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.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 5X5 and 3x3 filter sizes and depth between 64 and 3.
The model includes RELU layers to introduce nonlinearity (code line 55-61), and the data is normalized in the model using Keras lambda layer (code line 54)

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 19-25). 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 was 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. I also augmented the data by flipping the image and angle.

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

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to create a neural network that can be trained with large amount of images and corresponding steering angel.

My first step was to use a convolution neural network model published by the autonomous vehicle team at NVIDIA. This model should be appropriate because it contains 5 convolution layers and 3 fully-connected layers.

I collected 46,182 images from simulator software, which contain images from center, left and right camera. In order to augment the data. I flip the images and steering measurement to solve the left turn bias problems.

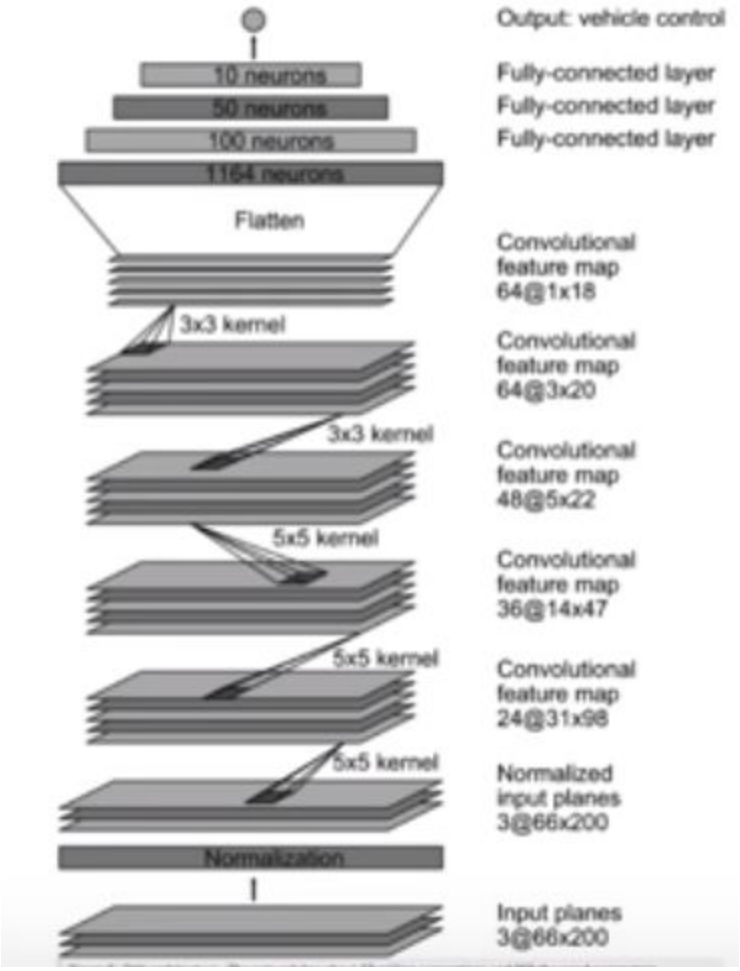
Also to improve the driving behavior, I particularly gather more images at big turn to better train the model when the vehicle is turning big.

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

2. Final Model Architecture

The final model architecture (model.py lines 52-64) consisted of a convolution neural network with the following layers and layer sizes ...

Here is a visualization of the architecture



3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to steer back to the center once it is deviating from the track.

The I repeated this process and I drive the vehicle for 5 laps.

To augmented the data set, I flipped the images and steering angles to mitigate the over steering problem.

When training the model, I also use the left camera image and the steering angel -0.2 as my corresponding steering angel. As for the right camera, the corresponding steering angel would be steering angel + 0.2.

After collection, I had 46182 of images and I then preprocessed this data by converting them into 160x320x3 numpy arrays. The arrays and corresponding steering angels are used as input for training the model.

I finally randomly shuffled the data set and put 20% of the data into a validation set.