```markdown
#**Behavioral Cloning**

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


[//]: # (Image References)
[image0]: ./models/model_1/model_1_summary.PNG "Model Summary"
[image1]: ./models/model_1/tb_losses.png "Final loss"
[image2]: ./models/model_1/equalized_shadowed.PNG "Equalized with shadows"
[image3]: ./models/model_1/high_validation_loss.png "High validation loss"
[image4]: ./models/model_1/cropping.PNG "Image Cropping"
[image5]: ./models/model_1/flipped.PNG "Flip Image"
[image6]: ./models/model_1/shifted.PNG "Shift Image"
[image7]: ./models/model_1/hueadjusted.PNG "Hue adjusted"
[image8]: ./models/model_1/shadowed.PNG "Shadowed image"
[image9]: ./models/model_1/model_1_hist.jpg "Steering Histogram"


## 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.


---
###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:
* main.py the script to train the model
```

* model.py containing the script to create the model
* my_model.py holding various models tested
* drive.py for driving the car in autonomous mode
* model_1.h5 containing a trained convolution neural network
* model_1_weights.h5 the weights from the trained cnn
* model_1.mp4 record of driving track1 2 rounds in autonomous mode
* prepare.py a class for reading and preprocessing images
* visual.py a helper class for visualizing results
* conv_visualization a class for generating activation images for layers
* video.py for generating videos from recorded autonomous driving images
* evaluate.py for running the conv_layer visualizations
* generator.py the generator class for training and validation set
* writeup_report.md this file - summarizing the results

Some files are only for experimental uses like conv_visualization.py/visual.py

####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 models\model_1\model_1.h5 30
```

The quality chosen was fantastic in window mode on 1280x960 display size. Steering speed desired was set to 30.

####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 is based on the nvidia model and consists of a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 and 128 (my_model.py lines 59-84)

The model includes ELU activation layers on each layer to speed up learning based on following a discussion on [ML Reddit](https://www.reddit.com/r/MachineLearning/comments/3u6ppw/exponential_linear_units_yielded_the_best/?st=izx2u5u9&sh=010a4b84)
and [paper](https://arxiv.org/abs/1511.07289)

The data is normalized in the model using a Keras lambda layer (my_model.py code line 63).
Instead of dropout layers i used batch normalization on channels axis for the first two convolutional layers to make the network more robust to bad initialization.

####2. Attempts to reduce overfitting in the model

The model contains 2 batch normalization layers (my_model.py lines 64,66) which should overfitting less likely.

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py line 102) by splitting
from the test set wit a factor of 0.2.
The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track for multiple laps.

####3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (my_model.py line 82).

####4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving and additional weak spot image recording.
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 implement a well known model like nvidia and finetune where it's necessary.
As dataset at first only the provided udacity dataset was used.

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 chose to validate on only the center images where i had exact labels and not on the steering corrected additional left and right images.

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.

![High validation loss][image3]

To combat the overfitting, I modified the model and introduced batch normalization layers

The final step was to run the simulator to see how well the car was driving around track one. As some spots were difficult and the car left the road i
created additonal datasets made on my own.

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

####2. Final Model Architecture

The final model architecture (my_model.py lines 59-84) consisted of a convolution neural network with the following layers and layer sizes:

![Model summary][image0]

![Final loss][image1]

####3. Creation of the Training Set & Training Process

I used the provided udacity dataset. After identifying a few weak spots where the car left the track, i decided to record additonal recovery images.
Doing so with the keyboard or gamecontroller failed badly as the steering angle was too fixed and could not be so well controlled, which was
in the end the most important point of the whole project. Only using the mouse steering control on the beta simulator let me
drive the track in a way that i good continous steering result on lots of small degree steps

Creating recovery images from off track on the road again was in my set not necessary for track one. Instead i cherry picked good steering angles for the weak spots
and added them to the training data.

To prevent biasing of bad angles like zero, ones or lots of the same steering angles sequentially i added a queue of
5 which discards the same values after the fifth same appearance. Also I set the threshold of zeroes to a maximum of the
second highest other steering angle which lead to the following final steering histogram:

![Steering histogram][image9]

To augment the data set, i randomly flipped the images and measurements as well as shifted them vertically or taking left or right camera image instead
of center images. when taking left or right camera images, the steering angle has been corrected by a value of 0.25 for right images and -0.25 for left images.

Example of cropping the image from 160x320 to 66x200:

![Cropping][image4]

Example of flipping the image vertically:

![Flipping][image5]

Example of shifting the image:

![Shifting][image6]

Example of adjusting the hue of the image:

![Hue adjust][image7]

Example of adding a shadow section to the image:

![Shadowing][image8]

The image input itself has been 0/1 normalized on the keras lambda layer (my_model.py line 63).
Additionally i added hue randomness and randomly generated shadow sections to overcome the hurdles in the challenge videos. at the end this did not succeed,
so i only was able to make track 1 working.

A test and validation set generator (generator.py) has been created to batch work on the fly all data. Since using the nvidia model quickly ran out of memory
even on my gtx 1080. Also i resized the images to the nvidia chosen size of 66x200, as using 100x320 made out about 12 million parameters, where the resized one
needed only 1.6 million which lowered the size of the model from about 140MB to 14MB.

I finally randomly shuffled the data set and put 0.2 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 3 as evidenced by an introduced early stopping layer with patience 1.

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

additionally i added also a tensorboard layer (model.py line 111) to experiment with it's logging capabilities and generated a loss diagram out of it.
The model is unfortunately so big that the tensorboard

dashboard was not able to handle it's size.

To get always the best checkpoint i added a checkpoint callback which always compared the current with the last run and saved the best out of it, as well as a checkpoint file in addition to the best fit.