Project 4 Behavioral Cloning

1.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, this model.h5 can let the car drive on track 1 successfully without issues.
- model-2.h5 this is the model which I used for both track1 and track2, but it still has some problems.
- README_P4.pdf has the final report which summarized the results and process

https://github.com/monkshao/CarND-Behavioral-Cloning-P3

2. MODEL ARCHITECTURE:

My model is based on the Nvidia model https://devblogs.nvidia.com/deep-learning-self-driving-cars/

This model consists of 5 convolution neural network with 5x5 kernel sizes for first 3 layers and 3x3 kernel sizes for next 2 layers, and filter depth between 24 and 64 (model.py lines 129-134).
I added a few layers to make the model performance better after a few tries and errors: The model includes ELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer (code line 126). I also added an additional dropout layer to avoid overfitting after the convolution layers.
(for track1 only, I didn't use max pooling layer, and and got very good val_loss, around 0.015)

3. DETAILS OF THE ARCHITECTURE:

keras2 api is used to make tensorflow code easy to understand. Here is the details of the model architecture.

- Image normalization
- Imgae cropping layer
- Convolution: 5x5, filter: 24, strides: 2x2, padding: same, activation: ELU
- Convolution: 5x5, filter: 36, strides: 2x2, padding: same, activation: ELU
- Convolution: 5x5, filter: 48, strides: 2x2, padding: same, activation: ELU
- Convolution: 3x3, filter: 64, strides: 1x1, padding: same, activation: ELU
- Convolution: 3x3, filter: 64, strides: 1x1, padding: same, activation: ELU
- Drop out (0.5)
- Flatten layer
- Fully connected: neurons: 100, activation: ELU
- Fully connected: neurons: 50, activation: ELU
- Fully connected: neurons: 10, activation: ELU

- Fully connected: neurons: 1 (output)

some layer details:

a. The data is normalized in the model by doing image normalization:
```
model.add(Lambda(lambda x: x/255.0 - 0.5,
input_shape=(160,320,3)))
```

b. As the goal of this model is to identify the right steering angles based on road images, so features like road edges are most important. Other unimportant features such as skys, trees are less important. So we can apply similar techniques as we did in previous project, such as image cropping. Here we can use Keras api to do this.
```
model.add(Cropping2D(cropping=((70,25), (0,0)),
input_shape=(3,160,320)))
```

c. Attempts to reduce overfitting in the model
tried with both dropout layer and max-pooling layer, turns out dropout layer with dropout rate 0.5 works well in this case.The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

d.Adam optimizer is used.
```
model.compile(loss='mse', optimizer='adam')
```

4. APPROPRIATE TRAINING DATA
Here for track1, I used the existing data from the workspace '/opt/carnd_p3/data/' to train the model for track1. and eventually this model works well and the car can drive autonomously without running out of the road.

to improve the model to be compatible with track2, because it is hard to use the online unity tool to drive the car manually within the track, so I also downloaded the simulator, and run the simulator locally to produce some images data for track2.
Then I uploaded the self-collected data to workspace, and used for training as well.

Some approaches taken:
a. Train/validation/test splits have been used,. 80% training data, 20% validation data
```
training_data_csv_lines, validation_data_csv_lines =
train_test_split(lines, test_size = 0.2)
```

b. If the image is read by opencv imread() function, it will load as BGR. Based on the forum discussion, I need to convert it to RGB for eras to work
```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

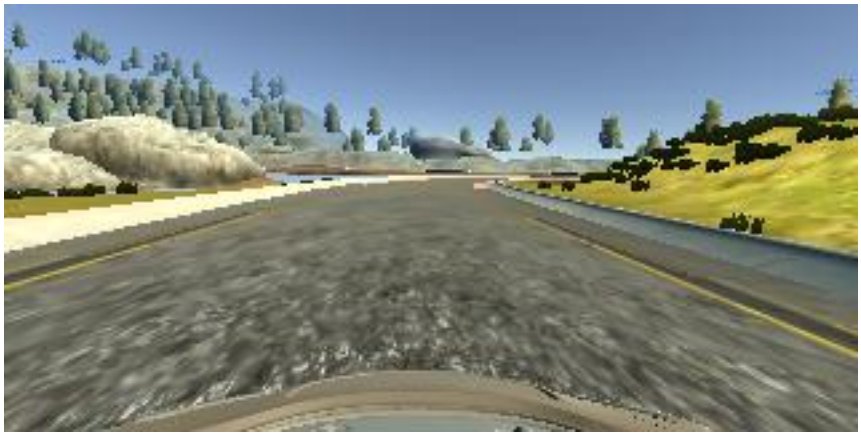
c. similar to what the course material suggests, I also used the left-right camera images, and also used numpy method to flip the center-image, with adjusted steering angles.

```
images.append(np.fliplr(center_img))
angles.append(-steering_angle)
```

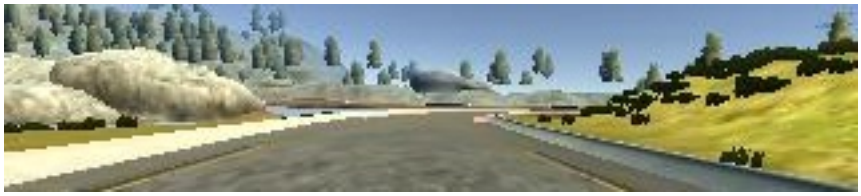
d. used model.fit_generator to load training data in memory batch by batch, which has smaller memory footprint.
e. Code to handle anomalies. during the image reading process, I found there are possible some missing images from the existing data. So I added some checks during image reading, to skip this image if the loading is not successful. (similarly, in the self-collected data for track2, the steering data might not be a valid float value, so also I also added some checks in the code)

Here are some visualizations of the image transformation I applied before I used them as training dataset.

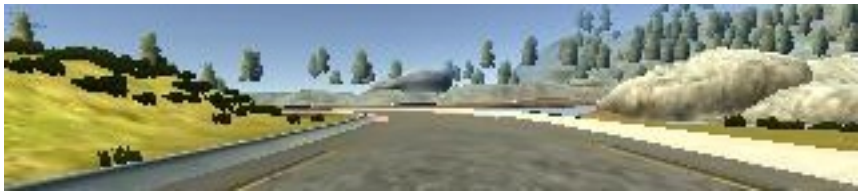
center image



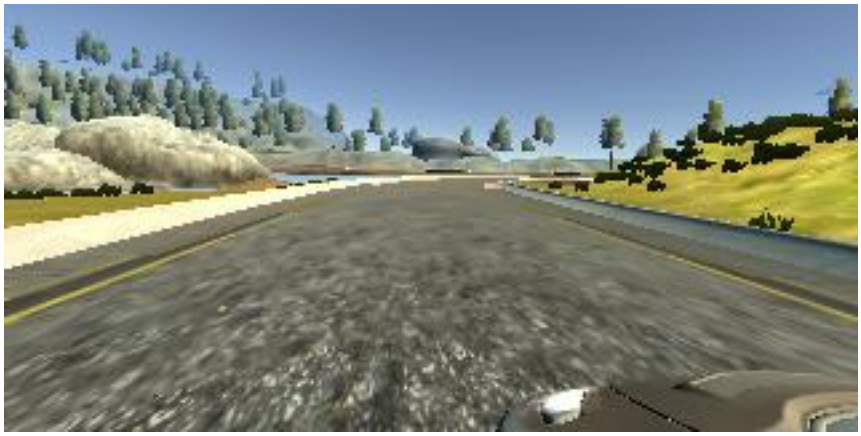
center cropped image



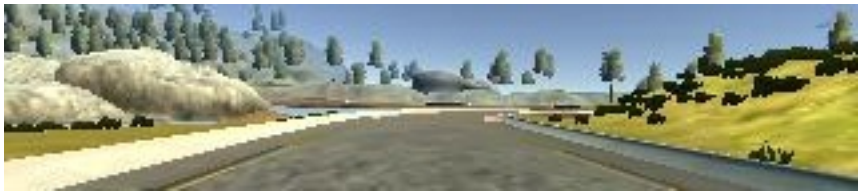
center flipped image



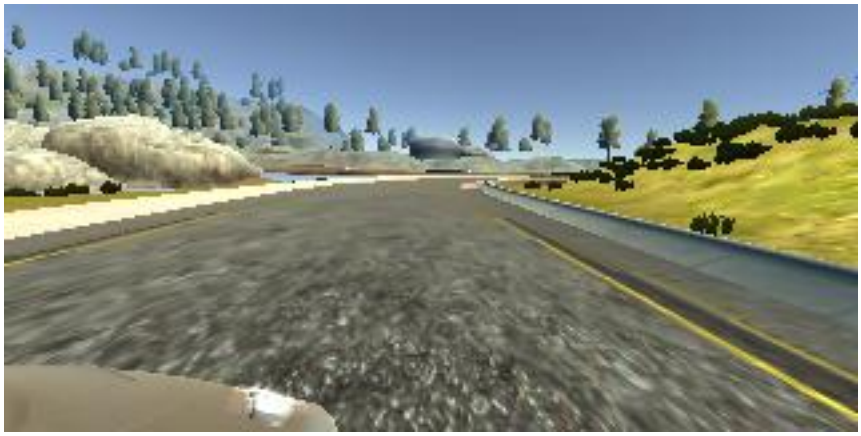
left image



left cropped image



right image



right cropped image

5. MODEL TRAINING PROCESS

a.In the submission, I have 2 models, **model.h5** is the good one for track1 which is being used to produce the video **lap1_run3_successful.mp4**, the mean squared error loss on the validation dataset is less than 0.015 by using this model after some parameter tuning, such as change the receptive fields of the convolutional layer, and adjust the left-right camera steering angle offset values. (04, 0.3 and 0.2; 0,2 works best)

b. after passing track1, I also tried with track2. I added about two 1.5 laps self-collected images to the input dataset. Also split the dataset into training and validation dataset as 80/20. And re-trained the model together with lap1's data, which is model-2.h5
Here is the code i used to train, which has not been committed to master branch.
https://github.com/monkshao/CarND-Behavioral-Cloning-P3/blob/lap2/model.py

However, I found this model-2.h5 still couldn't get a good val_loss less than 0.1; and also tested autonomous driving on lap2, it doesn't work well. (Though it still works fine for Lap1)
In track2, the problem is after driving for a while, the car veers off to the side, and couldn't recover.

6. FUTURE IMPROVEMENTS

Several things I could try to improve the model for track2.

a. there seems to be an overfitting in the model. As the training loss is 0.09, but the val_loss is 0.1, which is above the training loss. I could add some L2 regularization to the model architecture.
b. I think my recording for track2 is too "perfect", which means the car was mostly driving on the center lane. I probably need to collect more data which has vehicle recovering from the left side and right side back to the center of the road.
c. I think right now my problem is most likely a data collection issue, and there might be some other ways to adjust the input training dataset. But I don't have many good ideas at this moment…