Writeup Template

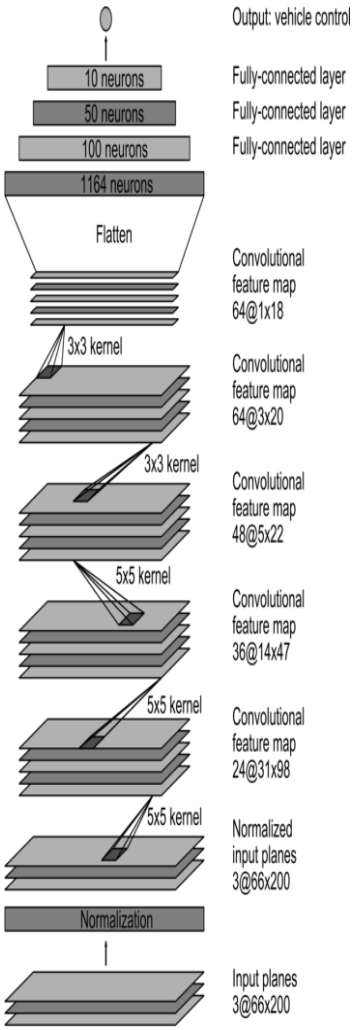
You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

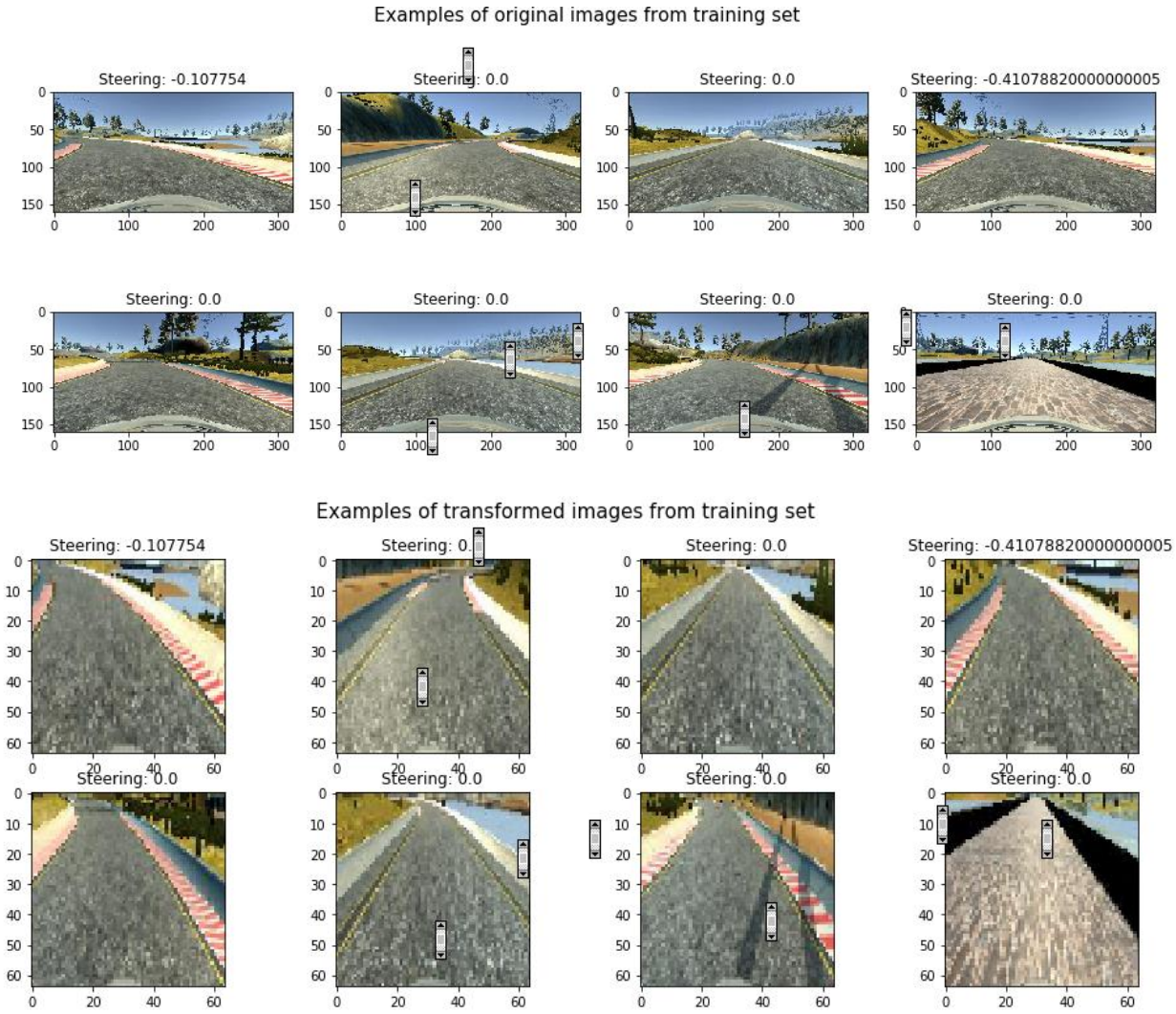
****Behavioral Cloning Project****

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

*****Model Visualisation****





Rubric Points

Here I will consider the [rubric points] (https://review.udacity.com/#!/rubrics/432/view) individually and describe how I addressed each point in my implementation. The link to the github repository is given below…
https://github.com/bnnair/behavioural_cloning_p4.git

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_Behavioural_Cloning.pdf - summarizes the results
* runl.mp4 – this is the video of the autonomous driving car
* video.py – this is used to create the mp4 video.

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 has been employed in the project.

My model consists of a convolution neural network with 3x3 filter sizes and depths between 24 and 64.

The model includes ELU layers to introduce nonlinearity , and the data is normalized in the model using a Keras lambda layer.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting.

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 is an Adam optimizer and the learning rate was tuned.
Also the batchsize and epoch were adjusted not to overfit the model.

4. Appropriate training data
```

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, reverse lane driving, recovering from the left and right sides of the road.

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 model architecture was to go through some of the famous models and check whether any of the model would work.

My first step was to use a convolution neural network model similar to the LeNet model. I thought this model might be appropriate because it worked well in previous projects utilizing CNN for e.g. handwritten digits recognition. After testing these models, adding/removing some layers I observed that the Nvidia model works the best in general.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

To combat the overfitting, I modified the model so that after each convolutional layer there is a dropout layer. Also, to increase nonlinearity, "ELU" activation layers were placed after each layer of the model. I tried also to reduce the model complexity but it didn't help much. I would say that it was important to have not so much collected data (about 6(3 in both directions) laps in total were just enough).

The model used an Adam optimizer, so the learning rate was automated during the training process. I tuned a bit the batch size which finally equaled 32. Also, I was experimenting with the number of epochs between 10 and 50. Finally, it was sufficient to run the training only on 10 epochs.

The final step was to run the simulator to see how well the car was driving around track one. However, it seemed that the vehicle was able to drive

autonomously around the track without leaving the road at the first instance itself.

2. Final Model Architecture

The final model architecture (model.py lines 18-24) consisted of a convolution neural network with the following layers and layer sizes .

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)

Image normalization - using Keras lambda layer
Convolution: 5x5, no. filters: 24, strides: 2x2, activation: ELU
Dropout (0.4)
Convolution: 5x5, no. filters: 36, strides: 2x2, activation: ELU
Dropout (0.4)
Convolution: 5x5, no. filters: 48, strides: 2x2, activation: ELU
Dropout (0.4)
Convolution: 3x3, no. filters: 64, strides: 1x1, activation: ELU
Dropout (0.4)
Convolution: 3x3, no. filters: 64, strides: 1x1, activation: ELU
Dropout (0.4)
Flattening
Fully connected: neurons: 100, activation: ELU
Fully connected: neurons: 50, activation: ELU
Fully connected: neurons: 10, activation: ELU
Fully connected: neurons: 1 (output)

3. Creation of the Training Set & Training Process

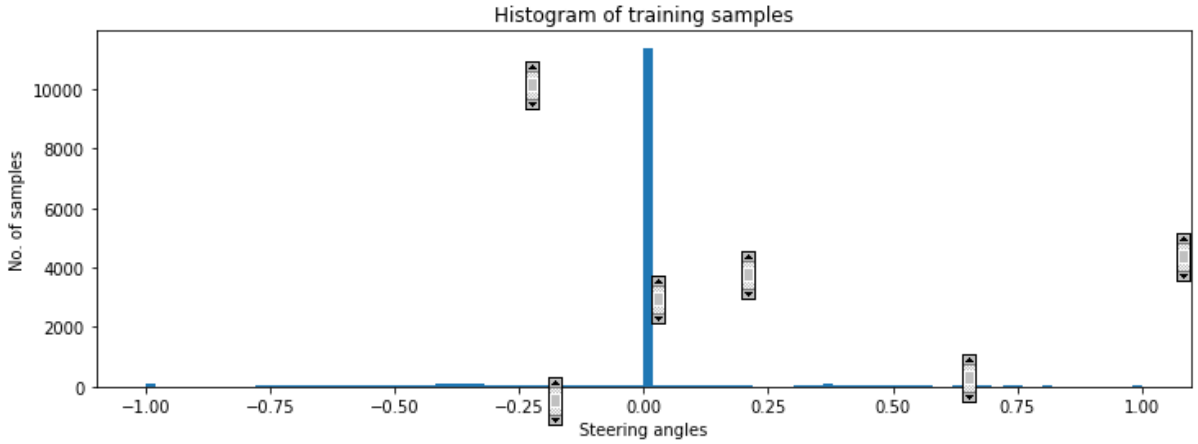
To capture good driving behavior, I first recorded three laps on track one using center lane driving on one side. Then I reversed the car and used three laps for the other side driving.

While driving the car in both directions, I had already taken care of the recovery of the car from the left and right sides of the road back to the center so that the vehicle would learn to recover whenever it went through the sides of the road.

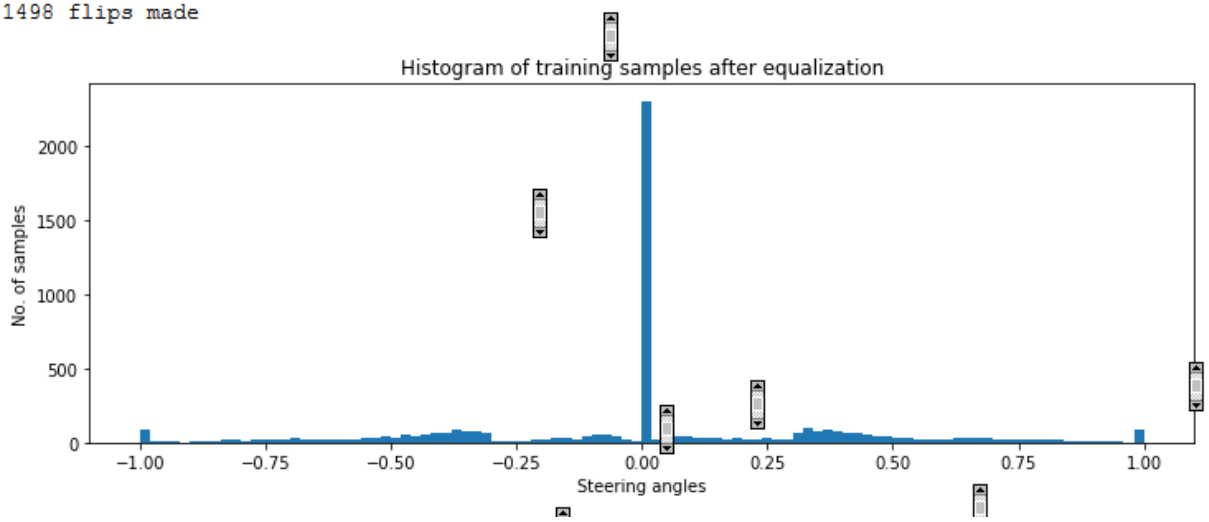
To augment the data sat, I cropped the image so that the model can only see what is relevant and train accordingly.

After drawing a histogram of steering angles for all collected samples, one conclusion comes straightaway. Although we did a "recovery driving" there is still a huge number of samples with angles close to 0. This could bias the model towards predicting 0 angle. To improve this situation I rejected about 80% of samples whose steering angle is really, really close to 0. Below, there are histograms of training samples before and after this operation:

```
Number of training samples: 18107
(14485, 7)
(14485, 7)
0 flips made
```




```
(5320, 7)
(5320, 7)
1498 flips made
```



We can see that there are still many 0 angles among the samples but it's significantly lower value than before.

I also did some random flipping of the images so as to reduce the bias on the left side of the road. It means flipping an image horizontally and changing the sign of corresponding steering angle.

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 10. I used an Adam optimizer so that manually training the learning rate wasn't necessary.

I also added some pre-processing code for the input images in the drive.py from the simulator.