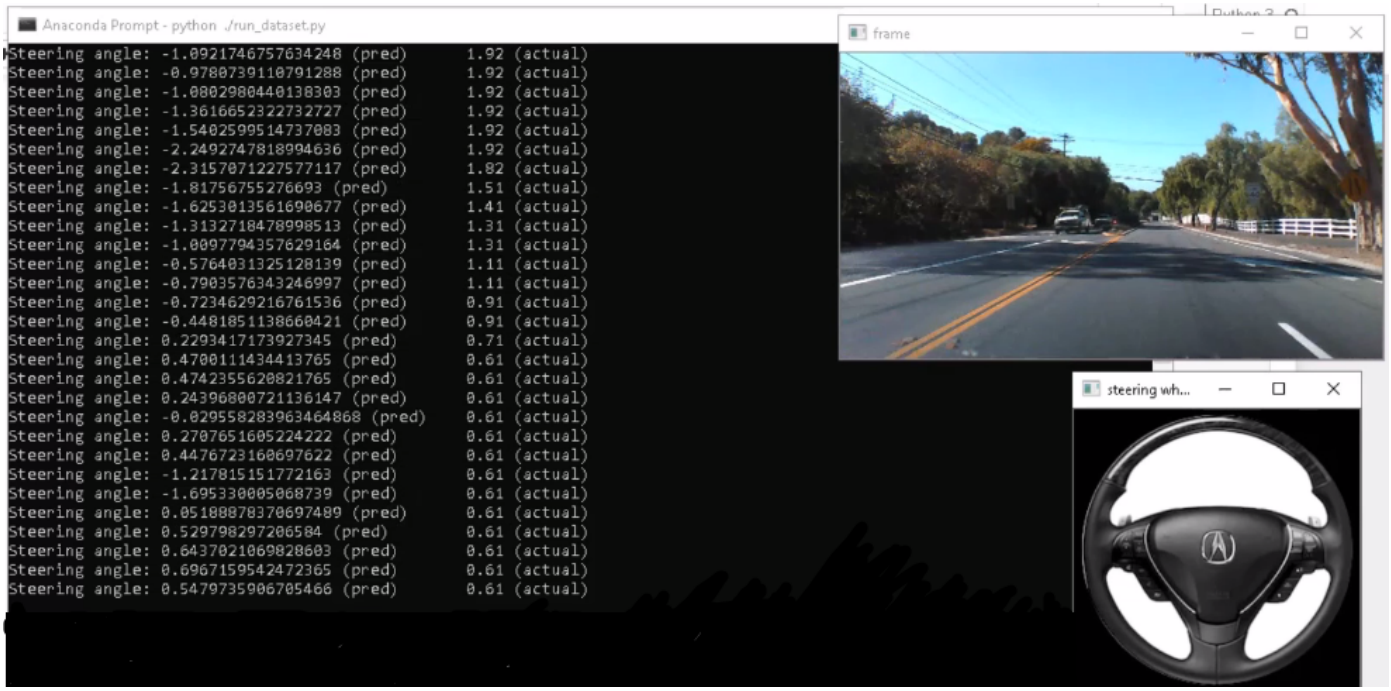
Document Created Date:12-03-2019

# Self Driving Car - Case Study




## Dataset

- Refer this: https://github.com/SullyChen/Autopilot-TensorFlow (https://github.com/SullyChen/Autopilot-TensorFlow)
- There are total 45406 images in the dataset along with their steering angles. We will split the dataset into train and test in a ratio of 70:30 sequentially.

## Objective

- Our objective is to predict the correct steering angle from the given test image of the road. Here, our loss is Mean Squared Error(MSE). Our goal is to reduce the MSE error as low as possible.

## Prerequisites

- we need to installed following softwares and libraries in our machine before running this project.
- Python 3: https://www.python.org/downloads/ (https://www.python.org/downloads/)
- Anaconda: It will install ipython notebook and most of the libraries which are needed like pandas, matplotlib, numpy and scipy: https://www.anaconda.com/download/ (https://www.anaconda.com/download/)
- Libraries:
- Tensorflow: It is a deep learning library.
- pip install tensorflow
- OpenCV: It is used for processing images.
- pip install opencv-python

In [2]:

```
# Credits: https://github.com/SullyChen/Autopilot-TensorFlow
# Research paper: End to End Learning for Self-Driving Cars by Nvidia. [https://arxiv.o

# NVidia dataset: 72 hrs of video => 72*60*60*30 = 7,776,000 images
# Nvidia blog: https://devblogs.nvidia.com/deep-learning-self-driving-cars/


# Our Dataset: https://github.com/SullyChen/Autopilot-TensorFlow [https://drive.google.
# Size: 25 minutes = 25*60*30 = 45,000 images ~ 2.3 GB


# If you want to try on a slightly large dataset: 70 minutes of data ~ 223GB
# Refer: https://medium.com/udacity/open-sourcing-223gb-of-mountain-view-driving-data-f
# Format: Image, latitude, longitude, gear, brake, throttle, steering angles and speed



# Additional Installations:
# pip3 install h5py


# AWS: https://aws.amazon.com/blogs/machine-learning/get-started-with-deep-learning-usi

# Youtube:https://www.youtube.com/watch?v=qhUvQiKec2U
# Further reading and extensions: https://medium.com/udacity/teaching-a-machine-to-steer
# More data: https://medium.com/udacity/open-sourcing-223gb-of-mountain-view-driving-da
```

## 1. Train Test Split (70:30)

In [1]:

```
# read images and steering angles from driving_dataset folder

from __future__ import division

import os
import numpy as np
import random

from scipy import pi
from itertools import islice



DATA_FOLDER = './driving_dataset/' # change this to your folder
TRAIN_FILE = os.path.join(DATA_FOLDER, 'data.txt')


split =0.7
X = []
y = []
with open(TRAIN_FILE) as fp:
    for line in fp:
        path, angle = line.strip().split()
        full_path = os.path.join(DATA_FOLDER, path)
        X.append(full_path)

        # converting angle from degrees to radians
        y.append(float(angle) * pi / 180 )


y = np.array(y)
print("Completed processing data.txt")

split_index = int(len(y)*0.7)

train_y = y[:split_index]
test_y = y[split_index:]
```

```
Completed processing data.txt
```

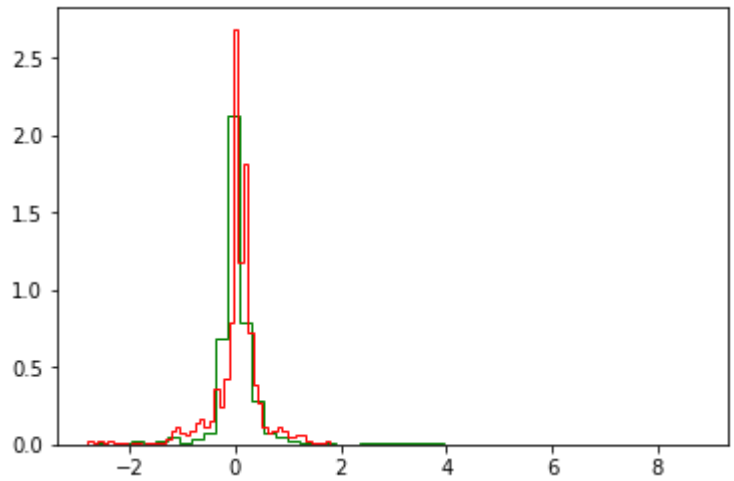
In [20]:

```
import scipy.misc
import random

xs = []
ys = []

#points to the end of the last batch
train_batch_pointer = 0
val_batch_pointer = 0

#read data.txt
with open("driving_dataset/data.txt") as f:
    for line in f:
        xs.append("driving_dataset/" + line.split()[0])
        #the paper by Nvidia uses the inverse of the turning radius,
        #but steering wheel angle is proportional to the inverse of turning radius
        #so the steering wheel angle in radians is used as the output
        ys.append(float(line.split()[1]) * scipy.pi / 180)

#get number of images
num_images = len(xs)


train_xs = xs[:int(len(xs) * 0.7)]
train_ys = ys[:int(len(xs) * 0.7)]

val_xs = xs[-int(len(xs) * 0.3):]
val_ys = ys[-int(len(xs) * 0.3):]

num_train_images = len(train_xs)
num_val_images = len(val_xs)

def LoadTrainBatch(batch_size):
    global train_batch_pointer
    x_out = []
    y_out = []
    for i in range(0, batch_size):
        x_out.append(scipy.misc.imresize(scipy.misc.imread(train_xs[(train_batch_pointe
        y_out.append([train_ys[(train_batch_pointer + i) % num_train_images]])
    train_batch_pointer += batch_size
    return x_out, y_out

def LoadValBatch(batch_size):
    global val_batch_pointer
    x_out = []
    y_out = []
    for i in range(0, batch_size):
        x_out.append(scipy.misc.imresize(scipy.misc.imread(val_xs[(val_batch_pointer +
        y_out.append([val_ys[(val_batch_pointer + i) % num_val_images]])
    val_batch_pointer += batch_size
    return x_out, y_out
```

In [3]:

```
import warnings
warnings.filterwarnings("ignore")
import numpy;

# PDF of train and test 'y' values.
import matplotlib.pyplot as plt
plt.hist(train_y, bins=50, normed=1, color='green', histtype ='step');
plt.hist(test_y, bins=50, normed=1, color='red', histtype ='step');
plt.show()
```

```
C:\Users\ramesh\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462:
UserWarning: The 'normed' kwarg is deprecated, and has been replaced by th
e 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```




In [4]:

```
#Model 0: Base line Model: y_test_pred = mean(y_train_i)
del np
import numpy as np
train_mean_y = np.mean(train_y)

print('Test_MSE(MEAN):%f' % np.mean(np.square(test_y-train_mean_y)) )

print('Test_MSE(ZERO):%f' % np.mean(np.square(test_y-0.0)) )
```

```
Test_MSE(MEAN):0.241561
Test_MSE(ZERO):0.241107
```

## Model Code:

In [12]:

```
# model.py
import tensorflow as tf
import scipy

def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

def conv2d(x, W, stride):
  return tf.nn.conv2d(x, W, strides=[1, stride, stride, 1], padding='VALID')

x = tf.placeholder(tf.float32, shape=[None, 66, 200, 3])
y_ = tf.placeholder(tf.float32, shape=[None, 1])

x_image = x

#first convolutional layer
W_conv1 = weight_variable([5, 5, 3, 24])
b_conv1 = bias_variable([24])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1, 2) + b_conv1)

#second convolutional layer
W_conv2 = weight_variable([5, 5, 24, 36])
b_conv2 = bias_variable([36])

h_conv2 = tf.nn.relu(conv2d(h_conv1, W_conv2, 2) + b_conv2)

#third convolutional layer
W_conv3 = weight_variable([5, 5, 36, 48])
b_conv3 = bias_variable([48])

h_conv3 = tf.nn.relu(conv2d(h_conv2, W_conv3, 2) + b_conv3)

#fourth convolutional layer
W_conv4 = weight_variable([3, 3, 48, 64])
b_conv4 = bias_variable([64])

h_conv4 = tf.nn.relu(conv2d(h_conv3, W_conv4, 1) + b_conv4)

#fifth convolutional layer
W_conv5 = weight_variable([3, 3, 64, 64])
b_conv5 = bias_variable([64])

h_conv5 = tf.nn.relu(conv2d(h_conv4, W_conv5, 1) + b_conv5)

#Fully Connected Layer 1
W_fc1 = weight_variable([1152, 1164])
b_fc1 = bias_variable([1164])

h_conv5_flat = tf.reshape(h_conv5, [-1, 1152])
h_fc1 = tf.nn.relu(tf.matmul(h_conv5_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

```
#ully Connected Layer 2
W_fc2 = weight_variable([1164, 100])
b_fc2 = bias_variable([100])

h_fc2 = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)

#ully Connected Layer 3
W_fc3 = weight_variable([100, 50])
b_fc3 = bias_variable([50])

h_fc3 = tf.nn.relu(tf.matmul(h_fc2_drop, W_fc3) + b_fc3)

h_fc3_drop = tf.nn.dropout(h_fc3, keep_prob)

#FCL 4
W_fc4 = weight_variable([50, 10])
b_fc4 = bias_variable([10])

h_fc4 = tf.nn.relu(tf.matmul(h_fc3_drop, W_fc4) + b_fc4)

h_fc4_drop = tf.nn.dropout(h_fc4, keep_prob)

#Output
W_fc5 = weight_variable([10, 1])
b_fc5 = bias_variable([1])

#y = tf.multiply(tf.atan(tf.matmul(h_fc4_drop, W_fc5) + b_fc5), 2) #scale the atan outpu
y = tf.multiply((tf.matmul(h_fc4_drop, W_fc5) + b_fc5), 2) #scale the atan output
```

In [10]:

```
%%time
import tensorflow as tf
from tensorflow.core.protobuf import saver_pb2
import driving_data
import model


LOGDIR = './My Final Save'

sess = tf.InteractiveSession()


L2NormConst = 0.001

train_vars = tf.trainable_variables()

loss = tf.reduce_mean(tf.square(tf.subtract(model.y_, model.y))) + tf.add_n([tf.nn.l2_l
train_step = tf.train.AdamOptimizer(1e-4).minimize(loss)
sess.run(tf.initialize_all_variables())

# create a summary to monitor cost tensor
tf.summary.scalar("loss", loss)
# merge all summaries into a single op
merged_summary_op =  tf.summary.merge_all()

saver = tf.train.Saver(write_version = saver_pb2.SaverDef.V1)

# op to write logs to Tensorboard
logs_path = './logs'
summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_graph())

epochs = 30
batch_size = 100

# train over the dataset about 30 times
for epoch in range(epochs):
  for i in range(int(driving_data.num_images/batch_size)):
    xs, ys = driving_data.LoadTrainBatch(batch_size)
    train_step.run(feed_dict={model.x: xs, model.y_: ys, model.keep_prob: 0.5})
    if i % 10 == 0:
      xs, ys = driving_data.LoadValBatch(batch_size)
      loss_value = loss.eval(feed_dict={model.x:xs, model.y_: ys, model.keep_prob: 1.0}
      print("Epoch: %d, Step: %d, Loss: %g" % (epoch, epoch * batch_size + i, loss_valu

    # write logs at every iteration
    summary = merged_summary_op.eval(feed_dict={model.x:xs, model.y_: ys, model.keep_pr
    summary_writer.add_summary(summary, epoch * driving_data.num_images/batch_size + i)

    if i % batch_size == 0:
      if not os.path.exists(LOGDIR):
        os.makedirs(LOGDIR)
      checkpoint_path = os.path.join(LOGDIR, "model.ckpt")
      filename = saver.save(sess, checkpoint_path)
  print("Model saved in file: %s" % filename)
print("Run the command line:\n" \
          "--> tensorboard --logdir=./logs " \
          "\nThen open http://0.0.0.0:6006/ into your web browser")
```

```
Epoch: 0, Step: 0, Loss: 7.75127
WARNING:tensorflow:**************************************************
**
WARNING:tensorflow:TensorFlow's V1 checkpoint format has been deprecate
d.
WARNING:tensorflow:Consider switching to the more efficient V2 format:
WARNING:tensorflow:   `tf.train.Saver(write_version=tf.train.SaverDef.V
2)`
WARNING:tensorflow:now on by default.
WARNING:tensorflow:**************************************************
**
Epoch: 0, Step: 10, Loss: 6.20833
Epoch: 0, Step: 20, Loss: 6.05425
Epoch: 0, Step: 30, Loss: 6.02217
Epoch: 0, Step: 40, Loss: 5.97837
Epoch: 0, Step: 50, Loss: 5.94729
Epoch: 0, Step: 60, Loss: 5.89532
Epoch: 0, Step: 70, Loss: 5.8629
Epoch: 0, Step: 80, Loss: 5.83926
```

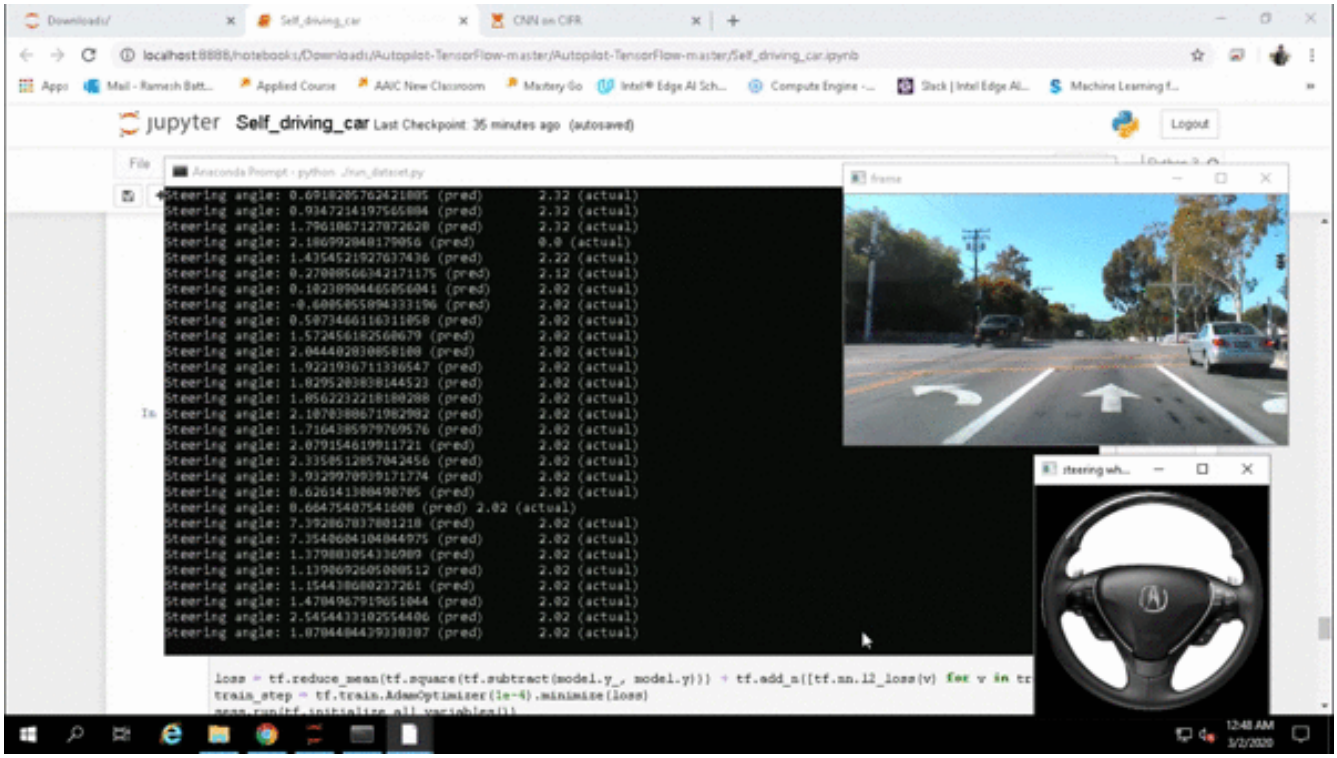
---

- **I build a minimal version of self driving car. Here, I have a front camera view. This will transfer input to the computer.**
- **Then Deep Learning algorithm in computer predicts the steering angle to avoid all sorts of collisions.**
- **Predicting steering angle can be thought of as a regression problem.**
- **I will feed images to Convolutional Neural Network and the label will be the steering angle in that image.**
- **Model will learn the steering angle from the as per the turns in the image and will finally predicts steering angle for unknown images.**

---

## Final Results in video

---




# Conclusions:

- I have splitted the data into train and test by 70%,30%
- Then i have used Adam optimizer with 10^-4 Lr
- dropout has been changed to 0.5
- For the activation function, instead of giving it to a linear function,after multiplying with weights ,i took that directly as output without passing it to any activation function and got good results.

## Thank You.

**Sign Off RAMESH BATTU** (https://www.linkedin.com/in/rameshbattuai/)