PROJECT 3: BEHAVIORAL CLONING

This project includes the following files:

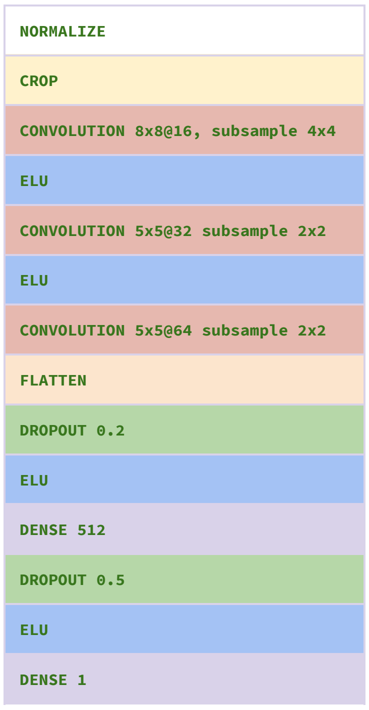
`video.py , drive.py`	`script to drive the car, and record the video`
`data/driving_log.csv`	`CSV of all the data samples`
`run1.mp4, run2.mp4`	`videos recording of your vehicle driving autonomously around the track for at least one full lap`
`model.h5, model.json`	`a trained Keras model`
`model.py`	`script used to create and train the model`
`writeup_report.pdf`	`This file which explains structure of network and training approach`

THE MODEL

The figure below summarizes the model architecture that I have employed. This architecture is modelled after comma.ai's research.

- https://arxiv.org/pdf/1608.01230.pdf
- https://github.com/commaai/research/blob/master/train_steering_model.py)

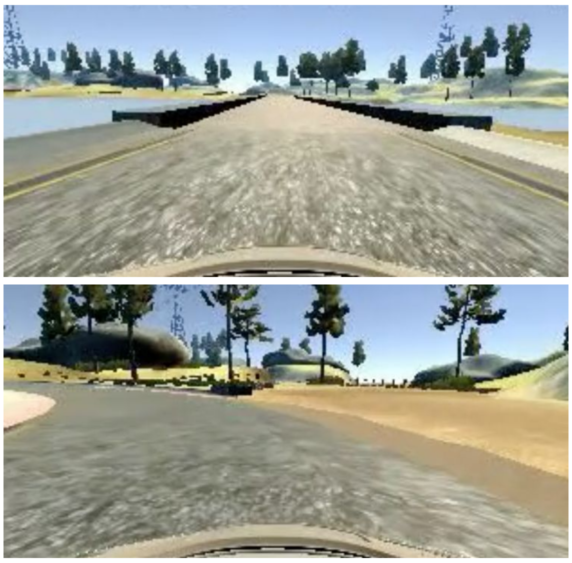
Before I decided to employ this architecture, I tried using the recommended NVIDIA model, but I was always running out of memory using my AWS server (g2.2large) - https://arxiv.org/pdf/1604.07316v1.pdf



NORMALIZE
CROP
CONVOLUTION 8x8@16, subsample 4x4
ELU
CONVOLUTION 5x5@32 subsample 2x2
ELU
CONVOLUTION 5x5@64 subsample 2x2
FLATTEN
DROPOUT 0.2
ELU
DENSE 512
DROPOUT 0.5
ELU
DENSE 1

```
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_1 (Lambda)                (None, 160, 320, 3)   0           lambda_input_1[0][0]
____________________________________________________________________________________________________
cropping2d_1 (Cropping2D)        (None, 65, 320, 3)    0           lambda_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 17, 80, 16)    3088        cropping2d_1[0][0]
____________________________________________________________________________________________________
elu_1 (ELU)                      (None, 17, 80, 16)    0           convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 9, 40, 32)     12832       elu_1[0][0]
____________________________________________________________________________________________________
elu_2 (ELU)                      (None, 9, 40, 32)     0           convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 5, 20, 64)     51264       elu_2[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 6400)          0           convolution2d_3[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 6400)          0           flatten_1[0][0]
____________________________________________________________________________________________________
elu_3 (ELU)                      (None, 6400)          0           dropout_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 512)           3277312     elu_3[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 512)           0           dense_1[0][0]
____________________________________________________________________________________________________
elu_4 (ELU)                      (None, 512)           0           dropout_2[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 1)             513         elu_4[0][0]
====================================================================================================
Total Trainable params: 3,345,009
```

The first step is normalizing the dataset (x: x/127.5 - 1.) by using the Lambda function as prescribed by the paper. I cropped the top 75 pixels and the bottom 25 pixels as these pixels do not have useful information. We are not interested in the sky or the hood of the car. We have cropped within the model, as it is known that this function is relatively fast, because the model is parallelized on the GPU.



```
model = Sequential()

model.add(Lambda(lambda x: (x / 127.5) - 1., input_shape = (160, 320, 3)))
model.add(Cropping2D(cropping=((70, 25), (0, 0)), input_shape = (160, 320, 3)))

model.add(Convolution2D(16, 8, 8, subsample=(4, 4), border_mode="same"))
model.add(ELU())

model.add(Convolution2D(32, 5, 5, subsample=(2, 2), border_mode="same"))
model.add(ELU())

model.add(Convolution2D(64, 5, 5, subsample=(2, 2), border_mode="same"))

model.add(Flatten())
model.add(Dropout(.2))
model.add(ELU())

model.add(Dense(512))
model.add(Dropout(.5))
model.add(ELU())

model.add(Dense(1))

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

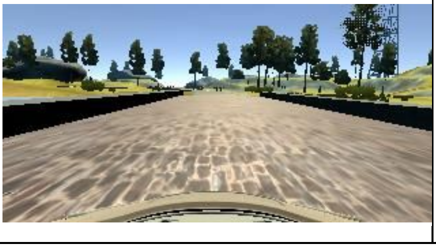
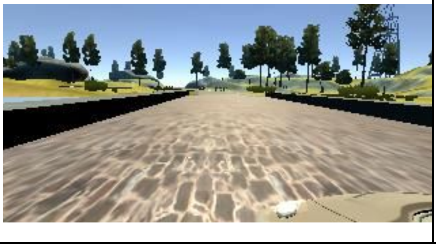
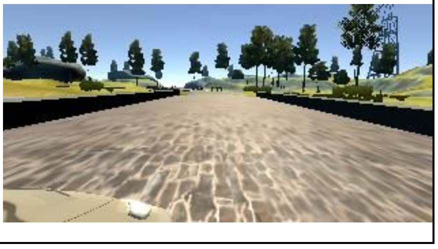
As you can infer from the code above, the cropping is followed by three convolutions and two fully connected layers. ELU (Exponential Linear Units) are introduced as activates within network which are said to speed up learning and lead to higher accuracies (https://arxiv.org/pdf/1511.07289v1.pdf). Subsampling was also used in each convolutional layer This architecture also employs aggressive dropout probabilities (0.2 and 0.5) to reduce overfitting. The model used an adam optimizer, so the learning rate was not tuned manually. The loss function used for optimization is the mean square error function.

TRAINING AND DATA AUGMENTATION STRATEGY

We have used a batch size of 64 and an epoch of 10 is used for training the data. Having a higher batch size could make the server run out of memory. A higher epoch seems to not make a difference with the loss function output, and make it prone to overfitting. The aggressive dropouts (0.2 and 0.5) are also employed to combat overfitting.

```
Epoch 1/10 6830/6830 [==============================] - 16s - loss: 0.2282 - val_loss: 0.0472
Epoch 2/10 6830/6830 [==============================] - 14s - loss: 0.0509 - val_loss: 0.0377
Epoch 3/10 6830/6830 [==============================] - 14s - loss: 0.0432 - val_loss: 0.0359
Epoch 4/10 6830/6830 [==============================] - 14s - loss: 0.0382 - val_loss: 0.0349
Epoch 5/10 6830/6830 [==============================] - 14s - loss: 0.0340 - val_loss: 0.0294
Epoch 6/10 6830/6830 [==============================] - 15s - loss: 0.0327 - val_loss: 0.0275
Epoch 7/10 6830/6830 [==============================] - 14s - loss: 0.0315 - val_loss: 0.0313
Epoch 8/10 6830/6830 [==============================] - 14s - loss: 0.0297 - val_loss: 0.0298
Epoch 9/10 6830/6830 [==============================] - 14s - loss: 0.0280 - val_loss: 0.0329
Epoch 10/10 6830/6830 [==============================] - 14s - loss: 0.0284 - val_loss: 0.0294
```

Although I have played with the simulator and gathered by own data using the data collection strategies (center driving, recovery driving) I have learned from the lectures, I have decided not to use this data, because I didn't feel like I was good at driving with the keyboard as I didn't have a joystick with me.

Center	Left	Right
		

I have used the sample driving data provided by udacity to train my network (https://d17h27t6h515a5.cloudfront.net/topher/2016/December/584f6edd_data/data.zip) This consists of **8036 data samples**. 85% of the samples were used to train while the remaining 15% was used for validation. Each data sample contains the steering measurement as well as three images captured from three cameras installed at three different locations in the car [left, center, right]. About 4000 of these samples are within the [-0.05, -0.05] as shown when plotted in a histogram. Also, the data is biased towards left turns because of the track used to record this data.

```
def flipped(image, measurement):
  return np.fliplr(image), -measurement
```

```
def get_image(i, data):

  positions, corrections = ['left', 'center', 'right'], [.25, 0, -.25]
  ID, r = data.index[i], random.choice([0, 1, 2])

  measurement = data['steering'][ID] + corrections[r]

  path = PATH + data[positions[r]][ID][1:]
  if r == 1: path = PATH + data[positions[r]][ID]
  image = imread(path)

  if random.random() > 0.5:
    image, measurement = flipped(image, measurement)

  #print(i, ID)
  return image, measurement
```

To combat this biases, we have used the following data augmentation techniques (as seen in the code above):

- For each data point we use, we randomly choose among the three camera positions (left, center, right), and employ a steering correction of 0.25. The value of steering correction is adjusted based on how training the network was behaving
- We also randomly flip the image and change the sign of the steering angle.
- Doing those two things above increases our data set by a factor of 6. (as discussed in this article: http://navoshta.com/end-to-end-deep-learning/)

A python generator (as seen in the code below) was used to generate samples for each batch of data that would be fed when training and validating the network. We used a generator so that we don't need to store a lot of data unnecessarily and only use the memory that we need to use at a time. Notice that in my code I don't use the last batch of data which size is less than the batch_size I have chosen. This seems to be not a problem, so I just left it as is.

```
def generate_samples(data, batch_size):

  while True:

    SIZE = len(data)
    data.sample(frac = 1)

    for start in range(0, SIZE, batch_size):
      images, measurements = [], []

      for this_id in range(start, start + batch_size):
        if this_id < SIZE:
          image, measurement = get_image(this_id, data)
          measurements.append(measurement)
          images.append(image)

      yield np.array(images), np.array(measurements)
```

For this project I have:

- Used a simulator to collect data that could be used as samples of driving behavior
- Built a CNN in keras that predicts steering angles form images
- Trained and validated the model with training and validation set with the data provided by udacity
- Tested that the model could successfully drive around track one without leaving the road

For track 1, I have recorded a video of how no tire has left the drivable portion of the track surface (**run1.mp4**). The model has not generalized for track 2 (**run2.mp4**) as first it was swerving left to right and eventually crashed, As a recommendation for further improvement aside from gathering more data we can can use the augmentation techniques at pre existing data like adding shadows, random brightness and contrasts, and vertical and horizontal shifting.

Mithi Sevilla, March 11, 2017

OTHER THINGS I READ

- https://medium.com/@sujaybabruwad/teaching-a-car-to-ride-itself-by-showing-it-how-a-human-driver-does-it-797cc9c2462b#.6kp5b4a88
- https://github.com/upul/Behavioral-Cloning
- https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9#.5dpi87xzi