Project: Behavioral Cloning
Name: Dongdong Wu

Brief Description:
This project is try to clone driving behavior of human to drive a car in simulator by design and train a convolutional neural network(CNN). In my solution, the behavior cloning use a simple CNN network based on preprocessed images. Before describing my solution, I have to say the reason why i choose this way to do this.

At the very beginning, I directly load all input images of all angles(left, middle, right) with original size and also with normalization. I build a large CNN network which is similar with tutorial in Carnd-class. but soon i found there exists problems there some of which i still not figure out why it happen:

a. Training time is extremelly long when on my CPU PC, so i move the project on AWS where trainning speed is not a problem any more.

b. however when i run project on AWS, it frequently report memory error when i uncarefully restart project and load images. That is so boring. so I resized the input image and shrink convolution network based on reference case from website.

c. main function files:

Model.py provide image pre-process function and model architecture

drive.py simulate drivine behavior based on simulater image

1. Preprocess:

This main idea here is to eliminate background and unnecessary information but keep lane info of original image. which will increase trainning speed.

a. transfrom image to HSV format but only keep S value. from next class, for lane detection, S channgel is most imortant to detect lane

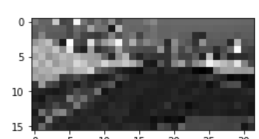
b. resize image to 32 * 16 size comparing original size = 320 * 160 which help for increasing trainning speed

c. load all images and measurement of all angels

d. load flipped data of original and add to training set

sample of image/preprocess image:

left_2016_12_01_13_39_27_420.jpg



2. Model architechture

a. Normalization layer to normalize image value to within (0, 1)

b. 2D convolution with valid padding and relu activation: kernel size = (3,3).

c. Max pooling layer with valid padding: kernel size = (4,4).

d. Dropout layer with ratio = 0.25 (which is not necessary since this is very small CNN network and dropout will make training unstable)

e. Flatten layer.

f. Dense layer to sum up to 1 value which represent steering angle.

3. Training
Since the input data is resized to 32 * 16, i don't use generator to fit memory.
version 1:
[I use Epoch = 10, bache_size = 128 without any validation data. Since the small image, the training speed is much faster which is about dozens of seconds on CPU. During training, the loss will down roughly from 0.1 to 0.006.]

version 2:
add validation data to avoid overfitting. the validation data picked from training data is about 0.2 proportion of all data. Others are the same
report:
Train on 38572 samples, validate on 9644 samples
Epoch 1/10

```
Train on 38572 samples, validate on 9644 samples

Epoch 1/10

38572/38572 [==============================] - 6s - loss: 0.0788 - val_loss: 0.0582 - ETA: 2s - loss: 0.0862

Epoch 2/10

38572/38572 [==============================] - 5s - loss: 0.0559 - val_loss: 0.0447

Epoch 3/10

38572/38572 [==============================] - 5s - loss: 0.0474 - val_loss: 0.0403

Epoch 4/10

38572/38572 [==============================] - 5s - loss: 0.0439 - val_loss: 0.0390 - ETA: 2s - loss: 0.0446

Epoch 5/10

38572/38572 [==============================] - 5s - loss: 0.0423 - val_loss: 0.0378 - ETA: 2s - loss: 0.0427- ETA: 2s - loss: 0.0428- ETA: 0s - loss: 0.0425

Epoch 6/10

38572/38572 [==============================] - 5s - loss: 0.0416 - val_loss: 0.0370 - ETA: 2s - loss: 0.0417

Epoch 7/10

38572/38572 [==============================] - 5s - loss: 0.0414 - val_loss: 0.0367

Epoch 8/10

38572/38572 [==============================] - 5s - loss: 0.0405 - val_loss: 0.0369
```

```
Epoch 9/10

38572/38572 [==============================] - 5s - loss: 0.0406 - val_loss:
0.0365

Epoch 10/10

38572/38572 [==============================] - 5s - loss: 0.0404 - val_loss:
0.0367
```