In [ ]:

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

# 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.com/
# 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-f6b55
# 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-using-t

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

In [ ]:

```
# Importing required libraries

import warnings
warnings.filterwarnings("ignore")

import os
import random
import cv2
import math
import numpy as np
import scipy
import scipy.misc
from scipy import pi
from subprocess import call
from datetime import datetime
from itertools import islice
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [3]:

```
# Installing Required Libraries used for unpackin rar file
!pip install pyunpack
!pip install patool
```

```
Collecting pyunpack
  Downloading https://files.pythonhosted.org/packages/79/dc/44cd41fb99d184ae
7c2eac439a52ca624d5ece62b0302c3437fcc4ce3b58/pyunpack-0.1.2.tar.gz (https://
files.pythonhosted.org/packages/79/dc/44cd41fb99d184ae7c2eac439a52ca624d5ece
62b0302c3437fcc4ce3b58/pyunpack-0.1.2.tar.gz)
Collecting easyprocess (from pyunpack)
  Downloading https://files.pythonhosted.org/packages/45/3a/4eecc0c7995a13a6
4739bbedc0d3691fc574245b7e79cff81905aa0c2b38/EasyProcess-0.2.5.tar.gz (http
s://files.pythonhosted.org/packages/45/3a/4eecc0c7995a13a64739bbedc0d3691fc5
74245b7e79cff81905aa0c2b38/EasyProcess-0.2.5.tar.gz)
Building wheels for collected packages: pyunpack, easyprocess
  Building wheel for pyunpack (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/af/44/08/60613970881e542c0baa
d1f2dea5ed8e6716bc573f49197b7e
  Building wheel for easyprocess (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/41/22/19/af15ef6264c58b625a82
641ed7483ad05e258fbd8925505227
Successfully built pyunpack easyprocess
Installing collected packages: easyprocess, pyunpack
Successfully installed easyprocess-0.2.5 pyunpack-0.1.2
Collecting patool
  Downloading https://files.pythonhosted.org/packages/43/94/52243ddff508780d
d2d8110964320ab4851134a55ab102285b46e740f76a/patool-1.12-py2.py3-none-any.wh
l (https://files.pythonhosted.org/packages/43/94/52243ddff508780dd2d81109643
20ab4851134a55ab102285b46e740f76a/patool-1.12-py2.py3-none-any.whl) (77kB)
    100% |████████████████████████████████| 81kB 3.8MB/s
Installing collected packages: patool
Successfully installed patool-1.12
```

In [ ]:

```
# Creating a Directiory to store unpacked dataset

os.mkdir("Driving Data")
```

In [ ]:

```
from pyunpack import Archive
Archive('drive/My Drive/Autopilot-TensorFlow-master.rar').extractall('Driving Data')
```

In [ ]:

```
# Dataset Preparation

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 Data/Autopilot-TensorFlow-master/Autopilot-TensorFlow-master/driving_dat
    for line in f:
        xs.append("Driving Data/Autopilot-TensorFlow-master/Autopilot-TensorFlow-master/dri
        #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)] # splitting data into 70:30 ratio, as per the task assig
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_pointer +
        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 + i) %
        y_out.append([val_ys[(val_batch_pointer + i) % num_val_images]])
    val_batch_pointer += batch_size
    return x_out, y_out
```

In [9]:

```
print(num_train_images)
print(num_val_images)
```

```
31784
13621
```

1. Activation Used -- Identity
2. Splitted Data into 70 : 30
3. Optimizer Used Adam(learning rate = 1e-3)
4. Dropout 0.5

In [ ]:

```
# Model Architecture

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)

#FCL 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)

#FCL 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)

#FCL 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 3
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.identity(tf.matmul(h_fc4_drop, W_fc5) + b_fc5), 2) #scale the atan outpu
```

In [14]:

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

LOGDIR = '/content/drive/My Drive/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_loss(
train_step = tf.train.AdamOptimizer(1e-3).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.V2)

# 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 27 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_value)

    # write logs at every iteration
    summary = merged_summary_op.eval(feed_dict={model.x:xs, model.y_: ys, model.keep_prob:
    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: 12.5401
Epoch: 0, Step: 10, Loss: 12.7512
Epoch: 0, Step: 20, Loss: 12.4309
Epoch: 0, Step: 30, Loss: 12.419
Epoch: 0, Step: 40, Loss: 12.6505
Epoch: 0, Step: 50, Loss: 12.3569
Epoch: 0, Step: 60, Loss: 12.4944
Epoch: 0, Step: 70, Loss: 12.7282
Epoch: 0, Step: 80, Loss: 12.6205
Epoch: 0, Step: 90, Loss: 12.395
Epoch: 0, Step: 100, Loss: 12.3711
Epoch: 0, Step: 110, Loss: 12.3539
Epoch: 0, Step: 120, Loss: 12.3955
Epoch: 0, Step: 130, Loss: 12.6471
Epoch: 0, Step: 140, Loss: 13.0785
Epoch: 0, Step: 150, Loss: 12.5807
Epoch: 0, Step: 160, Loss: 13.2772
Epoch: 0, Step: 170, Loss: 12.3556
Epoch: 0, Step: 180, Loss: 12.5243
```

In [ ]:

```
degrees_predicted = []
for i in range(len(val_xs)):
    full_image = scipy.misc.imread(val_xs[i], mode="RGB")
    image = scipy.misc.imresize(full_image[-150:], [66, 200]) / 255.0
    degrees = sess.run(y,feed_dict={x: [image], keep_prob: 1.0})[0][0] * 180.0 / scipy.pi
    #call("clear")
    #print("Predicted Steering angle: " + str(degrees))
    #print("Steering angle: " + str(degrees) + " (pred)\t" + str(val_ys[i]*180/scipy.pi) +
    #cv2.imshow("frame", cv2.cvtColor(full_image, cv2.COLOR_RGB2BGR))
    #make smooth angle transitions by turning the steering wheel based on the difference of
    #and the predicted angle
    #smoothed_angle += 0.2 * pow(abs((degrees - smoothed_angle)), 2.0 / 3.0) * (degrees - s
    #M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)
    #dst = cv2.warpAffine(img,M,(cols,rows))
    #cv2.imshow("steering wheel", dst)
    #i += 1
    degrees_predicted.append(degrees)
```

In [33]:

```
len(val_ys)
```

Out[33]:

```
13621
```

In [ ]:

```
import pandas as pd
```

In [ ]:

```
data = pd.DataFrame({"degrees":degrees_predicted,"original":[val_ys[i]*180/scipy.pi for i i
```

In [41]:

```
data.head()
```

Out[41]:

	degrees	original
0	-40.534075	-28.34
1	-36.923139	-28.84
2	-33.461322	-29.75
3	-35.143335	-31.06
4	-32.249397	-32.27

In [ ]:

```
data.to_csv("results_adam_linear.csv")
```

In [ ]:

```

```