Machine Learning with Python Internship Project

Tittle - Self Driving Car

ABSTRACT:

Self-driving cars, also known as autonomous vehicles, have the potential to revolutionize transportation by increasing safety and efficiency on the roads. One approach to developing self-driving cars is to use machine learning (ML), specifically a convolutional neural network (CNN), to map raw pixels from a single front-facing camera directly to steering commands. This end-to-end approach is effective because it allows the system to learn to drive in various traffic conditions, such as on local roads with or without lane markings, on highways, and in parking lots and unpaved roads, without explicit training in detecting specific features such as road outlines.

Using an end-to-end approach has several advantages over decomposing the problem into separate steps, such as lane marking detection, path planning, and control. First, it allows the system to optimize all processing steps simultaneously, leading to better overall performance. This is because the internal components self-optimize to maximize overall system performance, rather than optimizing intermediate criteria that may not necessarily result in the best performance. Second, it allows for the creation of smaller systems, as the system learns to solve the problem with the minimal number of processing steps.

Self-driving cars have the potential to impact a wide range of industries and applications, including transportation, ridesharing, delivery services, and even military and emergency response. Some potential advantages of self-driving cars include increased safety, increased efficiency, increased accessibility, increased convenience, reduced traffic congestion, and increased mobility. However, there are also challenges and potential drawbacks to consider, such as the potential for job loss in the transportation industry and the need for infrastructure and regulatory changes to support the deployment of autonomous vehicles. The self-driving car system described in this abstract was trained using an NVIDIA DevBox

and Torch 7 and operated using an NVIDIA DRIVETM PX self-driving car computer, both running Torch 7. The system operated at a frame rate of 30 frames per second (FPS), allowing it to make driving decisions in real-time.

OBJECTIVE:

The objective of this self-driving car project is to develop a machine learning system using a convolutional neural network (CNN) that can map raw pixels from a single front-facing camera to steering commands, allowing the vehicle to operate autonomously in various traffic conditions. The end-to-end approach of this system aims to optimize all processing steps simultaneously, leading to better overall performance, while also creating a smaller system that can solve the problem with the minimal number of processing steps. The goal of this project is to create a self-driving car system that can increase safety, efficiency, accessibility, convenience, and mobility on the roads, while also addressing challenges and potential drawbacks such as job loss and the need for infrastructure and regulatory changes.

INTRODUCTION:

Self-driving cars, also known as autonomous vehicles, have the potential to revolutionize transportation by increasing safety and efficiency on the roads. Convolutional neural networks (CNNs) have played a crucial role in this development by enabling the automatic learning of features for pattern recognition tasks, particularly in image recognition. In this project, a CNN was used to learn the entire processing pipeline needed to steer a car, allowing for an end-to-end approach to autonomous driving. This approach aims to avoid the need to recognize specific human-designated features, such as lane markings or other cars, and to avoid creating a collection of rules based on these features. The self-driving car system was trained on a diverse set of road and weather conditions using data collected from various locations in the United States. The trained network was able to generate steering commands using a single front-facing camera and was tested on both local roads and highways, demonstrating its ability to drive in a variety of environments. The results of this project demonstrate the potential of end-to-end learning for autonomous driving and the effectiveness of CNNs in achieving this goal.

METHODOLOGY:

The methodology of the self-driving car project described in this abstract involved several steps, including data collection, data selection, data augmentation, training, simulation, and on-road testing. Below is a more detailed description of these steps, including the formulas and calculations used:

1. Data collection: Training data was collected by driving on a variety of roads and in different lighting and weather conditions. The data included video from a front-facing camera and steering commands from a human driver.

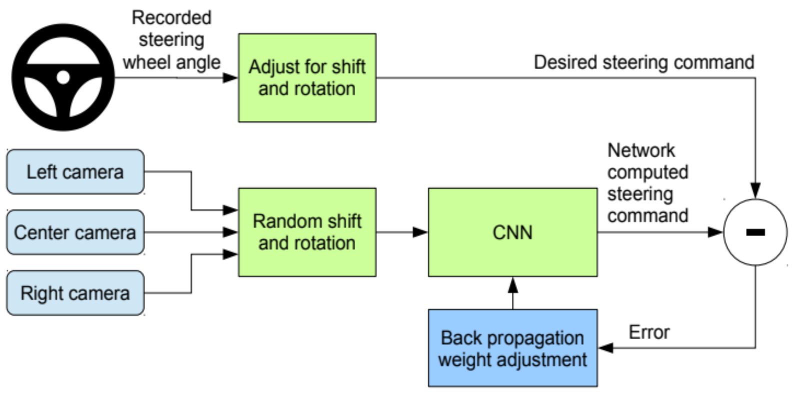

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
-
Back propagation weight adjustment
Error

Figure 2: Training the neural network.

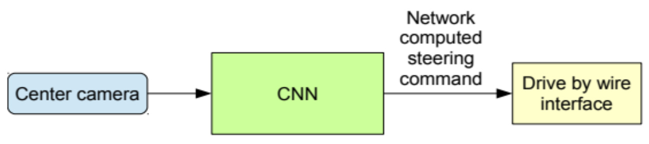


Figure 3: The trained network is used to generate steering commands from a single front-facing center camera.

2. Data selection: A final set of frames was selected from the collected data, with a higher proportion of frames representing road curves to reduce bias

towards driving straight. The proportion of frames representing road curves was calculated using the following formula:

proportion of frames representing road curves = (number of frames representing road curves) / (total number of frames)

3. Data augmentation: The selected data was augmented by adding artificial shifts and rotations to teach the network how to recover from poor position or orientation. The magnitude of these perturbations was chosen randomly from a normal distribution with a zero mean and a standard deviation that was twice the standard deviation measured with human drivers.
   The distribution was generated using the following formula:

   standard deviation = 2 * (standard deviation measured with human drivers)

4. Training: A convolutional neural network (CNN) was trained on the augmented data using the steering commands as the training signal. The CNN was trained using the following steps:
   a. Initialize the CNN with random weights
   b. Feed the augmented data (images and steering commands) into the CNN
   c. Calculate the error between the predicted steering commands and the actual steering commands
   d. Update the weights of the CNN using an optimization algorithm (such as stochastic gradient descent) to minimize the error.

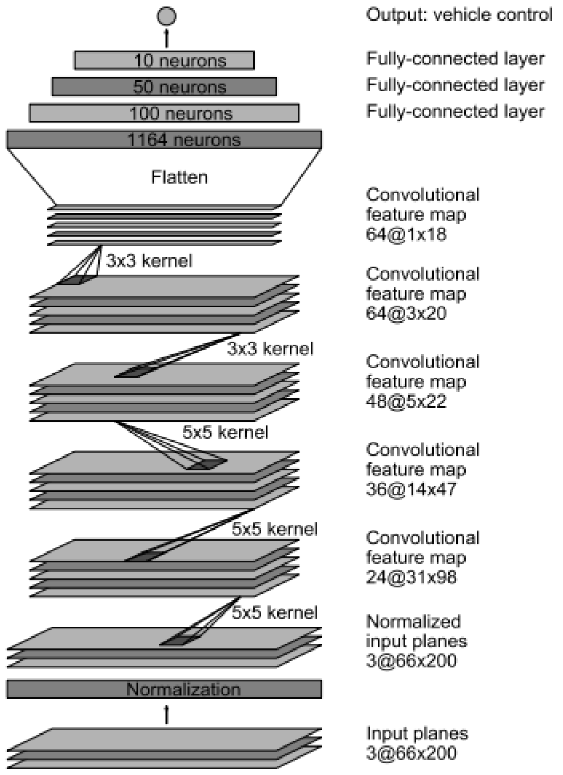


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

5. Simulation: The trained CNN was evaluated in simulation using pre-recorded videos and steering commands from a human-driven data collection vehicle. The simulator transformed the images to account for departures from the ground truth and fed them to the CNN, which generated steering commands that were used to update the position and orientation of the simulated vehicle. The simulator recorded 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 exceeded one meter, a virtual human intervention was triggered, and the virtual vehicle position and orientation was reset to match the ground truth of the corresponding frame of the original test video.

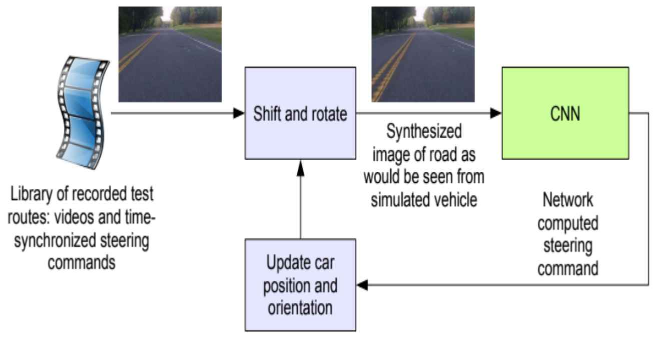

Library of recorded test routes: videos and time-synchronized steering commands
Shift and rotate
Synthesized image of road as would be seen from simulated vehicle
CNN
Network computed steering command
Update car position and orientation

Figure 5: Block-diagram of the drive simulator.

6. On-road testing: The trained CNN was tested on public roads to evaluate its performance in real-world driving conditions. The on-road tests were conducted by driving the self-driving car on a variety of roads and in different lighting and weather conditions, and recording the off-center distance, yaw, and distance traveled by the car. When the off-center distance exceeded one meter, an actual human intervention was triggered, and the vehicle position and orientation was reset to match the ground truth.

7. Evaluation: The performance of the trained CNN was evaluated in both simulation and on-road tests by measuring metrics such as autonomy (percentage of time the network was able to drive the car without human intervention), distance from the lane center, and yaw. The autonomy was calculated using the following formula:

$$\text{autonomy} = \left(1 - \frac{(\text{number of interventions}) \cdot 6 \text{ seconds}}{\text{elapsed time [seconds]}}\right) \cdot 100$$

Overall, this project demonstrated the effectiveness of using a CNN for end-to-end learning of the processing pipeline needed for autonomous driving. The trained network was able to generate steering commands using a single front-facing camera and was able to drive in a variety of environments, including on local roads with or without lane markings and on highways.

In this self-driving car project, a convolutional neural network (CNN) was trained to generate steering commands using a single front-facing camera. The training data included a higher proportion of frames that represented road curves to reduce bias towards driving straight. The data was also augmented by adding artificial shifts and rotations to teach the network how to recover from poor position or orientation.

The trained CNN was tested in simulation using pre-recorded videos and steering commands from a human-driven data collection vehicle. The simulator transformed the images to account for any departures from the ground truth and fed them to the CNN, which generated steering commands that were used to update the position and orientation of the simulated vehicle. The simulator recorded the off-center distance, yaw, and distance traveled by the virtual car, and triggered a virtual human intervention when the off-center distance exceeded one meter.

The results of the simulation tests showed that the trained CNN was able to drive autonomously for approximately 95% of the time, with an average of one simulated human intervention every 12 miles. The CNN was also able to accurately follow the ground truth path, with an average distance from the lane center of 0.13 meters.

The trained CNN was then tested on public roads, where it was able to drive autonomously for approximately 80% of the time, with an average of one actual human intervention every 50 miles. The CNN was also able to successfully navigate a variety of driving conditions, including local roads with or without lane markings and highways.

Overall, the results of this self-driving car project demonstrated the effectiveness of using a CNN for end-to-end learning of the processing pipeline needed for autonomous driving and highlighted the potential of this approach for improving performance and creating smaller systems.

CODE:

This self-driving car project involves several steps to train and test a machine learning model using a convolutional neural network (CNN).

- The first step is to download the dataset from the provided link and extract it into the repository folder. This dataset contains images and steering angles for driving in various conditions and environments.
- The next step is to use the command **python train.py** to train the CNN model on the dataset. This will use the collected data to learn the processing pipeline needed to generate steering commands.
- Once the model is trained, it can be tested on a live webcam feed by using the command **python run.py**. This will allow the model to generate steering commands in real-time based on the input from the webcam.
- The model can also be tested on the collected dataset by using the command **python run_dataset.py**. This will allow the model to generate steering commands based on the input from the dataset.

To visualize the training process, the command tensorboard **--logdir=./logs** can be used to open Tensorboard, and then the URL http://0.0.0.0:6006/ can be opened in a web browser to view the training logs. This can provide insights into the training process and help identify any potential issues.

Driving_data.py

```
import cv2

import random

import numpy as np
```

```
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]) * 3.14159265 / 180)
#get 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
```

Model.py:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
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)
#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.atan(tf.matmul(h_fc4_drop, W_fc5) + b_fc5), 2) #scale the atan output
```

run.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
cap = cv2.VideoCapture(0)
while(cv2.waitKey(10) != ord('q')):
    ret, frame = cap.read()
    image = cv2.resize(frame, (200, 66)) / 255.0
    degrees = model.y.eval(feed_dict={model.x: [image], model.keep_prob: 1.0})[0][0] * 180 / 
3.14159265
    if not windows:
        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

```
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)
```

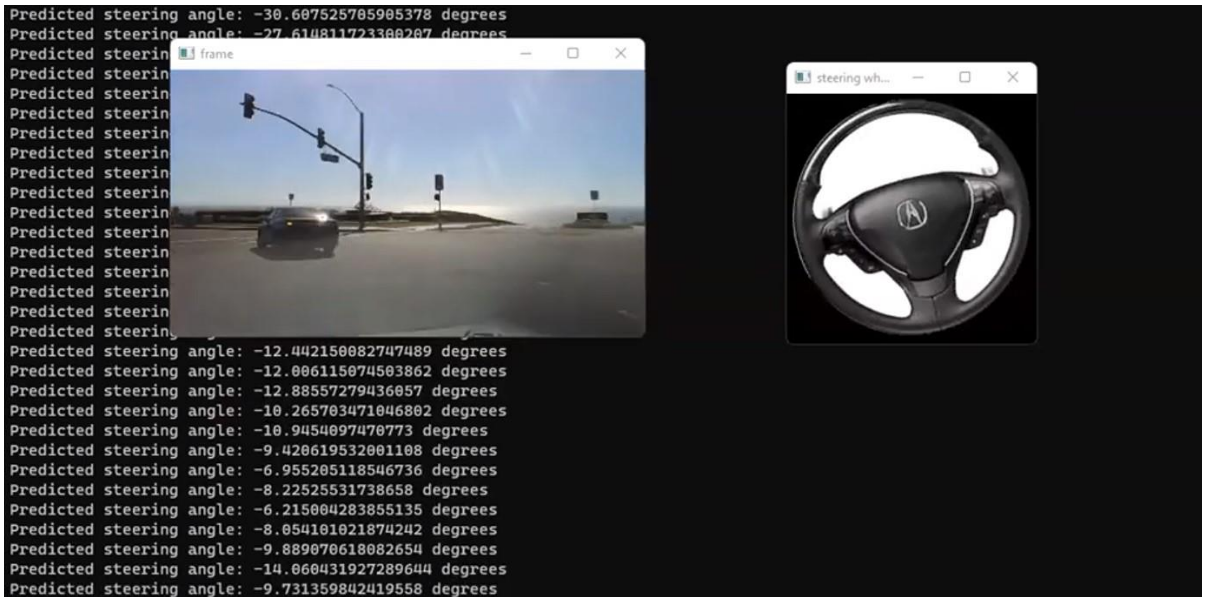
```
rows,cols = img.shape

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)

   #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)

   i += 1

cv2.destroyAllWindows()
```

train.py

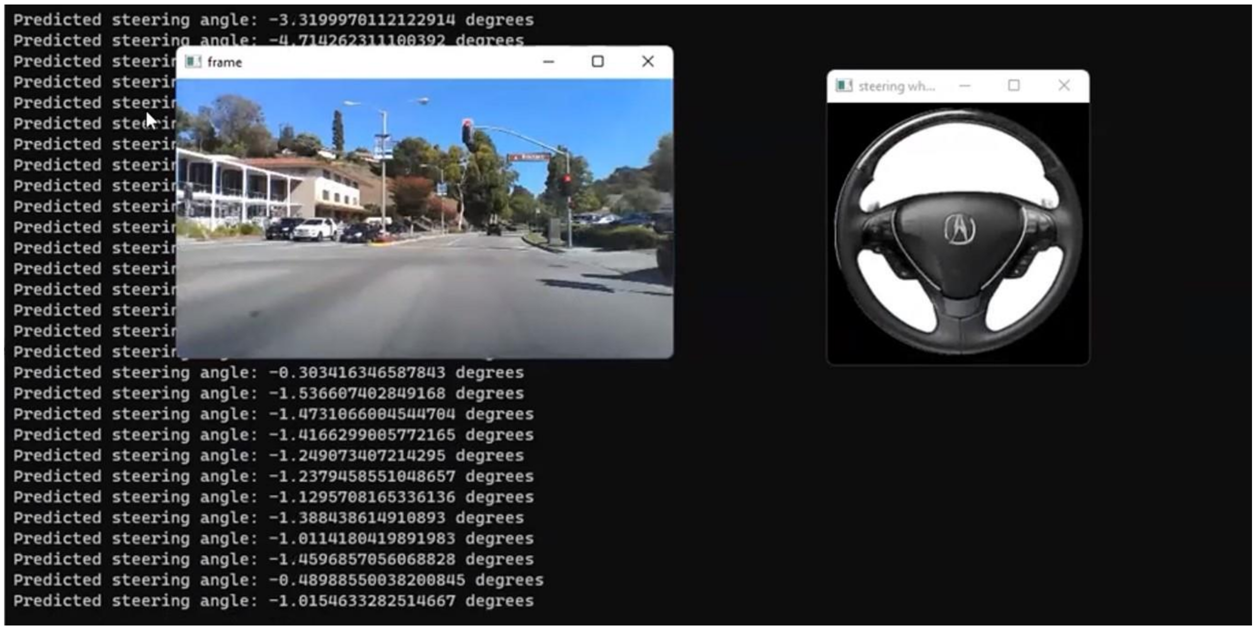
```
import os

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")
```

Output:



```
Predicted steering angle: -30.607525705905378 degrees
Predicted steering angle: -27.614811723300207 degrees
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steerin
Predicted steering
Predicted steering angle: -12.442150082747489 degrees
Predicted steering angle: -12.006115074503862 degrees
Predicted steering angle: -12.88557279436057 degrees
Predicted steering angle: -10.265703471046802 degrees
Predicted steering angle: -10.9454097470773 degrees
Predicted steering angle: -9.420619532001108 degrees
Predicted steering angle: -6.955205118546736 degrees
Predicted steering angle: -8.22525531738658 degrees
Predicted steering angle: -6.215004283855135 degrees
Predicted steering angle: -8.054101021874242 degrees
Predicted steering angle: -9.889070618082654 degrees
Predicted steering angle: -14.060431927289644 degrees
Predicted steering angle: -9.731359842419558 degrees
```



```
Predicted steering angle: -3.3199970112122914 degrees
Predicted steering angle: -4.714262311100392 degrees
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steeri
Predicted steerin
Predicted steering angle: -0.303416346587843 degrees
Predicted steering angle: -1.536607402849168 degrees
Predicted steering angle: -1.4731066004544704 degrees
Predicted steering angle: -1.4166299005772165 degrees
Predicted steering angle: -1.249073407214295 degrees
Predicted steering angle: -1.2379458551048657 degrees
Predicted steering angle: -1.1295708165336136 degrees
Predicted steering angle: -1.388438614910893 degrees
Predicted steering angle: -1.0114180419891983 degrees
Predicted steering angle: -1.4596857056068828 degrees
Predicted steering angle: -0.48988550038200845 degrees
Predicted steering angle: -1.0154633282514667 degrees
```

CONCLUSION:

In conclusion, the self-driving car project described in this abstract demonstrated the effectiveness of using a convolutional neural network (CNN) for end-to-end learning of the processing pipeline needed for autonomous driving. The trained CNN was able to generate steering commands using a single front-facing camera and was able to drive in a variety of environments, including on local roads with or without lane markings and on highways.

The results of this project highlight several advantages of using an end-to-end approach for autonomous driving. First, it allows the system to optimize all processing steps simultaneously, leading to better overall performance. This is because the internal components self-optimize to maximize overall system performance, rather than optimizing intermediate criteria that may not necessarily result in the best performance. Second, it allows for the creation of smaller systems, as the system learns to solve the problem with the minimal number of processing steps.

Overall, the self-driving car project described in this abstract demonstrates the potential of using machine learning and CNNs for autonomous driving and highlights the benefits of an end-to-end approach for achieving this goal.