Behavioral-Cloning

Write up

1st submit: August 17th, Kenta Kumazaki

1. Purpose

In this project, I used what I've learned about deep neural networks and convolutional neural networks to clone driving behavior. I trained, validated and tested a model using Keras. The model outputs a steering angle to an autonomous vehicle to drive autonomously around the track.

2. Goals/Steps

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

3. Submission

(1) GitHub

https://github.com/kkumazaki/Self-Drivig-Car_Project4_Behavioral-Cloning.git

(2) Directory

- **Writeup_of_Lesson17.pdf**: This file
- **drive.py**: Script to drive the car (I didn't modify the original file)
- **model.py**: Script used to create and train the model
- **model.h5**: A trained Keras model by using collected data from simulator
- **video.mp4**: A video recording of my vehicle driving autonomously around the track for at least one full lap

4. Reflection

(1)Collecting training data

To be honest, it was very difficult to drive the track smoothly in Training Mode by hand.

I did many cycles of “collecting training data” and “making the Model”, then I finally got a good result.

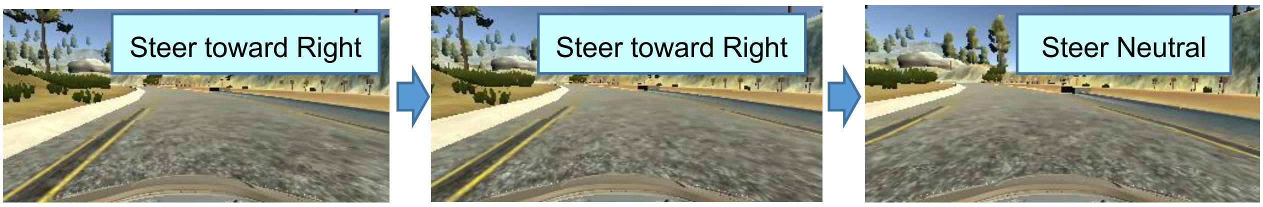
I got the following training data, and finally I chose Data 1, 3, 6, 7, 9, 10.

Data	Purpose	Result	Final Choice
Data1	Drive the center as much as possible.	There were a couple of times I reached the edge of the road, but it’s mostly good.	O
Data2	Drive around the edge and try to avoid lane departure.	I saved the unnecessary data to “try to approach to the road edge”, so I deleted it.	X
Data3	Drive the curve smoothly as much as possible.	I mostly drove the track smoothly.	O
Data4	Drive around the edge and try to avoid lane departure.	I saved the unnecessary data to “try to approach to the road edge”, so I deleted it.	X
Data5	Drive around the edge and try to avoid lane departure.	I saved the unnecessary data to “try to approach to the road edge”, so I deleted it.	X
Data6	Drive the center as much as possible.	There were a couple of times I reached the edge of the road, but it’s mostly good.	O
Data7	Drive the center as much as possible.	There were a couple of times I reached the edge of the road, but it’s mostly good.	O
Data8	Drive around the edge and try to avoid lane departure.	I saved the unnecessary data to “try to approach to the road edge”, so I deleted it.	X
Data9	Drive the curve smoothly as much as possible.	I mostly drove the track smoothly.	O
Data10	Drive around the edge and try to avoid lane departure.	I save the data only when “try to avoid lane departure”, not save data when “try to approach to the road edge”.	O

When I collected “Recovery Laps” with Data 2, 4, 5, 8, I saved the unnecessary data to “try to approach to the road edge”, so when I trained the Model by using those data, the vehicle wandered a lot even at the straight road. After I collected “Recovery Laps” with Data 10 as below, the vehicle behavior became very stable.

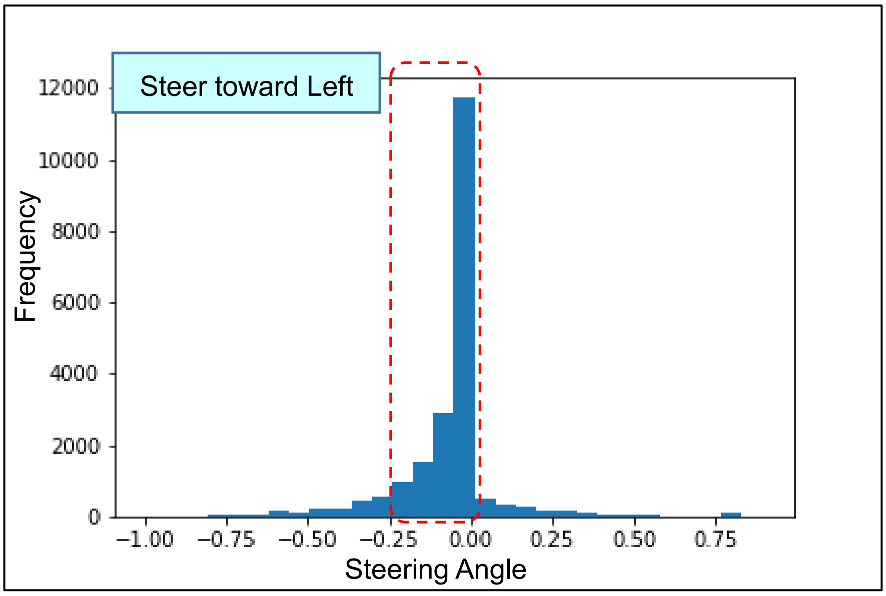
<Step1> Go to the road edge without saving data.

<Step2> After start saving data, try to avoid lane departure.



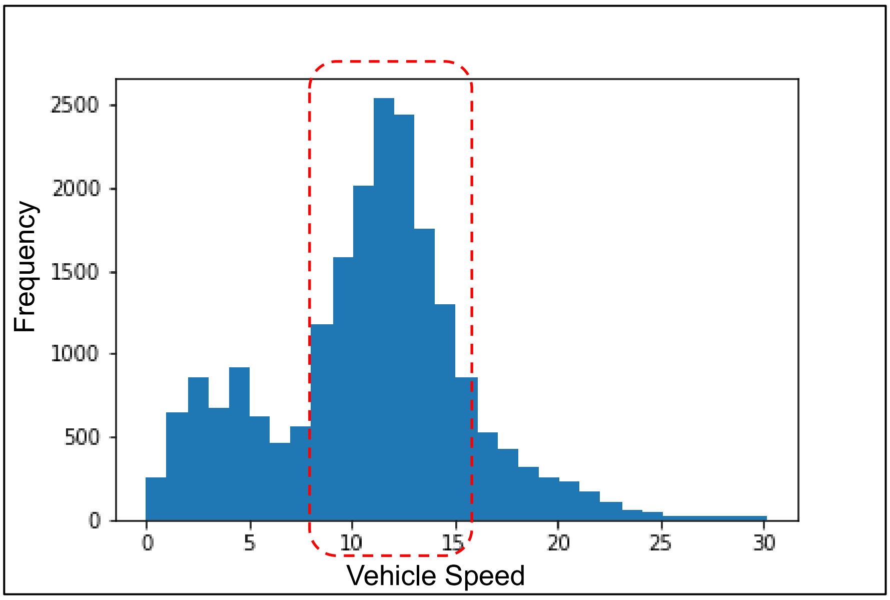
The following graph shows the histogram of Steering Angle of the Training Data (Final Choice).

I trained the data at the Track1, so the frequency of Left Side is higher than Right Side.



The following graph shows the histogram of Vehicle Speed of the Training Data (Final Choice).

To avoid unnecessary lane deviation, I drove slowly about 10-15 mph.



The total number of Training Data (image file and steering angle) is “20,928”.

(2)Description of my pipeline

My pipeline consisted of 3 steps as following.

Step 1: Preparation (import, read data, etc)

Step 2: Augment Images (Flip the image and steering right/left)

Step 3: Make Model

Step 1: Preparation

Load the Trained Datasets at the workspace.

As I explained before, I took multiple datasets but finally comment out unnecessary data.

At the end of this step, I generated “train_samples” and “validation_samples” by splitting the dataset.

As recommended in the Lecture, I set 80%: Training Data, and 20%: Validation Data (A).

```
#(1)Preparation
import csv
import cv2
import math
import random
import numpy as np
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

lines = []
with open('data1/driving_log.csv') as csvfile:
    reader = csv.reader(csvfile)
    for line in reader:
        lines.append(line)

#with open('data2/driving_log.csv') as csvfile:
#    reader = csv.reader(csvfile)
#    for line in reader:
#        lines.append(line)

with open('data3/driving_log.csv') as csvfile:
    reader = csv.reader(csvfile)
    for line in reader:
        lines.append(line)

#with open('data4/driving_log.csv') as csvfile:
#    reader = csv.reader(csvfile)
#    for line in reader:
#        lines.append(line)

#with open('data5/driving_log.csv') as csvfile:
#    reader = csv.reader(csvfile)
#    for line in reader:
#        lines.append(line)

with open('data6/driving_log.csv') as csvfile:
    reader = csv.reader(csvfile)
    for line in reader:
        lines.append(line)

with open('data7/driving_log.csv') as csvfile:
    reader = csv.reader(csvfile)
    for line in reader:
        lines.append(line)

with open('data9/driving_log.csv') as csvfile:
    reader = csv.reader(csvfile)
    for line in reader:
        lines.append(line)

with open('data10/driving_log.csv') as csvfile:
    reader = csv.reader(csvfile)
    for line in reader:
        lines.append(line)

# To use "model.fit_generator" instead of "model.fit", split training data here
train_samples, validation_samples = train_test_split(lines, test_size=0.2)
```

A

Step 2: Augment Images

At first, I was trying to make the Model without generator, so the codes for augmentation was quite simple.

But as I write later I created generator, so I commented out it here.

```
#(2)Augmentate Images (Flip the image and steering right/left)
#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)
```

I created the generator as below. I followed the Lesson and I set the same batch_size = 32.

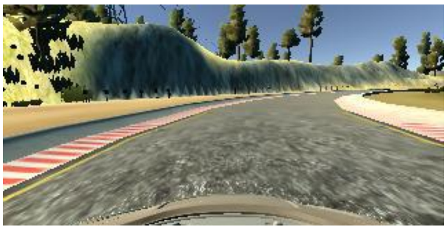
Here, I augmented the image data by flipping left/right (B) to generalize the training data.

By shuffling the training data and validation data, I can avoid overfitting (C).

```
# To use "model.fit_generator" instead of "model.fit", create generator
num_lines = len(lines)
batchsize = 32

def generator(lines, batch_size=batchsize):
    num_lines = len(lines)
    while 1:
        random.shuffle(lines)

        for offset in range(0, num_lines, batch_size):
            batch_samples = lines[offset:offset+batch_size]
            augmented_images = []
            augmented_measurements = []

            for batch_sample in batch_samples:
                    source_path = batch_sample[0]
                    filename = source_path.split('IMG')[-1]
                    if source_path.split('/')[2] == 'drive1':
                        current_path = 'data1/IMG/' + filename
                    elif source_path.split('/')[2] == 'drive2':
                        current_path = 'data2/IMG/' + filename
                    elif source_path.split('/')[2] == 'drive3':
                        current_path = 'data3/IMG/' + filename
                    elif source_path.split('/')[2] == 'drive4':
                        current_path = 'data4/IMG/' + filename
                    elif source_path.split('/')[2] == 'drive5':
                        current_path = 'data5/IMG/' + filename
                    elif source_path.split('/')[2] == 'drive6':
                        current_path = 'data6/IMG/' + filename
                    elif source_path.split('/')[2] == 'drive7':
                        current_path = 'data7/IMG/' + filename
                    elif source_path.split('/')[2] == 'drive9':
                        current_path = 'data9/IMG/' + filename
                    elif source_path.split('/')[2] == 'drive10':
                        current_path = 'data10/IMG/' + filename
                    image = cv2.imread(current_path)
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                    measurement = float(batch_sample[3])
                    augmented_images.append(image)
                    augmented_measurements.append(measurement)
                    augmented_images.append(cv2.flip(image,1)) # Addition the flipped image
                    augmented_measurements.append(measurement* -1.0) # Addition the flipped steering angle
            # trim image to only see section with road
            X_train = np.array(augmented_images)
            y_train = np.array(augmented_measurements)
            yield sklearn.utils.shuffle(X_train, y_train)

train_generator = generator(train_samples, batch_size=batchsize)
validation_generator = generator(validation_samples, batch_size=batchsize)

print("Augmentation and creating generator is done.")
```

Show in the Lesson, the original image and flipped image will be as following. It helps generalization.

Original Image



Flipped Image

Step 3: Make Model

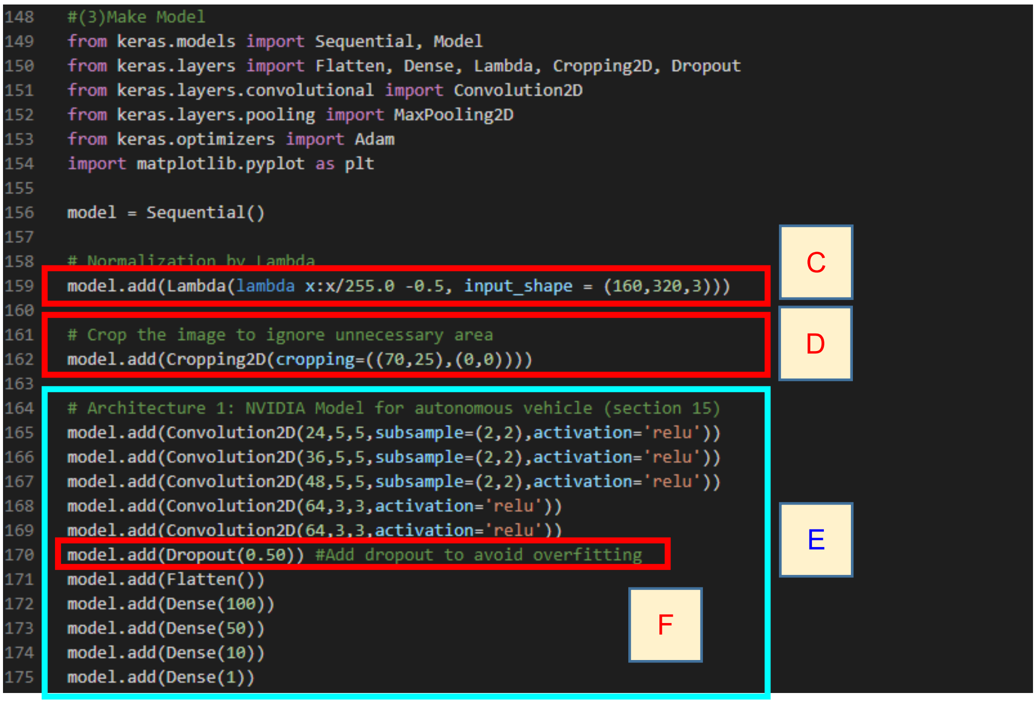
I added the Lambda Layer at first (C) to normalize the image data.

I added Cropping Layer (D) to eliminate the unnecessary area of image, such as image of sky, trees, etc.

I modified the Architecture of NVIDIA (E), which was recommended in the Lesson. I will explain it later.

I added Dropout Layer between Convolution Layers and Flatten Layer (F) to avoid overfitting.

I set the dropout parameter = 0.5, which looks common parameter in many Models.



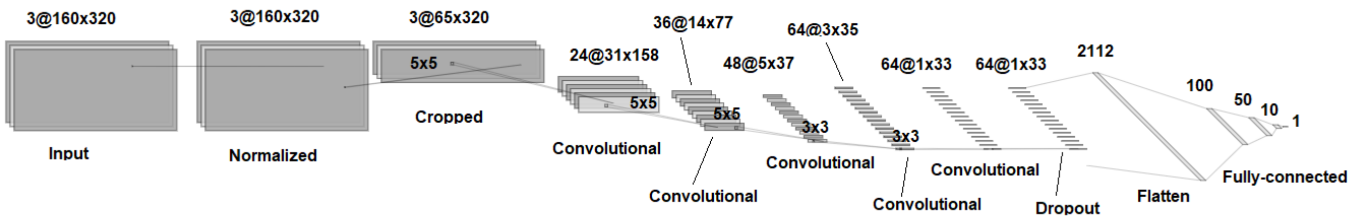
```
#(3)Make Model
from keras.models import Sequential, Model
from keras.layers import Flatten, Dense, Lambda, Cropping2D, Dropout
from keras.layers.convolutional import Convolution2D
from keras.layers.pooling import MaxPooling2D
from keras.optimizers import Adam
import matplotlib.pyplot as plt

model = Sequential()

# Normalization by Lambda
model.add(Lambda(lambda x:x/255.0 -0.5, input_shape = (160,320,3)))

# Crop the image to ignore unnecessary area
model.add(Cropping2D(cropping=((70,25),(0,0))))

# Architecture 1: NVIDIA Model for autonomous vehicle (section 15)
model.add(Convolution2D(24,5,5,subsample=(2,2),activation='relu'))
model.add(Convolution2D(36,5,5,subsample=(2,2),activation='relu'))
model.add(Convolution2D(48,5,5,subsample=(2,2),activation='relu'))
model.add(Convolution2D(64,3,3,activation='relu'))
model.add(Convolution2D(64,3,3,activation='relu'))
model.add(Dropout(0.50)) #Add dropout to avoid overfitting
model.add(Flatten())
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
```


The architecture of my Model is show below.

My model consists of a convolution neural network based on NVIDIA model.

There are 5 Convolution Layers with 5x5 or 3x3 filter sizes and depth between 24 and 64, and there are 4 Fully-connected Layers with neurons size between 1 and 100.

In the Convolution Layer, I chose the activation function = RELU to introduce nonlinearity.



I chose Adam Optimizer and set the Learning Rate = 0.0001 (G).

I searched Internet and people set the parameter for Adam Optimizer from 0.001 to 0.0001 range, so I chose the small rate to get better results even though it takes longer time. The training data number is “20,928” as previously written, so I thought it’s not so big data if I can use GPU Mode.

I tried learning many times by changing data sets and the number of epoch, and I finally found that the epoch number = 10 is good for my Dataset and Model (H).

```
#num_epoch = 15
num_epoch = 10
#num_epoch = 5
#num_epoch = 3
#num_epoch = 1

# Start Learning and make Model by "model.fit"
#model.compile(loss = 'mse', optimizer = 'adam')
#model.fit(X_train, y_train, validation_split = 0.2, shuffle = True, nb_epoch = num_epoch)
#model.save('model1.h5')

# Start Learning and make Model by "model.fit_generator"
LEARNING_RATE =0.0001
model.compile(loss='mean_squared_error', optimizer=Adam(lr=LEARNING_RATE))
history_object = model.fit_generator(train_generator,
            steps_per_epoch=math.ceil(len(train_samples)/batchsize),
            validation_data=validation_generator,
            validation_steps=math.ceil(len(validation_samples)/batchsize),
            epochs=num_epoch, verbose=1)

model.save('model.h5')
print("Model is made.")

#plot the graph for training rate and valiation rate
print(history_object.history.keys())
plt.plot(history_object.history['loss'])
plt.plot(history_object.history['val_loss'])
plt.title('model mean squared error loss')
plt.ylabel('mean squared error loss')
plt.xlabel('epoch')
plt.legend(['training set', 'validation set'], loc='upper right')
#plt.show()
plt.savefig("result.png")

print("Show the graph.")

exit()
```

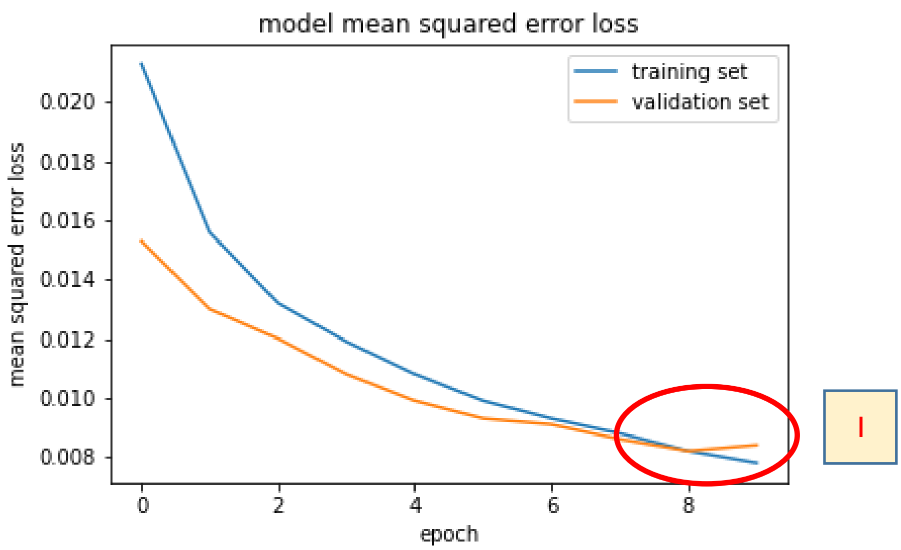
(3)Result

The following is the console log of the Udacity Workspace when I ran the final set of parameters and Datasets.

```
BASH
 K80, pci bus id: 0000:00:04.0)
524/524 [==============================] - 58s 111ms/step - loss: 0.0213 - val_loss: 0.0153
Epoch 2/10
524/524 [==============================] - 55s 105ms/step - loss: 0.0156 - val_loss: 0.0130
Epoch 3/10
524/524 [==============================] - 55s 105ms/step - loss: 0.0132 - val_loss: 0.0120
Epoch 4/10
524/524 [==============================] - 55s 105ms/step - loss: 0.0119 - val_loss: 0.0108
Epoch 5/10
524/524 [==============================] - 56s 107ms/step - loss: 0.0108 - val_loss: 0.0099
Epoch 6/10
524/524 [==============================] - 55s 106ms/step - loss: 0.0099 - val_loss: 0.0093
Epoch 7/10
524/524 [==============================] - 55s 106ms/step - loss: 0.0093 - val_loss: 0.0091
Epoch 8/10
524/524 [==============================] - 55s 106ms/step - loss: 0.0088 - val_loss: 0.0086
Epoch 9/10
524/524 [==============================] - 55s 106ms/step - loss: 0.0082 - val_loss: 0.0082
Epoch 10/10
524/524 [==============================] - 55s 106ms/step - loss: 0.0078 - val_loss: 0.0084
Model is made.
dict_keys(['val_loss', 'loss'])
^CTraceback (most recent call last):
```

The resulting graph is shown below. The vertical axis means Error Loss.

Loss of Training Dataset keeps decreasing, and Loss of Validation Dataset became saturate at the 9th and 10th epoch. It looks good timing to finish the learning (I).



In the console of Workspace, I ran “python drive.py model.h5 <folder name>” and I ran the Simulation with Autonomous Mode. After that, I ran “python video.py <folder name>” and created **video.mp4** file.

(* Before running video.py, I had to install “ffmpeg” in the Workspace.)

The vehicle drove well in the track 1. There was one time it drove very near from the right line, but it recovered soon thanks to the “Recovery Lap” training data. I think it was able to recover faster if I add more “Recovery Lap” training data.