CarND-Behavioral-Cloning-P3

In this project, I used a deep neural network (built with Keras) to clone car driving behavior.

The dataset used to sample data, which nums are 8036*3.

Sample data consists of images taken from three different camera angles (Center - Left - Right), in addition to the steering angle, throttle, brake, and speed during each frame.

The network is based on NVIDIA's paper End to End Learning for Self-Driving Cars, which has been proven to work in this problem domain.

Pipeline architecture:

- **Data Loading.**
- **Data Augmentation.**
- **Data Preprocessing.**
- **Model Architecture.**
- **Model Training and Evaluation.**
- **Model Testing on the simulator.**

I'll explain each step in details below.

Environement:

- Ubuntu 16.04
- CUDA 9.0
- CuDNN 7.0.5
- Python 3.5
- Keras 1.2.1
- TensorFlow 1.10

```
# Importing Python libraries
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.layers import Lambda, Conv2D, MaxPooling2D, Dropout, Dense, Flatten
import csv
import cv2
import matplotlib.pyplot as plt
import random
import matplotlib.image as mpimg
```

```
Using TensorFlow backend.
```

Step 1: Data Loading

Download the dataset from here. This dataset contains more than 8,000 frame images taken from the 3 cameras (3 images for each frame), in addition to a `csv` file with the steering angle, throttle, brake, and speed during each frame.

```
data_dir = './data/'
labels_file = './data/driving_log.csv'
```

```
def load_data(labels_file, test_size):
    """
    Display a list of images in a single figure with matplotlib.
        Parameters:
```

```
            labels_file: The labels CSV file.
            test_size: The size of the testing set.
    """
    # center,left,right,steering,throttle,brake,speed
    labels = pd.read_csv(labels_file)
    X = labels[['center', 'left', 'right']].values
    y = labels['steering'].values
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=test_size, random_state=0)

    return X_train, X_valid, y_train, y_valid
```

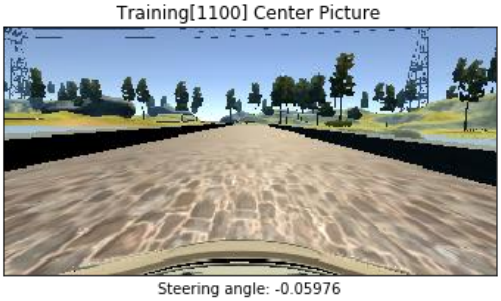
```
def load_image(data_dir, image_file):
    """
    Load RGB image.
        @Parameters:
            data_dir: The directory where the images are.
            image_file: The image file name.
    """
    return mpimg.imread(os.path.join(data_dir, image_file.strip()))
```

```
data = load_data(labels_file, 0.2)
print ((data[0][1100]))
```

```
['IMG/center_2016_12_01_13_41_58_530.jpg'
 ' IMG/left_2016_12_01_13_41_58_530.jpg'
 ' IMG/right_2016_12_01_13_41_58_530.jpg']
```

```
def display(image, angle, label):
    plt.imshow(image)
    plt.xlabel("Steering angle: {:.5f}".format(angle))
    plt.title(label)
    plt.xticks([])
    plt.yticks([])
    plt.show()
```

```
# data[0~3] is x_train set, x_valid set, y_train_steering set and y_valid_steering set
# data[0][0] are include center , left and right img name
# data[0][0][0-3] image name
image = load_image(data_dir, str(data[0][1100][0]))
# print(data[0][1100][0])
steering_angle = data[2][1100]
label = "Training[1100] Center Picture"
display(image, steering_angle, label)
```



Step 1: Data Preprocessing

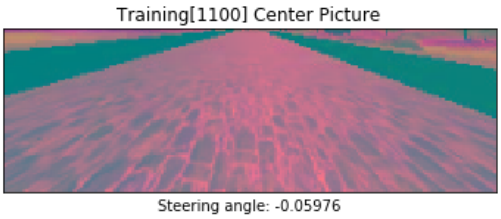
Preprocessing steps:

- Cropping the image to cut off the sky scene and the car front.
- Resizing the image to (66 * 200), the image size that the selected model expects.
- Converting the image to the YUV color space that paper required.

```
# Nvidia Model
IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 66, 200, 3
INPUT_SHAPE = (IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)
# Output: vehicle control
```

```
def preprocess(img):
    """
    Preprocessing (Crop - Resize - Convert to YUV) the input image.
        Parameters:
            img: The input image to be preprocessed.
    """
    # Cropping the image
    img = img[62:-23, :, :]
    # Resizing the image
    img = cv2.resize(img, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
    # Converting the image to YUV
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    return img
```

```
image = preprocess(load_image(data_dir, str(data[0][1100][0])))
steering_angle = data[2][1100]
label = "Training[1100] Center Picture"
display(image, steering_angle, label)
```



Step 2: Data Augmentation

Augmentation steps:

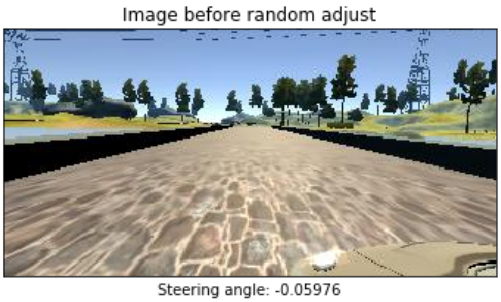
- Adjusting the steering angle of random images.
- Flipping random images horizontaly, with steering angle adjustment.
- Shifting (Translating) random images, with steering angle adjustment.
- Adding shadows to random images.
- Altering the brightness of random images.

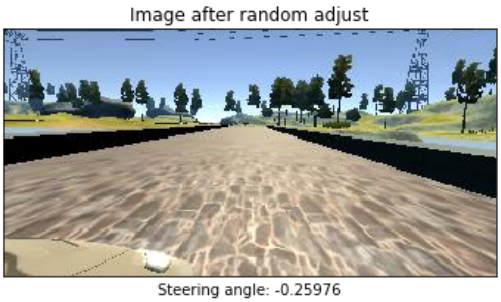
```
def random_adjust(data_dir, center, left, right, steering_angle):
    """
    Adjusting the steering angle of random images.
        Parameters:
            data_dir: The directory where the images are.
            center: Center image.
            left: Left image.
            right: Right image
            steering_angle: The steering angle of the input frame.
    """
    choice = np.random.choice(3)
    # print (choice)
    if choice == 0:
        return load_image(data_dir, left), steering_angle + 0.2
    elif choice == 1:
```

```
        return load_image(data_dir, right), steering_angle - 0.2
    return load_image(data_dir, center), steering_angle
```

```
image = load_image(data_dir, str(data[0][1100][1]))
steering_angle = data[2][1100]
label = "Image before random adjust"
display(image, steering_angle, label)

image = random_adjust(data_dir,str(data[0][1100][0]),str(data[0][1100][1]),str(data[0][1100][2]),
steering_angle)
label = "Image after random adjust"
display(image[0], image[1], label)
```





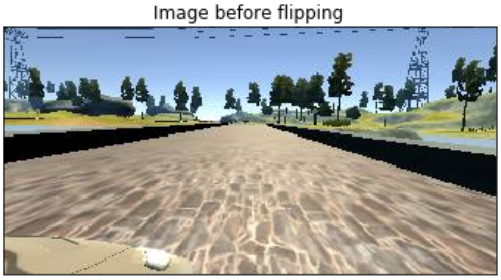

```
def random_flip(image, steering_angle):
    """
    Randomly flipping the input image horizontaly, with steering angle adjustment.
        Parameters:
            image: The input image.
            steering_angle: The steering angle related to the input image.
    """
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)
        steering_angle = -steering_angle
    return image, steering_angle
```

```
image = load_image(data_dir, str(data[0][1100][2]))
print (image.shape)
steering_angle = data[2][1100]
label = "Image before flipping"
display(image, steering_angle, label)

image = random_flip(image, steering_angle)
label = "Image after flipping"
display(image[0], image[1], label)
```

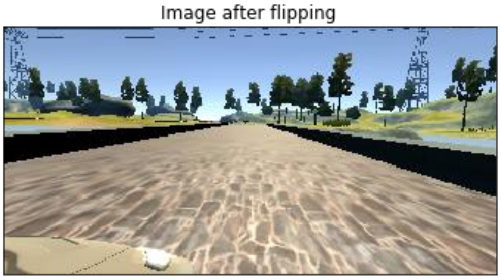
```
(160, 320, 3)
```

Image before flipping



Steering angle: -0.05976

Image after flipping



Steering angle: -0.05976

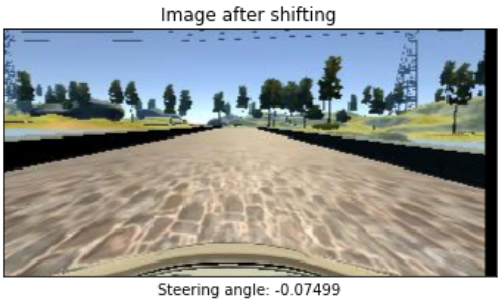
```
def random_shift(image, steering_angle, range_x, range_y):
    """
    Shifting (Translating) the input images, with steering angle adjustment.
        Parameters:
            image: The input image.
            steering_angle: The steering angle related to the input image.
            range_x: Horizontal translation range.
            range_y: Vertival translation range.
    """
    trans_x = range_x * (np.random.rand() - 0.5)
    trans_y = range_y * (np.random.rand() - 0.5)
    steering_angle += trans_x * 0.002
    trans_m = np.float32([[1, 0, trans_x], [0, 1, trans_y]])
    height, width = image.shape[:2]
    image = cv2.warpAffine(image, trans_m, (width, height))
    return image, steering_angle
```

```
image = load_image(data_dir, str(data[0][1100][0]))
steering_angle = data[2][1100]
label = "Image before shifting"
display(image, steering_angle, label)

image = random_shift(image, steering_angle, 100, 10)
label = "Image after shifting"
display(image[0], image[1], label)
```

Image before shifting

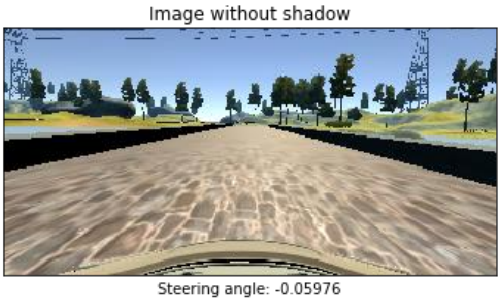
Steering angle: -0.05976

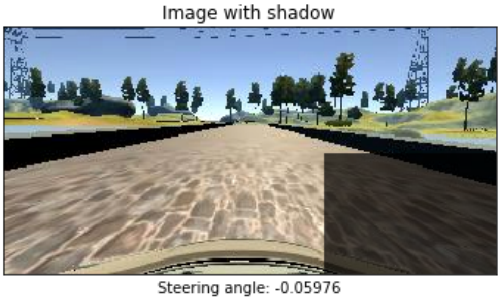



```
def random_shadow(image):
    """
    Adding shadow to the input image.
        Parameters:
            image: The input image.
    """
    bright_factor = 0.3
    x = random.randint(0, image.shape[1])
    y = random.randint(0, image.shape[0])
    width = random.randint(image.shape[1], image.shape[1])
    if(x + width > image.shape[1]):
        x = image.shape[1] - x
    height = random.randint(image.shape[0], image.shape[0])
    if(y + height > image.shape[0]):
        y = image.shape[0] - y
    image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    image[y:y+height,x:x+width,2] = image[y:y+height,x:x+width,2]*bright_factor
    return cv2.cvtColor(image, cv2.COLOR_HSV2RGB)
```

```
image = load_image(data_dir, str(data[0][1100][0]))
steering_angle = data[2][1100]
label = "Image without shadow"
display(image, steering_angle, label)

image = random_shadow(image)
label = "Image with shadow"
display(image, steering_angle, label)
```



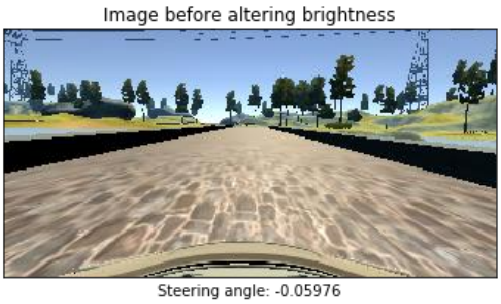


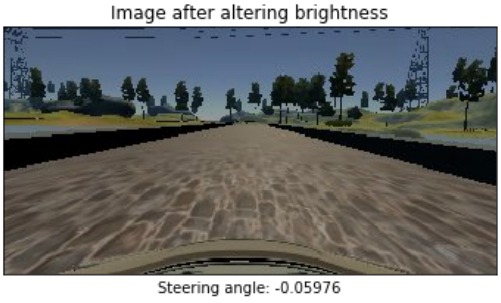

```
def random_brightness(image):
    """
    Altering the brightness of the input image.
```

```
        Parameters:
            image: The input image.
    """
    # HSV (Hue, Saturation, Value) is also called HSB ('B' for Brightness).
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    ratio = 1.0 + (np.random.rand() - 0.5)
    hsv[:,:,2] =  hsv[:,:,2] * ratio
    return cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)
```

```
image = load_image(data_dir, str(data[0][1100][0]))
steering_angle = data[2][1100]
label = "Image before altering brightness"
display(image, steering_angle, label)

image = random_brightness(image)
label = "Image after altering brightness"
display(image, steering_angle, label)
```



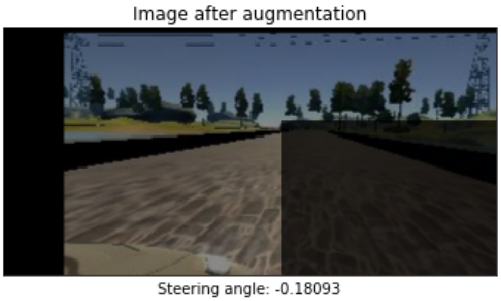



```
def augument(data_dir, center, left, right, steering_angle, range_x=100, range_y=10):
    """
    Generate an augumented image and adjust the associated steering angle.
        Parameters:
            data_dir: The directory where the images are.
            center: Center image.
            left: Left image.
            right: Right image
            steering_angle: The steering angle related to the input frame.
            range_x (Default = 100): Horizontal translation range.
            range_y (Default = 10): Vertival translation range.
    """
    image, steering_angle = random_adjust(data_dir, center, left, right, steering_angle)
    image, steering_angle = random_flip(image, steering_angle)
    image, steering_angle = random_shift(image, steering_angle, range_x, range_y)
    image = random_shadow(image)
    image = random_brightness(image)
    return image, steering_angle
```

```
steering_angle = data[2][1100]
center = data[0][1100][0]
left = data[0][1100][1]
right = data[0][1100][2]
range_x = 100
```

```
range_y = 10

image, steering_angle = augument(data_dir, center, left, right, steering_angle, range_x, range_y)
label = "Image after augmentation"
display(image, steering_angle, label)
```



Step 3: Model Architecture

In this step, we will design and implement a deep learning model that can clone the vehicle's behavior. We'll use a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands.

We'll use the ConvNet from NVIDIA's paper End to End Learning for Self-Driving Cars, which has been proven to work in this problem domain.

According to the paper: *"**Network Architecture** We train the weights of our network to minimize the mean squared error between the steering command output by the network and the command of either the human driver, or the adjusted steering command for off-center and rotated images. Our network architecture is shown in Figure 4. The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network. The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing. The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. We use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers. We follow the five convolutional layers with three fully connected layers leading to an output control value which is the inverse turning radius. The fully connected layers are designed to function as a controller for steering, but we note that by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractor and which serve as controller."*

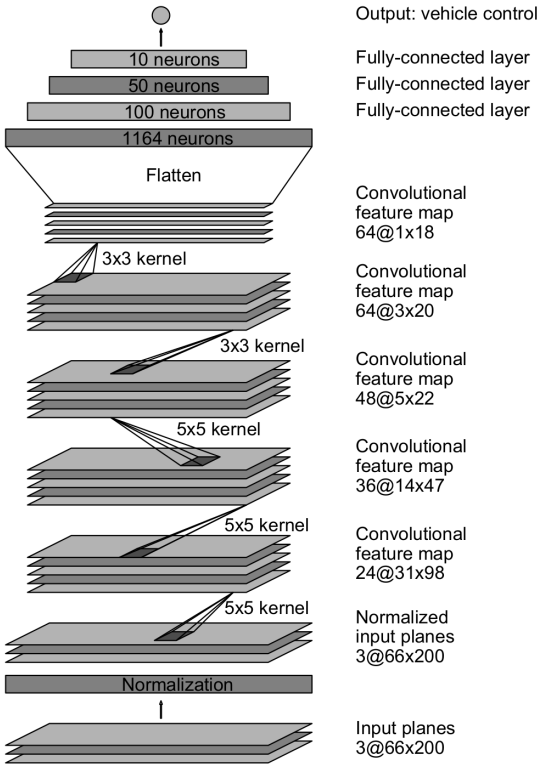


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

Lamda_function Normalizing the images (by dividing image data by 127.5 and subtracting 1.0). As stated in the Model Architecture section, this is to avoid saturation and make gradients work better).

```
def NVIDIA_model():
    model = Sequential()
    model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
    model.add(Conv2D(24, 5, 5, activation='relu', subsample=(2, 2)))
    model.add(Conv2D(36, 5, 5, activation='relu', subsample=(2, 2)))
    model.add(Conv2D(48, 5, 5, activation='relu', subsample=(2, 2)))
    model.add(Conv2D(64, 3, 3, activation='relu'))
    model.add(Conv2D(64, 3, 3, activation='relu'))
    model.add(Dropout(0.5))
    # Added a dropout layer to the model to prevent overfitting.
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1))
    model.summary()
    return model
```

Step 4: Model Training and Evaluation

- I've splitted the data into 80% training set and 20% validation set to measure the performance after each epoch.
- I used Mean Squared Error (MSE) as a loss function to measure how close the model predicts to the given steering angle for each input frame.
- I used the Adaptive Moment Estimation (Adam) Algorithm minimize to the loss function. Adam is an optimization algorithm introduced by D. Kingma and J. Lei Ba in a 2015 paper named Adam: A Method for Stochastic Optimization. Adam algorithm computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop algorithms, Adam also keeps an exponentially decaying average of past gradients mtmt, similar to momentum algorithm, which in turn produce better results.
- I used `ModelCheckpoint` from Keras to check the validation loss after each epoch and save the model only if the validation loss reduced.

```
batch_size = 128
samples_per_epoch = 20000
nb_epoch = 10
```

```
def batcher(data_dir, image_paths, steering_angles, batch_size, training_flag):
    """
    Generate a training image given image paths and the associated steering angles
        Parameters:
            data_dir: The directory where the images are.
            image_paths: Paths to the input images.
            steering_angle: The steering angle related to the input frame.
            batch_size: The batch size used to train the model.
            training_flag: A flag to determine whether we're in training or validation mode.
    """
    images = np.empty([batch_size, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS])
    steers = np.empty(batch_size)
    while True:
        i = 0
        for index in np.random.permutation(image_paths.shape[0]):
            center, left, right = image_paths[index]
            steering_angle = steering_angles[index]
            if training_flag and np.random.rand() < 0.6:
                image, steering_angle = augument(data_dir, center, left, right, steering_angle)
            else:
                image = load_image(data_dir, center)
            images[i] = preprocess(image)
            steers[i] = steering_angle
            i += 1
            if i == batch_size:
                break
        yield images, steers
```

```
def train_model(model, X_train, X_valid, y_train, y_valid):
    checkpoint = ModelCheckpoint('Nvidia_model-{val_loss:03f}.h5',
                                 monitor='val_loss',
                                 verbose=0,
                                 save_best_only=True,
                                 mode='auto')
    model.compile(loss='mse', optimizer=Adam(lr=1.0e-4))
    model.fit_generator(batcher(data_dir, X_train, y_train, batch_size, True),
                        samples_per_epoch,
                        nb_epoch,
                        max_q_size=1,
                        validation_data=batcher(data_dir, X_valid, y_valid, batch_size, False),
                        nb_val_samples=len(X_valid),
                        callbacks=[checkpoint],
                        verbose=1)
```

```
model = NVIDIA_model()
train_model(model, *data)
```

```
WARNING:tensorflow:From /home/henry_pan/.local/lib/python3.5/site-
packages/keras/backend/tensorflow_backend.py:1023: calling reduce_prod (from
tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
lambda_1 (Lambda)            (None, 66, 200, 3)        0           lambda_input_1[0][0]
_________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 31, 98, 24)    1824        lambda_1[0][0]
_________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 14, 47, 36)    21636       convolution2d_1[0][0]
_________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 5, 22, 48)     43248       convolution2d_2[0][0]
_________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 3, 20, 64)     27712       convolution2d_3[0][0]
_________________________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 1, 18, 64)     36928       convolution2d_4[0][0]
_________________________________________________________________________________________________
dropout_1 (Dropout)          (None, 1, 18, 64)         0           convolution2d_5[0][0]
_________________________________________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0           dropout_1[0][0]
_________________________________________________________________________________________________
dense_1 (Dense)              (None, 100)               115300      flatten_1[0][0]
_________________________________________________________________________________________________
dense_2 (Dense)              (None, 50)                5050        dense_1[0][0]
_________________________________________________________________________________________________
dense_3 (Dense)              (None, 10)                510         dense_2[0][0]
_________________________________________________________________________________________________
dense_4 (Dense)              (None, 1)                 11          dense_3[0][0]
=================================================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________________________________________
WARNING:tensorflow:From /home/henry_pan/.local/lib/python3.5/site-
packages/keras/backend/tensorflow_backend.py:1084: calling reduce_mean (from
tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
Epoch 1/10
19968/20000 [============================>.] - ETA: 0s - loss: 0.0286

/home/henry_pan/.local/lib/python3.5/site-packages/keras/engine/training.py:1569: UserWarning: Epoch
comprised more than `samples_per_epoch` samples, which might affect learning results. Set
`samples_per_epoch` correctly to avoid this warning.
  warnings.warn('Epoch comprised more than '
```

```
20096/20000 [==============================] - 58s - loss: 0.0287 - val_loss: 0.0122
Epoch 2/10
20096/20000 [==============================] - 55s - loss: 0.0234 - val_loss: 0.0110
Epoch 3/10
20096/20000 [==============================] - 52s - loss: 0.0210 - val_loss: 0.0115
Epoch 4/10
20096/20000 [==============================] - 51s - loss: 0.0187 - val_loss: 0.0114
Epoch 5/10
20096/20000 [==============================] - 54s - loss: 0.0175 - val_loss: 0.0093
Epoch 6/10
20096/20000 [==============================] - 55s - loss: 0.0167 - val_loss: 0.0093
Epoch 7/10
20096/20000 [==============================] - 53s - loss: 0.0166 - val_loss: 0.0104
Epoch 8/10
20096/20000 [==============================] - 50s - loss: 0.0159 - val_loss: 0.0100
Epoch 9/10
20096/20000 [==============================] - 49s - loss: 0.0160 - val_loss: 0.0086
Epoch 10/10
20096/20000 [==============================] - 51s - loss: 0.0154 - val_loss: 0.0101
```

Step 5: Model Testing on the simulator

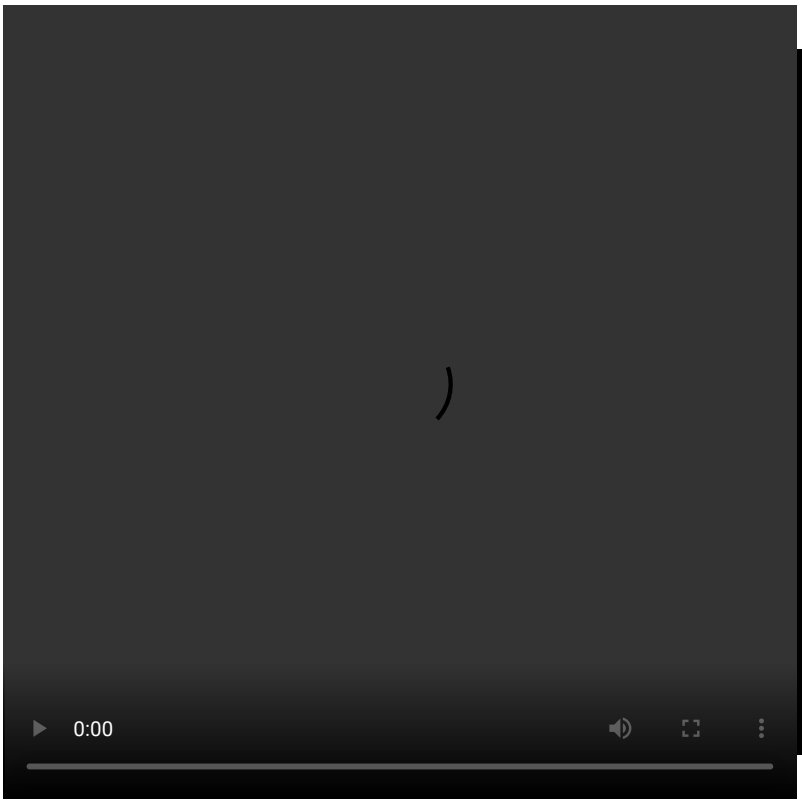
The model was able to drive the car safely through the track without leaving the drivable portion of the track surface.

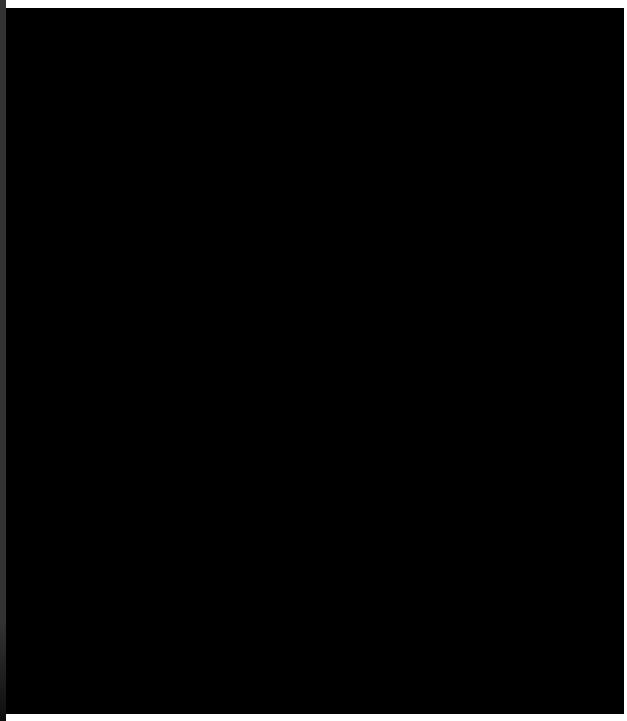
```
CUDA_VISIBLE_DEVICES=1 python3 drive.py Nvidia_model-0.008571.h5 run_result
```

```
# Import everything needed to edit/save/watch video clips
from moviepy.editor import VideoFileClip
from IPython.display import HTML
```

```


run_result = './run_result.mp4'
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(run_result))
```





Conclusion

Using NVIDIA's End to End learning network, the model was able to drive the car through the first track. I've only used the training data provided by Udacity. One way to improve the model is to collect more data from the mountain track, and train the model to go through this challenging track as well. We can also modify `drive.py` script to change the maximum speed and make it related to the steering angle, in order to make the car speed up in the straight parts of the track, and slow down in the curved parts.