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:

- clone.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
- this file 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 clone.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. The clone.py takes in up to four arguments:

- '-td' – path to training data relative to clone.py
- '-e' – number of epochs to train the model
- '-w' – trained weights file (*.h5) to initialize parameters
- '-f' – filename to save weights and models to

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model is based off of the NVIDIA architecture which consists of 9 layers, including a normalization layer, 5 convolutional layers, and 4 fully connected layers.
(clone.py lines 137-155) The model includes RELU layers to introduce nonlinearity (code line 20), a cropping layer to reduce the amount of data we need to process and is then normalized in the model using a Keras lambda layer (code line 141). The chart for the model can be seen below:

Layer (type)	Output Shape	Param #	Connected to
cropping2d_1 (Cropping2D)	(None, 80, 310, 3)	0	cropping2d_input_1[0][0]
lambda_1 (Lambda)	(None, 80, 310, 3)	0	cropping2d_1[0][0]
convolution2d_1 (Convolution2D)	(None, 38, 153, 24)	1824	lambda_1[0][0]
dropout_1 (Dropout)	(None, 38, 153, 24)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 17, 75, 36)	21636	dropout_1[0][0]
convolution2d_3 (Convolution2D)	(None, 7, 36, 48)	43248	convolution2d_2[0][0]
convolution2d_4 (Convolution2D)	(None, 5, 34, 64)	27712	convolution2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 3, 32, 64)	36928	convolution2d_4[0][0]
flatten_1 (Flatten)	(None, 6144)	0	convolution2d_5[0][0]
dense_1 (Dense)	(None, 100)	614500	flatten_1[0][0]
dropout_2 (Dropout)	(None, 100)	0	dense_1[0][0]
dense_2 (Dense)	(None, 50)	5050	dropout_2[0][0]
dense_3 (Dense)	(None, 10)	510	dense_2[0][0]
dense_4 (Dense)	(None, 1)	11	dense_3[0][0]

Total params: 751,419
Trainable params: 751,419
Non-trainable params: 0

Figure 1: Model structure

2. Attempts to reduce overfitting in the model

- The model contains dropout layers in order to reduce overfitting (clone.py lines 144, 151).
- The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 27-46). I also augmented the data by flipping every image and used both left and right cameras and shuffling the data sets in each batch. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. (code line 85-117)

3. Model parameter tuning

- The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 25).

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, driving backwards on the track, and flipping the images to better generalize.
- For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

My first step was to use a model with only a basic fully connected layer. This did not provide a very good structure even after adding more data and using both cameras. I then continued to use the convolution neural network model from NVIDIA as suggested and was able to get much smoother results. Since I had GPU's available, I decided this would be the best method and that It was time to fine tune from there. To make the training more efficient, I cropped the images by removing components that added no value to the training. I then normalized and zero centered the data during preprocessing.

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 had a hard time initially because my training and validation error were both low, and with my training/validation data my car would make it to different portions of the track. I noticed that it had a left turn bias so I added more data to reduce this. Looking at the histogram of steering angles, I decided to remove 70% of the data near the center (absolute angle less than 0.0005) so the car would be less biased to go straight and speed up the training.

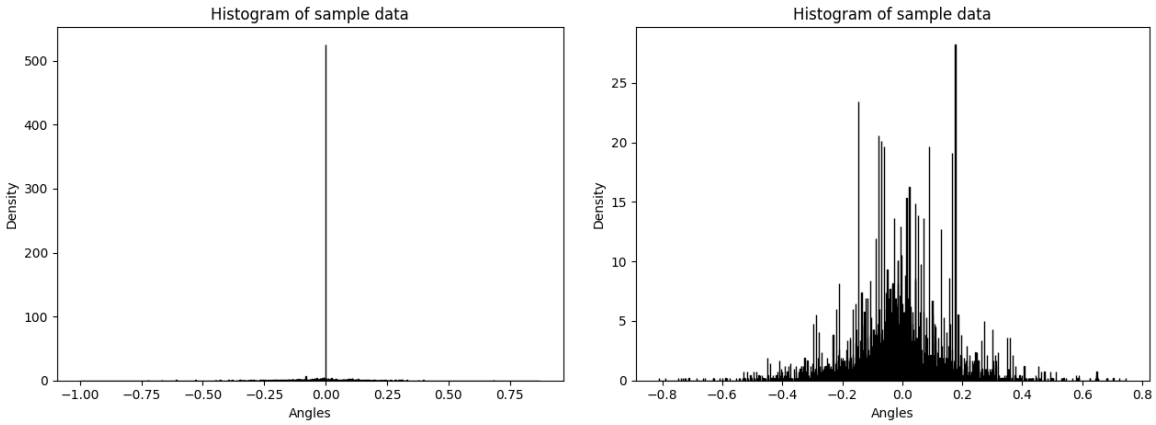


Figure 2: Before and after removing center biased data

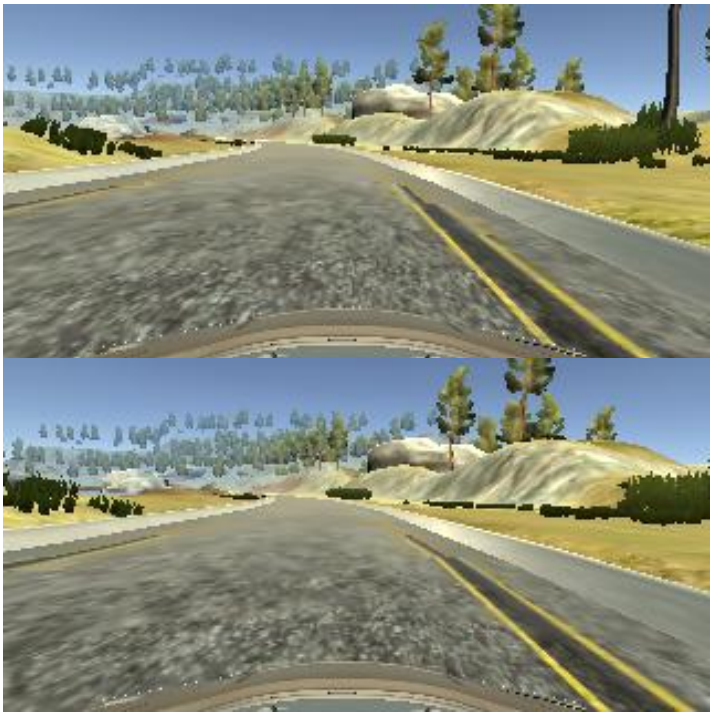
To combat the overfitting, I modified the model by adding a dropout layer after the first convolution and fully connected layer. This prevented the model from overfitting in these areas and provided training and validation with low root mean square errors.

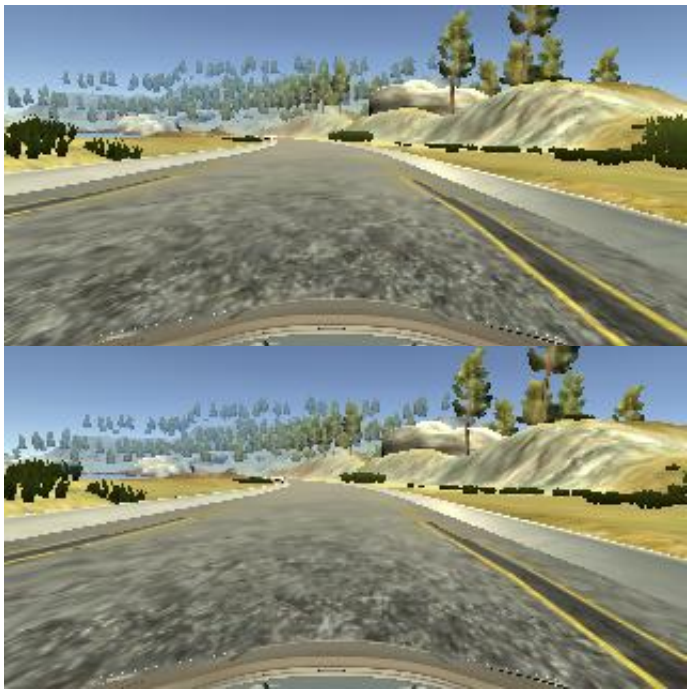
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road which can be seen in the video: https://github.com/suprnrdy/CarND-Behavioral-Cloning-P3/run1.mp4

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 make small adjustments and recover to center. These images show what a recovery looks like starting from the right side of the track and recovering to the center:





To augment the data set, I also flipped images and angles thinking that this would help generalize the training. For example, here is an image that has then been flipped:

After the collection process, I had 88,000 of data points. I then preprocessed this data by removing 70% of the images that were near center and then normalizing/zero center and cropping the images. This helped by reducing the amount of pixels the model would need to process and help reduce the possible error values.

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 4 as evidenced by where the root mean square error graph was settling to. I used an adam optimizer so that manually training the learning rate wasn't necessary.

Project Notes:

One of the big problems I had was that my model started to drive only straight. After many iterations of data collection ad running through the simulator, I determined that my

calculations of the left and right camera steering adjustments were wrong by adding adjustments to both left and right, creating right turn biased data. Once I fixed this my network was training as I expected it to.