Use Deep Learning to Clone Driving Behavior

The goal of this project is to drive the car in simulator mode. First the car is run on training mode and the data is collected. Udacity has provided its data on

https://d17h27t6h515a5.cloudfront.net/topher/2016/December/584f6edd_data/data.zip

and this data is used in this project.

By using this data (After preprocessing) and is fed in to Convolutional Neural Network and get the model trained. After training the model, model is used to provide data to simulator (Steering angle) to run the car in the simulator. So Learning, Feed in to model, and by using model intelligence drive the car.

Environmental setup:

I had used a google cloud machine to TESLA K80 GPUs 2 in number

Installed the following drivers in windows server 2016

Name	Publisher	Installed On	Size	Version
Notepad++ (32-bit x86)	Notepad++ Team	6/28/2019	8.07 MB	7.7.1
NVIDIA CUDA Development 9.0	NVIDIA Corporation	6/25/2019	54.3 MB	9.0
NVIDIA CUDA Documentation 9.0	NVIDIA Corporation	6/25/2019	54.3 MB	9.0
NVIDIA CUDA Runtime 9.0	NVIDIA Corporation	6/25/2019	4.00 KB	9.0
NVIDIA CUDA Samples 9.0	NVIDIA Corporation	6/25/2019	319 MB	9.0
NVIDIA CUDA Visual Studio Integration 9.0	NVIDIA Corporation	6/25/2019	55.2 MB	9.0
NVIDIA Nsight Compute 2019.3.0	NVIDIA Corporation	6/26/2019	767 MB	19.3.0.0
NVIDIA Nsight Systems v2019.3.3	NVIDIA Corporation	6/26/2019	414 MB	19.3.3.3
NVIDIA Nsight Visual Studio Edition 2019.2.0.19109	NVIDIA Corporation	6/26/2019	1.79 GB	19.2.0.19109
NVIDIA PhysX System Software 9.19.0218	NVIDIA Corporation	6/26/2019	410 MB	9.19.0218
NVIDIA Tools Extension SDK (NVTX) - 64 bit	NVIDIA Corporation	6/25/2019	1.37 MB	1.00.00.00
Python 3.5.2 (Anaconda3 4.1.1 64-bit)	Continuum Analytics, Inc.	6/25/2019		4.1.1
Unity	Unity Technologies ApS	6/27/2019		2018.3.14f1
Unity Hub 2.0.2	Unity Technologies Inc.	6/27/2019	225 MB	2.0.2
Visual Studio Community 2017	Microsoft Corporation	6/24/2019	206 MB	15.9.28307.718
WinZip 23.0	Corel Corporation	6/28/2019	66.3 MB	23.0.13431

Environmental File(env_cloning.yml) has the following list

name: test

channels:

- conda-forge

- defaults

dependencies:

- asn1crypto=0.24.0=py35_3

- ca-certificates=2019.6.16=hecc5488_0

- cffi=1.11.5=py35hfa6e2cd_1

- cryptography=2.3.1=py35h74b6da3_0

- cryptography-vectors=2.3.1=py35_0
- decorator=4.4.0=py_0
- eventlet=0.23.0=py35_0
- ffmpeg=4.1.3=h6538335_0
- freetype=2.10.0=h5db478b_0
- greenlet=0.4.13=py35_0
- idna=2.7=py35_2
- imageio=2.3.0=py_1
- jpeg=9c=hfa6e2cd_1001
- libpng=1.6.37=h7602738_0
- libtiff=4.0.10=h36446d0_1001
- moviepy=0.2.3.5=py_0
- olefile=0.46=py_0
- openssl=1.0.2r=hfa6e2cd_0
- pillow=5.3.0=py35h9a613e6_0
- pycparser=2.19=py_0
- pyopenssl=18.0.0=py35_0
- python-engineio=3.0.0=py_0
- python-socketio=4.0.3=py_0
- six=1.11.0=py35_1
- tk=8.6.9=hfa6e2cd_1002
- tqdm=4.32.2=py_0
- zlib=1.2.11=h2fa13f4_1004
- blas=1.0=mkl
- certifi=2018.8.24=py35_1
- chardet=3.0.4=py35_1
- icc_rt=2019.0.0=h0cc432a_1
- intel-openmp=2019.4=245
- mkl=2018.0.3=1

- mkl_fft=1.0.6=py35hdbbee80_0
- mkl_random=1.0.1=py35h77b88f5_1
- numpy=1.15.2=py35ha559c80_0
- numpy-base=1.15.2=py35h8128ebf_0
- pandas=0.23.4=py35h830ac7b_0
- pip=9.0.1=py35_1
- pysocks=1.6.8=py35_0
- python=3.5.4=0
- python-dateutil=2.7.3=py35_0
- pytz=2019.1=py_0
- requests=2.19.1=py35_0
- setuptools=36.4.0=py35_1
- urllib3=1.23=py35_0
- vc=14=0
- vs2015_runtime=14.0.25420=0
- wheel=0.29.0=py35_0
- win_inet_pton=1.0.1=py35_1
- wincertstore=0.2=py35_0

How to run the files:

1. Open the command prompt
2. Activate the environment
3. Change the directories to where the python files reside
4. python model.py
5. After the model is generated, you can see model.h5 present in the same folder
6. Open the simulator in autonomous mode for first path
7. python drive.py model.h5 Run
8. python video.py Run

Required files:

Files submitted

- model.py consists of selected model
- drive.py
- model.h5
- Writeup report
- Run.mp4
- env_cloning.yml

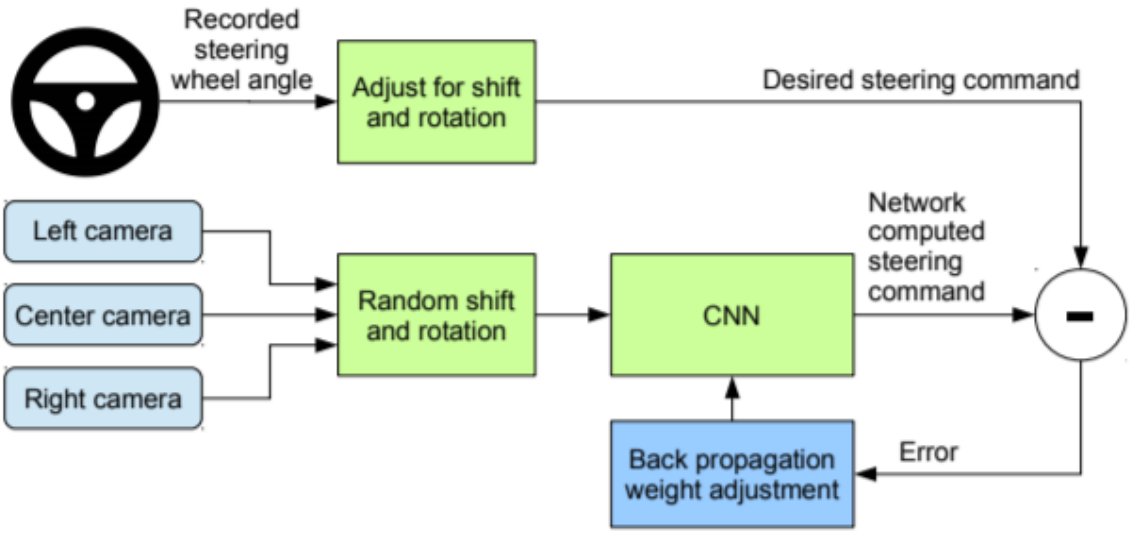
The code is compiling and running successfully in Google cloud virtual machine

The code in model.py downloads the training data from Udacity share and then extract relevant data from the driving log and images generated by the simulator

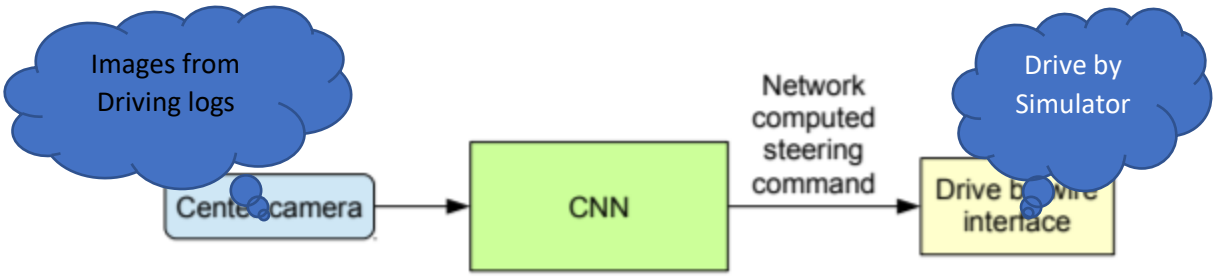
Model Architecture and Training Strategy:

The model I have chosen for training is well known NVIDIA for self-driving cars.

For their DAVE-2 system, NVDIA has three cameras mounted behind the windshield of the data acquisition car. Time stamped video from the cameras is captured simultaneously with the steering angle applied by human driver. I visualized the same scenario is being applied in our case where the Udacity simulator is trying to generate driving_log.csv with images and speed, throttle angle, etc

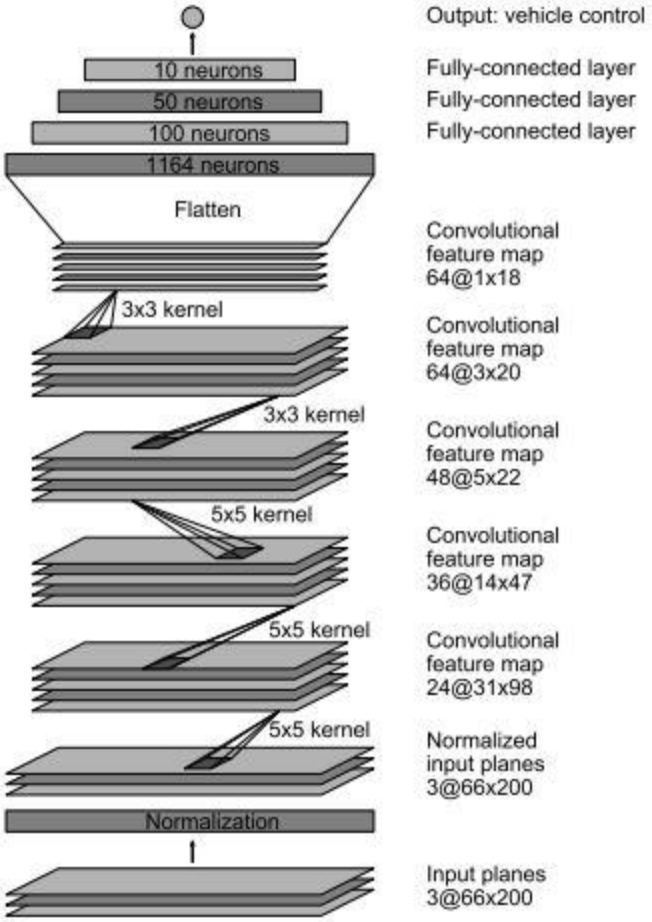


Once the CNN is trained well enough, the objective is to drive the Vehicle in simulator using Autonomous mode.



Model Architecture:

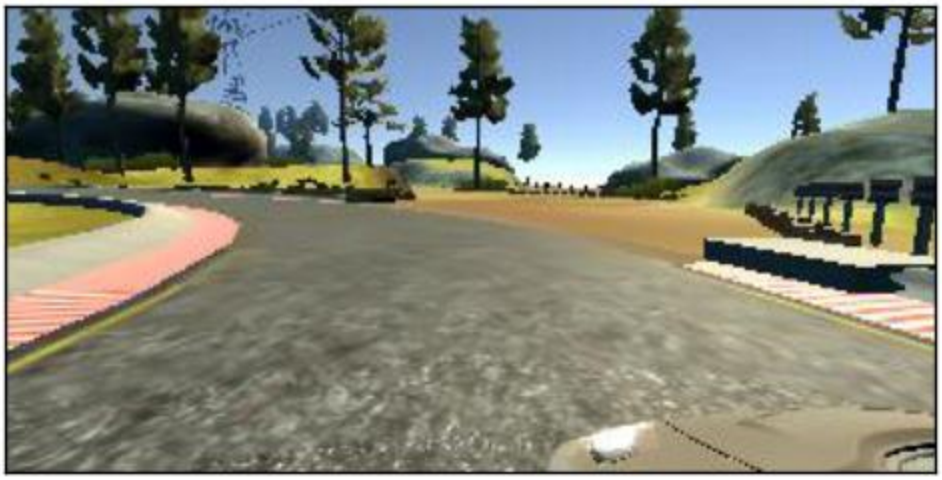
Our dashboard/training images is of size (160,320,3). The original model has different input shape of image but I am confident in pursuing this model and tried feeding the input image of different size.



Loading the data:

- I used the training data provided by Udacity.
- Used OpenCV to load the images, in BGR format
- In drive.py it is being loaded in other format (RGB), so I am forced to convert the image in to RGB format
- Since Steering angle is used, it is desirable to use a correction factor. So tried using a correction factor of 0.2 for left and right images and it worked
- For right images I have decreased the correction factor by 0.2 and for left images I have increased the correction factor by 0.2

A left image is as follows



A flipped image is shown as follows:

Preprocessing:

During preprocessing I decided to flip and shuffle the images.

Steering angle of the flipped image is obtained by multiplying -1

```
                    if(i==0):
                        angles.append(center_angle*-1)
                    elif(i==1):
                        angles.append((center_angle+0.2)*-1)
                    elif(i==2):
                        angles.append((center_angle-0.2)*-1)
```

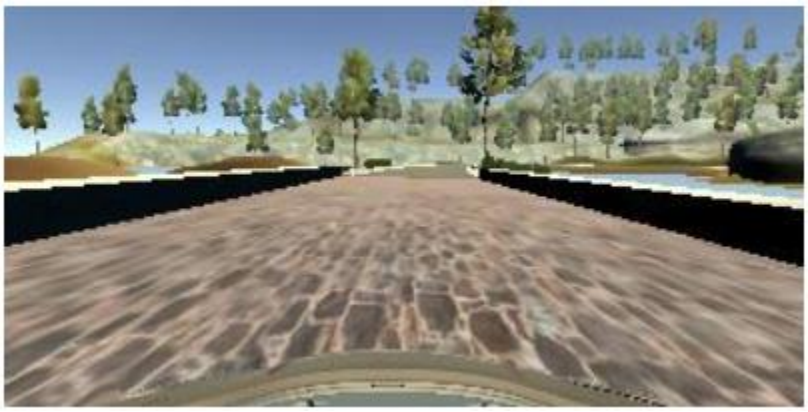
Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
conv2d_1 (Conv2D)	(None, 31, 158, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 77, 36)	21636
conv2d_3 (Conv2D)	(None, 5, 37, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 35, 64)	27712
conv2d_5 (Conv2D)	(None, 1, 33, 64)	36928
flatten_1 (Flatten)	(None, 2112)	0
dense_1 (Dense)	(None, 100)	211300
activation_1 (Activation)	(None, 100)	0
dropout_1 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 50)	5050
activation_2 (Activation)	(None, 50)	0
dense_3 (Dense)	(None, 10)	510
activation_3 (Activation)	(None, 10)	0
dense_4 (Dense)	(None, 1)	11

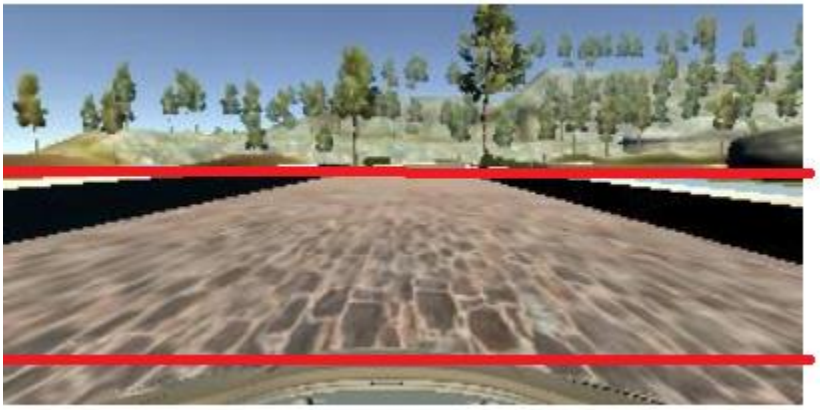
Total params: 348,219
Trainable params: 348,219
Non-trainable params: 0

1. Normalization of images is my first step
2. Second step is to crop the unwanted portions of image

model.add(Cropping2D(cropping=((70,25),(0,0))))

What is this 70 and 25? Those are the indexes to filter from top and bottom of the image to extract only the images which are of useful to the model.





3. First convolutional layer with filter depth as 24 and filter size as (5,5) with (2,2) stride followed by ELU activation function
4. Second convolutional layer with filter depth as 36 and filter size as (5,5) with (2,2) stride followed by ELU Activation function
5. The Third Convolutional layer has filter dept of 48 and filter size as (5,5) with (2,2) stride followed by ELU activation function
6. This is followed by two convolutional layers with filter depth as 6 and filter size as (3,3) and (1,1) stride followed by ELU activation function
7. Follow this by flatten the output from 2D to sides
8. Next apply fully connected layer with 100 outputs
9. Now it's time to apply dropouts with 0.25 to fight the overfitting
10. Now reduce the outputs to 50 with fully connected layer
11. Next the outputs are reduced to 10 and further to 1 by using fully connected layers

The last layer need not be activated.

Training and Validation Set

- Udacity training data consists of 9 laps of track1 and recovery data
- I am quite satisfied with it
- The data set was divided in to Training and Validation Sets using the sklearn.utils.shuffle(X_train, y_train) -> sklearn preprocessing library
- Used generator to avoid huge memory consumption

Tuning of the parameters:

Used the following parameters and tuned them for many iterations and finally arrived in the following which seems optimal

Number of epochs = 5

Learning rate = 0.001 (default)

Validation data split = 0.15

Correction factor = 0.2

Loss function used is Mean Square Error (MSE) along with Adam Optimizer using the function

```
model.compile(loss='mse',optimizer='adam')
```

Output video is archived in the following link:

https://youtu.be/ywduGOOiyVY