# Behavioral Cloning

1. **Purpose :**
   The purpose of project is to use Deep Learning to clone the Driving behavior on the track.

2. **Project Inputs :**
   a. A Github repo with starter code(Workspace can also be used)
   *b.* Files in the project folder includes –
      i. drive.py
      ii. model.py
      iii. video.py
      iv. Pre-recorded track images
      v. Driving log excel
   c. A Simulator which provides facility to drive the Car on the test track and record the images to support Deep learning.


*Figure 1: Simulator*

## 3. Project Outputs :

a. To satisfy project rubric following artifacts are expected –

i. Modified model.py
ii. model.h5
iii. drive.py
iv. video.mp4
v. writeup_report

b. To satisfy project rubric following specifications to be satisfied –

i. Functional code - The generated model shall successfully operate the simulation.
ii. Usable and Readable code - The code in model.py uses a Python generator, if needed, to generate data for training rather than storing the training data in memory. The model.py code is clearly organized and comments are included where needed.
iii. Appropriate model architecture - The neural network uses convolution layers with appropriate filter sizes. Layers exist to introduce nonlinearity into the model. The data is normalized in the model.
iv. Attempt to reduce overfitting of the model - Train/validation/test splits have been used, and the model uses dropout layers or other methods to reduce overfitting.
v. The model parameters been tuned appropriately - Learning rate parameters are chosen with explanation, or an Adam optimizer is used.
vi. Training data chosen appropriately - Training data has been chosen to induce the desired behavior in the simulation (i.e. keeping the car on the track).
vii. The solution design documented - The README thoroughly discusses the approach taken for deriving and designing a model architecture fit for solving the given problem.
viii. The model architecture documented - The README provides sufficient details of the characteristics and qualities of the architecture, such as the type of model

used, the number of layers, the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged.

ix. The creation of the training dataset and training process documented - The README describes how the model was trained and what the characteristics of the dataset are. Information such as how the dataset was generated and examples of images from the dataset must be included.

x. The car able to navigate correctly on test data - No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

4. **Description :**

a. Submitted Files – Following files are submitted in project repository

i. model.py file
ii. drive.py,
iii. model.h5
iv. writeup report : Project 4 – Behavioral Cloning
v. video.mp4

b. Functional Code – The model.py and drive.py files include the functional code modified to control the car.

c. Code Usable and Readable – The comments are included to define the purpose of section of code along with End of code section statements. The data generators are used to generate the training data.

d. Model Architecture – I used the model NVIDIA model architecture as mentioned in the Udacity tutorials.

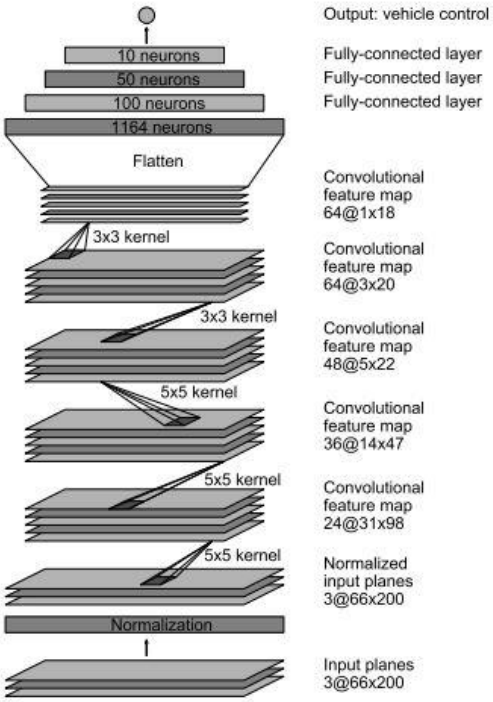


*Figure 2: NVIDIA Model Architecture*

e. Training Data –

i. Data set – I utilized the data set provided by Udacity along with the Driving log excel file with steering angle information.
ii. Loading Data – I used csv.reader function to load the images.
iii. Steering Angle – The steering angle respective to each image is appended to an array.
iv. To access the image files from the workspace was quiet tricky for me. Faced some issues because I was not putting the file path correctly. After referring to the github update at following link I was able to resolve the issue.

https://github.com/udacity/CarND-ehavioral-Cloning-P3/issues/19 Paths in example data file don't match when generating your own. #19

f. Preprocessing –

i. Crop images – The images are randomly cropped to remove the un useful portion like sky and bonnet. The images are then resized to 64x64 pixel size.
ii. Flip – For data augmentation I used flip function to flip the image as mentioned in the Udacity videos. This process helped to create more images for training from the available image data set.
iii. For other data augmentation I referred available resources on web and implemented random image shear and brightness adjustments. (Note: Image is only for example not with actual crop.)

*Figure 3: Preprocessed Image*

g. Training and Validation split –

i. The Udacity Dataset and found out that it contains 9 laps of track 1 with recovery data.
ii. The Dataset is divided into training and validation set using sklearn preprocessing library.
iii. I decided to keep 10% of the data in Validation Set and remaining in Training Set.
iv. I used generator to generate the data to avoid loading all the images in the memory and instead generate it at the run time in batches of 32. Even Augmented images are generated inside the generators.

h. Parameter Tuning –

i. Optimizer Used- Adam
ii. No of epochs= 25
iii. Learning Rate- Default 0.001
iv. Generator batch size= 32
v. Loss Function Used- MSE(Mean Squared Error as it is efficient for regression problem).

i. Final Model – Following is the final model architecture

```
## Model Implementation
model = Sequential()
model.add(Lambda(lambda x:x/127.5 -1., input_shape = (rows,cols,3)))  #Lambda layer for normalization
model.add(Conv2D(24,(5,5),input_shape=(rows,cols,3),padding = 'valid' ))
model.add(ELU())
model.add(Conv2D(36,(5,5), padding ='valid',strides =(2,2)))
model.add(ELU())
model.add(Conv2D(48,(5,5), padding ='valid',strides=(2,2)))
model.add(ELU())
model.add(Conv2D(64,(3,3), padding ='valid',strides=(1,1)))
model.add(ELU())
model.add(Conv2D(64,(3,3), padding ='valid',strides=(1,1)))
model.add(ELU()) #Change -1
model.add(Flatten())
model.add(Dropout(0.5)) #Change -2 : Add dropout
model.add(Dense(1164,kernel_initializer ='he_normal'))
model.add(ELU())
model.add(Dense(100,kernel_initializer='he_normal'))
model.add(ELU())
model.add(Dense(50,kernel_initializer ='he_normal'))
model.add(ELU())
model.add(Dropout(0.5)) #Change -3 : Add dropout
model.add(Dense(10,kernel_initializer ='he_normal'))
model.add(ELU())
model.add(Dense(1,name='output',kernel_initializer = 'he_normal'))

model.summary()
```

*Figure 4: Model Architecture*

j. Solution Design –
    i. The first step is to apply normalization to all the images. The Lambda layers in Keras are used for normalization as explained in tutorials. A lambda layer is a convenient way to parallelize image normalization. The lambda layer will also ensure that the model will normalize input images when making predictions in drive.py.
    ii. The Lambda layer expects the input image of the size 64x64 pixels.
    iii. Next Step is to define the first convolutional layer with filter depth as 24 and filter size as (5,5) followed by ELU activation function
    iv. Moving on to the second convolutional layer with filter depth as 36 and filter size as (5,5) with (2,2) stride followed by ELU activation function
    v. The third convolutional layer with filter depth as 48 and filter size as (5,5) with (2,2) stride followed by ELU activation function
    vi. Next we define two convolutional layer with filter depth as 64 and filter size as (3,3) and (1,1) stride followed by ELU activation function
    vii. Next step is to flatten the output from 2D to side by side
    viii. Then we have used a Dropout layer to reduce overfitting.
    ix. Here we apply first fully connected layer with 1164 outputs
    x. Then we apply second fully connected layer with 100 outputs
    xi. Next we introduce third fully connected layer with 50 outputs
    xii. Again a second Dropout layer is implemented.
    xiii. Then comes a fourth connected layer with 10 outputs
    xiv. And finally the layer with one output.

k. Desired Behavior –

After trying multiple times with different augmentation techniques, training the model multiple times with different number of epochs. Finally I got the expected results available in the workspace repository and also at following youtube link.

https://www.youtube.com/watch?v=-56KmUzXmpc

l. Side Notes –

i. Due to lag between code execution in workspace and the Simulator, the motion is little bit jerky.

ii. The training process takes lot of time and consumes lot of GPU time, so fine tuning becomes difficult for students who are first time executing such project with lot of images.

iii. Sometimes workspace exited while training was going on due to no action by user.

m. Observed Errors during code compilation -

i. Image size mismatch between cropped image and Lambda layer

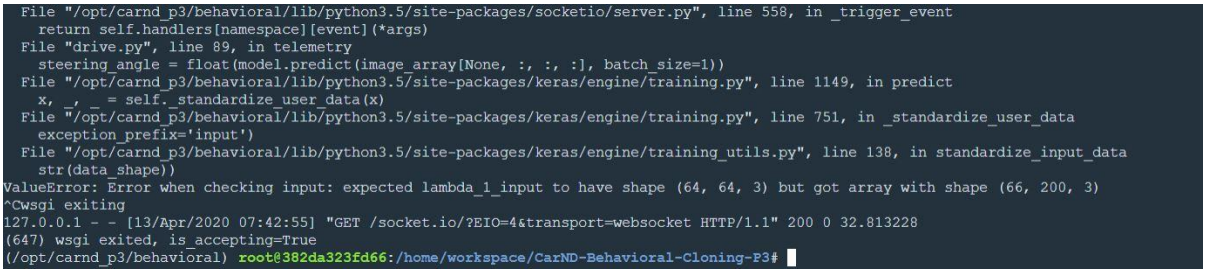


*Figure 5: Image Size Mismatch*

ii. Memory Error due to large array size

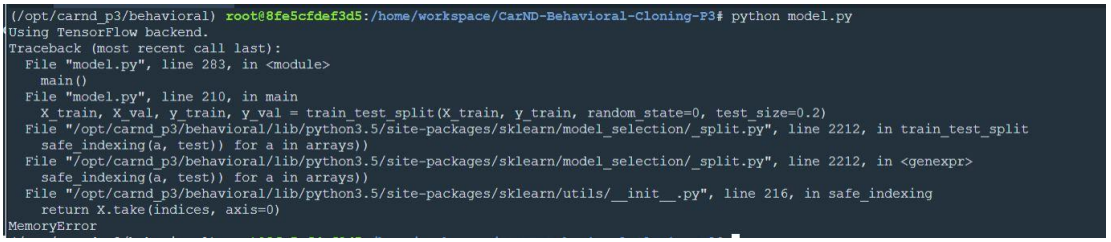


*Figure 6: Memory Error*

## iii. Tuple Indices

```
Traceback (most recent call last):
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/engineio/server.py", line 505, in _trigger_event
    return self.handlers[event](*args)
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/socketio/server.py", line 590, in _handle_eio_message
    self._handle_event(sid, pkt.namespace, pkt.id, pkt.data)
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/socketio/server.py", line 526, in _handle_event
    self._handle_event_internal(self, sid, data, namespace, id)
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/socketio/server.py", line 529, in _handle_event_internal
    r = server._trigger_event(data[0], namespace, sid, *data[1:])
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/socketio/server.py", line 558, in _trigger_event
    return self.handlers[namespace][event](*args)
  File "drive.py", line 101, in telemetry
    transformed_image_array = image_array[None, :, :, :]
TypeError: tuple indices must be integers or slices, not tuple
```

*Figure 7: Tuple Indices*

## iv. Video file creation issue

```
    return imageio_ffmpeg.get_ffmpeg_exe()
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/imageio_ffmpeg/_utils.py", line 50, in get_ffmpeg_exe
    "No ffmpeg exe could be found. Install ffmpeg on your system, "
RuntimeError: No ffmpeg exe could be found. Install ffmpeg on your system, or set the IMAGEIO_FFMPEG_EXE environment variable.
(/opt/carnd_p3/behavioral) root@11f6bf672073:/home/workspace/CarND-Behavioral-Cloning-P3# python video.py run1
Traceback (most recent call last):
  File "video.py", line 1, in <module>
    from moviepy.editor import ImageSequenceClip
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/moviepy/editor.py", line 33, in <module>
    from .video.io.VideoFileClip import VideoFileClip
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/moviepy/video/io/VideoFileClip.py", line 3, in <module>
    from moviepy.video.VideoClip import VideoClip
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/moviepy/video/VideoClip.py", line 18, in <module>
    from ..config import get_setting
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/moviepy/config.py", line 35, in <module>
    FFMPEG_BINARY = get_exe()
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/imageio/plugins/ffmpeg.py", line 51, in get_exe
    return imageio_ffmpeg.get_ffmpeg_exe()
  File "/opt/carnd_p3/behavioral/lib/python3.5/site-packages/imageio_ffmpeg/_utils.py", line 50, in get_ffmpeg_exe
    "No ffmpeg exe could be found. Install ffmpeg on your system, "
RuntimeError: No ffmpeg exe could be found. Install ffmpeg on your system, or set the IMAGEIO_FFMPEG_EXE environment variable.
(/opt/carnd_p3/behavioral) root@11f6bf672073:/home/workspace/CarND-Behavioral-Cloning-P3#
```

*Figure 8: Video File Creation Error*