Project: Behavioral Cloning

This project aims at building a network which can successfully steer car around the track in given simulator. A video that shows the performance of the network in driving the simulated car around the track is provided (/home/workspace/CarND-Behavioral-Cloning-P3/video.mp4).

The steps followed in the building the model is:

1) Collect Training Data from simulator
2) Split the collected dataset into training and validation datasets
3) Build convolutional neural network model
4) Train and save the model
5) Use saved model to drive car around the simulator track

Collect Training Data from Simulator:

The workspace from the project provides access to a simulator with Training Mode and Autonomous Mode. In training mode, we can collect training and validation dataset to a choice of folder by driving the car around the track.

Inside the folder of choice (chosen to save the training mode data), a CSV file ("**driving_log.csv**") and an "**IMG/**" folder gets created. The CSV file has following columns.

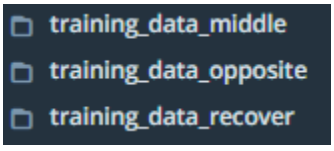
1. **Center Image:** Complete folder path along with the image name for images taken from center camera.
2. **Left Image:** Complete folder path along with the image name for images taken from left camera.
3. **Right Image:** Complete folder path along with the image name for images taken from right camera.
4. **Steering Angle:** The steering angle at the time the above three images are captured.
5. **Throttle:** The throttle used when the images in first three columns are captured.
6. **Break:** Like throttle, break used.
7. **Speed:** Speed of car when the center, left and right image are captured.

Out of the above seven columns, we use first three (center, left and right images) and the fourth column (Steering Angle) for training the network to be described below. The below image shows the entries in the CSV file.

	A	B	C	D	E	F	G
1	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	2.73E-07
2	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	4.89E-07
3	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	1.30E-06
4	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	1.58E-06
5	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	1.98E-06
6	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	2.44E-06
7	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	3.04E-06
8	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	1.25E-06
9	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	4.30E-06
10	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	4.90E-06
11	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	3.10E-06
12	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	3.37E-06
13	/home/workspace/CarND-	/home/workspace/CarND-Beh	/home/workspace/CarND	0	0	0	6.43E-06

Below are the images that have entries in the above-mentioned CSV file. They are show with Center Image to left, followed by Image from Left Camera and Image from right camera to the right. (Note: Each of the below images are associated with same steering angle).

Like mentioned above three sets of training data is collected. They are i) By driving car in the middle of road, ii) By driving the car in opposite direction and in middle of the road and iii) By driving the car from sides of the road into middle of the road. Each data is saved into three different folders, as shown below.



Splitting the collected data into training and validation sets:

As mentioned above the CSV files (each named "**driving_log.csv**") contain the path to images taken from center, left and right cameras respectively along with the associated steering angle. So, each

from the log can be considered a data point. So, all the lines on three of the log files along with the folder where the log is present are loaded into a python list using the "csv" python library.

These entries are then split into training and validation sets using the **Sklearn's "train_test_split"** module. 70% of entries are taken as training set and remaining 30% as validation set.

Generator: A Generator is defined with a batch size of 32 to process the test and validation sets (lines from csv files), so images related can be made available to train the model. Each of the center, left and right camera images are loaded into a **numpy array** using **"cv2"** library (please note with batch size of 32 each time). Also, the associated steering angle reading are loaded into numpy array (steering angle is what the network must predict), for left and right images an angle correction of **"+0.2"** and **"-0.2"** is applied respectively. Along with these a flipped image is generated for each of center, left and right images and are loaded into numpy array (their steering angles are also loaded by multiplying respective steering angles with "-1"). The generated data is shuffled and yielded to the model to train and validate upon.

Build Convolutional Neural Network Model:

A model like NVIDIA's Neural Network described in the lecture is built for the project.

Keras python library is used here to build the required Convolutional Neural Network. Modules used from Keras are "**Sequential, Dense, Flatten, Activation, Dropout, Convolution2D, Lambda, Cropping2D**".

The network built has below layers.

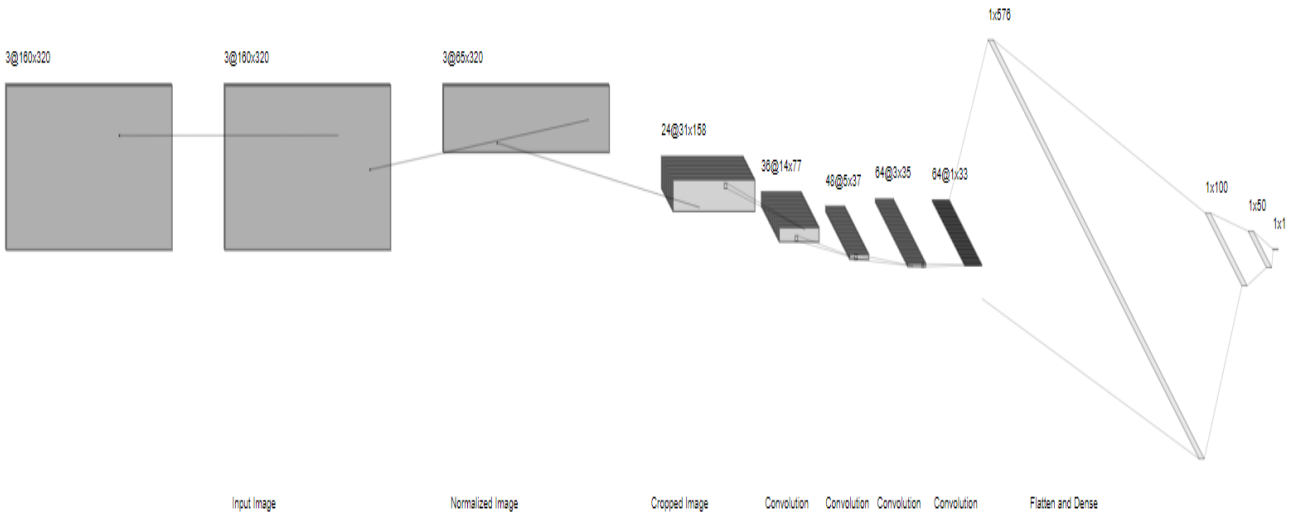
1. **Input Layer:** The images generated are of shape (160, 320, 3).
2. **Normalization Layer:** This layer normalizes input images to the model using **Lambda** module of Keras. Each pixel value is first divided by "**255**" and then subtracted by "**0.5**".
3. **Cropping Layer:** As can be seen from above images, the top and few rows at the bottom of the images taken from center, left and right cameras are not really important (top part does not have road and bottom part shows hood of the car), so the image are cropped to remove the top and bottom parts as shown below. This operation is performed using Keras's **Cropping2D.**

4. **Convolutional Layer:** This layer is implemented using Keras's **Convolution2D.** A filter shape of **5x5,** depth of **24,** and a stride of **2x2** is applied on the cropped images. The activation used after the convolution is **RELU**.
5. **Convolutional Layer:** This layer is implemented using Keras's **Convolution2D.** A filter shape of **5x5,** depth of **36,** and a stride of **2x2** is applied on the cropped images. The activation used after the convolution is **RELU**.

6. **Convolutional Layer:** This layer is implemented using Keras's **Convolution2D.** A filter shape of **5x5,** depth of **48,** and a stride of **2x2** is applied on the cropped images. The activation used after the convolution is **RELU**.
7. **Convolutional Layer:** This layer is implemented using Keras's **Convolution2D.** A filter shape of **3x3,** depth of **64,** and a stride of **1x1** is applied on the cropped images. The activation used after the convolution is **RELU**.
8. **Convolutional Layer:** This layer is implemented using Keras's **Convolution2D.** A filter shape of **3x3,** depth of **64,** and a stride of **1x1** is applied on the cropped images. The activation used after the convolution is **RELU**.
9. **Flatten Layer:** This layer is implemented using Keras's **Flatten.** The output of the above layer is flattened.
10. **Fully Connected Layer followed by Dropout Layer:** Implemented using Kera's **Dense and Dropout**. Has **576** neurons with **RELU** activation followed by a Dropout layer with retain probability of **0.5**.
11. **Fully Connected Layer followed by Dropout Layer:** Implemented using Kera's **Dense and Dropout**. Has **100** neurons with **RELU** activation followed by a Dropout layer with retain probability of **0.5**.
12. **Fully Connected Layer:** Implemented using Kera's **Dense**. Has **50** neurons with **RELU** activation.
13. **Output Layer:** It has one neuron and predicts the steering angle of the car on simulator.

NOTE: Dropout is used to reduce overfitting.

Below images shows a visualization of the network. The shape of output of each layer are shown in the next section.



Train and Save the Model:

The Keras model created above is compiled using "**Mean Squared Error**" as loss function and **AdamOptimizer** as the Optimizer.

It is then trained and validated using the Keras's "**fit_generator**" function, to which the training and validation generators (described above) are passed as parameters. Also as mentioned above, for each image, its left and right images, and the flipped images of all the three images are yielded by the generator. So, "**steps_per_epoch**" is calculated as **(((number of training samples) * 3 *2) / batch size)**. Similarly, "**validation_steps**" is calculated as (**((number of validation samples) * 3 *2) / batch size)**. Number of epochs are taken as **5**.

The trained model is saved as "model.h5" using the Keras's "**save**" function.

Below is the image of summary of trained model, along with the shape of the oupur generated by each layer.

```
742/742 [==============================] - 209s 281ms/step - loss: 0.0409 - val_loss: 0.0368
Epoch 2/5
742/742 [==============================] - 206s 278ms/step - loss: 0.0291 - val_loss: 0.0342
Epoch 3/5
742/742 [==============================] - 207s 280ms/step - loss: 0.0207 - val_loss: 0.0306
Epoch 4/5
742/742 [==============================] - 209s 282ms/step - loss: 0.0142 - val_loss: 0.0324
Epoch 5/5
742/742 [==============================] - 209s 281ms/step - loss: 0.0103 - val_loss: 0.0305
Model Trained and saved!!!!!!!

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 160, 320, 3)       0
_________________________________________________________________
cropping2d_1 (Cropping2D)    (None, 65, 320, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 31, 158, 24)       1824
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 77, 36)        21636
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 37, 48)         43248
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 35, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 33, 64)         36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 2112)              0
_________________________________________________________________
dense_1 (Dense)              (None, 576)               1217088
_________________________________________________________________
dropout_1 (Dropout)          (None, 576)               0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               57700
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
=================================================================
Total params: 1,411,237
Trainable params: 1,411,237
Non-trainable params: 0
_________________________________________________________________
```

Use Saved Model to autonomously drive car around the track:

The saved model “model.h5” is used to autonomously drive the car around the track. The provided “drive.py” and “video.py” programs are used to run autonomously and record a video (**/home/workspace/CarND-Behavioral-Cloning-P3/video.mp4**).