### 1-Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- model.ipynb 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

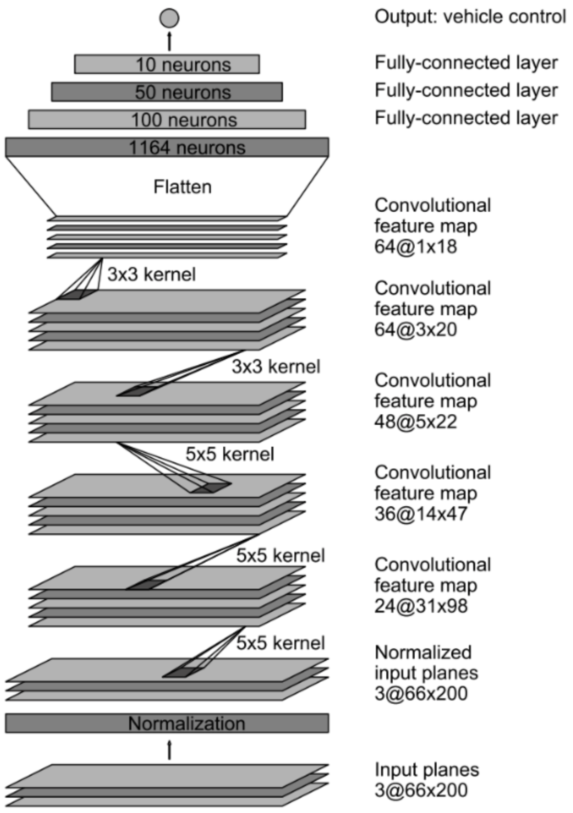
*python drive.py model.h5*

### 3-Submission code is usable and readable

The model.ipynb file contains the code for data gathering, preprocessing and saving convolution neural network with comments in cells.

## Model Architecture

As suggested in project instructions started with Nvidia model and as I see in slack or forums it works well. You can see below the model diagram.



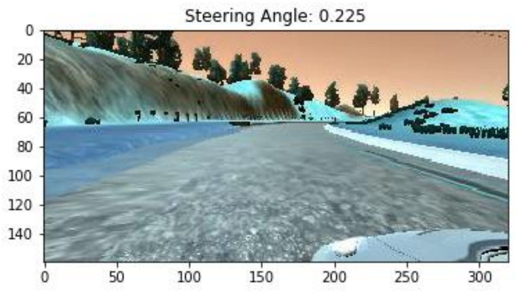
The model consists of 5 convolutional layers. After image normalization started with kernel size of 5x5 until 3. Layer by using filter 24,36,48 respectively. Then continue with 2 Layer kernel size of 3x3. Both 3x3 Layer uses 64 filters. Although Paper doesn't mention any activation function or regularization method, I started using "ELu" activation function and dropout with keep probability 0.5 in three fully connected layer to overcome overfitting. Adam optimizer was chosen, therefore learning rate is in this project not a hyperparameter.

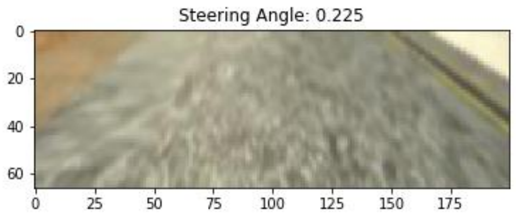
## Data Collecting

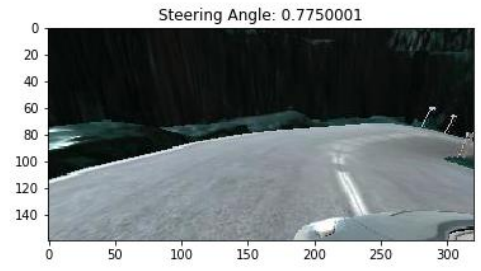
Although Udacity provides a dataset, I collected my own data. It was funny producing my own dataset. Track 1 was driven 6 times clockwise and counter-clockwise to generalize the model as suggested in data collection strategies. Track2 was driven 4 times clockwise. The recovery data collected just for track1. I am not good at using mouse by playing such a computer game, that's why samples collected with keyboard controls.

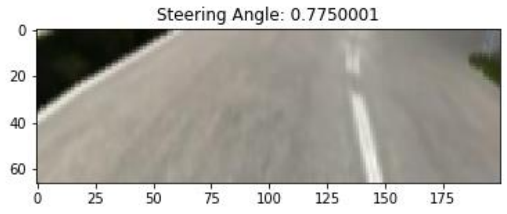
## Data Preprocess

I used left and right camera images besides center images as training data with correction factor for track1 and track2 -0.225, 0.225. All images are cropped in y axes and the region between [90:135] is used in training. The removed part of images contains blue sky, trees which objects are unnecessary by learning process and for staying in track. (200,66) resized images like in Nvdia Paper were used to train the model. Both cropping and resizing are necessary to reduce compute time. Instead of using YUV color space as suggested in paper I decided to use RGB color space. (cell 6, 1-14). Below you can see the original image(left) and the preprocessed image(right) for both track.



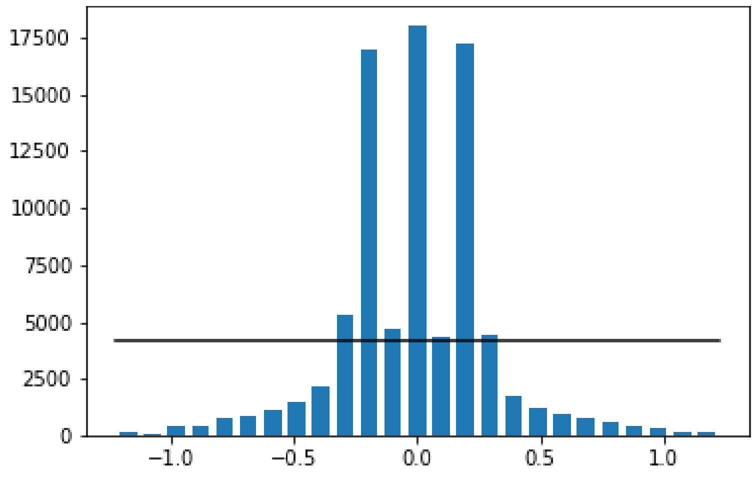


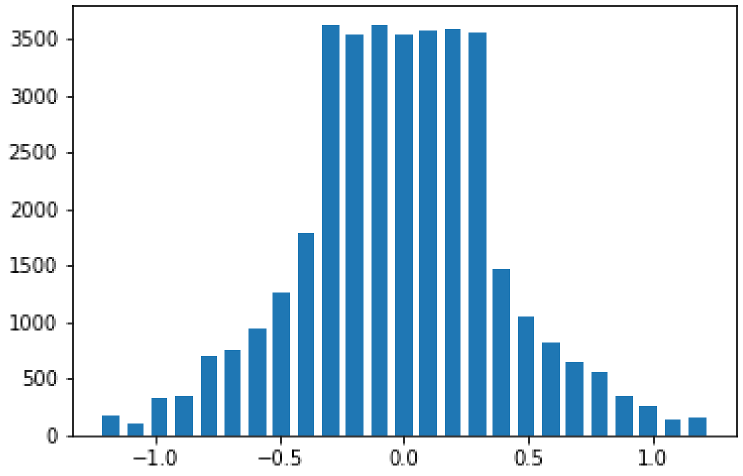




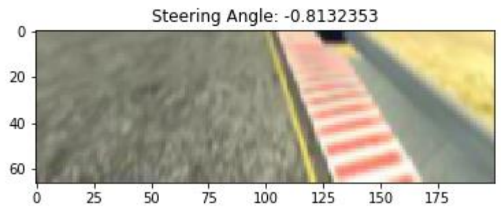
## Data Augmentation and preparing the dataset to train

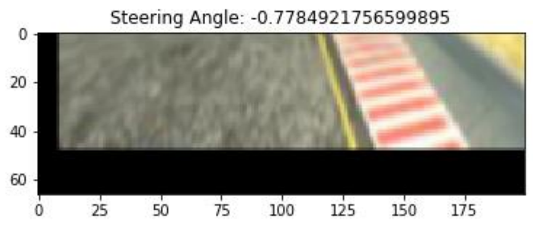
As you can see below in first histogram the steering angles was splited in 25 bins out (cell 9) and there are biased bins. I would remove images from them randomly to get more even distribution of steering angles (cell 10). As a target value was chosen 4200 images. Why 4200? Decision was made by looking quantities in other bins and tried to have two times bigger data in most used bins that is necessary especially in track 1. And of course, this can be introduced as a hyperparameter in my project. The data set was splited again to 85% train, 15% validation data set (see 2. histogram). At the end I have 36888 images to train and 6510 images to validate.





Model was feed with batch size 120 by using generator (cell 8). In generator all images are shifted horizontally and vertically except images in validation data set and then batch sample goes in to the Model. Below you can see example shifted images for both track, left is preprocessed image and right is shifted.

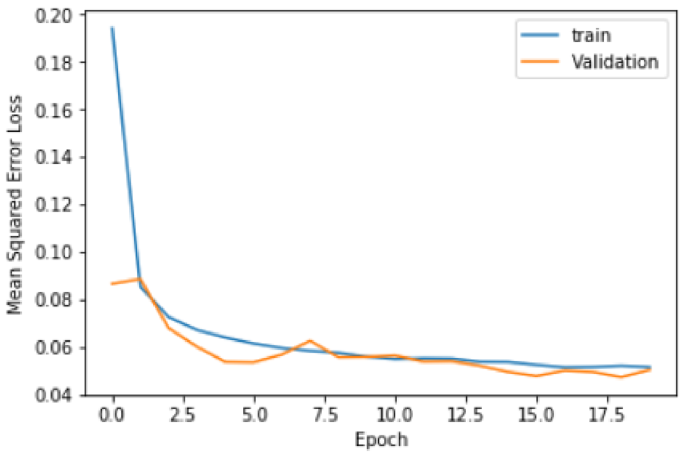




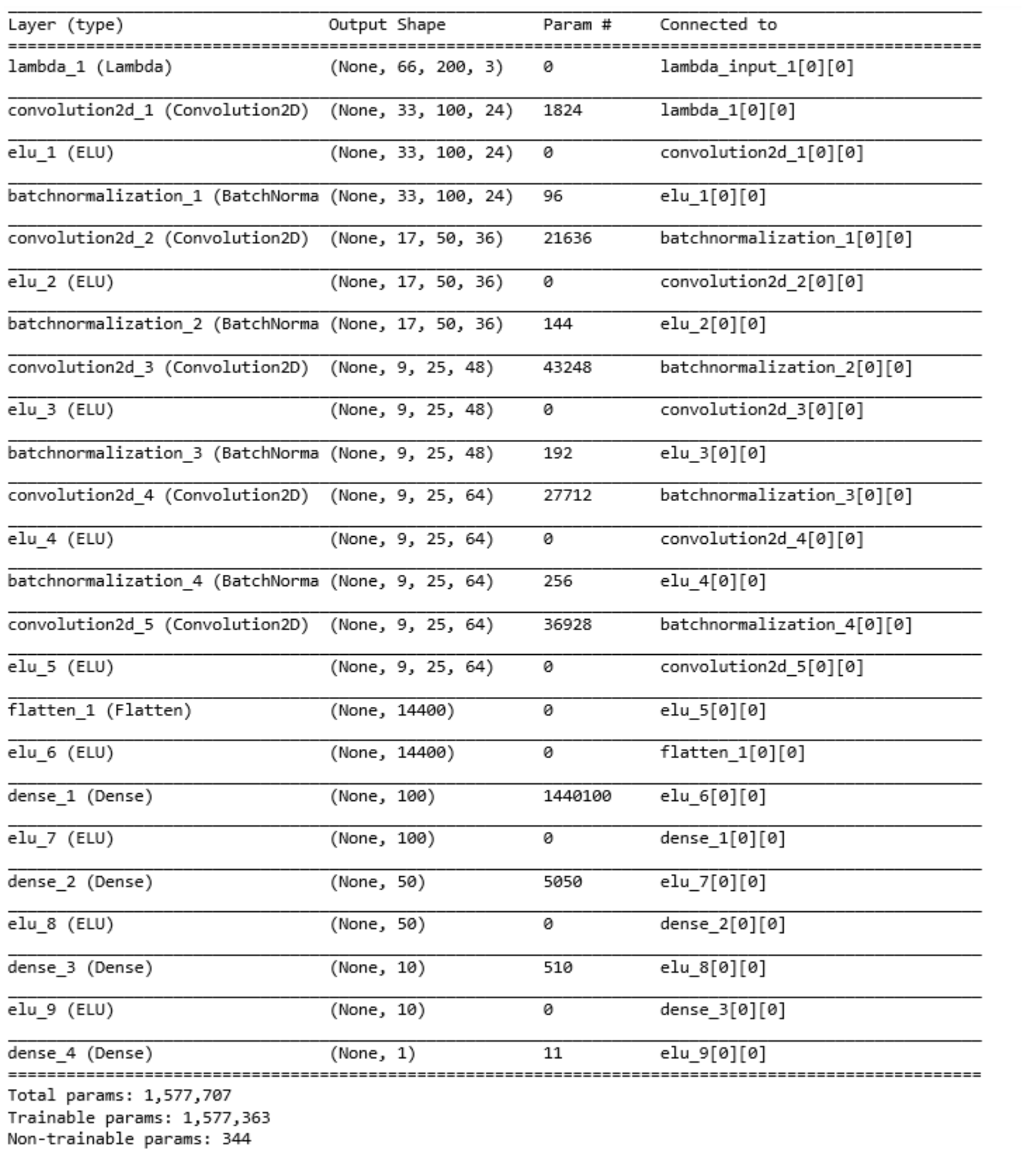
## Final Model

My final Model is a little bit different from starting point. With dropout, model was most of the time underfit, that's why I searched another solution to apply different method, even something different than L2 Regularization, which I used in Project 2. I found Batch normalization, which was explained perfectly in this paper Batch Normalization: Accelerating Deep Network Training by reducing Internal Covariate Shift. Some says batch normalization less about regularization and more about conditioning of the input to each layer. I think it is easy to apply and the best part of it is not hyperparameter!.

Because I have simple data Augmentation and the model trains with them actually I was not so much concerned about overfitting. Epoch number was chosen 20, but thanks to keras I can just save the best model while training. Below you can see loss-epoch graphic and in next page the final model summary.



In Track 1 you can realize that my car moves zigzag especially after the starting point until the first curve, but then smoother. The first reason for that, I think, I used keyboard to data collecting and yes it was hard to stay on center particularly on slight-long curves. This happened when I cropped images harder than before. On the other hand, this brings me smoother driving behavior on the challenge track, which satisfies me more. I hope you too=)



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_1 (Lambda)                (None, 66, 200, 3)    0           lambda_input_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 33, 100, 24)   1824        lambda_1[0][0]
____________________________________________________________________________________________________
elu_1 (ELU)                      (None, 33, 100, 24)   0           convolution2d_1[0][0]
____________________________________________________________________________________________________
batchnormalization_1 (BatchNorma (None, 33, 100, 24)   96          elu_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 17, 50, 36)    21636       batchnormalization_1[0][0]
____________________________________________________________________________________________________
elu_2 (ELU)                      (None, 17, 50, 36)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
batchnormalization_2 (BatchNorma (None, 17, 50, 36)    144         elu_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 9, 25, 48)     43248       batchnormalization_2[0][0]
____________________________________________________________________________________________________
elu_3 (ELU)                      (None, 9, 25, 48)     0           convolution2d_3[0][0]
____________________________________________________________________________________________________
batchnormalization_3 (BatchNorma (None, 9, 25, 48)     192         elu_3[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 9, 25, 64)     27712       batchnormalization_3[0][0]
____________________________________________________________________________________________________
elu_4 (ELU)                      (None, 9, 25, 64)     0           convolution2d_4[0][0]
____________________________________________________________________________________________________
batchnormalization_4 (BatchNorma (None, 9, 25, 64)     256         elu_4[0][0]
____________________________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 9, 25, 64)     36928       batchnormalization_4[0][0]
____________________________________________________________________________________________________
elu_5 (ELU)                      (None, 9, 25, 64)     0           convolution2d_5[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 14400)         0           elu_5[0][0]
____________________________________________________________________________________________________
elu_6 (ELU)                      (None, 14400)         0           flatten_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 100)           1440100     elu_6[0][0]
____________________________________________________________________________________________________
elu_7 (ELU)                      (None, 100)           0           dense_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        elu_7[0][0]
____________________________________________________________________________________________________
elu_8 (ELU)                      (None, 50)            0           dense_2[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 10)            510         elu_8[0][0]
____________________________________________________________________________________________________
elu_9 (ELU)                      (None, 10)            0           dense_3[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 1)             11          elu_9[0][0]
====================================================================================================
Total params: 1,577,707
Trainable params: 1,577,363
Non-trainable params: 344
```