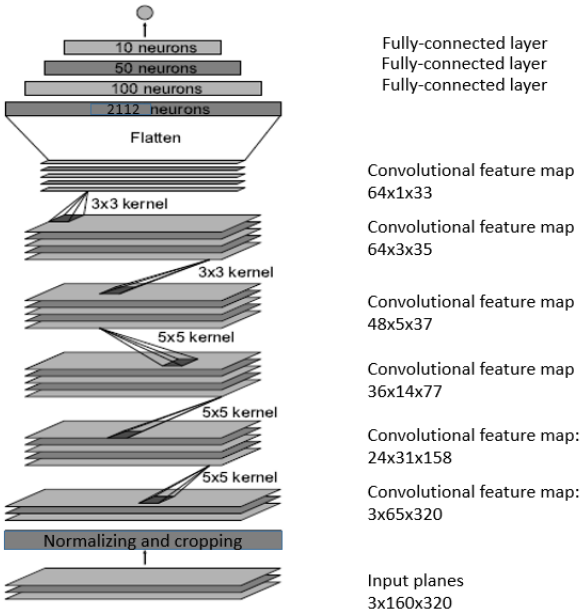
```
python drive.py model.h5
```

The model.ipynb or model.generator.ipynb 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 NVIDIA pipeline introduced in the course video was utilized to build the network architecture because it has been proved effective and not complex. There are totally 10 layers in the network, including 1 normalization layer, 1 cropping layer, 5 convolutional layers and 3 fully connected layers. The input to the network is RGB image arrays in size of 160x320x3 and it is then normalized after the first layer (code line 8 in the 3rd code cell). The following three layers are convolutional layers with 2x2 strides and 5x5 kernel (code line 9 to 14 in the 3rd code cell). Then two non-strided convolutional layers with 3x3 kernel were built (code line 17 to 19 in the 3rd code cell). The five convolutional layers were followed by three fully connected layers and the final output controls the steering angle. The shape of each layer is shown in the figure below.



2. Attempts to reduce overfitting in the model

To prevent overfitting, a dropout layer with dropout rate 0.5 was created after the final

convolutional layer. (code line 15 in the 3rd code cell)

The validation_split parameter in the model.fit() was set to be 0.2, which means 13176 (80% of 16470) samples were used to train the model and 3294 (the rest 20%) samples were used for validation. The training and validation loss during the three epochs are shown in the figure below. The loss of training data is lower than that of validation data, but the difference is not too much, which indicates a little overfitting. I have tried different learning rate and dropout rate, and 0.0003 learning rate and 0.5 dropout rate performed best. The loss of validation set decreased but it is hard for it to reach the training loss level. I tend to attribute this to the low quality of the discontinuous input command data from the keyboard when I was gathering the training data. Sometimes the steering angle controlled by me is not very reasonable (especially in sharp curves) and this can make it hard for the model to predict its control command signal value.

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. The result video with frame in view of the front camera is shown in the run1.mp4.

3. **Model parameters tuning**

An adam optimizer was used, and the learning rate was set to be 0.0003 because with higher or lower or default values, the validation loss would be fluctuating or even increasing as epochs goes.

Images from three cameras (positions as shown below) were applied to train the model. The corresponding steering angle correction of the left and right camera is set to be 0.2 as the codes below illustrates.

```
measurement = float(line[3])
if i==0:
    measurements.append(measurement)
elif i==1:
    measurements.append(measurement+correction)
elif i==2:
    measurements.append(measurement-correction)
```

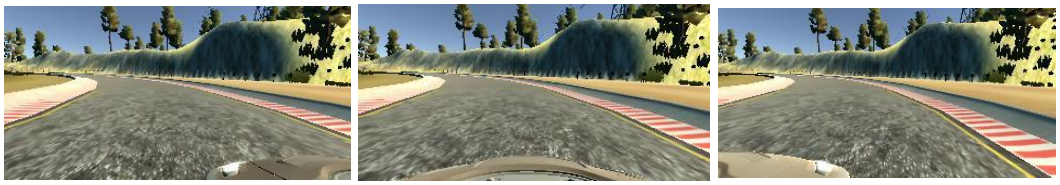
Also, the cropping areas of the input image is pixels 65 pixels from the top and 30 pixels from the bottom. No pixels were cropped in the horizontal direction.

3 epochs in the training process seems enough for training the model because more epochs will raise the loss up.

4. **Appropriate training data**

I configured all the required program including tensorflow-gpu, keras, socketio and eventlet etc in a local environment on my computer. Although the NVIDIA GeForce 940M is a small gpu, it trains the model a lot faster than single cpu does, which is amazing. Initially, to test if the simulator runs properly in autonomous mode, I just drove the car through the first two corners, and used the model trained by these small amount of data to drive the car. The simulator worked well.

I then started to gather training data for completing the track. Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving and recovering from the left and right sides of the road. I drove the car by keyboard and there are one complete clockwise lap, one anti-clockwise lap of the track and some recovering from the left and right sides driving actions in the sharp corners. The images in the three cameras were gathered simultaneously and examples are shown below.



I used generator to prevent occupying the memory space of the computer (in the model_generator.py code file). However, I found it relatively slower than the normal training way. Since I have enough memory in my computer (20G), I decided to still use the computer memory to store the image data for training, which allows me to train large amount of data faster (in the model.py code file).

The car driven by model.h5 kept in the track and finished one complete lap (in run1.mp4)

5. **Drive in the track 2**

As suggested in the course, I drove the car manually in the track 2 to gather more training data for training a more generalized model. The strategy for track 2 is also keeping the car the in middle of the lane. I finished one clockwise lap and one anti-clockwise lap, stored them in the previous directory, and then used the merged data set to train the model. The model can still drove the car to finish the track 1, but in track 2, it rush out in the first 180° sharp curve. I then repeated to drive the car through a few extremely sharp corners more slowly and smoothly, and reduced the cruise velocity from 25km/h to 15km/h. Finallym the new model trained by the upgraded data set drove the car to finishe the track 2 (run2.mp4)

and also the track 1.

The training the validation loss are shown below. It can be seen that the loss of the validation set is higher than that of training set, but I attribute this to the low level of input consistency caused by the keyboard input rather than serious overfitting.