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

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 layers of convolution along with max pooling and relu to account for complex curves. The layer is then followed by fully connected layers separated by relu activation functions and dropout to account for overfitting. The image is also cropped to only keep useful objects in the field of view with the lambda layer.

The initial 2 convolution layers used a 5x5 filter size while the rest used a 3x3 filter size.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce 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 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.

Model Architecture and Training Strategy

The overall strategy was for a model archetecture used in the lecture with convolution layers and dense layers, this model was driving the car but it wasn't driving it well. The car was unable to handle the bridge and dirt ground well.

To create a more efficient model, I added relu activation layers along with dropout layers and max pooling layers. At the end of the process the vehicle was able to autonomously drive around the track as seen in the video.mp4 file.

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:

Model = sequential

Lambda layer with input shape of (120,320,3)

Cropped 2D (70,25) from top and bottom, nothing removed from the sides

Convolution 2D with (24,5,5)

MaxPooling 2D with size (2,2)

Activation = relu

Convolution 2D with (36,5,5)

MaxPooling 2D with size (2,2)

Activation = relu

Convolution 2D with (48,3,3)

MaxPooling 2D with size (2,2)

Activation = relu

Convolution 2D with (64,3,3)

Flatten

Dropout(0.1)

Activation = relu

Dense = 1164

Dropout(0.1)

Activation = relu

Dense = 100

Dropout(0.1)

Activation = relu

Dense = 50

Dropout(0.1)

Activation = relu

Dense = 10

Dropout(0.1)

Activation = relu

Dense = 1

Creation of the Training Set & Training Process

To capture more data, I went around the track 2 to 3 times followed by going around the track in the reverse direction 1 time. This gave me more data but factors such as lag and frame drops caused me to use the sample data.

To augment the data set, I also flipped images and angles thinking that this would give it more data by effectively giving a new track with a more right turn focus. This would help ensure that the car can turn right and left while also giving a larger data set. The brightness was also adjusted for any light variations and kept the image in a more focused darker format.

Preprocessing the data involved standardising it with the formula x/127.5 - 1.0 , this ensures that the data is treated under the same criteria.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 5 as evidenced by seeing no major increase or decrease in the loss. I used an adam optimizer so that manually training the learning rate wasn't necessary.