Project 4: Behavioral Cloning

The goal of this project is to design a deep neural network to copy and replicate a driving behavior based on pre-collected data in the simulator.

This is achieved in following steps:

1. Collect data from simulator with driving characteristics such as – steering angle, throttle, brake, speed. The data collection is in the form of an image from three dashboard cams – left, right and center
2. Preprocess the data for model improvement – 1. Shuffling 2. Data augmentation by image flipping
3. Creating training and validation set
4. Model architecture – normalize and crop image. Implement the NVIDIA network architecture and parameter tuning
5. Implement Keras deep neural network framework to create a model.h5 file which is used to run the car autonomously on the simulator. Drive.py is used to connect the car to the simulator

---

Required Files and Quality of Code

- Required Files
  - *model.py* – script used to create the neural network framework and train the model
  - *drive.py* – Python scripts used to drive the vehicle autonomously once the deep neural network model is trained
  - *model.h5* – has the trained convolutional neural network
  - *video.py* – script used to record video of the vehicle when it is driving autonomously
  - *run1.mp4* – video file
- Quality of code
  - Model.py contains a generator to generate data for training rather than storing it in memory. It has a clearly organized and labeled neural network framework with pipeline to train and validate the model.
  - Run1.mp4 shows the video output of car driving autonomously around the track with command – *python drive.py model.h5 run1*

Data Collection

- Data collection –
  - I decided to use the sample driving data provided by Udacity for the first track to start my project.
  - The data consists of information such as steering angle, throttle, brake, speed. The data collection is in the form of an image from three dashboard cams – left, right and center
  - In the sample images shown below, images from left, right and center camera can be identified by the position of the car hood.

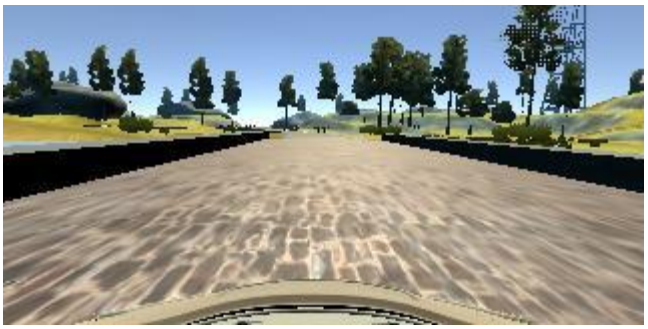


Figure 1:Sample image – Center

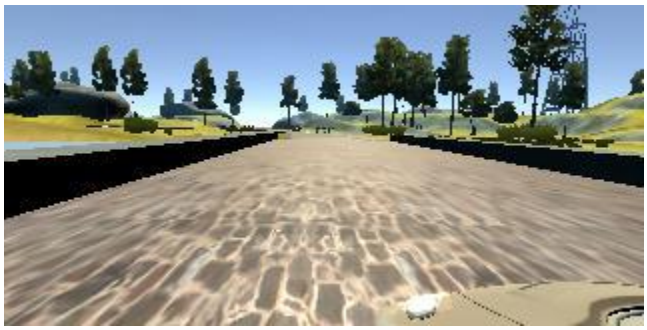


Figure 2- Sample image – left

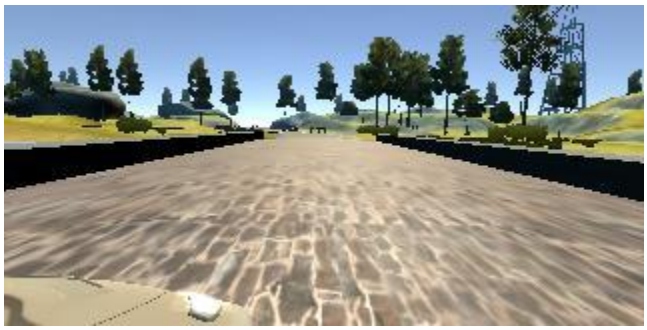


Figure 3 - Sample image - right

- Training and validation set
  - Using sklearn preprocessing library, I split the dataset into training and validation set
  - Validation set size being 15% of the total samples
- Preprocessing the data and generator function
  - To remove the left turn bias from the image samples, as a first step of the image preprocessing and data augmentation, I flipped each image by taking negative of the associated steering angle. So now, I have 6 images (2 each from each camera) for one entry in csv file.

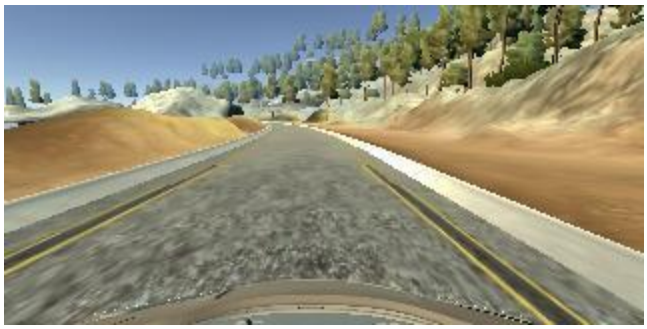


Figure 4: Original image

Figure 5: flipped image

- I used generator function to make my code memory efficient. Instead of saving processed images in the memory it processes each image on the fly as required
- As a next step, I shuffled the images to avoid order of the image to be an influencing factor in training of the neural network
- I introduced a correction factor of 0.2 to the steering angle recorded by left and right camera. This is subtracted from the steering angle captured by the right camera and added to the steering angle captured by the left camera
- The generator function is only used to hold the values of images and angles (X_train and y_train) while the function is running. This is done by using yield instead of return

Model Architecture

- Cropping images in Keras –
  - Before setting up the model, I cropped the image to focus only on the road portion of the image and remove top part which has unnecessary information likes sky, trees and hills.
  - This step allows the model to train faster without losing useful information required to predict steering angle

Figure 6: Original image

Figure 7: Cropped image

- Model architecture
  - For deep neural network architecture, I began by implementing NVIDIA model as explained in the lectures

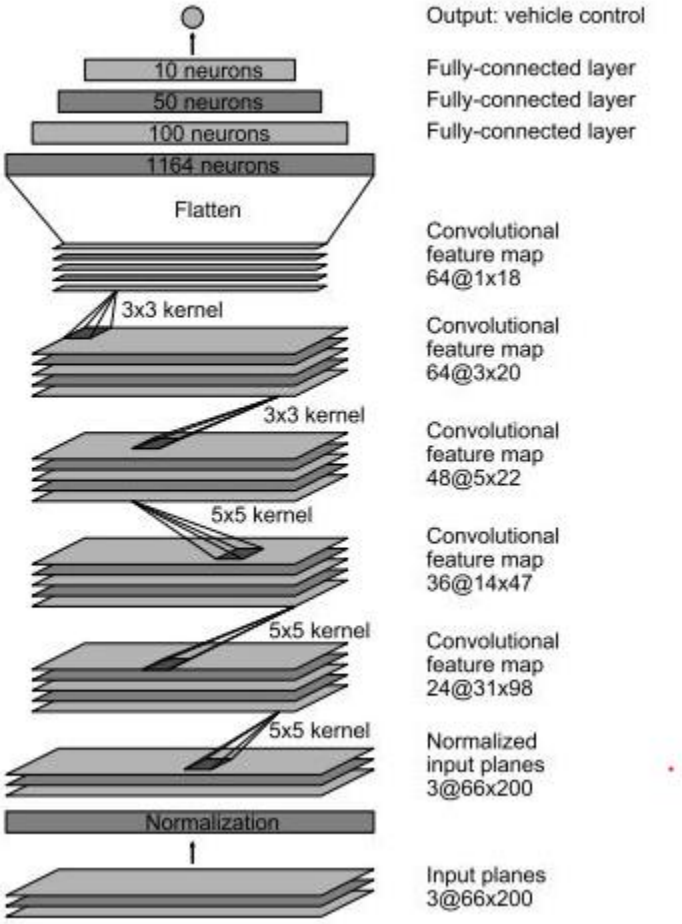
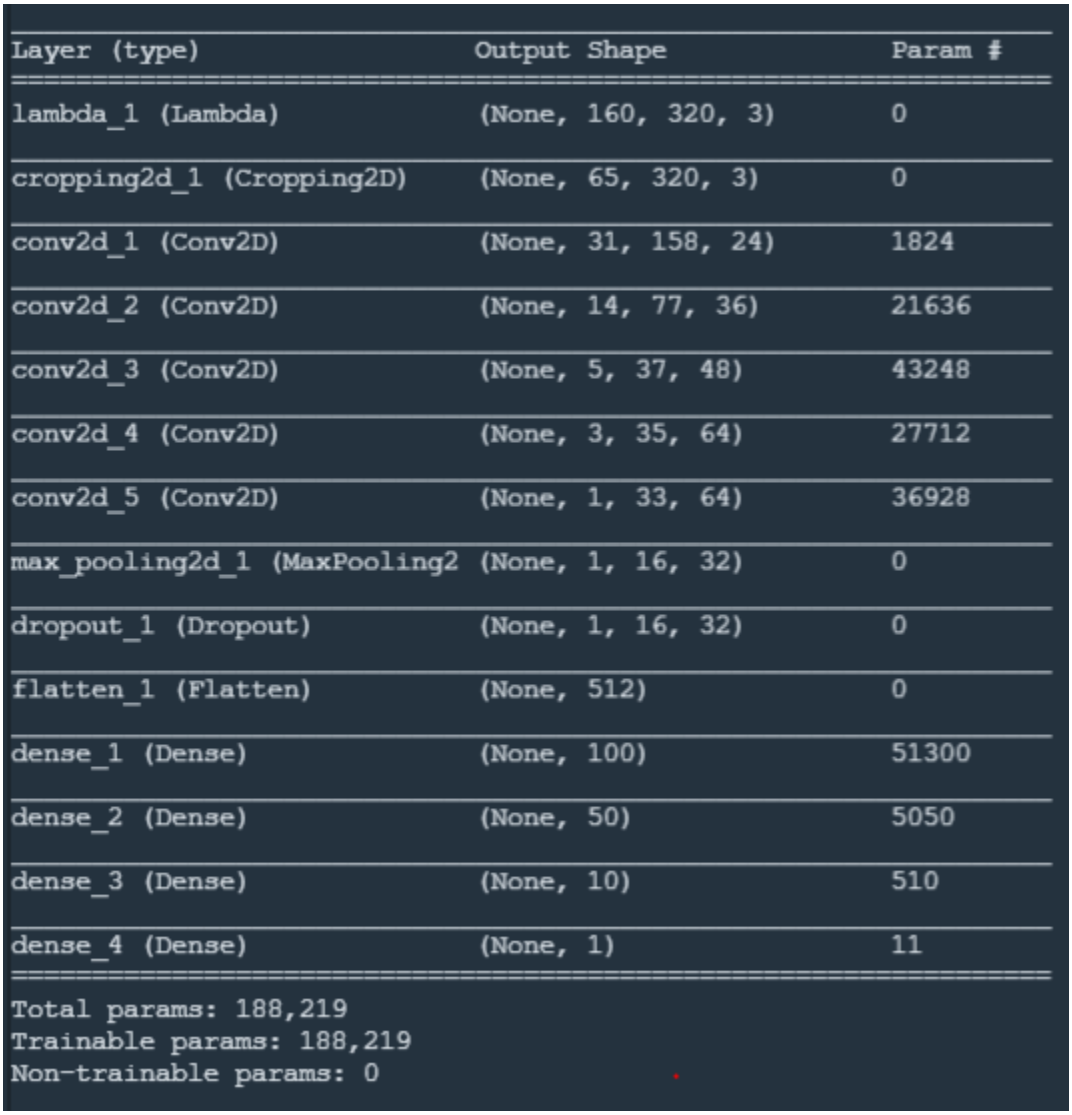


Figure 8: NVIDIA deep neural network

  - After normalization and cropping the image to (70,25) from (top, bottom), my final model architecture looks like shown in the summary below
  - Model architecture –
    - Three 5x5 convolution layers (output depth 24, 36, and 48), each with 2x2 stride, followed by a RELU activation
    - Followed by two 3x3 convolution layers (output depth 64), each with 1x1 stride and 3x3 filer size
    - Followed by a max pooling layer with pooling size 2x2
    - Followed by a dropout layer to avoid overfitting with probability of 0.5. Output shape of the image after this layer is (1x16x32)
    - Next step is to flatten the 2D image with output shape 512

- This is followed by four fully connected layers of output shapes 100, 50, 10 and 1 respectively
- Output of this model is the predicted steering angle



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
max_pooling2d_1 (MaxPooling2	(None, 1, 16, 32)	0
dropout_1 (Dropout)	(None, 1, 16, 32)	0
flatten_1 (Flatten)	(None, 512)	0
dense_1 (Dense)	(None, 100)	51300
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11

Total params: 188,219
Trainable params: 188,219
Non-trainable params: 0

Figure 9: Final model architecture

- Tuning parameters
  - Number of epochs – 5
  - Optimizer – Adam optimizer
  - Validation test set size – 0.15 x total samples
  - Correction factor – 0.2
  - Generator batch size -32
  - Loss function – MSE

Output video is saved as run1.mp4