Predicting Steering Angles using Deep Learning: Behaviour Cloning

Harveen Singh Chadha
Mar 3, 2018 · 10 min read

I am into my first term of Udacity's Self Driving Car Nanodegree and I want to share my experiences regarding one of my recent projects. The complete code can be found here

Figure 1. Driving Car on Simulator Provided By Udacity

Introduction

is this even possible ? Sounds Weird, Right? So Let's Get Started.

Collecting Data

First things first, it is not magic but it really feels like magic. With just a bunch of Python libraries, some lines of Code and **huge amount of Data** we can teach a car to drive itself. Remember I have emphasized on huge bunch of Data because Data plays the most important role in this project and it has to be in Abundance as in any other Deep Learning problem.

So what type of data do we need to collect? We will capturing three images from a dashboard camera placed at slightly different positions center, left and right in addition to steering angle, throttle, brake and speed. So basically we have a 8 tuple (center image, left image, right image, steering, throttle, brake, speed).

The next question is how? Well, that is pretty simple, thanks to Udacity for providing a simulator which does the required job here. The simulator is built on unity engine and all you need is to just drive around the track in the training mode. If you didn't play video games in your childhood then you are going to have a tough time collecting good training data. Make sure you record the training data once you start driving around the track.

The next point is about the **quality** of training data to collect. If you do not drive well enough, then you car won't either in the autonomous mode. So make sure to keep your car in the center as much as possible. Your goal is to collect maximum different data that is possible. First drive 3-5 laps and record data, then drive 3-5 reverse laps on the same track.

Is that enough? **No!** If you just teach your car to drive in the center of the track then it will never know what to do if it comes on the side path, so you need to collect recovery data for that as well. So if a car goes off track, then it should try to come back to center as quickly as possible. Once you are done you will be able to see the captured images and the corresponding data.csv file containing the mapping of all the images and tuple defined above.

center	left	right	steering	throttle	brake	speed
IMG/center_2016_12_01_13_30_48_287.jpg	IMG/left_2016_12_01_13_30_48_287.jpg	IMG/right_2016_12_01_13_30_48_287.jpg	0	0	0	22.14829

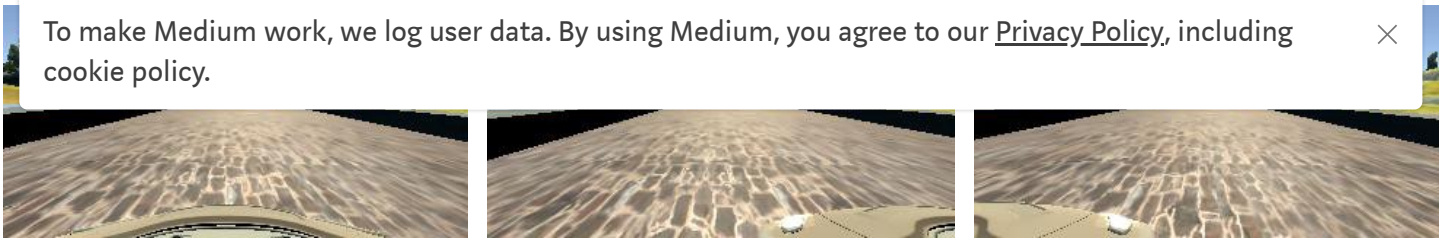


Figure 2. L to R- Center Image, Left Image, Right Image

Data Preprocessing

In our task to drive a car itself, our main focus is just to steer the car around the curves. So basically we will be building a model that learns itself to steer to car according to the curvature of the road. So the only parameter that is important to us is the Steering Angle. The first step is to visualize the data set.

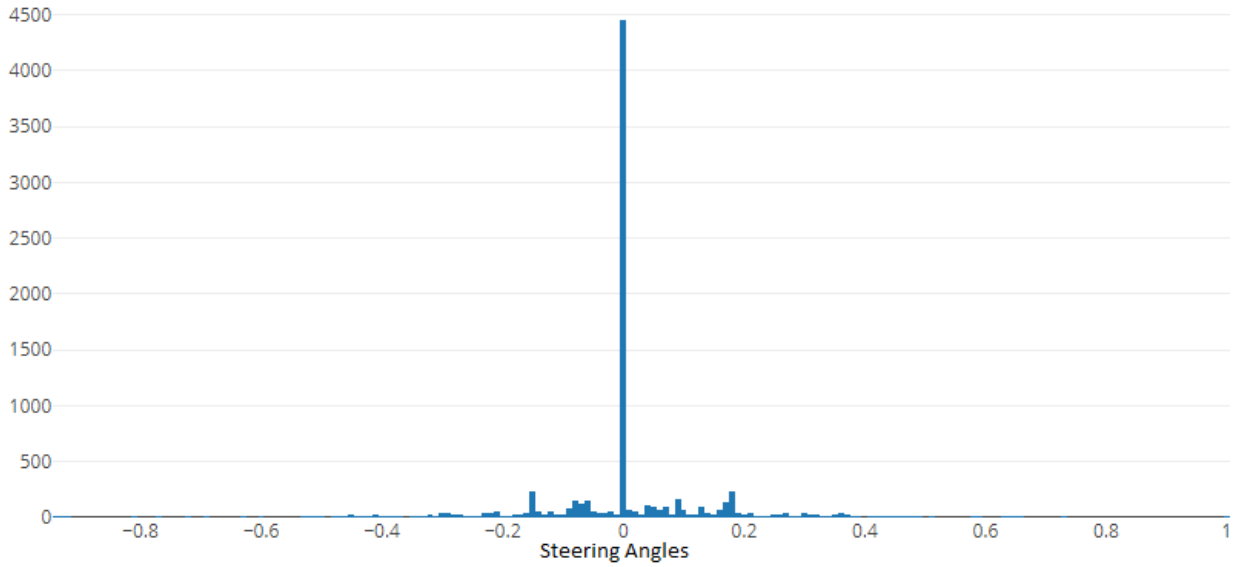


Figure 3. Data Visualization

As you can observe, we have too much data for the images with zero steering angle, so we need to create more data for other steering angles. And this can be done by Data Augmentation. But please make a note, we cannot apply every type of Data Augmentation.

When we look at the data, we observe that we have 3 images corresponding to one steering angle and that is so because the steering angle is supposed to be captured from the center position. *So does that mean our right and left images also have the same steering angle?* **No!**

subtract in the images taken from right dashboard camera in order to keep our vehicle in the center. So, lets suppose our steering correction angle is 0.2. So in case of left images

> *steering angle = steering angle + 0.2*

and in case of right images

> *steering angle= steering angle - 0.2*

Flipping: Perfect! That was the first step of data preprocessing. Moving on, what else can be done to generate more data. *Well how about a flip*? Yes, We can flip images horizontally and negate the corresponding steering angle to get an altogether different record. Awesome! So that means by 1 record in csv we are able to generate 6 different records. One for center, one for left , one for right, one for center flipped, one for left flipped, one for right flipped.

So how does this helps? Well, In the simulator there are a lot of left turns, so we don't want that our model to know only how to turn to left so we are flipping to generalize so that it can make a right turn too.

Normalizing: Anything else? Yes of Course! How can we forget to Normalize the model? Well it is a perfect step to normalize the data before processing so we normalize all the images mean centered. Since we have a lot of images so it is not a good idea to apply it to all images in one go, since you may go out of memory we will later add this step in keras model and will give keras the opportunity to do it for us.

Cropping: Oh god! how can we not crop the image! Well if you observed the image, the trees and sky do not play any important role in our image, instead they are more likely to distract our model. So we don't need them. Also if you observed, car dash is visible in the bottom of the image so we crop that too. We avoid cropping anything horizontally because we do not want to loose information of the curvature of the road. I am still not cropping images initially, we will give keras this golden opportunity to crop that for us.

Shuffling: Of course, we can shuffle the data set so that the order in which these images are processed by the CNN does not matters.

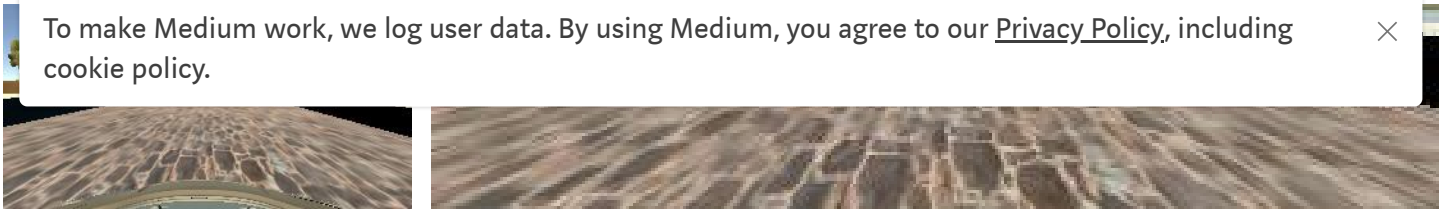


Figure 4. L to R- Original Image and Cropped Image

There are many more Augmentation techniques that can be applied like varying lighting conditions and brightness, a little bit of transformations as well but let's be happy with the current data set and feed this into our model and see what happens.

> ***Important + Bonus: Please check the Automould Library by one of my colleagues. This library is for Data Augmentation in real world.***

Model Architecture

As Udacity provided insight about the NVIDIA self driving car model, I decided to implement that initially.

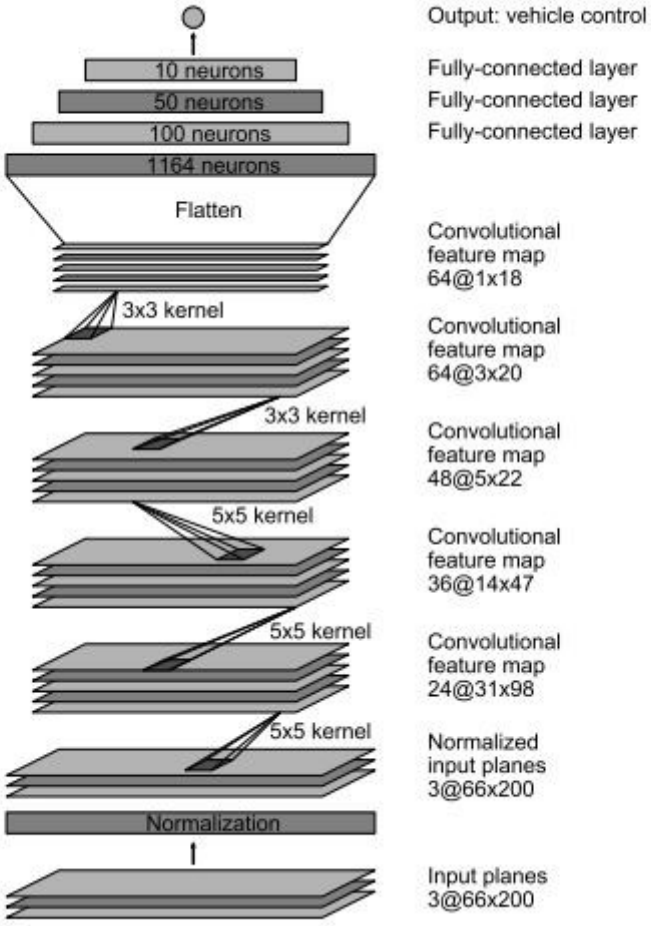


Figure 5. NVIDIA Comma.ai model for Self Driving Car

fully connected layers and change our inputs to each layer.

Generators

Important Note- If we have 10,000 samples in data.csv then we will have 10,000 * 6 images in our data set and it is not feasible to load these many images in one shot, since it will consume all the memory. So we have to take the help of generators that generate data in batches and use "yield" keyword, which keeps on appending images one after the other. Also, it is recommended to perform data augmentation inside generators.

```
def generator(samples, batch_size=32):
    num_samples = len(samples)

    while 1:
        shuffle(samples) #shuffling the total images
        for offset in range(0, num_samples, batch_size):

            batch_samples = samples[offset:offset+batch_size]

            images = []
            angles = []
            for batch_sample in batch_samples:
                    for i in range(0,3):
                      #we are taking 3 images, first one is center,
second is left and third is right

                        name =
'./data/data/IMG/'+batch_sample[i].split('/')[-1]
                        center_image =
cv2.cvtColor(cv2.imread(name), cv2.COLOR_BGR2RGB) #since CV2 reads
an image in BGR we need to convert it to RGB since in drive.py it is
RGB
                        center_angle = float(batch_sample[3])
#getting the steering angle measurement
                        images.append(center_image)

                        # introducing correction for left and right
images
                        # if image is in left we increase the
steering angle by 0.2
                        # if image is in right we decrease the
steering angle by 0.2

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

```
negate the measurement

                        images.append(cv2.flip(center_image,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)
                        #here we got 6 images from one image


            X_train = np.array(images)
            y_train = np.array(angles)

            yield sklearn.utils.shuffle(X_train, y_train) #here we
do not hold the values of X_train and y_train instead we yield the
values which means we hold until the generator is running
```

To test our model how it is performing, lets divide our dataset into two parts, training set and validation set. This is also helpful to judge the performance as well to combact overfitting.

```
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

train_samples, validation_samples =
train_test_split(samples,test_size=0.15)
#simply splitting the dataset to train and validation set usking
sklearn. .15 indicates 15% of the dataset is validation set
```

Since the starting of article, Keras is waiting for its turn so let's call it. Some more code coming your way! Let's use Keras with Tensorflow as backend. I am using keras 1.2.1 so some of the calling notations may appear as obsolete.

```
from keras.models import Sequential
from keras.layers.core import Dense, Flatten, Activation, Dropout
from keras.layers.convolutional import Convolution2D
from keras.layers import Lambda, Cropping2D

model = Sequential()

# Preprocess incoming data, centered around zero with small standard
deviation
model.add(Lambda(lambda x: (x / 255.0) - 0.5, input_shape=
```

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

#layer 1- Convolution, no of filters- 24, filter size= 5x5, stride= 2x2
model.add(Convolution2D(24,5,5,subsample=(2,2)))
model.add(Activation('elu'))

#layer 2- Convolution, no of filters- 36, filter size= 5x5, stride= 2x2
model.add(Convolution2D(36,5,5,subsample=(2,2)))
model.add(Activation('elu'))

#layer 3- Convolution, no of filters- 48, filter size= 5x5, stride= 2x2
model.add(Convolution2D(48,5,5,subsample=(2,2)))
model.add(Activation('elu'))

#layer 4- Convolution, no of filters- 64, filter size= 3x3, stride= 1x1
model.add(Convolution2D(64,3,3))
model.add(Activation('elu'))

#layer 5- Convolution, no of filters- 64, filter size= 3x3, stride= 1x1
model.add(Convolution2D(64,3,3))
model.add(Activation('elu'))

#flatten image from 2D to side by side
model.add(Flatten())

#layer 6- fully connected layer 1
model.add(Dense(100))
model.add(Activation('elu'))

#Adding a dropout layer to avoid overfitting. Here we are have given the dropout rate as 25% after first fully connected layer
model.add(Dropout(0.25))

#layer 7- fully connected layer 1
model.add(Dense(50))
model.add(Activation('elu'))


#layer 8- fully connected layer 1
model.add(Dense(10))
model.add(Activation('elu'))

#layer 9- fully connected layer 1
model.add(Dense(1)) #here the final layer will contain one value as this is a regression problem and not classification


# the output is the steering angle
# using mean squared error loss function is the right choice for this regression problem
```

```
#fit generator is used here as the number of images are generated by
the generator
# no of epochs : 5

model.fit_generator(train_generator, samples_per_epoch=
len(train_samples), validation_data=validation_generator,
nb_val_samples=len(validation_samples), nb_epoch=5, verbose=1)

#saving model
model.save('model.h5')

print('Done! Model Saved!')
```

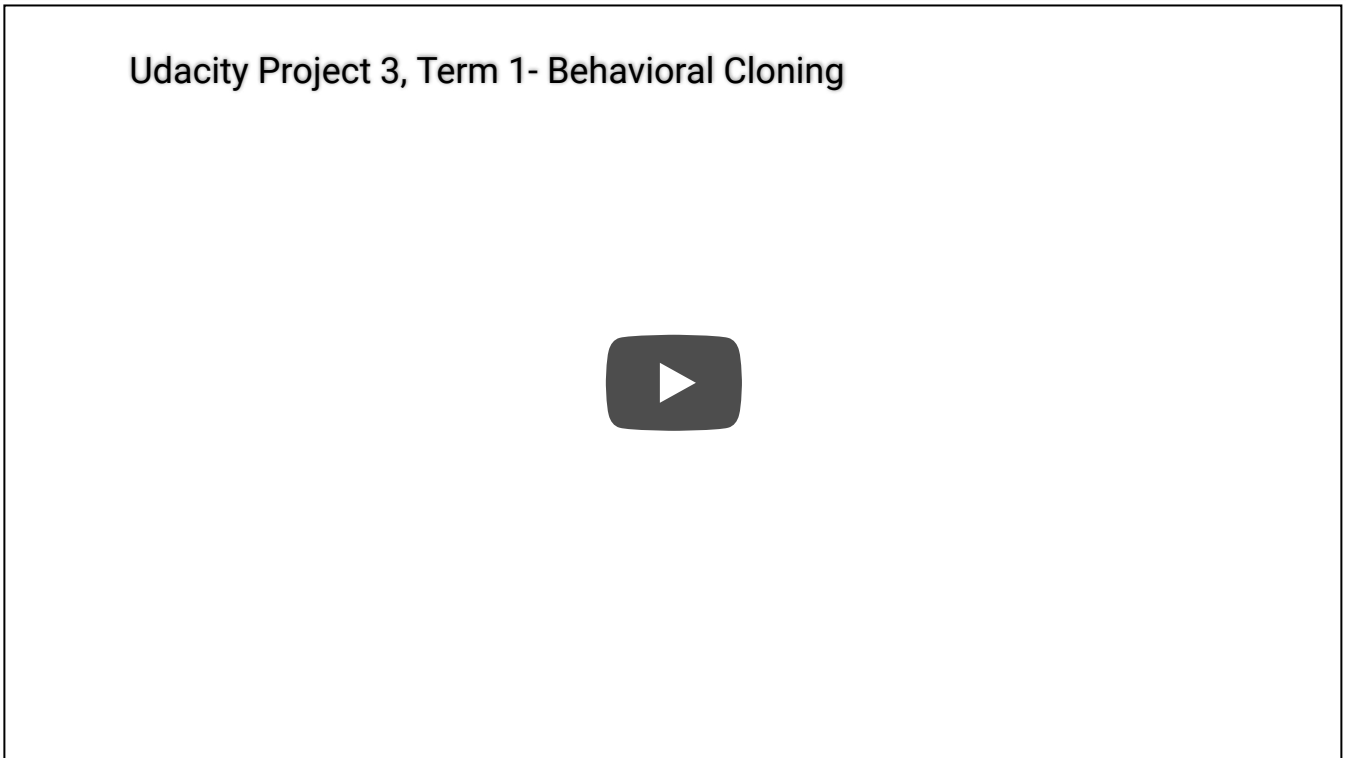
There are a number of parameters that were our choice, so we can definitely play with them to get different results. Some of the parameters I am listing here-

- No of epochs= 5
- Optimizer Used- Adam
- Learning Rate- Default 0.001
- Validation Data split- 0.15
- Generator batch size= 32
- Correction factor- 0.2
- Loss Function Used- MSE(Mean Squared Error as it is efficient for regression problem).
- Activation Function- ELU
- Dropout- 20% (after 1st activation layer to combact overfitting)

So is that it? Yes! I would say. Now how do we test this model in the simulator? Well, some cool people at udacity have already provided a drive.py file that connects your model to simulator with the following command

> *python drive.py model.h5 ./OutputShots*

Run the following command and select autonomous mode to check the performance of our car and whether or not it meets our expectations to drive itself. I captured a video



Well, this project was really cool. We learnt a lot of stuff, generators, data preprocessing techniques in keras etc. But the most important thing learnt is the **Importance of GOOD training data.** Things at a point do depend on the model architecture but the training data is the ultimate decider.

> ***Important + Bonus:*** *I have read blog posts where people were able to drive even with one convolutional layer and 2 fully connected layers, that shows the strength of training data.* ***Please see this blog post from one of my colleagues who talks whether we require a complex structure or not!*** *But importantly we have to make sure our model is not overfitting by testing the model on other tracks as well. Our car may not drive the full lap in the other track but we have to make sure our model is able to generalize.*

In case you have any suggestions to share, please leave a note or drop me a message at LinkedIn. Also full project is present on the github here.

Special thanks to ujjwal saxena and Prathmesh Dali.

Email

Sign up

I agree to leave Becominghuman.ai and submit this information, which will be collected and used according to Upscribe's privacy policy.