Downloading the dataset

In [0]:
```
!pip install pydrive
```

In [0]:
```
import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

In [0]:
```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:
```
# https://drive.google.com/open?id=1QiEgrF7OMNgAkoO4f7pm6vWpVb7Em2cW
download = drive.CreateFile({'id': '1QiEgrF7OMNgAkoO4f7pm6vWpVb7Em2cW'
})
download.GetContentFile("Autopilot-TensorFlow-master.zip")
```

In [ ]:
```
!unzip "Autopilot-TensorFlow-master.zip"
```

In [0]:
```
!pip install scipy==1.2.1 --user
```

```
Requirement already satisfied: scipy==1.2.1 in /root/.local/lib/python
3.6/site-packages (1.2.1)
Requirement already satisfied: numpy>=1.8.2 in /usr/local/lib/python3.
6/dist-packages (from scipy==1.2.1) (1.16.5)
```

In [0]:
```
pip install pillow
```

```
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-
```

```
packages (4.3.0)
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist
-packages (from pillow) (0.46)
```

In [0]:
```
# Checking if imread is working or not
import scipy.misc
scipy.misc.imread
```

Out[0]:
```
<function numpy.lib.utils._Deprecate.__call__.<locals>.newfunc>
```

In [0]:
```
pip install h5py
```

```
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-pa
ckages (2.8.0)
Requirement already satisfied: numpy>=1.7 in /usr/local/lib/python3.6/d
ist-packages (from h5py) (1.16.5)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-pac
kages (from h5py) (1.12.0)
```

In [0]:
```
# Credits: https://github.com/SullyChen/Autopilot-TensorFlow
# Research paper: End to End Learning for Self-Driving Cars by Nvidia.
 [https://arxiv.org/pdf/1604.07316.pdf]

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


# Our Dataset: https://github.com/SullyChen/Autopilot-TensorFlow [http
s://drive.google.com/file/d/0B-KJCaaF7elleG1RbzVPZWV4Tlk/view]
# 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-vie
w-driving-data-f6b5593fbfa5
# Format: Image, latitude, longitude, gear, brake, throttle, steering a
ngles and speed
```

```
# Additional Installations:
pip install h5py


# AWS: https://aws.amazon.com/blogs/machine-learning/get-started-with-d
eep-learning-using-the-aws-deep-learning-ami/

# Youtube:https://www.youtube.com/watch?v=qhUvQiKec2U
# Further reading and extensions: https://medium.com/udacity/teaching-a
-machine-to-steer-a-car-d73217f2492c
# More data: https://medium.com/udacity/open-sourcing-223gb-of-mountain
-view-driving-data-f6b5593fbfa5
```

Splitting the data and creating batches

Now we are going to split the dataset into 70-30 split, this will be a temporal split which means we will use the first 70% of the dataset (17.5 mins of driving data) for training our model and the remaining 30% of the dataset will be used for testing purposes.

We are also going to define to functions which will help us load batches of data from both train and validation datasets so that we can easily train our final model.

In [0]:
```
import scipy.misc
import random

xs = []
ys = []

#points to the end of the last batch
train_batch_pointer = 0
val_batch_pointer = 0
#content/Autopilot-TensorFlow-master
#read data.txt
with open("Autopilot-TensorFlow-master/driving_dataset/data.txt") as f:
```

```
    for line in f:
        xs.append("Autopilot-TensorFlow-master/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 tur
ning 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)

# Splitting the data
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[(tr
ain_batch_pointer + i) % num_train_images])[-150:], [66, 200]) / 255.0)
        y_out.append([train_ys[(train_batch_pointer + i) % num_train_im
ages]])
    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) % num_val_images])[-150:], [66, 200]) / 255.0)
        y_out.append([val_ys[(val_batch_pointer + i) % num_val_images
]])
    val_batch_pointer += batch_size
    return x_out, y_out
```

EDA

Now we will convert the steering angle from degree to radian unit and look at the distribution of the data, which means for the dataset how the steering angle has changed for both training and validation dataset.

The reason we are converting angles from degree to radian is that when we convert the angles, the range reduces to [-2, 2] as seen in the below pdf plot, this can be thought of as a form of normalization where we are reducing the range of values to easily train our model.

In [0]:

```
# 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

LIMIT = None

DATA_FOLDER = './Autopilot-TensorFlow-master/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 islice(fp, LIMIT):
        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
```

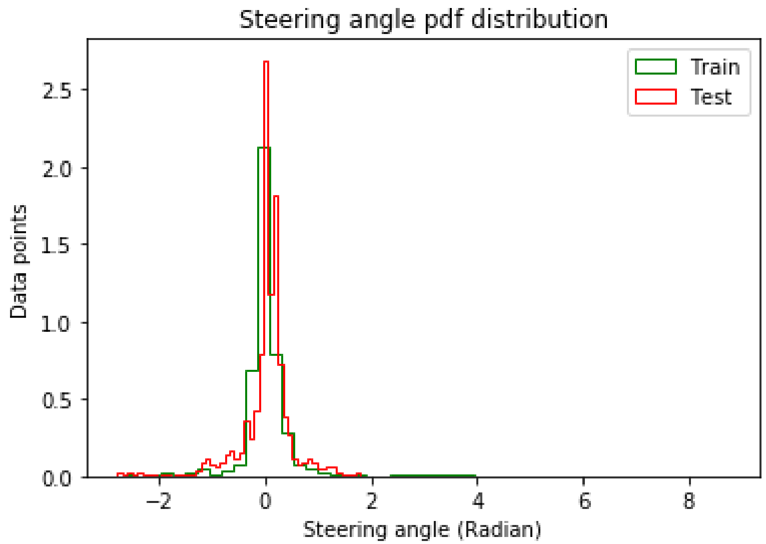
Now we will plot the steering angle pdf graph for both train and validation datasets.

In [0]:

```
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', label="Train");
plt.hist(test_y, bins=50, normed=1, color='red', histtype ='step', label="Test");
plt.xlabel("Steering angle (Radian)")
plt.ylabel("Data points")
plt.title("Steering angle pdf distribution")
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py:6521: M
atplotlibDeprecationWarning:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed
in 3.1. Use 'density' instead.
  alternative="'density'", removal="3.1")
```



From the above plot we can see that the most probable angle for the steering wheel is 0 for both train and test datasets. Which is normal as normally most of the time roads will be straight which means we will have the steering angle set to 0.

Create a simple baseline model

As we saw in the above plot that most of the time the steering angle is at 0 radian, so we can build a base line model with this finding.

We will create two baseline model, where for the first model we will simply return the mean of all the steering angle and for the other model we will return 0. Then we will measure the MSE of these two models to figure out our baseline.

In [0]:
```
#Model 0: Base line Model: y_test_pred = mean(y_train_i)
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
```

Building the CNN model

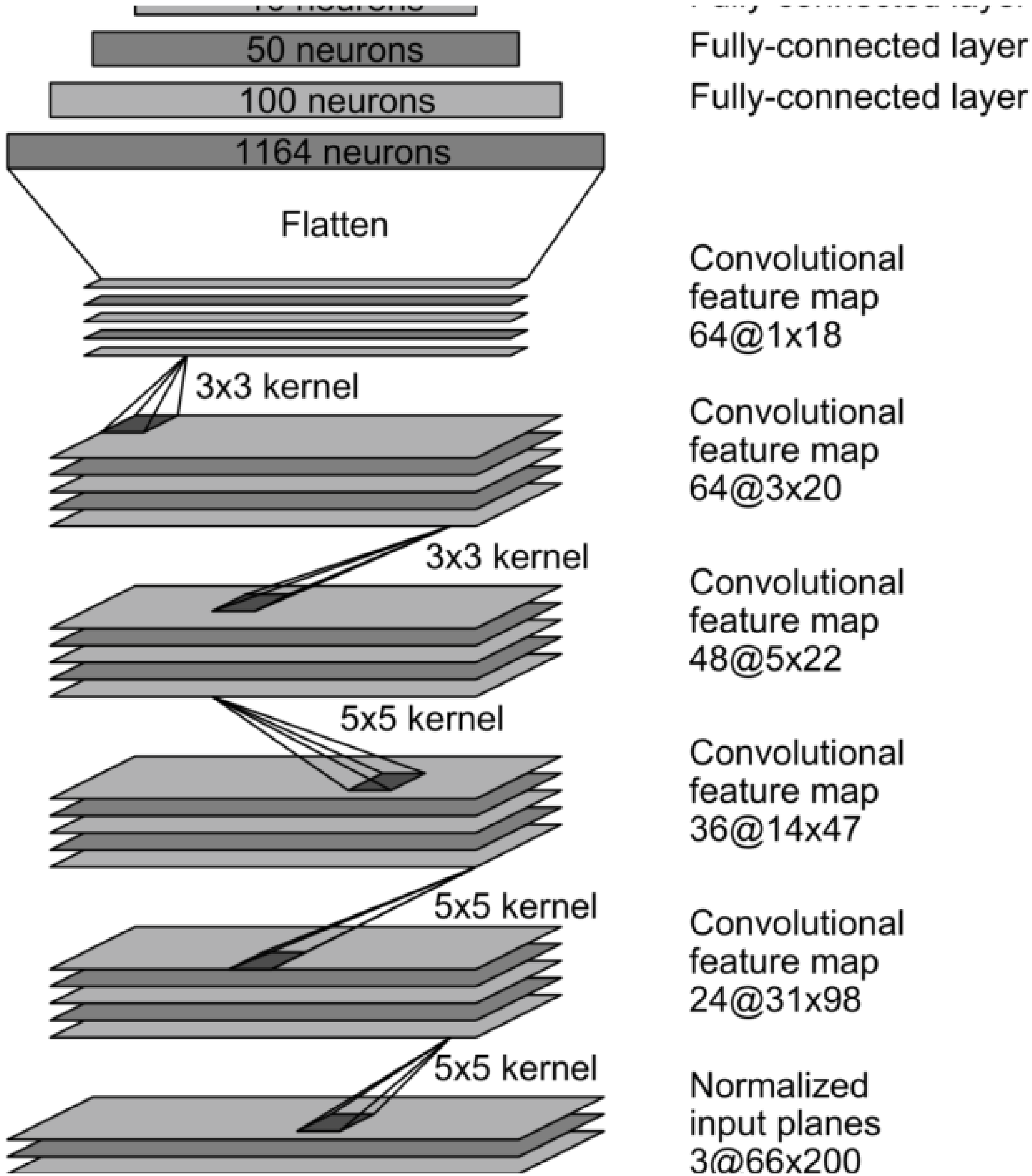
To create the model, we will be following NVDIA's research paper on self driving car.

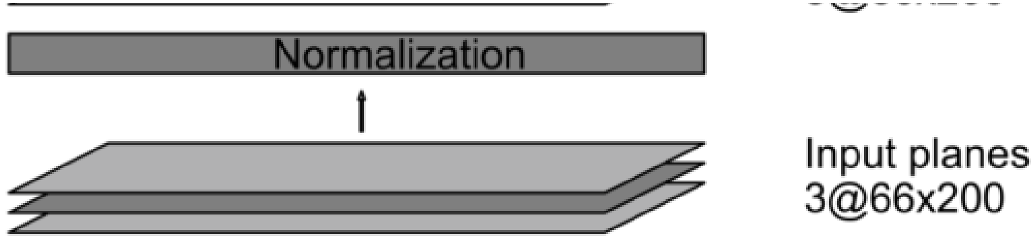
- Research Paper
- Developer's blog

The model we will be implementing is given in the below image, with slight modifications such as adding drop out layers in between the fully connected layers.

In [0]:
```
from google.colab.patches import cv2_imshow
!curl -o logo.png https://devblogs.nvidia.com/wp-content/uploads/2016/0
8/cnn-architecture-624x890.png
import cv2
img = cv2.imread('logo.png', cv2.IMREAD_UNCHANGED)
cv2_imshow(img)
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100  189k  100  189k    0     0  2081k      0 --:--:-- --:--:-- --:--:-
- 2081k
```


50 neurons
Fully-connected layer
100 neurons
Fully-connected layer
1164 neurons
Flatten
Convolutional
feature map
64@1x18
3x3 kernel
Convolutional
feature map
64@3x20
3x3 kernel
Convolutional
feature map
48@5x22
5x5 kernel
Convolutional
feature map
36@14x47
5x5 kernel
Convolutional
feature map
24@31x98
5x5 kernel
Normalized
input planes
3@66x200



In [0]:

```
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='VA
LID')


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])

# Apply Linear activation function
y = tf.matmul(h_fc4_drop, W_fc5) + b_fc5
```

```
WARNING:tensorflow:From <ipython-input-11-d15671f37e82>:59: calling dro
pout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated a
nd will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate =
1 - keep_prob`.
```

Training the model

In [ ]:

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

LOGDIR = './save'
```

```
sess = tf.InteractiveSession()

L2NormConst = 0.001

train_vars = tf.trainable_variables()

loss = tf.reduce_mean(tf.square(tf.subtract(y_, y))) + tf.add_n([tf.nn.
l2_loss(v) for v in train_vars]) * L2NormConst
train_step = tf.train.AdamOptimizer(1e-2).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(num_images/batch_size)):
    xs, ys = LoadTrainBatch(batch_size)
    train_step.run(feed_dict={x: xs, y_: ys, keep_prob: 0.5})
    if i % 10 == 0:
      xs, ys = LoadValBatch(batch_size)
      loss_value = loss.eval(feed_dict={x:xs, y_: ys, keep_prob: 0.5})
      print("Epoch: %d, Step: %d, Loss: %g" % (epoch, epoch * batch_siz
e + i, loss_value))

    # write logs at every iteration
```

```
    summary = merged_summary_op.eval(feed_dict={x:xs, y_: ys, keep_prob
: 0.5})
    summary_writer.add_summary(summary, epoch * 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")
```

WARNING:tensorflow:***

WARNING:tensorflow:TensorFlow's V1 checkpoint format has been deprecated.

WARNING:tensorflow:Consider switching to the more efficient V2 format:

WARNING:tensorflow: `tf.train.Saver(write_version=tf.train.SaverDef.V2)`

WARNING:tensorflow:now on by default.

WARNING:tensorflow:***

Epoch: 29, Step: 3310, Loss: 0.0444565

Epoch: 29, Step: 3320, Loss: 0.0120469

Epoch: 29, Step: 3330, Loss: 0.0138884

Epoch: 29, Step: 3340, Loss: 0.151224

Validating the model

In [0]:

```
import tensorflow as tf
import scipy.misc
imort model
import cv2
from subprocess import call

sess = tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, "save/model.ckpt")

img = cv2.imread('steering_wheel_image.jpg',0)
rows,cols = img.shape

smoothed_angle = 0

cap = cv2.VideoCapture(0)
while(cv2.waitKey(10) != ord('q')):
    ret, frame = cap.read()
    image = scipy.misc.imresize(frame, [66, 200]) / 255.0
    degrees = model.y.eval(feed_dict={model.x: [image], model.keep_prob
: 1.0})[0][0] * 180 / scipy.pi
    call("clear")
    print("Predicted steering angle: " + str(degrees) + " degrees")
    cv2.imshow('frame', frame)
    #make smooth angle transitions by turning the steering wheel based
 on the difference of the current angle
    #and the predicted angle
    smoothed_angle += 0.2 * pow(abs((degrees - smoothed_angle)), 2.0 /
3.0) * (degrees - smoothed_angle) / abs(degrees - smoothed_angle)
    M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)
    dst = cv2.warpAffine(img,M,(cols,rows))
    cv2.imshow("steering wheel", dst)

cap.release()
cv2.destroyAllWindows()
```

run_dataset.py

In [0]:
```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.
6/dist-packages (3.4.5.20)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.
6/dist-packages (from opencv-python) (1.16.4)
```

In [0]:
```
from google.colab.patches import cv2_imshow
```

In [0]:
```
#pip3 install opencv-python

import tensorflow as tf
import scipy.misc
import model
import cv2
from subprocess import call
import math

sess = tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, "save/model.ckpt")

img = cv2.imread('Autopilot-TensorFlow-master/steering_wheel_image.jpg'
,0)
rows,cols = img.shape

smoothed_angle = 0


#read data.txt
xs = []
ys = []
with open("Autopilot-TensorFlow-master/driving_dataset/data.txt") as f:
    for line in f:
        xs.append("Autopilot-TensorFlow-master/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 tur
```

```
ning 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)


i = math.ceil(num_images*0.7)
print("Starting frameofvideo:" +str(i))

while(cv2.waitKey(10) != ord('q')):
    full_image = scipy.misc.imread("Autopilot-TensorFlow-master/driving
_dataset/" + str(i) + ".jpg", mode="RGB")
    image = scipy.misc.imresize(full_image[-150:], [66, 200]) / 255.0
    degrees = y.eval(feed_dict={x: [image], keep_prob: 1.0})[0][0] * 18
0.0 / scipy.pi
    #call("clear")
    #print("Predicted Steering angle: " + str(degrees))
    print("Steering angle: " + str(degrees) + " (pred)\t" + str(ys[i]*1
80/scipy.pi) + " (actual)")
    cv2_imshow(cv2.cvtColor(full_image, cv2.COLOR_RGB2BGR))
    #make smooth angle transitions by turning the steering wheel based
 on the difference of the current angle
    #and the predicted angle
    smoothed_angle += 0.2 * pow(abs((degrees - smoothed_angle)), 2.0 /
3.0) * (degrees - smoothed_angle) / abs(degrees - smoothed_angle)
    M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)
    dst = cv2.warpAffine(img,M,(cols,rows))
    cv2_imshow(dst)
    i += 1

cv2.destroyAllWindows()
```

Conclusion

All models implemented:

Split	Adam	Dropout	Activation	Test MSE
70-30	1e-2	0.4	Linear	.40
70-30	1e-3	0.5	Linear	0.16
70-30	1e-4	0.4	Linear	0.207

Steps followed:

-1) The objective of this case study was to predict the steering angle by analysing an image. This is a regression problem.

- 2)The data we had consists of 25mins of driving data taken from the front of a car by Sully chen. The data can be found at GitHub. We had more than 45000 images as our data.
- 3)As a first step to solve this case study we splitted the dataset into 70-30 split where the first 70% of the data (17.5mins approx) was used to train the model and the last 30% of the dataset was used for validation.
- 4)As an EDA step, we first converted the steering angle of both train and test dataset from degree to radian unit. We did it because converting everything to radian reduced the range of value for the target variable (steering angle). This can be thought of as a normalizing step which helped in training our model.
- 5)As a next step we plotted the pdf of the steering angle for both train and test dataset to see what distribution does it follow. We figured out though the train and test dataset differ a little in the distribution (which is normal as we are splitting the dataset in a temporal fashion), but most of the time the steering angle is at 0 as the pdf was highest at that point for both train and test dataset.
- 6)From the above finding, we created two baseline models where for one model we simply returned the mean of all the steering angle and in another model we returned 0. In both cases we got a MSE of 0.241.
- 7)After this we built the cnn model consisting of 5 cnn layers and 3 Fully connected layers inspired by the NVIDA end to end model, we also made some modifications to the model by adding 4 dropout layers in between the fully connected layers to prevent overfitting of the model.

- After trying out different combinations of values for the hyperparameter tuning I figured out that we are getting the best MSE of 0.167 when we are using a linear activation with Adam optimatizer (1e-4) and dropout of 0.4. For all the other configurations I tried I was facing a problem where the predictions were not changing.
- I thought of using different activation functions such as sigmoid, tanh but as they were mainly for classification, I did not proceed with the idea (As training this model takes significant amount of time) and so at the end I used Linear activation function, which in many blogs I followed was mentioned as the go to activation function for any regression problem.

Furthur improvements:

I guess we can furthur improve the model by creating a CNN-RNN model where we can also leverage the sequence information of the driving data images, which may help our model learn some interesting insights about the driving data and may help us get a better result. But as I don't have that much of computing resource to train a CNN-RNN model, I used only CNN model in this case study.