Udacity Self-Driving Car Nanodegree

Behavioral Cloning Project

Overview

In this project, I use a neural network to clone car driving behavior. It is a supervised regression problem between the car steering angles and the road images in front of a car.

Those images were taken from three different camera angles (from the center, the left and the right of the car).

The network is based on The NVIDIA model, which has been proven to work in this problem domain.

As image processing is involved, the model is using convolutional layers for automated feature engineering.

Files included

- model.py The script used to create and train the model.
- drive.py The script to drive the car. You can feel free to resubmit the original drive.py or make modifications and submit your modified version.
- preprocess.py The script to provide useful functionalities (i.e. image preprocessing)
- Model.h5 The model weights.
- environments.yml conda environment (Use TensorFlow without GPU)
- environments-gpu.yml conda environment (Use TensorFlow with GPU)

Note: drive.py is originally from the Udacity Behavioral Cloning project GitHub

Quick Start Guide

Install required python libraries:

You need a anaconda or miniconda to use the environment setting.

```
# Use TensorFlow with GPU
conda env create -f environments-gpu.yml
```

Or you can manually install the required libraries (see the contents of the environemnt*.yml files) using pip.

Run the pretrained model

Start up the Udacity self-driving simulator, choose a scene and press the Autonomous Mode button. Then, run the model as follows:

```
Python drive.py model.json
```

To train the model

You'll need the data folder which contains the training images.

```
Python model.py carnd.pickle
```

`Here carnd.pickle is` the **pickle** file from the local drive and train the data using model that I built.

This will generate a file `model.json` whenever the performance in the epoch is better than the previous best. For example, the first epoch will generate a file called `model.h5`.

There are **3** python scripts: **preprocess.py**, **model.py**, and **drive.py**.

preprocess.py

This python script imports the raw image data and resizes them.

I resized the image because image contains unnecessary background noises such as sky, river, and trees.

I decided to remove them and reduced the size of the image by **25%**, then I used only one channel from each image. I found that the image data do not have to have a lot of pixels when training the model. I found reducing the size by 25% and using just one channel were more efficient in terms of time and space.

I saved them as **features** and saved the data of steering angels as **labels**.

Then, I splitted the data into **train** and **validation**, and saved them as **frontcamera.pickle** file.

model.py

The main purpose of this script is to train the model using the data saved from the above python script.

First, it imports the **pickle** file from the local drive and train the data using model that I built.

The detail of the model can be found in the script.

When the training is done, the model and weights are saved as **model.json** and **model.h5**.

drive.py

This is the python script that receives the data from the Training data which I collected in training mode, predicts the steering angle using the deep learning model, and send the throttle and the predicted angles back to the program.

Since the images were reshaped and normalized during training, the image from the program is reshaped and normalized just as in **preprocess.py** and **model.py**

Preprocessing

As mentioned briefly above, the images are loaded from the local drive and reshaped by the function called **load_image**.

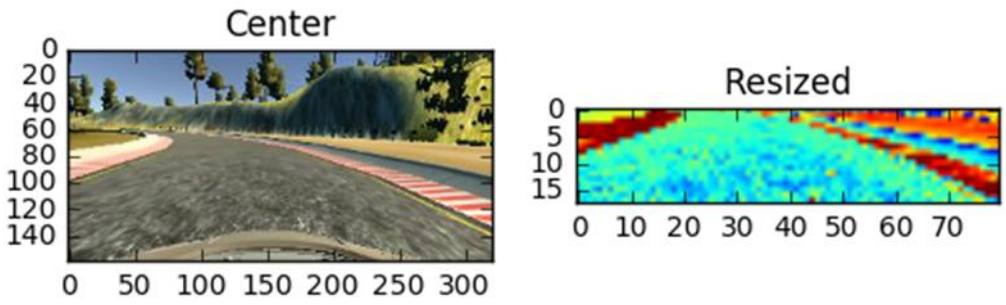
Below are the original images from center, left, and right cameras and reshaped images at the right of each original image.

```
### Import data
import argparse
import os
import csv
import base64
import numpy as np
import matplotlib.pyplot as plt

folder_path = C:\Users\vikas\Desktop\behaviourial_cloning\training_data\IMG"
label_path = "{}/driving_log.csv".format(folder_path)

data = []
with open(label_path) as F:
    reader = csv.reader(F)
    for i in reader:
        data.append(i)

print("data imported")
```

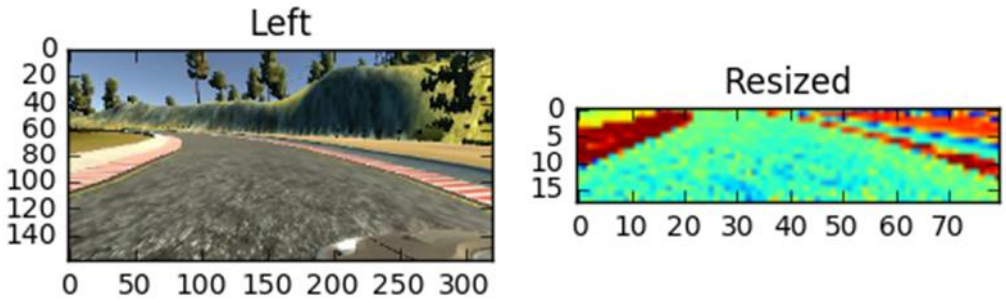
```
data imported
```

```
def load_image(data_line, j):
    img = plt.imread(data_line[j].strip())[65:135:4,0:-1:4,0]
    lis = img.flatten().tolist()
    return lis

i = 0
for j in range(3):
    plt.subplot(121)
    img = plt.imread(data[i][j].strip())
    plt.imshow(img)
    if j == 0:
        plt.title("Center")
    elif j == 1:
        plt.title("Left")
    elif j == 2:
        plt.title("Right")
    plt.subplot(122)
    a = np.array(load_image(data[i], j)).reshape(1, 18, 80, 1)
    # a = load_image(data[img_num])
    print(a.shape)
    plt.imshow(a[0,:,:,0])
    plt.title("Resized")
    plt.show()
del(a)
```

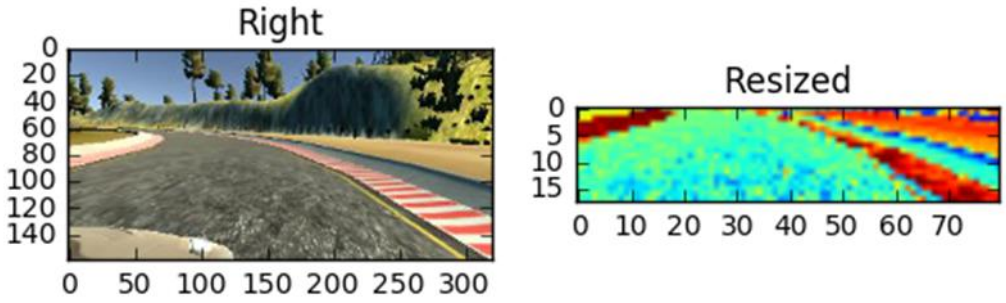
```
(1, 18, 80, 1)
```




```
(1, 18, 80, 1)
```




```
(1, 18, 80, 1)
```



Training

Below is the summary of the model I implemented to train the data.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 16, 78, 16)        160
_________________________________________________________________
activation_1 (Activation)    (None, 16, 78, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 76, 8)         1160
```

```
_________________________________________________________________
activation_2 (Activation)    (None, 14, 76, 8)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 74, 4)         292
_________________________________________________________________
activation_3 (Activation)    (None, 12, 74, 4)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 72, 2)         74
_________________________________________________________________
activation_4 (Activation)    (None, 10, 72, 2)         0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 36, 2)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 36, 2)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 360)               0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                5776
_________________________________________________________________
activation_5 (Activation)    (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
activation_6 (Activation)    (None, 16)                0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                272
_________________________________________________________________
activation_7 (Activation)    (None, 16)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 8,023.0
Trainable params: 8,023.0
Non-trainable params: 0.0
```

Conclusion

I found that the whole image can confuse the model due to unncessary background noises such as tries, skies, etc. I decided to cut those unncessary pixels and reduced the size by 25%. I only used red channel of the image because I assumed that red channel contains the better information for identifying the road and lanes than green and blue channels. As a result, the size of the image was 18 x 80 x 1.

In my model, I used 4 convolutional layers with 1 max pooling layer, and 3 more dense layers after flatten the matrix. For each convolutional layer, I decreased the channel size by half. When the size of the channel became 2 in the fourth convolutional layer, I applied max pooling with dropout with 25%. After flatten the matrix, the size of features became 360. I used dense layers with 16 features 4 times. Each **epoch** took

about **100** seconds and I used **10 epoches** to train the data. The interesting thing I noticed was even though the model allowed the car to drive itself, the accuracy was only about **58%**. So the accuracy did not have to be high for car to drive autonomously. I believe that to increase the accuracy, I would need more data set and more epoches.