ABSTRACT

The fundamental idea behind the project is to develop an automated car that can sense its environment and move without human input. This paper proposes Car automation, which is accomplished by recognizing the road, signals, obstacles, stop signs, responding and making decisions, such as changing the course of the vehicle, stopping red signals, stopping signs, and moving on green signals using Neural Network. Self-driving car processes input tracks a track and sends instructions to the actuators that control acceleration, braking, and steering. The software tracks traffic by means of hard-coded rules, preventive algorithms, predictive modeling, and "smart" discrimination on objects, helping the software to follow rules on transport.

CONTENTS

OBJECTIVE

To create a prototype car which will be able to lap around a track, after it has been trained using supervised learning.

The goal of Self driving vehicle is to make a completely practical mechanized vehicle that can decrease human exertion. Lessen the mishap rate, give better fuel utilization and better traffic stream. Self-driving vehicles are made for giving advantages to the general public, for example, giving transportation to those individuals who can't drive due to being old or physical impairment.

INTRODUCTION

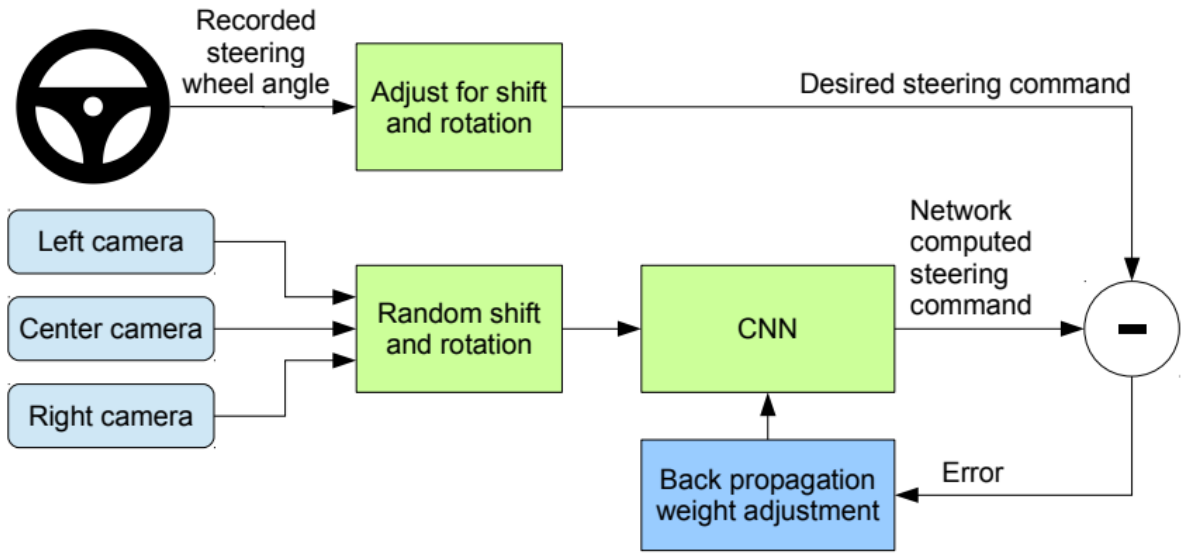
Every year, traffic accidents account for 2.2% of global deaths. That stacks up to roughly 1.3 million a year — 3,287 a day. On top of this, some 20–50 million people are seriously injured in auto-related accidents each year.

“Autonomous cars are no longer beholden to Hollywood sci-fi films” — Elon Musk, the founder of Tesla Inc. and SpaceX believes within a decade, self-driving cars will be as common as elevators. Industry experts say the technology is going to disrupt and revolutionize the future of transportation as we know it.

METHODOLOGY

Train an end-to-end deep learning model that would let a car drive by itself around the track in a driving simulator. It is a supervised regression problem between the car steering angles and the road images in real-time from the cameras of a car.

Data Collection

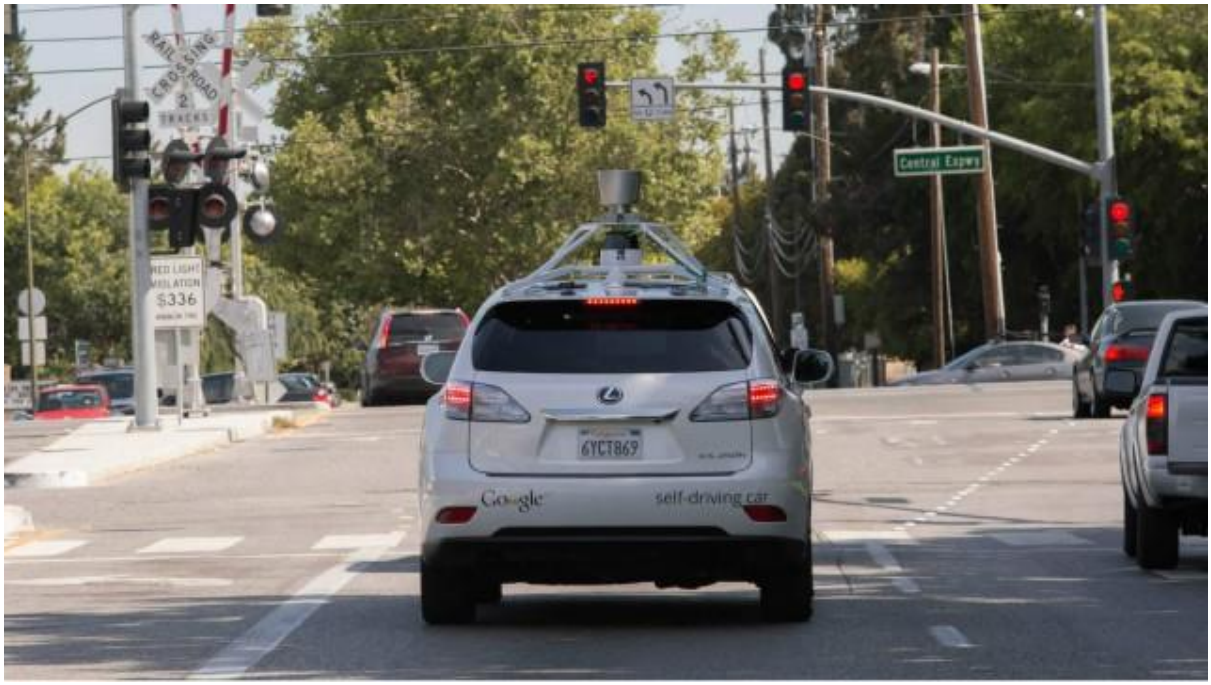


In this project, Udacity driving simulator has been used which has two different tracks. One of them was used for collecting training data, and the other one — never seen by the model — as a substitute for the test set.

The driving simulator would save frames from three front-facing “cameras”, recording data from the car’s point of view; as well as various driving statistics like throttle, speed, and steering angle. We are going to use camera data as model input and expect it to predict the steering angle in the [-1, 1] range.

Environment and Tools

1. matplotlib
2. keras
3. numpy
4. pandas
5. scikit-learn



Google’s self-driving car — Waymo

CODE

I started with loading all the required libraries and dependencies.

```
1
2   import numpy as np
3   import matplotlib.pyplot as plt
4   import matplotlib.image as npimg
5   import os
6
7   ## Keras
8   import keras
9   from keras.models import Sequential
10  from keras.optimizers import Adam
11  from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense
12
13  import cv2
14  import pandas as pd
15  import random
16  import ntpath
17
18  ## Sklearn
19  from sklearn.utils import shuffle
20  from sklearn.model_selection import train_test_split
```

The dataset has 6 columns — center, left, right (camera image paths), steering, throttle, reverse, speed (values). I have used pandas dataframe to display the first five rows in the dataset.

```
datadir = 'Self-Driving-Car'
columns = ['center', 'left', 'right', 'steering', 'throttle', 'reverse', 'speed']
data = pd.read_csv(os.path.join(datadir, 'driving_log.csv'), names = columns)
pd.set_option('display.max_colwidth', -1)
data.head()
```

right	steering	throttle	reverse	speed
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_382.jpg	0.0	0.0	0	0.000079
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_526.jpg	0.0	0.0	0	0.000082
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_669.jpg	0.0	0.0	0	0.000078
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_802.jpg	0.0	0.0	0	0.000078
C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_937.jpg	0.0	0.0	0	0.000080

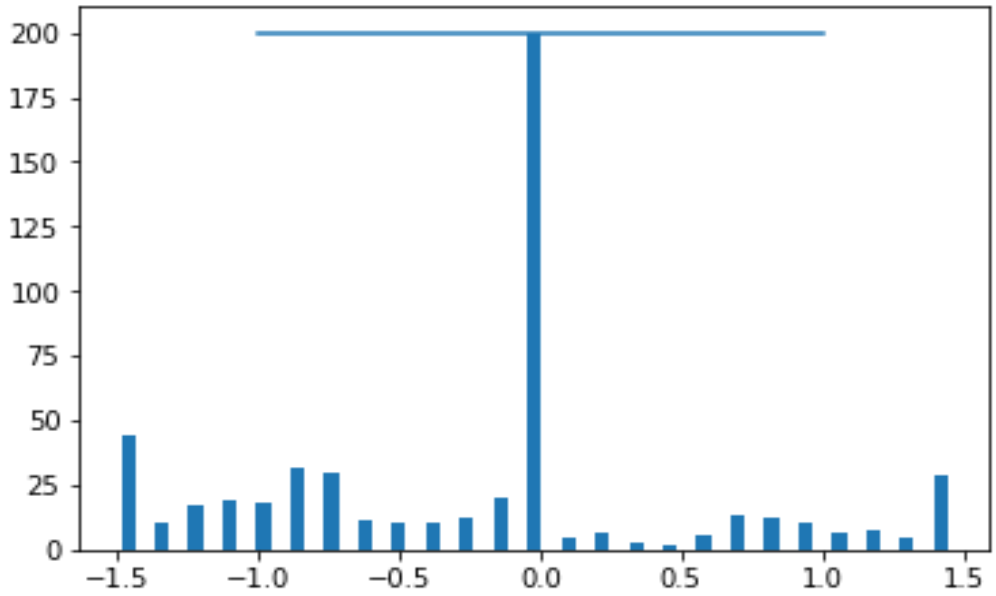
Since the prefix of the left, right and center image paths was the same for all the rows so I decided to remove the prefix part throughout the dataset.

```
def path_leaf(path):
  head, tail = ntpath.split(path)
  return tail

data['center'] = data['center'].apply(path_leaf)
data['left'] = data['left'].apply(path_leaf)
data['right'] = data['right'].apply(path_leaf)
data.head()
```

right	steering	throttle	reverse	speed
right_2019_07_22_20_38_15_382.jpg	0.0	0.0	0	0.000079
right_2019_07_22_20_38_15_526.jpg	0.0	0.0	0	0.000082
right_2019_07_22_20_38_15_669.jpg	0.0	0.0	0	0.000078
right_2019_07_22_20_38_15_802.jpg	0.0	0.0	0	0.000078
right_2019_07_22_20_38_15_937.jpg	0.0	0.0	0	0.000080

Next, I plotted distribution of the steering wheel angle values. As one can see there is a huge spike near zero which means that most of the times the car is driving straight.

```
num_bins = 25
samples_per_bin = 200
hist, bins = np.histogram(data['steering'], num_bins)
center = bins[:-1] + bins[1:] * 0.5  # center the bins to 0

plt.bar(center, hist, width=0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])), (samples_per_bin,
samples_per_bin))
```

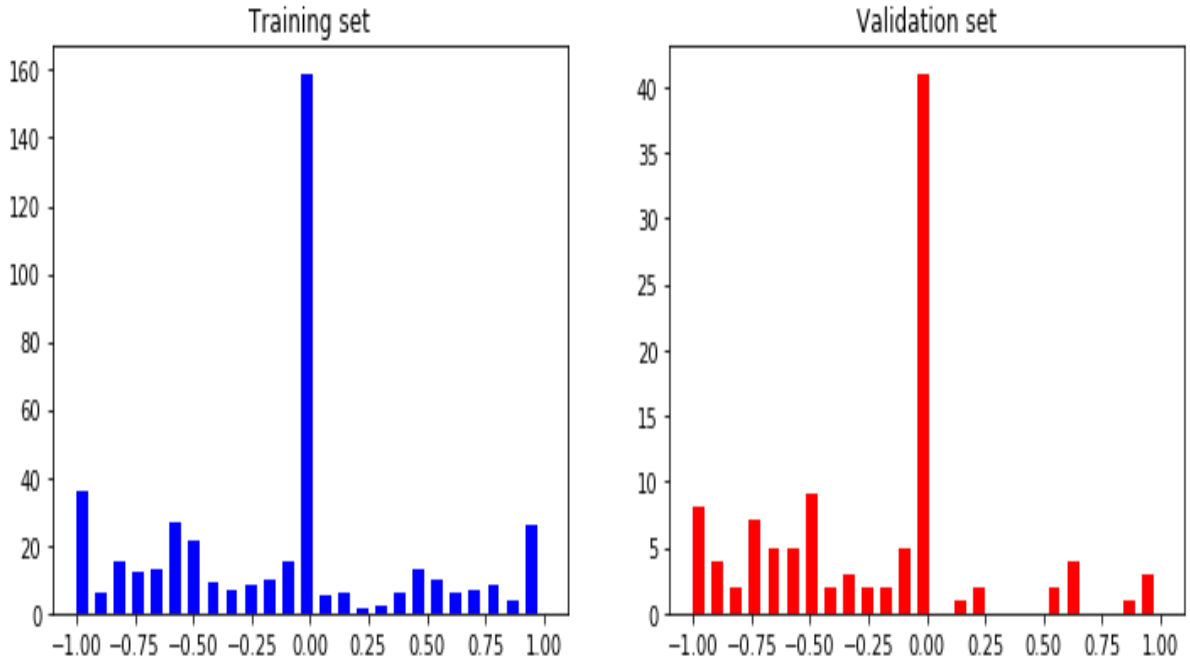


Then I made a function to load all the images as well as the steering wheel angle values in a numpy array.

```
def load_img_steering(datadir, df):
  image_path = []
  steering = []
  for i in range(len(data)):
    indexed_data = data.iloc[i]
    center, left, right = indexed_data[0], indexed_data[1], indexed_data[2]
    image_path.append(os.path.join(datadir, center.strip()))
    steering.append(float(indexed_data[3]))
  image_paths = np.asarray(image_path)
  steerings = np.asarray(steering)
  return image_paths, steerings

image_paths, steerings = load_img_steering(datadir + '/IMG', data)
```

The next step was to split the data using the 80–20 rule which means using 80% of the data for training while the rest for testing the model on unseen images. Also, I plotted the sample training and validation steering angle distributions.

```
X_train, X_valid, Y_train, Y_valid = train_test_split(image_paths, steerings, test_size=0.2,
random_state=0)

print("Training Samples: {}\nValid Samples: {}".format(len(X_train), len(X_valid)))
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
axes[0].hist(Y_train, bins=num_bins, width=0.05, color='blue')
axes[0].set_title('Training set')
axes[1].hist(Y_valid, bins=num_bins, width=0.05, color='red')
axes[1].set_title('Validation set')
```

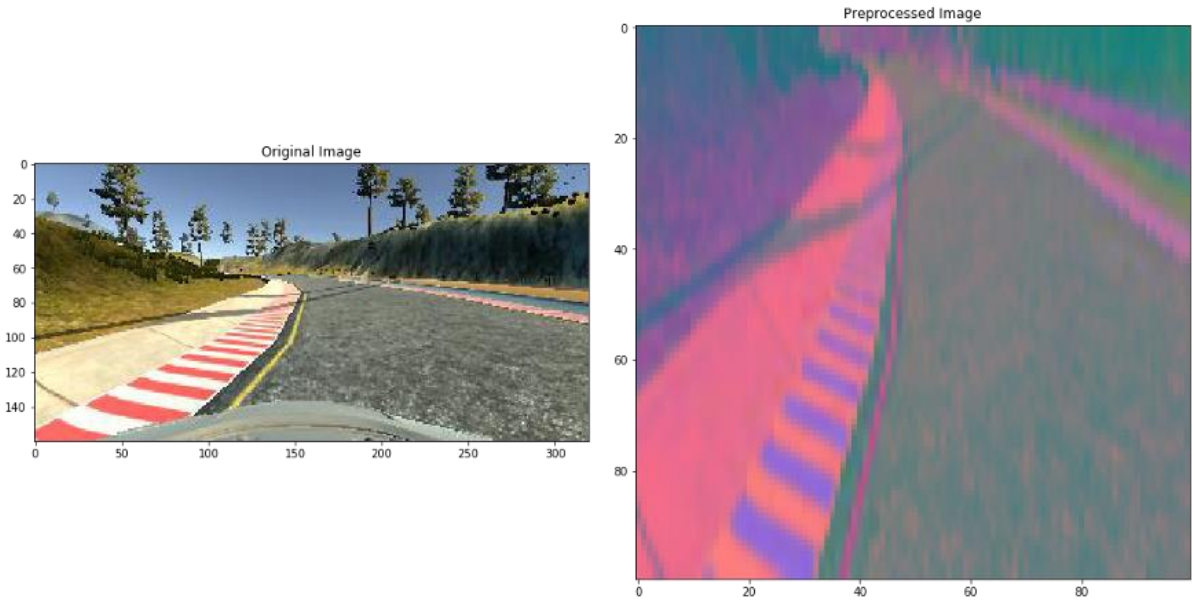


I continued by doing some image processing. I cropped the image to remove the unnecessary features, changes the images to YUV format, used gaussian blur, decreased the size for easier processing and normalized the values.

```
def img_preprocess(img):
    img = npimg.imread(img)

    ## Crop image to remove unnecessary features
    img = img[60:135, :, :]

    ## Change to YUV image
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)

    ## Gaussian blur
    img = cv2.GaussianBlur(img, (3, 3), 0)

    ## Decrease size for easier processing
    img = cv2.resize(img, (100, 100))

    ## Normalize values
    img = img / 255
    return img
```

To compare and visualize I plotted the original and the pre-processed image.

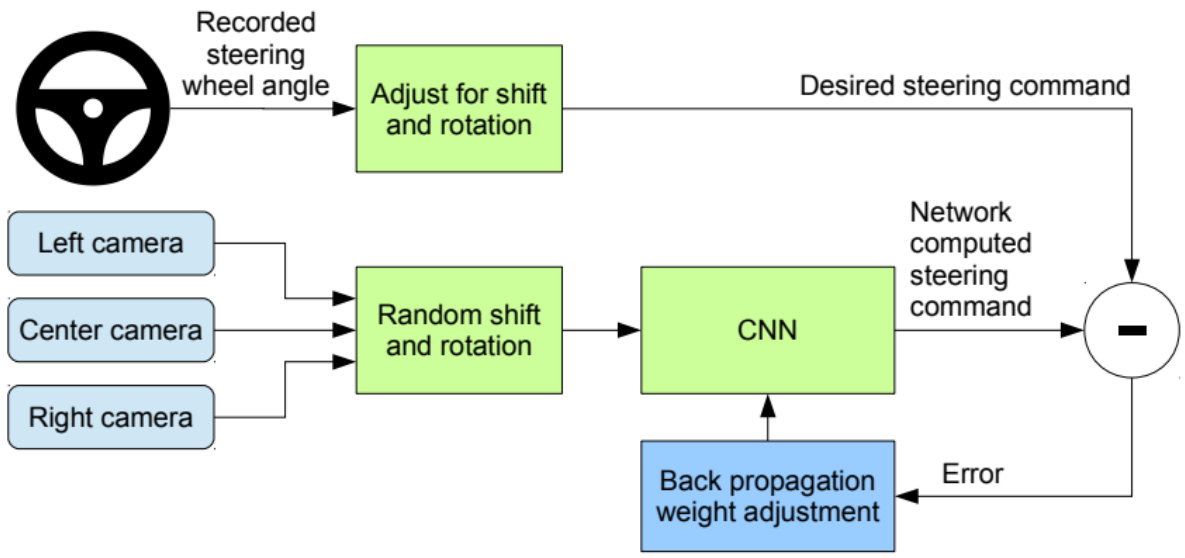
```
image = image_paths[100]
original_image = npimg.imread(image)
preprocessed_image = img_preprocess(image)

fig, axes = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()
axes[0].imshow(original_image)
axes[0].set_title('Original Image')
axes[1].imshow(preprocessed_image)
axes[1].set_title('Preprocessed Image')
```



So far so good. Next, I converted all the images into numpy array.

```
X_train = np.array(list(map(img_preprocess, X_train)))
X_valid = np.array(list(map(img_preprocess, X_valid)))
```

The next step was to build the model. I have used ResNet as the pre-trained weights. I have removed the last 4 layers to make my own custom neural network.



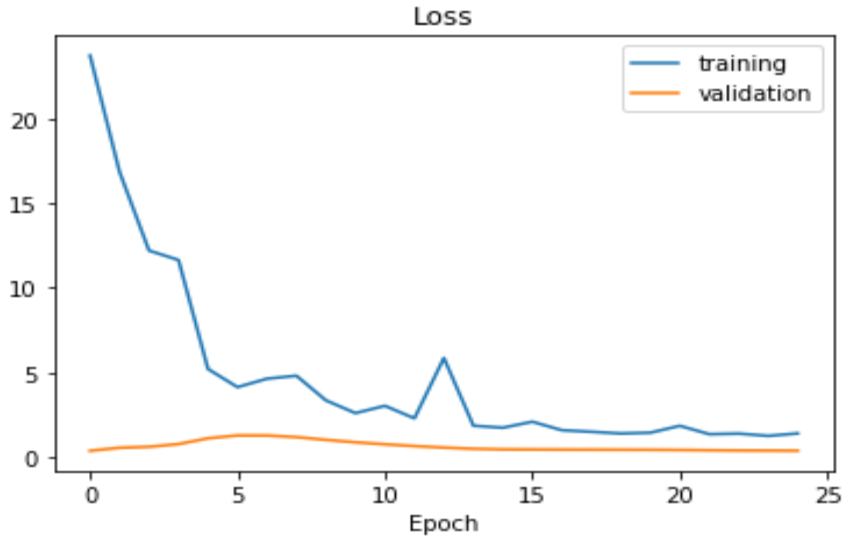
```
from keras.applications import ResNet50
resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(100, 100, 3))

for layer in resnet.layers[:-4]:
    layer.trainable = False

for layer in resnet.layers:
    print(layer, layer.trainable)
```

On top of the heavy resnet architecture, I have used the flatten layer to normalize the weights. Next, I have used three dense layers with 100, 50 and 10 neurons respectively and elu as the activation function. Also in between, I have used 50% dropouts to reduce over-fitting the values to the training set.

```
def nvidia_model():
  model = Sequential()
  model.add(resnet)
  model.add(Dropout(0.5))
  model.add(Flatten())
  model.add(Dense(100, activation='elu'))
  model.add(Dropout(0.5))
  model.add(Dense(50, activation='elu'))
  model.add(Dropout(0.5))
  model.add(Dense(10, activation='elu'))
  model.add(Dropout(0.5))
  model.add(Dense(1))
  optimizer = Adam(lr=1e-3)
  model.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
  return model

model = nvidia_model()
print(model.summary())
```

Finally, I trained the model for 25 epochs with a batch size of 128. Also, I plotted the training and the validation loss as a function of epochs.

```
history = model.fit(X_train, Y_train, epochs=25, validation_data=(X_valid, Y_valid),
batch_size=128, verbose=1, shuffle=1)

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training', 'validation'])
plt.title('Loss')
plt.xlabel('Epoch')
```



The model is converging quite good in just 25 epochs. It means that it is learning a fairly good policy to steer a car on unseen road environments. Feel free to play with the hyper-parameters for better results.

RESULTS

I was surprised with how well the car drove on the test track. The CNN had never seen this track. The performance on the training track was a little off but I think that's fine as it shows that the car was not merely memorizing the track. It recovered successfully from a few critical situations, even though none of those maneuvers had been performed during training.

CONCLUSIONS

Summarizing, this was a really interesting and at the same time challenging project to work on. Deep learning is an exciting field and we're lucky to live in these times of invention. In 10 years most of us probably won't own a car. We'll have some subscription with some company like an Uber… and we would pay $149 a month and every morning we wake up with a car in our driveway that will take us to work.

Developments in autonomous cars are continuing and the software in the car is continuing to be updated. Though it all started from a driverless thought to radiofrequency, cameras, sensors, more semi-autonomous features will come up, thus reducing the congestion, increasing safety with faster reactions, and fewer errors. Despite the inherent benefits, autonomous vehicle technology must overcome many social barriers.