Self Driving Car

Problem Definition

We are here building a minimal version of self driving car. Here, we 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. We 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.

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 80:20 **sequentially**.

In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import pi
import cv2
import scipy.misc
import tensorflow as tf
```

```
C:\Users\GauravP\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the second argument of i
ssubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).
type`.
  from ._conv import register_converters as _register_converters
```

1. Reading images from file

In [2]:
```
DATA_FOLDER = "../driving_dataset/"
DATA_FILE = os.path.join(DATA_FOLDER, "data.txt")

x = []
y = []

train_batch_pointer = 0
test_batch_pointer = 0
```

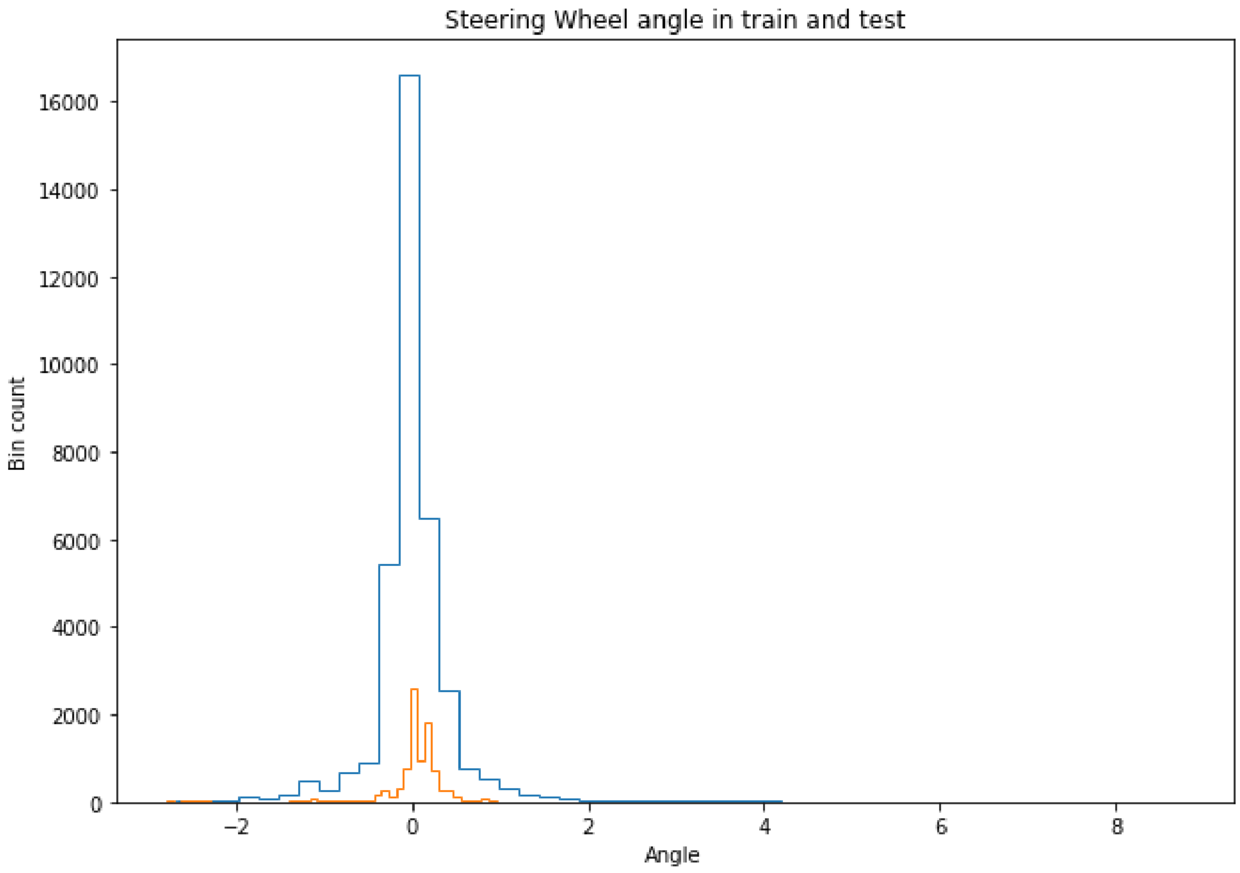
In [3]:
```
with open(DATA_FILE) as f:
    for line in f:
        image_name, angle = line.split()

        image_path = os.path.join(DATA_FOLDER, image_name)
        x.append(image_path)

        angle_radians = float(angle) * (pi / 180)  #converting angle into radians
        y.append(angle_radians)
y = np.array(y)
print(str(len(x))+" "+str(len(y)))
```

```
45406 45406
```

In [4]:
```
split_ratio = int(len(x) * 0.8)

train_x = x[:split_ratio]
train_y = y[:split_ratio]

test_x = x[split_ratio:]
test_y = y[split_ratio:]

len(train_x), len(train_y), len(test_x), len(test_y)
```

Out[4]:
```
(36324, 36324, 9082, 9082)
```

In [5]:
```
fig = plt.figure(figsize = (10, 7))
plt.hist(train_y, bins = 50, histtype = "step")
plt.hist(test_y, bins = 50, histtype = "step")
plt.title("Steering Wheel angle in train and test")
plt.xlabel("Angle")
plt.ylabel("Bin count")
plt.show()
```



Above histogram plot clearly shows that most of the values list on 0. This is obvious as well as most of the time car runs on straight road so therefore, steering wheel angle is 0 most of the time during driving.

2. Writing function for creating batch of images for training

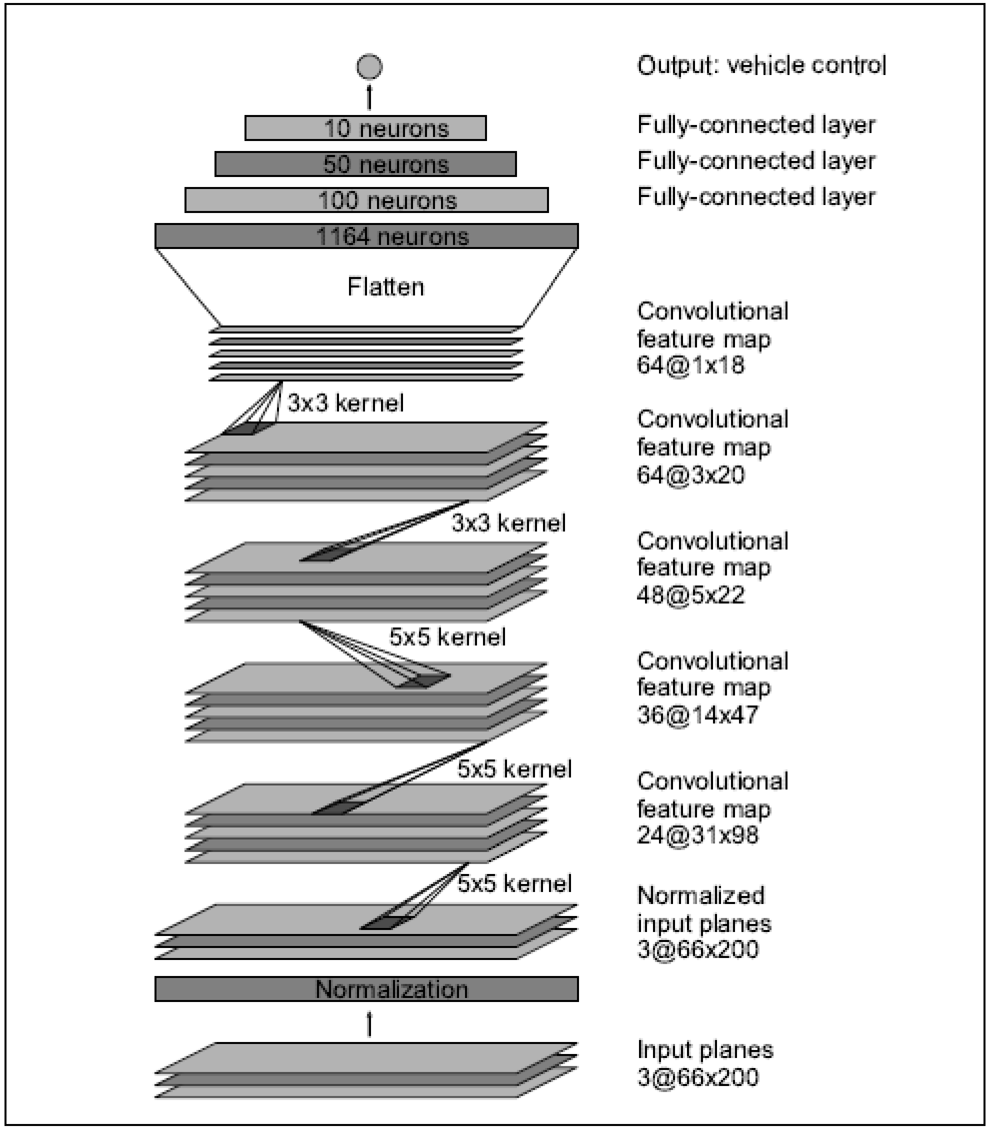
In [5]:
```
def loadTrainBatch(batch_size):
    global train_batch_pointer
    x_result = []
    y_result = []
    for i in range(batch_size):
        read_image = cv2.imread(train_x[(train_batch_pointer + i) % len(train_x)]) #here % len(train_x) is used to make s
        #"train_batch_pointer + i" should not cross the number of train images. As soon as the value of "train_batch_poin
        #equal to number of train images then it will again start reading the train images from the beginning means from
        #index onwards.
        read_image_road = read_image[-150:] #here, we are taking only the lower part of the images where there is a road
        #image. As, we are concern only with the curves of the road to predict angles so therefore, we are discarding the
        #part of the image. Hence, here -"150" is equivalent to the last 150 matrix pixels of the image.
        read_image_resize = cv2.resize(read_image_road, (200, 66)) #After, resizing, each image will be of size (66, 200,
        #now since we have kept only the last 150 matrices in the image so the size of our image is now (150, 455, 3).
        #Now 455/150 = 3.0303. Also 200/66 = 3.0303. Hence, here we are keeping the aspect ratio of images same.
        read_image_final = read_image_resize/255.0  #here, we are normalizing the images

        x_result.append(read_image_final) #finally appending the image pixel matrix

        y_result.append(train_y[(train_batch_pointer + i) % len(train_y)]) #appending corresponding labels

    train_batch_pointer += batch_size

    return x_result, y_result
```

In [6]:
```
def loadTestBatch(batch_size):
    global test_batch_pointer
    x_result = []
    y_result = []
    for i in range(batch_size):
        read_image = cv2.imread(test_x[(test_batch_pointer + i) % len(test_x)]) #here % len(test_x) is used to make sure
        #"test_batch_pointer + i" should not cross the number of test images. As soon as the value of "test_batch_pointer
        #equal to number of test images then it will again start reading the test images from the beginning means from 0t
        #index onwards.
        read_image_road = read_image[-150:] #here, we are taking only the lower part of the images where there is a road
        #image. As, we are concern only with the curves of the road to predict angles so therefore, we are discarding the
        #part of the image. Hence, here -"150" is equivalent to the last 150 matrix pixels of the image.
        read_image_resize = cv2.resize(read_image_road, (200, 66)) #After, resizing, each image will be of size (66, 200,
        #now since we have kept only the last 150 matrices in the image so the size of our image is now (150, 455, 3).
        #Now 455/150 = 3.0303. Also 200/66 = 3.0303. Hence, here we are keeping the aspect ratio of images same.
        read_image_final = read_image_resize/255.0  #here, we are normalizing the images

        x_result.append(read_image_final) #finally appending the image pixel matrix

        y_result.append(test_y[(test_batch_pointer + i) % len(test_y)]) #appending corresponding labels

    test_batch_pointer += batch_size

    return x_result, y_result
```

3. Making Model Architecture

In [7]:
```
def weightVariable(shape):
    initial = tf.truncated_normal(shape = shape, stddev = 0.1)
    return tf.Variable(initial)

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

def convolution(previous_input, filter_input, strides):
    return tf.nn.conv2d(previous_input, filter_input, strides = [1, strides, strides, 1], padding = "VALID")
```

In [8]:
```
x_input = tf.placeholder(tf.float32, shape = [None, 66, 200, 3], name = "Plc_1")
y_true = tf.placeholder(tf.float32, name = "Plc_2")

input_image = x_input
```



In [9]:
```
#Convolution Layers
#First convolution layer
W_Conv1 = weightVariable([5,5,3,24])
B_Conv1 = bias_variable([24])
Conv1 = tf.nn.relu(convolution(input_image, W_Conv1, 2) + B_Conv1)
#strides = 2
#Output size: 31*98*24

#Second convolution layer
W_Conv2 = weightVariable([5,5,24,36])
B_Conv2 = bias_variable([36])
Conv2 = tf.nn.relu(convolution(Conv1, W_Conv2, 2) + B_Conv2)
#strides = 2
#Output size: 14*47*36

#Third convolution layer
W_Conv3 = weightVariable([5,5,36,48])
B_Conv3 = bias_variable([48])
Conv3 = tf.nn.relu(convolution(Conv2, W_Conv3, 2) + B_Conv3)
#strides = 2
#Output size: 5*22*48

#Fourth convolution layer
W_Conv4 = weightVariable([3,3,48,64])
B_Conv4 = bias_variable([64])
Conv4 = tf.nn.relu(convolution(Conv3, W_Conv4, 1) + B_Conv4)
#strides = 1
#Output size: 3*20*64


#Fifth convolution layer
W_Conv5 = weightVariable([3,3,64,64])
B_Conv5 = bias_variable([64])
Conv5 = tf.nn.relu(convolution(Conv4, W_Conv5, 1) + B_Conv5)
#strides = 1
#Output size: 1*18*64

#Fully-Connected Dense Layers
keep_prob = tf.placeholder(tf.float32)
#First FC-Dense
#Input = 1*18*64 = 1152
```

```
W_FC1 = weightVariable([1152, 1164])
B_FC1 = bias_variable([1164])
FC1_Flatten = tf.reshape(Conv5, [-1, 1152]) #here, -1 indicates 1. It means that the shape of FC1_Flatten will be 1*1152
Output_FC1 = tf.nn.relu(tf.matmul(FC1_Flatten, W_FC1) + B_FC1) #so, here shape of FC1_Flatten is 1*1152 and shape of W_FC
#be 1152*1164. Therefore, there will be a matrix multiplication of matrices: (1*1152) * (1152*1164) = (1*1164).
Output_FC1_drop = tf.nn.dropout(Output_FC1, keep_prob)

#Second FC-Dense
#Input = 1*1164 = 1164
W_FC2 = weightVariable([1164, 100])
B_FC2 = bias_variable([100])
Output_FC2 = tf.nn.relu(tf.matmul(Output_FC1_drop, W_FC2) + B_FC2) #so, here shape of Output_FC1_drop is 1*1164 and shape
#W_FC2 will be 1164*100. Therefore, there will be a matrix multiplication of matrices: (1*1164) * (1164*100) = (1*100).
Output_FC2_drop = tf.nn.dropout(Output_FC2, keep_prob)

#Third FC-Dense
#Input = 1*100 = 100
W_FC3 = weightVariable([100, 50])
B_FC3 = bias_variable([50])
Output_FC3 = tf.nn.relu(tf.matmul(Output_FC2_drop, W_FC3) + B_FC3) #so, here shape of Output_FC2_drop is 1*100 and shape
#W_FC3 will be 100*50. Therefore, there will be a matrix multiplication of matrices: (1*100) * (100*50) = (1*50).
Output_FC3_drop = tf.nn.dropout(Output_FC3, keep_prob)

#Fourth FC-Dense
#Input = 1*50 = 50
W_FC4 = weightVariable([50, 10])
B_FC4 = bias_variable([10])
Output_FC4 = tf.nn.relu(tf.matmul(Output_FC3_drop, W_FC4) + B_FC4) #so, here shape of Output_FC3_drop is 1*50 and shape o
#W_FC4 will be 50*10. Therefore, there will be a matrix multiplication of matrices: (1*50) * (50*10) = (1*10).
Output_FC4_drop = tf.nn.dropout(Output_FC4, keep_prob)

#Final Output to one neuron with linear/identity function
#Input = 1*10 = 10
W_FC5 = weightVariable([10, 1])
B_FC5 = bias_variable([1])
y_predicted = tf.identity(tf.matmul(Output_FC4_drop, W_FC5) + B_FC5) #so, here shape of Output_FC4_drop is 1*10 and shape
#W_FC5 will be 10*1. Therefore, there will be a matrix multiplication of matrices: (1*10) * (10*1) = (1*1). Since, this i
#regression problem so we have applied identity fuction in the end. We can also apply "atan" function here. If computatio
#power is available then the model should be tested with both identity and atan functions. In the end, that function shou
#considered which gives better result.
```

4. Training the model

In [31]:

```
SAVEDIR = "../Saver/"
sess = tf.InteractiveSession()

L2NormConst = 0.001
train_vars = tf.trainable_variables() #it will return all the variables. Here, all the weights and biases are variables w
#are trainable.

loss = tf.reduce_mean(tf.square(tf.subtract(y_true, y_predicted))) + tf.add_n([tf.nn.l2_loss(w) for w in train_vars]) * L
#since this is a regression problem so above loss is mean-squared-error loss
train_step = tf.train.AdamOptimizer(learning_rate = 10**-4).minimize(loss)
sess.run(tf.global_variables_initializer())

saver = tf.train.Saver()

epochs = 30
batch_size = 100
epoch_number, train_loss, test_loss,  = [], [], []

for epoch in range(epochs):
    train_avg_loss = 0
    test_avg_loss = 0
    te_loss_old = 10000  #any big number can be given

    for i in range(int(len(x)/batch_size)):
        train_batch_x, train_batch_y = loadTrainBatch(batch_size)
        train_step.run(feed_dict = {x_input: train_batch_x, y_true: train_batch_y, keep_prob: 0.8})
        tr_loss = loss.eval(feed_dict = {x_input: train_batch_x, y_true: train_batch_y, keep_prob: 1.0})
        train_avg_loss += tr_loss / batch_size

        test_batch_x, test_batch_y = loadTestBatch(batch_size)
        te_loss_new = loss.eval(feed_dict = {x_input: test_batch_x, y_true: test_batch_y, keep_prob: 1.0})
        test_avg_loss += te_loss_new / batch_size

        if te_loss_new < te_loss_old:
            print("Epoch: {}, Train_Loss: {}, Test_Loss: {} *".format(epoch+1, tr_loss, te_loss_new))
        else:
            print("Epoch: {}, Train_Loss: {}, Test_Loss: {}".format(epoch+1, tr_loss, te_loss_new))
        te_loss_old = te_loss_new

        if (i+1) % batch_size == 0:
            if not os.path.exists(SAVEDIR):
```

```
                os.makedirs(SAVEDIR)
            save_path = os.path.join(SAVEDIR, "model.ckpt")
            saver.save(sess = sess, save_path = save_path)
            print("Model saved at location {} at epoch {}".format(save_path, epoch + 1))

    epoch_number.append(epoch)
    train_loss.append(train_avg_loss)
    test_loss.append(test_avg_loss)

#creating dataframe and record all the losses and accuracies at each epoch
log_frame = pd.DataFrame(columns = ["Epoch", "Train Loss", "Test Loss"])
log_frame["Epoch"] = epoch_number
log_frame["Train Loss"] = train_loss
log_frame["Test Loss"] = test_loss
log_frame.to_csv(os.path.join(SAVEDIR, "log.csv"), index = False)
```

In [13]:
```
frame = pd.read_csv(os.path.join(SAVEDIR, "log.csv"))
frame
```

Out[13]:

	Epoch	Train Loss	Test Loss
0	0	25.441976	24.565600
1	1	18.587419	18.186967
2	2	14.609151	14.172622
3	3	11.957517	11.540807
4	4	10.433561	9.667840
5	5	8.524213	8.287085
6	6	7.396868	7.110160
7	7	6.448679	6.113556
8	8	5.962348	5.384088
9	9	4.816765	4.692729
10	10	4.327291	4.103290
11	11	3.831057	3.698865
12	12	3.768895	3.492390
13	13	3.195319	3.057956
14	14	2.939912	2.794230
15	15	2.779324	2.687055
16	16	2.948726	2.751356
17	17	2.449967	2.380851
18	18	2.387339	2.277248
19	19	2.271806	2.301014
20	20	2.451325	2.404977
21	21	2.111570	2.093948
22	22	2.023531	2.142157
23	23	1.930803	2.180721

	Epoch	Train Loss	Test Loss
24	24	2.234076	2.202252
25	25	1.851828	2.028983
26	26	1.775510	2.035361
27	27	1.739461	1.888012
28	28	2.007302	2.063404
29	29	1.615547	1.970689

5. Making Predictions from the Model

In [14]:
```
sess = tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, "../Saver/model.ckpt")

img = cv2.imread('steering_wheel_image.jpg', 0) #here, second parameter '0' specifies that img.shape will return only hei
#width of the image and not the number of channels. It is a colored image so number of channels = 3, which it will not re
rows, cols = img.shape

i = 0
while(cv2.waitKey(60) != ord("q")):
    full_image = cv2.imread(test_x[i])
    cv2.imshow('Frame Window', full_image)
    image = ((cv2.resize(full_image[-150:], (200, 66)) / 255.0).reshape((1, 66, 200, 3)))
    degrees = sess.run(y_predicted, feed_dict = {x_input: image, keep_prob: 1.0})[0][0] *180 / pi #here, we have converte
    #predicted degrees from radians to degrees.
    M = cv2.getRotationMatrix2D((cols/2,rows/2), -degrees, 1) #this function rotate the image by a given degrees.
    dst = cv2.warpAffine(src = img, M = M, dsize = (cols, rows)) #warpAffine function applies rotation to the image
    cv2.imshow("Steering Wheel", dst)
    i += 1

cv2.destroyAllWindows()
```

```
INFO:tensorflow:Restoring parameters from ../Saver/model.ckpt

C:\Users\GauravP\Anaconda3\lib\site-packages\tensorflow\python\client\session.py:1711: UserWarning: An interactive sess
ion is already active. This can cause out-of-memory errors in some cases. You must explicitly call `InteractiveSession.
close()` to release resources held by the other session(s).
  warnings.warn('An interactive session is already active. This can '
```

Run the file "Visualize_Output.py" at command prompt to visualize the output better.