Project Behavioral Cloning

Aim of the project:

Collect data and train a deep neural network to clone a driving behavior from a simulator.

The project contains the following relevant files:

- model.py - creation & training of model
- drive.py - script to drive the car
- model.h5 - trained model that has been saved
- video.mp4 - video recording of the vehicle driving autonomously around the track based on the model saved in model.h5 file
- writeup_report - This file containing documentation of the project

(There are a few other files such as training_new, model_trial.py, model.h5.bkp which can be ignored as they are files/folders related to initial trial-and-error in building the data model)

Data collection:

The simulator can be run on “training” mode to collect data which includes camera images from 3 different angles (center, left, right), the corresponding steering angles, throttle, break and speed. The data is captured in the following format at the following location - *./training_data/driving_log.csv .*

A	B	C	D	E	F	G
Center Image	Left Image	Right Image	Steering Ang	Throttle	Break	Speed
IMG/center_2022_02_28_02_29_05_068.jpg	IMG/left_2022_02_28_02_29_05_068.jpg	IMG/right_2022_02_28_02_29_05_068.jpg	0	0	0	3.013167
IMG/center_2022_02_28_02_29_05_145.jpg	IMG/left_2022_02_28_02_29_05_145.jpg	IMG/right_2022_02_28_02_29_05_145.jpg	0	0	0	2.988954
IMG/center_2022_02_28_02_29_05_216.jpg	IMG/left_2022_02_28_02_29_05_216.jpg	IMG/right_2022_02_28_02_29_05_216.jpg	0	0	0	2.97092
IMG/center_2022_02_28_02_29_05_287.jpg	IMG/left_2022_02_28_02_29_05_287.jpg	IMG/right_2022_02_28_02_29_05_287.jpg	0	0	0	2.947046

The data has been collected by driving the vehicle in 3 loops to capture 3 different scenarios:

1. Keeping the vehicle in the middle of the lane lines as much as possible
2. Driving the vehicle in the opposite direction.
   (The data captured in step 1 has also been reused by flipping the images. This step mimics similar data, therefore the data collected is doubled.)
3. Purposely drifting the vehicle from the middle and bringing it back to the center of the lane lines.

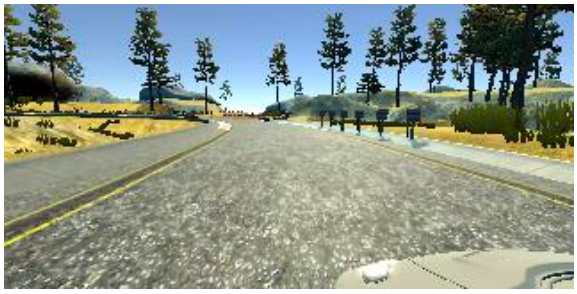
Pipeline:

1. The raw data is loaded from ./training_data/IMG by reading the image paths stored in *./training_data/driving_log.csv* .
2. The dataset contains 7464 images in total. A few samples of images captured are as below:

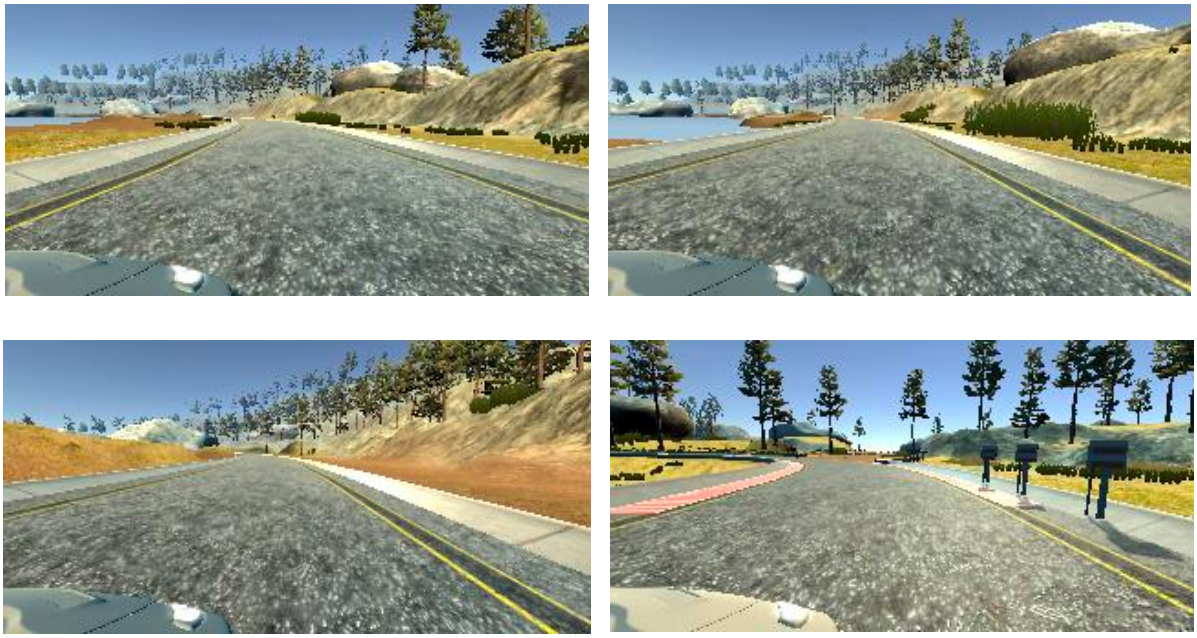
Center




Left




Right



3. The data is augmented by undergoing 2 steps:

- By cropping the upper part which is the portion of the image including the sky and the lower part which includes the vehicles' hood area. This required a lot of trial-and-error to set the appropriate parameters in order to capture only the required portion of the image within the frame.

```
#To remove parts of image that are not required
model.add(Cropping2D(cropping=((50,20), (0,0)), input_shape=(160,320,3)))
```

- By flipping the captured images and multiplying the corresponding steering angles by -1. This helps in doubling of the captured data.

```
for aug_img,aug_sa in zip(camera_images,steering_angles):
    augmented_images.append(aug_img)
    augmented_s_angles.append(aug_sa)

    #Flipping the image
    augmented_images.append(cv2.flip(aug_img,1))

    #Reversing the steering angle
    augmented_s_angles.append(aug_sa*-1.0)
```

4. As part of pre-processing of the data the image is resized. Following is the code snippet for the resizing method.

```
def preprocess(image):
    import tensorflow as tf
    #Resizing the image
    return tf.image.resize_images(image, (200, 66))
```

5. Model Architecture:

The CNN architecture has been used from NVIDIA's End to End Learning for Self-Driving Cars paper, published by the autonomous vehicle team at NVIDIA. (Reference to course concept "Even more powerful network")
The network consists of a normalization layer, followed by five convolutional layers, followed by four fully connected layers

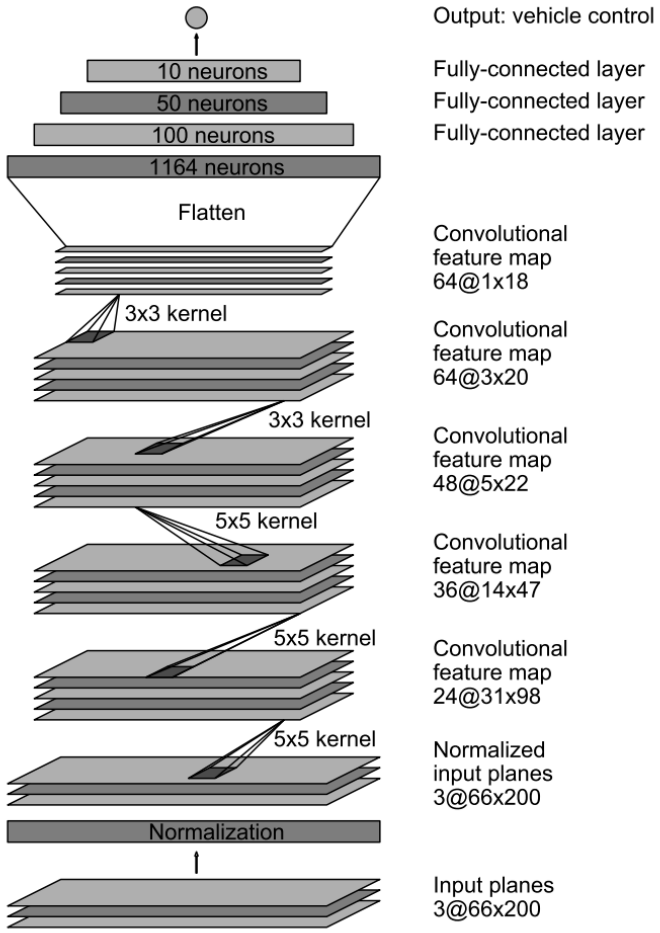


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
cropping2d_1 (Cropping2D)    (None, 90, 320, 3)        0
_________________________________________________________________
lambda_1 (Lambda)            (None, 200, 66, 3)        0
_________________________________________________________________
lambda_2 (Lambda)            (None, 200, 66, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 98, 31, 24)        1824
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 47, 14, 36)        21636
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 5, 48)         43248
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 20, 3, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 18, 1, 64)         36928
_________________________________________________________________
dropout_1 (Dropout)          (None, 18, 1, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               115300
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
```

6. Finally, the model is compiled using Mean Square Error (MSE) loss function and Adam optimizer (Adaptive moment estimation)
   Initially, the number of epochs is set to 13. However, it was observed after a couple of runs that the validation loss started to increase after the 10th epoch.
   The number of epochs is therefore set to 10.
   20% of the training data is used as validation data in each epoch.
   The final model is saved to model.h5

```
97  model.compile(loss='mse',optimizer='adam')
98  model.fit(X_train,y_train,validation_split=0.2,shuffle=True,nb_epoch=10)
99  model.save('model.h5')
```

```
(device: 0, name: Tesla K80, pci bus id: 0000:00:04.0)
5971/5971 [==============================] - 9s 1ms/step - loss: 0.0158 - val_loss: 0.0748
Epoch 2/10
5971/5971 [==============================] - 5s 850us/step - loss: 0.0078 - val_loss: 0.0693
Epoch 3/10
5971/5971 [==============================] - 5s 856us/step - loss: 0.0051 - val_loss: 0.0672
Epoch 4/10
5971/5971 [==============================] - 5s 854us/step - loss: 0.0035 - val_loss: 0.0647
Epoch 5/10
5971/5971 [==============================] - 5s 855us/step - loss: 0.0027 - val_loss: 0.0657
Epoch 6/10
5971/5971 [==============================] - 5s 851us/step - loss: 0.0022 - val_loss: 0.0692
Epoch 7/10
5971/5971 [==============================] - 5s 855us/step - loss: 0.0018 - val_loss: 0.0806
Epoch 8/10
5971/5971 [==============================] - 5s 852us/step - loss: 0.0015 - val_loss: 0.0701
Epoch 9/10
5971/5971 [==============================] - 5s 852us/step - loss: 0.0015 - val_loss: 0.0683
Epoch 10/10
5971/5971 [==============================] - 5s 850us/step - loss: 0.0014 - val_loss: 0.0777
```

7. A recorded video of the simulator in autonomous mode is saved as run1.mp4. It shows the vehicle driving 1 loop of the track.