Udacity Behavioral Cloning

This project uses convolutional neural network to predict steering angle from image and drive a car in the simulator.

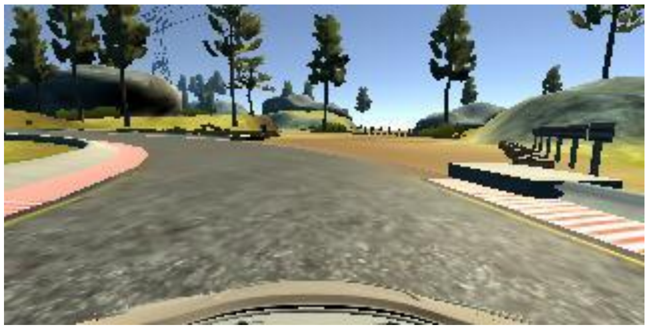
Overview

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Data Gathering

The data was gathered by driving the simulator for 2 forward and 1 reverse lap.



Sample Image Data

Generator function was used with a batch size of 32 to load the large set of data.

Data Preprocessing and Augmentation

All 3 camera (front, left and right) were used with correction factor added to the steering data of the left and the right camera.

Further, the images and the steering data were flipped to obtain more data.

All the images were cropped to obtain only the necessary information required for training.

Genrator function was incorporated to load the data and preprocessing.

```
def generator(samples, batch_size=32):
    num_samples = len(samples)
    while 1:
        for offset in range(0, num_samples, batch_size):
            batch_samples = samples[offset:offset+batch_size]

            images = []
            measurements = []
            for batch_sample in batch_samples:

                for i in range(3):
                    current_path = 'data/IMG/'+batch_sample[i].split('/')[-1]
                    image = cv2.imread(current_path)
                    images.append(image)
                    measurement = float(batch_sample[3])
                    if i == 0:
                        measurement = measurement

                    elif i == 1:
                        measurement = measurement + 0.2
                    else:
                        measurement = measurement - 0.2

                    measurements.append(measurement)


            #for image, measurement in zip(images, measurements):
             #   images.append(cv2.blur(image, (5,5)))
              #  measurements.append(measurement)

            augmented_images, augmented_measurements = [], []
            for image, measurement in zip(images, measurements):
                augmented_images.append(image)
                augmented_measurements.append(measurement)
                augmented_images.append(cv2.flip(image,1))
                augmented_measurements.append(measurement*-1.0)
            X_train = np.array(augmented_images)
            y_train = np.array(augmented_measurements)

            yield sklearn.utils.shuffle(X_train, y_train)
```

Model Architecture and Training

The model is based on the NVIDIA's paper on end-to-end deep learning for self driving cars. Slight modifications like dropout layers were incorporated to avoid overfitting.

The training uses mean squared error as cost function and Adam optimizer for 10 epochs.

```
model = Sequential()
model.add(Lambda(lambda x: x / 255.0 - 0.5, input_shape=(160, 320, 3)))
model.add(Cropping2D(cropping=((70,25),(0,0))))
model.add(Conv2D(24, (5, 5), strides=(2, 2), activation="relu"))
model.add(Conv2D(36, (5, 5), strides=(2, 2), activation="relu"))
model.add(Conv2D(48, (5, 5), strides=(2, 2), activation="relu"))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(Flatten())
model.add(Dense(500))
model.add(Dropout(0.2))
model.add(Dense(100))
model.add(Dropout(0.2))
model.add(Dense(10))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
model.fit_generator(train_generator,steps_per_epoch=len(train_samples)/batch_size,validati
on_data=validation_generator,validation_steps=(len(validation_samples)/batch_size),epochs=1
0, verbose=1)

model.save('model.h5')
```

The model and video of the simulation was stored for submission.