Use Deep Learning to Clone Driving Behavior

The Goal:
This project is to use deep learning to learn the driving steering from a collection of images. The images are taken from a driving simulator where you can control a simulated car to drive along a path, the driving steering is recorded together with the picture from car center, left and right.
After we use deep learning estimate the driving steering in the various situations and generate a model, we can use the model to clone the driving behavior.

Dataset Collection
The dataset was collected from a simulator. Usually you drive a few laps along the path, you got the data, but a few things to pay special attention while collecting the data:

- Drive both clockwise and count clockwise, it's obvious clockwise has more right turns than count clockwise while count clockwise has more left turns than clockwise. Drive both directions provides us balanced turn data

- Collection some recovery data especially for sharp turn area so the model would learn how to recovery for a shifted situation

- There are a few shadow and muddy areas, worth to drive serve times to get enough data to make sure we can go through those cases.
- Sample images

Dataset separation

The entire dataset was separated into 80% training data and 20% validation data, test data was not used since we use the simulator to test.

Dataset Augmentation

At each moment, the simulator takes 3 pictures: center picture, left picture and right picture, but only 1 steering angle provided.

- Left picture and right picture utilized the same angle as center picture, but with angle compensation applied.
- Flip each picture and steering angle to augment the data set

Dataset Normalization

- A special note: the image collected is in RGB format, but it will be BGR format if we read it from cv2; the most important it's that the image is in RGB format when we test drive in autonomous model; therefore, we need to convert the data to RGB format.
- Another normalization is to make each training data having a zero mean and equal variance. Since each pixel value ranges from 0 to 255, I applied a simple formula:
  x / 127.5 – 1.0

Dataset Crop
On the top of the image there are some trees; and car hood on the bottom of the image; they are irrelevant to the training model, thus crop them from the image.

Model Architecture
A typical NVidia network, implemented in Keras, other than a lambda layer for normalization and a crop layer for image cropping, it includes:

- A convolution layer with filter = 24, kernel = 5, followed by a (2,2) maxpooling layer
- A convolution layer with filter = 36, kernel = 5, followed by a (2,2) maxpooling layer
- A convolution layer with filter = 48, kernel = 5, followed by a (2,2) maxpooling layer
- A convolution layer with filter = 64, kernel = 5
- A convolution layer with filter = 64, kernel = 5
- A flatter layer
- A fully connected layer with size as 100
- A fully connected layer with size as 50
- A fully connected layer with size as 10
- Finally, a fully connected layer with size as 1 as the output

Training process

- There is no activation function used since it's a regression model instead of a classifier.
- The loss function is mean square error.
- The optimizer is Adam
- Two callbacks were utilized for the model training:
    - One ModelCheckingPoint callback to check val_loss and save the model with minimum val_loss
    - One EarlyStopping callback to terminate the training if the val_loss was not improved for certain amount of epochs.
- A generator was used for each batch so we would not run out of memory, each iteration takes 80% of the data for training, and rest 20% for validation,
- The training loss and validation loss were calculated for each epoch so the callbacks know what to do.