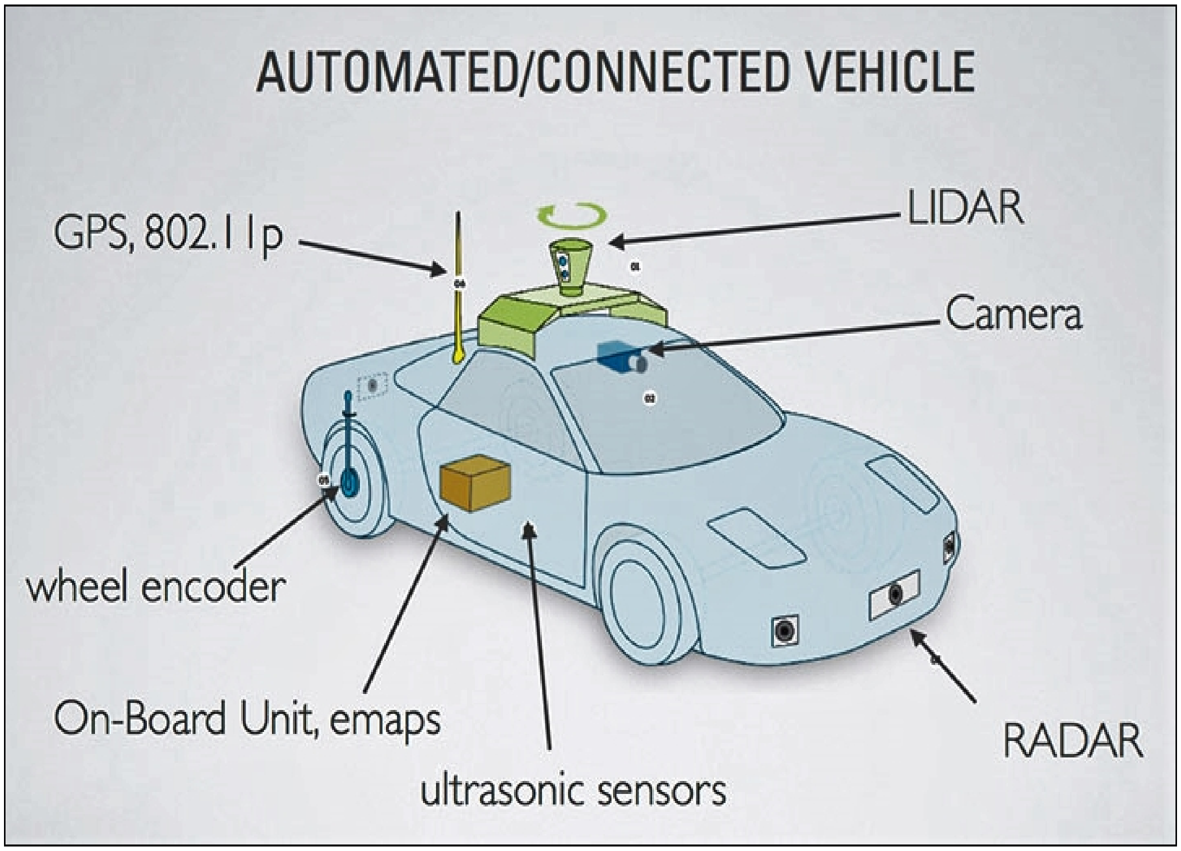
MACHINE LEARNING WITH PYTHON INTERNSHIP PROJECT

SELF DRIVING CAR



ABSTRACT

We trained a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. This end-to-end approach proved surprisingly powerful. With minimum training data from humans the system learns to drive in traffic on local roads with or without lane markings and on highways. It also operates in areas with unclear visual guidance such as in parking lots and on unpaved roads. The system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. We never explicitly trained it to detect, for example, the outline of roads. Compared to explicit decomposition of the problem, such as lane marking detection, path planning, and control, our end-to-end system optimizes all processing steps simultaneously. We argue that this will eventually lead to better performance and smaller systems. Better performance will result because the internal components self-optimize to maximize overall system performance, instead of optimizing human-selected intermediate criteria, e. g., lane detection. Such criteria understandably are selected for ease of human interpretation which doesn't automatically guarantee maximum system performance. Smaller networks are possible because the system learns to solve the problem with the minimal number of processing steps. We used an NVIDIA DevBox and Torch 7 for training and an NVIDIA DRIVETM PX self-driving car computer also running Torch 7 for determining where to drive. The system operates at 30 frames per second (FPS).

OBJECTIVE

Autonomous vehicles can visualize their environments with high-resolution digital camera images. Self-driving cars can use camera images to “see” and interpret environmental details (e.g. signs, traffic lights, animals) in ways that approximate human vision (aka computer vision)

INTRODUCTION

Self-driving vehicles are cars or trucks in which human drivers are never required to take control to safely operate the vehicle. Also known as autonomous or “driverless” cars, they combine sensors and software to control, navigate, and drive the vehicle.

FEATURE

Currently, there are no legally operating, fully-autonomous vehicles in the United States. There are, however, *partially*-autonomous vehicles—cars and trucks with varying amounts of self-automation, from conventional cars with brake and lane assistance to highly-independent, self-driving prototypes.

Though still in its infancy, self-driving technology is becoming increasingly common and could radically transform our transportation system (and by extension, our economy and society). Based on automaker and technology company estimates, level 4 self-driving cars could be for sale in the next several years (see the callout box for details on autonomy levels).

Layers of autonomy

Different cars are capable of different levels of self-driving, and are often described by researchers on a scale of 0-5.

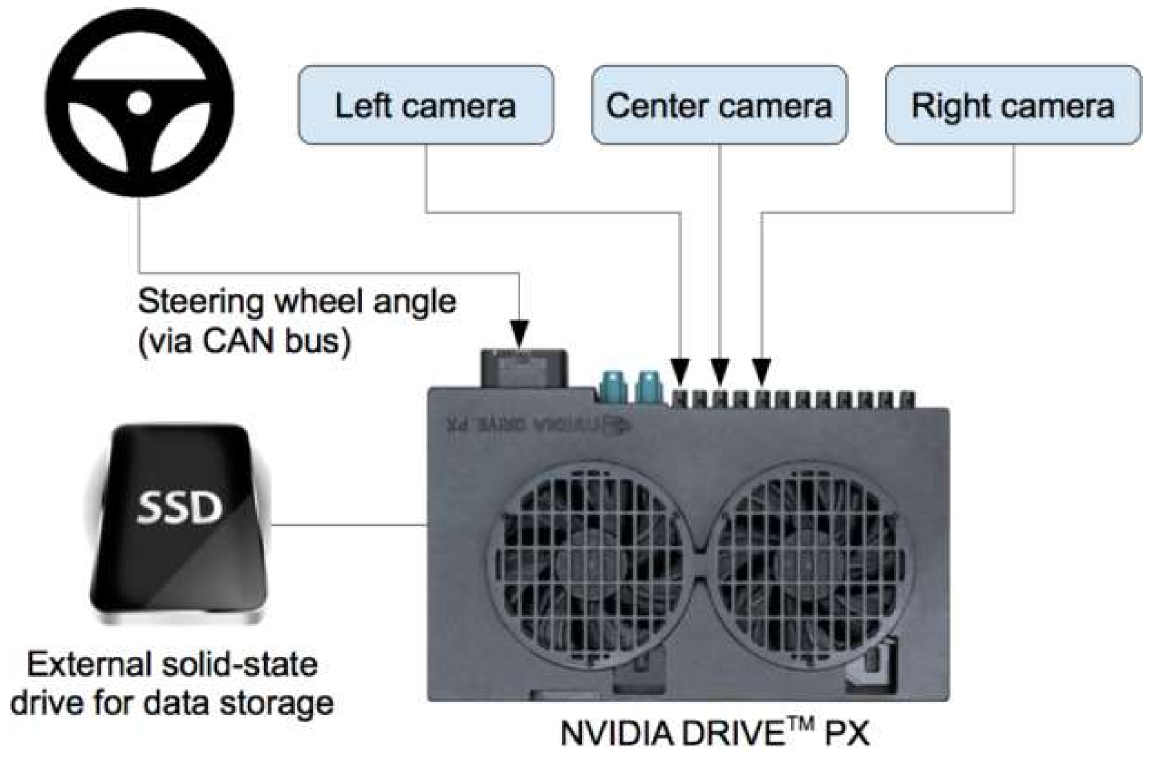
- **Level 0**: All major systems are controlled by humans
- **Level 1:** Certain systems, such as cruise control or automatic braking, may be controlled by the car, one at a time
- **Level 2**: The car offers at least two simultaneous automated functions, like acceleration and steering, but requires humans for safe operation
- **Level 3**: The car can manage all safety-critical functions under certain conditions, but the driver is expected to take over when alerted
- **Level 4**: The car is fully-autonomous in some driving scenarios, though not all
- **Level 5**: The car is completely capable of self-driving in every situation

METHODOLOGY

1. Overview od DAVE 2 system

A simplified block diagram of the collection system for training data for DAVE-2. Three cameras are mounted behind the windshield of the data-acquisition car. Time-stamped video from the cameras is captured simultaneously with the steering angle applied by the human driver. This steering command is obtained by tapping into the vehicle's Controller Area Network (CAN) bus. In order to make our system independent of the car geometry, we represent the steering command as 1/r where r is the turning radius in meters. We use 1/r instead of r to prevent a singularity when driving straight (the turning radius for driving straight is infinity). 1/r smoothly transitions through zero from left turns (negative values) to right turns (positive values). Training data contains single images sampled from the video, paired with the corresponding steering command (1/r). Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. Otherwise the car will slowly drift off the road. The training data is therefore augmented with additional images that show the car in different shifts from the center of the lane and rotations from the direction of the road.

Images for two specific off-center shifts can be obtained from the left and the right camera. Additional shifts between the cameras and all rotations are simulated by viewpoint transformation of the image from the nearest camera. Precise viewpoint transformation requires 3D scene knowledge which we don’t have. We therefore approximate the transformation by assuming all points below the horizon are on flat ground and all points above the horizon are infinitely far away. This works fine for flat terrain but it introduces distortions for objects that stick above the ground, such as cars, poles, trees, and buildings. Fortunately these distortions don’t pose a big problem for network training. The steering label for transformed images is adjusted to one that would steer the vehicle back to the desired location and orientation in two seconds. A block diagram of our training system is shown in Figure 2. Images are fed into a CNN which then computes a proposed steering command. The proposed command is compared to the desired command for that image and the weights of the CNN are adjusted to bring the CNN output closer to the desired output. The weight adjustment is accomplished using back propagation as implemented in the Torch 7 machine learning package. Left camera Center camera Right camera Random shift and rotation Adjust for shift and rotation CNN - Back propagation weight adjustment Recorded steering wheel angle Network computed steering command Desired steering command Error Figure 2: Training the neural network. Once trained, the network can generate steering from the video images of a single center camera. This configuration is shown in Figure 3. 3 Center camera CNN Network computed steering command Drive by wire interface Figure 3: The trained network is used to generate steering commands from a single front-facing


Left camera
Center camera
Right camera
Steering wheel angle (via CAN bus)
SSD
External solid-state drive for data storage
NVIDIA DRIVE™ PX

Training data was collected by driving on a wide variety of roads and in a diverse set of lighting and weather conditions. Most road data was collected in central New Jersey, although highway data was also collected from Illinois, Michigan, Pennsylvania, and New York. Other road types include two-lane roads (with and without lane markings), residential roads with parked cars, tunnels, and unpaved roads. Data was collected in clear, cloudy, foggy, snowy, and rainy weather, both day and night. In some instances, the sun was low in the sky, resulting in glare reflecting from the road surface and scattering from the windshield. Data was acquired using either our drive-by-wire test vehicle, which is a 2016 Lincoln MKZ, or using a 2013 Ford Focus with cameras placed in similar positions to those in the Lincoln. The system has no dependencies on any particular vehicle make or model. Drivers were encouraged to maintain full attentiveness, but otherwise drive as they usually do. As of March 28, 2016, about 72 hours of driving data was collected.

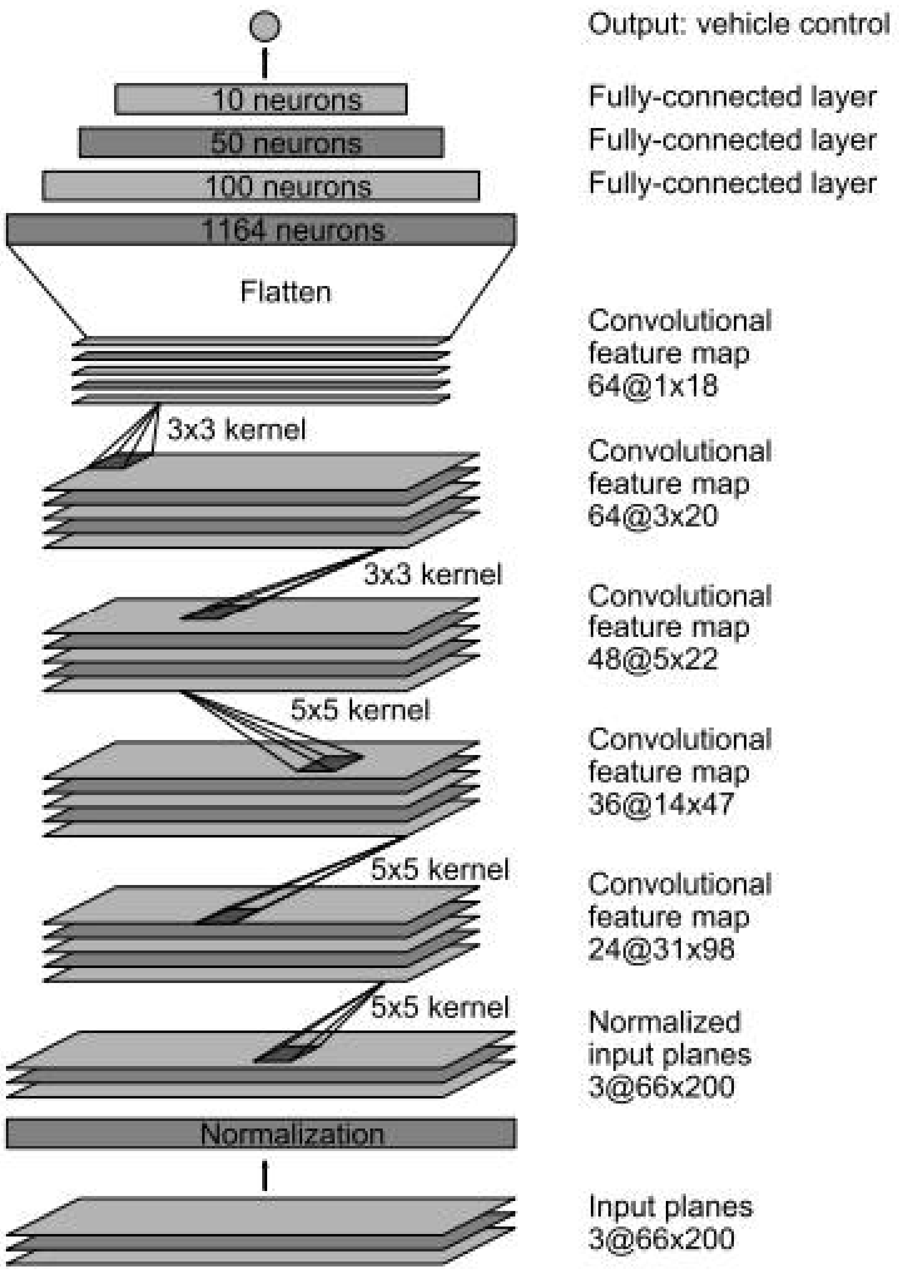
3. Network Architecture

We train the weights of our network to minimize the mean squared error between the steering command output by the network and the command of either the human driver, or the adjusted steering command for off-center and rotated images (see Section 5.2). Our network architecture is shown in Figure 4. The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network. The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing. The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. We use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers. We follow the five convolutional layers with three fully connected layers leading to an output control value which is the inverse turning radius. The fully connected layers are designed to function as a controller for steering, but we note that by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractor and which serve as controller

4. Training detail

4.1 Data Selection.

The first step to training a neural network is selecting the frames to use. Our collected data is labeled with road type, weather condition, and the driver's activity (staying in a lane, switching lanes, turning, and so forth). To train a CNN to do lane following we only select data where the driver was staying in a lane and discard the rest. We then sample that video at 10 FPS. A higher sampling rate would result in including images that are highly similar and thus not provide much useful information.

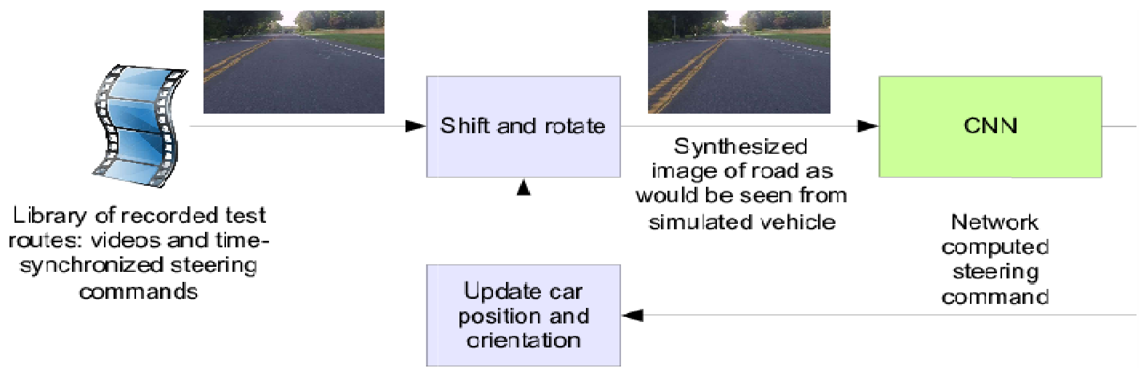


4.2 Augmentation

After selecting the final set of frames we augment the data by adding artificial shifts and rotations to teach the network how to recover from a poor position or orientation. The magnitude of these perturbations is chosen randomly from a normal distribution. The distribution has zero mean, and the standard deviation is twice the standard deviation that we measured with human drivers. Artificially augmenting the data does add undesirable artifacts as the magnitude increases

5.Simulation

Before road-testing a trained CNN, we first evaluate the networks performance in simulation. A simplified block diagram of the simulation system is shown in Figure 5. The simulator takes pre-recorded videos from a forward-facing on-board camera on a human-driven data-collection vehicle and generates images that approximate what would appear if the CNN were, instead, steering the vehicle. These test videos are time-synchronized with recorded steering commands generated by the human driver. 5 Since human drivers might not be driving in the center of the lane all the time, we manually calibrate the lane center associated with each frame in the video used by the simulator. We call this position the “ground truth”. The simulator transforms the original images to account for departures from the ground truth. Note that this transformation also includes any discrepancy between the human driven path and the ground truth. The transformation is accomplished by the same methods described in Section 2. The simulator accesses the recorded test video along with the synchronized steering commands that occurred when the video was captured. The simulator sends the first frame of the chosen test video, adjusted for any departures from the ground truth, to the input of the trained CNN. The CNN then returns a steering command for that frame. The CNN steering commands as well as the recorded human-driver commands are fed into the dynamic model [8] of the vehicle to update the position and orientation of the simulated vehicle. The simulator then modifies the next frame in the test video so that the image appears as if the vehicle were at the position that resulted by following steering commands from the CNN. This new image is then fed to the CNN and the process repeats. The simulator records the off-center distance (distance from the car to the lane center), the yaw, and the distance traveled by the virtual car. When the off-center distance exceeds one meter, a virtual human intervention is triggered, and the virtual vehicle position and orientation is reset to match the ground truth of the corresponding frame of the original test video.


Library of recorded test routes: videos and time-synchronized steering commands
Shift and rotate
Synthesized image of road as would be seen from simulated vehicle
CNN
Network computed steering command
Update car position and orientation

CODE

Driving.py

```python
import cv2
import random
import numpy as np

xs = []
ys = []

train_batch_pointer = 0
```

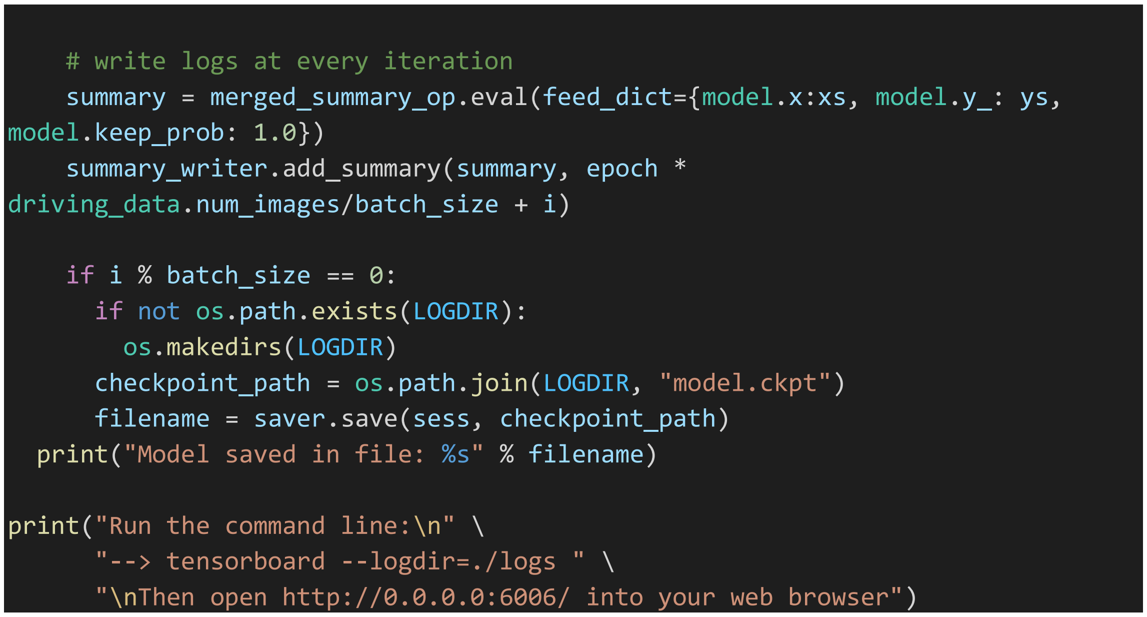
```python
val_batch_pointer = 0

with open("driving_dataset/data.txt") as f:
    for line in f:
        xs.append("driving_dataset/" + line.split()[0])
        ys.append(float(line.split()[1]) * 3.14159265 / 180)

#take number of images
num_images = len(xs)

#shuffle list of images
c = list(zip(xs, ys))
random.shuffle(c)
xs, ys = zip(*c)

train_xs = xs[:int(len(xs) * 0.8)]
train_ys = ys[:int(len(xs) * 0.8)]

val_xs = xs[-int(len(xs) * 0.2):]
val_ys = ys[-int(len(xs) * 0.2):]

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(cv2.resize(cv2.imread(train_xs[(train_batch_pointer + i)
% num_train_images])[-150:], (200, 66)) / 255.0)
        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(cv2.resize(cv2.imread(val_xs[(val_batch_pointer + i) %
num_val_images])[-150:], (200, 66)) / 255.0)
        y_out.append([val_ys[(val_batch_pointer + i) % num_val_images]])
    val_batch_pointer += batch_size
    return x_out, y_out
```

Run_Datasheet.py

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import model
import cv2
from subprocess import call
import os

#check if on windows OS
windows = False
if os.name == 'nt':
    windows = True

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

i = 0
while(cv2.waitKey(10) != ord('q')):
    full_image = cv2.imread("driving_dataset/" + str(i) + ".jpg")
    image = cv2.resize(full_image[-150:], (200, 66)) / 255.0
    degrees = model.y.eval(feed_dict={model.x: [image], model.keep_prob:
1.0})[0][0] * 180.0 / 3.14159265
    if not windows:
        call("clear")
    print("Predicted steering angle: " + str(degrees) + " degrees")
    cv2.imshow("frame", full_image)
    #making smooth angle transitions by turning the steering wheel based on
the difference of the current angle
    #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)
    i += 1

cv2.destroyAllWindows()
```

Train.py

```
import os
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
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(model.y_, model.y))) +
tf.add_n([tf.nn.l2_loss(v) for v in train_vars]) * L2NormConst
train_step = tf.train.AdamOptimizer(1e-4).minimize(loss)
sess.run(tf.global_variables_initializer())

# 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 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:
0.8})
    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: 1.0})
    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")
```

CONCLUSION

We have empirically demonstrated that CNNs are able to learn the entire task of lane and road following without manual decomposition into road or lane marking detection, semantic abstraction, path planning, and control. A small amount of training data from less than a hundred hours of driving was sufficient to train the car to operate in diverse conditions, on highways, local and residential roads in sunny, cloudy, and rainy conditions. The CNN is able to learn meaningful road features from a very sparse training signal (steering alone).

The system learns for example to detect the outline of a road without the need of explicit labels during training.

More work is needed to improve the robustness of the network, to find methods to verify the robustness, and to improve visualization of the network-internal processing steps.