Udacity Self-Driving Car Engineer

Report

- **Project:** Behavioural Cloning
- **Name:** Shao Hongxu

Let me just use some brief sentence to explain the thing I did, basically these are 6 big parts here:

Part 1: import necessary lib
Part 2: using training data generator function (read data real time while testing & spilt training & validation data)
Part 3: collect training data (most important part for this project)
Part 4: identify model
Part 5: use trained model to run simulation in Auto- mode.
Part 6: Summary

To be specific as below, you will find some note on Script, key take away and Area could be improved:

Part 1: import necessary lib:

In [ ]:
```python
import csv
import cv2
import numpy as np
from random import shuffle
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

Part 2: using training data generator function (read data real time while testing & spilt training & validation data)

```python
import cv2
import numpy as np
import sklearn

def generator(samples, batch_size=32):
    num_samples = len(samples)
    while 1: # Loop forever so the generator never terminates
        shuffle(samples)
        for offset in range(0, num_samples, batch_size):
            batch_samples = samples[offset:offset+batch_size]

            images = []
            angles = []
            for batch_sample in batch_samples:
                #name = 'G:/Udacity/Project 3/data collection/IMG/'+batch_sample[0].split('\\')[-1]
                name = 'C:/Users/HX/Self_drive_car_Project_3/data/data/IMG/'+batch_sample[0].split('/')
                center_image = cv2.imread(name)
                center_angle = float(batch_sample[3])
                images.append(center_image)
                angles.append(center_angle)
            augmented_images, augmented_angles=[],[]
            for image, angle in zip(images, angles):
                augmented_images.append(image)
                augmented_angles.append(angle)
                augmented_images.append(cv2.flip(image,1))
                augmented_angles.append(angle*-1.0)
            # trim image to only see section with road
            X_train = np.array(augmented_images)
            y_train = np.array(augmented_angles)
            yield sklearn.utils.shuffle(X_train, y_train)

# compile and train the model using the generator function
train_generator = generator(train_samples, batch_size=32)
validation_generator = generator(validation_samples, batch_size=32)
```

-> Key take away: This is a beautiful function especially for situation of a big data but a normal computing memory. Learnt from Udacity, it makes few Machine learning possible for me to have a try. (I have only one NVIDIA GTX 1080 with 8GB memory)

In the end, I select **batch size** as 32, which suits my GPU not running more than 50%.

Part 3: collect training data (most important part for this project)

1. Track one - one complete round. (forward) -> keep the car on the middle line
2. Track one - one complete round. (backward) -> keep the car on the middle line
3. Track one – collect data on scenario of car moving from one side into middle line. (most important data)
4. Training data provided by project
5. Track two - one complete round. (forward) -> keep the car on the middle line
6. Track two - one complete round. (backward) -> keep the car on the middle line

After all these data, still car sometime driving out of road at the point of road side next to lake & desert. After that, I was collecting a group of additional training data specific at the point of road side next to lake & desert. And replicate the data for these scenarios 4 time and put it into the training data. Finally, it works in the end.

Part 4: identify model

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, InputLayer, Lambda, Dropout
from keras.layers import Convolution2D, MaxPooling2D, Cropping2D
model=Sequential()
#model.add(InputLayer(input_shape=[160,320,3]))

model.add(Lambda(lambda x: (x / 255.0) - 0.5, input_shape=(160,320,3)))
model.add(Cropping2D(cropping=((50,20), (0,0))))
model.add(Convolution2D(24,5,5,subsample=(2,2),activation="relu"))
#model.add(MaxPooling2D())
model.add(Convolution2D(36,5,5,subsample=(2,2),activation="relu"))
#model.add(MaxPooling2D())
model.add(Convolution2D(48,5,5,subsample=(2,2),activation="relu"))
#model.add(MaxPooling2D())
model.add(Convolution2D(64,3,3,activation="relu"))
#model.add(MaxPooling2D())
model.add(Convolution2D(64,3,3,activation="relu"))
#model.add(MaxPooling2D())
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(100,activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(50,activation="relu"))
#model.add(Dropout(0.1))
model.add(Dense(10,activation="relu"))
#model.add(Dropout(0.1))
model.add(Dense(1))

model.compile(loss='mse', optimizer='adam')
#model.fit(x_train, y_train, validation_split=0.2, shuffle=True, nb_epoch=3)



from keras.models import Model
import matplotlib.pyplot as plt

history_object = model.fit_generator(train_generator, samples_per_epoch =
    len(train_samples), validation_data =
    validation_generator,
    nb_val_samples = len(validation_samples),
    nb_epoch=4, verbose=1)
model.save('model.h5')
```

Machine learning model structure:

- Input layer: 160*320*3
- 1st layer: cropping layer, to crop the image into area needed.
- 2nd layer: Convolution2D layer, 24, 5, 5
- 3rd layer: Convolution2D layer, 36, 5, 5
- 4th layer: Convolution2D layer, 48, 5, 5

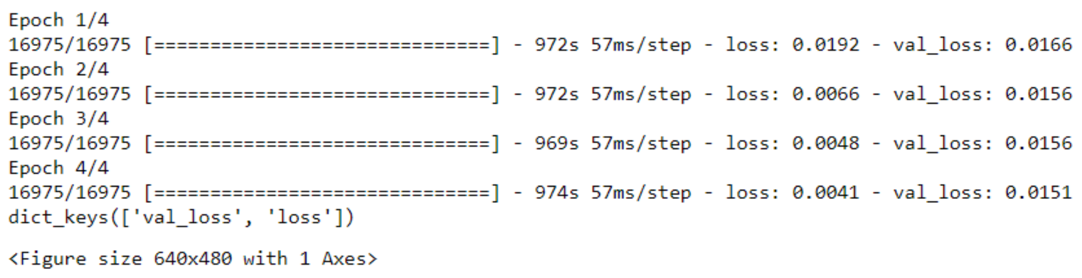
- 3rd layer: Convolution2D layer, 64, 5, 5
- 4th layer: Convolution2D layer, 64, 5, 5 , with 0.5 dropout
- 5th layer: flatten layer
- 6th layer: 100 unit, with 0.1 dropout
- 7th layer: 50 unit, with 0.1 dropout
- 8th layer: 10 unit
- Output layer: 1

Learning rate parameters is Adam optimizer for this model.
Dropout (0.5, 0.1) has been selected to protect overfitting.
In the end, only 4 epochs give a good enough result for simulation track one.
Training & validation set has been spilt into 80% and 20%

-> Key take away: dropout is needed, I tried model without dropout, validation set loss never decreased.

-> Area could be improved: last 3 layers (100, 50, 1) architecture is slightly simple, maybe it can be improved a little bit to make it more intelligent.

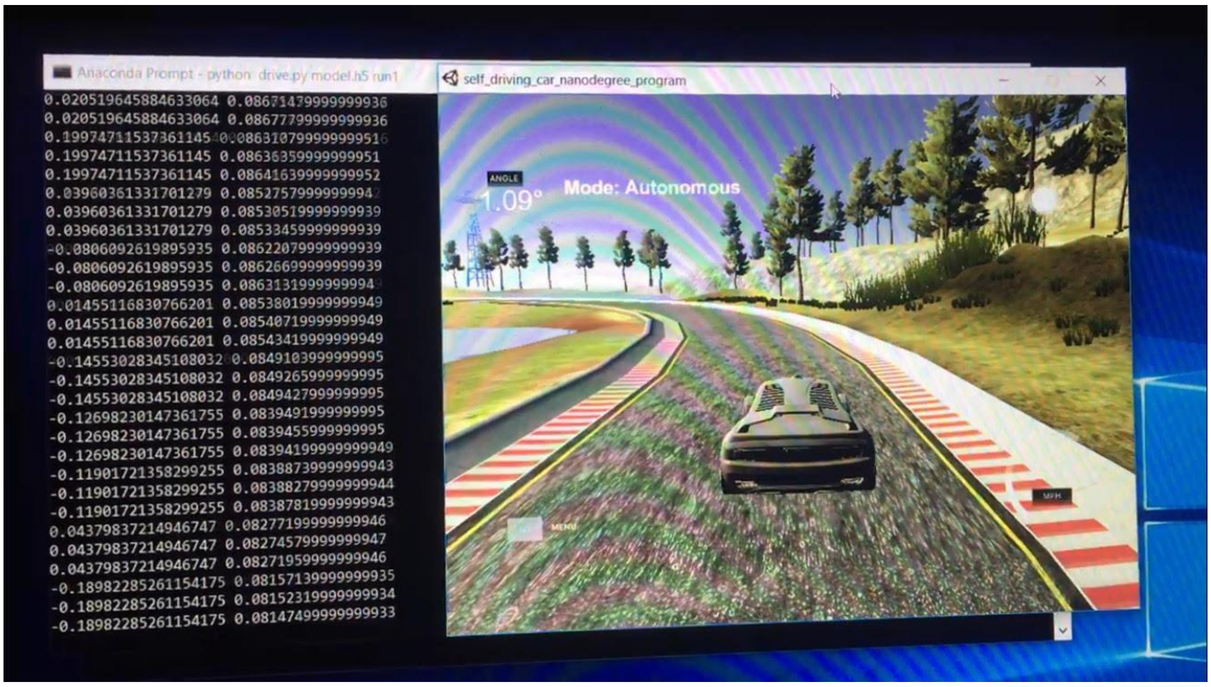
-> Result:



```
Epoch 1/4
16975/16975 [==============================] - 972s 57ms/step - loss: 0.0192 - val_loss: 0.0166
Epoch 2/4
16975/16975 [==============================] - 972s 57ms/step - loss: 0.0066 - val_loss: 0.0156
Epoch 3/4
16975/16975 [==============================] - 969s 57ms/step - loss: 0.0048 - val_loss: 0.0156
Epoch 4/4
16975/16975 [==============================] - 974s 57ms/step - loss: 0.0041 - val_loss: 0.0151
dict_keys(['val_loss', 'loss'])

<Figure size 640x480 with 1 Axes>
```

Part 5: use trained model to run simulation in Auto-mode.


Anaconda Prompt - python drive.py model.h5 run1
self_driving_car_nanodegree_program
ANGLE
1.09°
Mode: Autonomous
MENU
MPH

```
0.020519645884633064 0.08671479999999936
0.020519645884633064 0.08677799999999936
0.19974711537361145 0.08631079999999951
0.19974711537361145 0.08636359999999951
0.19974711537361145 0.08641639999999952
0.03960361331701279 0.0852757999999994
0.03960361331701279 0.08530519999999939
0.03960361331701279 0.08533459999999939
-0.0806092619895935 0.08622079999999939
-0.0806092619895935 0.08626699999999939
-0.0806092619895935 0.0863131999999994
0.01455116830766201 0.08538019999999949
0.01455116830766201 0.08540719999999949
0.01455116830766201 0.08543419999999949
-0.14553028345108032 0.0849103999999995
-0.14553028345108032 0.0849265999999995
-0.14553028345108032 0.0849427999999995
-0.12698230147361755 0.0839491999999995
-0.12698230147361755 0.0839455999999995
-0.12698230147361755 0.08394199999999949
-0.11901721358299255 0.08388739999999943
-0.11901721358299255 0.08388279999999944
-0.11901721358299255 0.08387819999999943
0.04379837214946747 0.08277199999999946
0.04379837214946747 0.08274579999999947
0.04379837214946747 0.08271959999999946
-0.18982285261154175 0.08157139999999935
-0.18982285261154175 0.08152319999999934
-0.18982285261154175 0.08147499999999933
```

In the end, car is able to run for one complete round in track one. You can check video attached in detail.

Part 6: Summary

Personally, I really like this cloning project very much. It brings me the sense of the most important area for machine learning/ deep learning project. Most of the time, for a machine learning competition, data has been provided. In that case, as a individual, most of the time we even couldn't start the project from data collection.

And project make me feel about the improvement from data you put in addition. That is really great!