SELF DRIVING CAR

15.04.2023

Sandipan Bhowmik

RollNo.181

Enrollment No.22021002002013

Somadrita Paul

RollNo.173

Enrollment No.22021002002005

Angana Pramanik

RollNo.187

Enrollment No.22021002002019

Niladri Dey

RollNo.218

Enrollment No.22021002002050

SELF DRIVING CAR

Abstract

In this proposed work, a prototype of a self-driving vehicle is designed and developed which uses OpenCV and machine learning technology. The vehicle can drive itself and follow traffic rules and regulations. Using data sets it can clearly detect objects and can drive over the roads. Using machine learning the data sets are utilized and give better results in the output.

Key Words: OpenCV, Machine Learning

Overview

The topic of self-driving cars involves the development and implementation of autonomous vehicles that can navigate and operate on the road without human intervention. Self-driving cars rely on a combination of advanced technologies, including machine learning, computer vision, robotics, and sensors, to perceive their surroundings and make decisions based on that perception. The development of self-driving cars is driven by a number of factors, including the potential for increased safety, improved efficiency, and increased accessibility for people with disabilities or limited mobility. The technology behind self-driving cars is still evolving and requires significant research and development to ensure that it is safe and reliable. Some of the key challenges involved in developing self-driving cars include ensuring that they can operate safely in a variety of road and weather conditions, navigating complex intersections and roadways, and avoiding collisions with other vehicles and pedestrians.

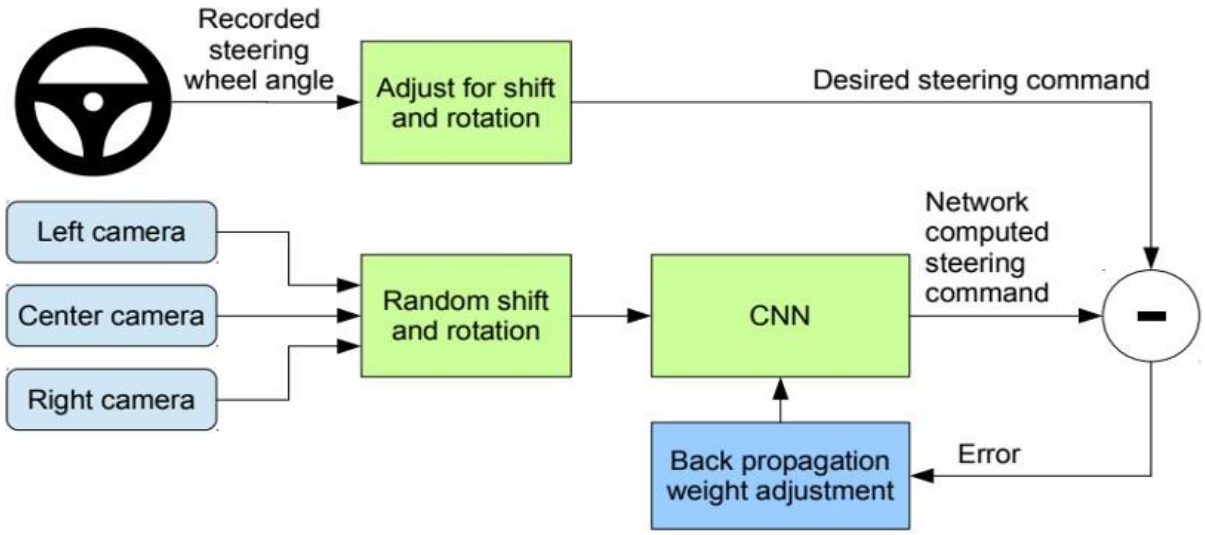
Objective

Road Safety is one of the major issues faced globally. Every year roughly 8 million people meet with road accidents and around 1.4 million lose their lives worldwide. These deaths are majorly caused by human negligence and human error. The most common reasons for road accidents are over speeding, drink & driving, distractions to drivers, red light jumping, lane changing and avoiding safety gears like seat belts. Traditional and manual cars require complete human control and attention over them while traveling, where most of the people make mistakes which eventually results in road accidents. To overcome this issue Self driving vehicles technology comes into picture. Self driving vehicles are those vehicles which are capable of sensing its environment and move safely with no human input. Various high-end sensors are situated in different parts of the car which detect its surroundings and send the data to the High Performance computing devices which then instructs the vehicle to follow a particular path and take decisions. Majorly with the help of video cameras, the vehicle detects the traffic lights, road signs, pedestrians and other cars on the road and sends those real time images to the processor. Machine learning is mostly used for perception and decision making in real-time. The algorithms need to make the decisions within a fraction of a second for applying a break or to make a turn right or left. The main goals of the project is to design and develop a low-cost prototype of an autonomous vehicle which uses raspberry pi as the core functioning unit and uses OpenCV and machine learning-technology.

Methodology

Autonomous cars rely on sensors, actuators, complex algorithms, machine learning systems, and powerful processors to execute software. Autonomous cars create and maintain a map of their surroundings based on a variety of sensors situated in different parts of the vehicle.

Working Diagram:



OpenCV:

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library developed by Intel. OpenCV is used to provide a common architecture for computer vision applications and to accelerate the use of machine learning in the commercial products. The library has more than 2500 algorithms, which includes a set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms are used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects.

Computer Vision tasks performed in this project are:

- Region of Interest (ROI)
- Perspective Transformation (Bird Eye View)
- Grayscale Conversion and Thresholding
- Canny Edge Detection
- Gaussian Blurring

Haar Cascade:

Haar cascade classifier is a machine learning object detection software that identifies objects in an image and video. The algorithm can be stated in four phases:

- Calculation of Haar Features
- Creating Integral Images using Adaboost
- Implementing Cascading Classifiers.

This algorithm requires a lot of positive images samples and negative images samples of its surroundings to train the machine learning model.

Other environments and tools are used like matplotlib, keras, numpy, pandas, scikit-learn.

Code

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as npimg
import os

## Keras
import keras
from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense

import cv2
import pandas as pd
import random
import ntpath

# # Sklearn
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
```

Out[12]:

	center	left	right	steering
0	C:\Users\Win 10\Desktop\benign\IMG\center_2019_07_22_20_38_15_382.jpg	C:\Users\Win 10\Desktop\benign\IMG\left_2019_07_22_20_38_15_382.jpg	C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_382.jpg	0.0
1	C:\Users\Win 10\Desktop\benign\IMG\center_2019_07_22_20_38_15_526.jpg	C:\Users\Win 10\Desktop\benign\IMG\left_2019_07_22_20_38_15_526.jpg	C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_526.jpg	0.0
2	C:\Users\Win 10\Desktop\benign\IMG\center_2019_07_22_20_38_15_669.jpg	C:\Users\Win 10\Desktop\benign\IMG\left_2019_07_22_20_38_15_669.jpg	C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_669.jpg	0.0
3	C:\Users\Win 10\Desktop\benign\IMG\center_2019_07_22_20_38_15_802.jpg	C:\Users\Win 10\Desktop\benign\IMG\left_2019_07_22_20_38_15_802.jpg	C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_802.jpg	0.0
4	C:\Users\Win 10\Desktop\benign\IMG\center_2019_07_22_20_38_15_937.jpg	C:\Users\Win 10\Desktop\benign\IMG\left_2019_07_22_20_38_15_937.jpg	C:\Users\Win 10\Desktop\benign\IMG\right_2019_07_22_20_38_15_937.jpg	0.0

Store data

```
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()
```

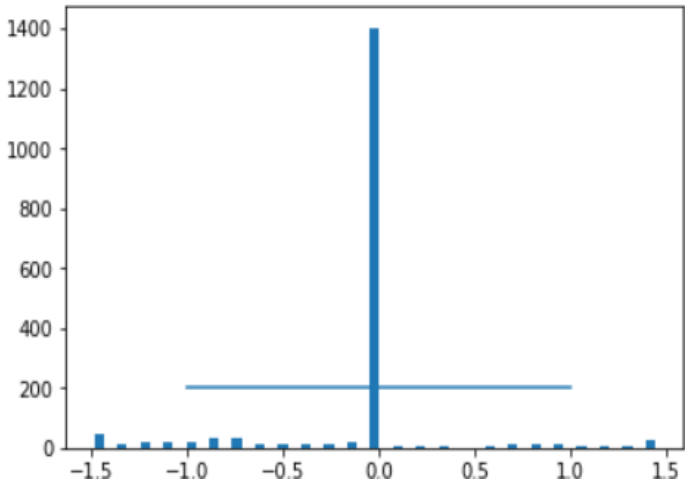
```
def path_leaf(path):
##Get tail of path
head, tail = ntpath.split(path)return
tail

## Remove path of images
data['center'] = data['center'].apply(path_leaf)
data['left'] = data['left'].apply(path_leaf) data['right'] =
data['right'].apply(path_leaf) data.head()
```

Out[13]:

	center	left	right	steering	throttle	reverse	speed
0	center_2019_07_22_20_38_15_382.jpg	left_2019_07_22_20_38_15_382.jpg	right_2019_07_22_20_38_15_382.jpg	0.0	0.0	0	0.000079
1	center_2019_07_22_20_38_15_526.jpg	left_2019_07_22_20_38_15_526.jpg	right_2019_07_22_20_38_15_526.jpg	0.0	0.0	0	0.000082
2	center_2019_07_22_20_38_15_669.jpg	left_2019_07_22_20_38_15_669.jpg	right_2019_07_22_20_38_15_669.jpg	0.0	0.0	0	0.000078
3	center_2019_07_22_20_38_15_802.jpg	left_2019_07_22_20_38_15_802.jpg	right_2019_07_22_20_38_15_802.jpg	0.0	0.0	0	0.000078
4	center_2019_07_22_20_38_15_937.jpg	left_2019_07_22_20_38_15_937.jpg	right_2019_07_22_20_38_15_937.jpg	0.0	0.0	0	0.000080

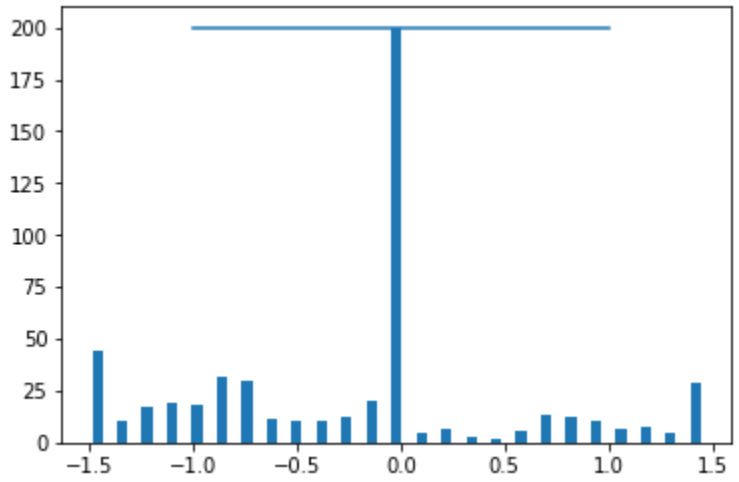
```
## Visualize data
num_bins = 25
samples_per_bin = 200
hist, bins = np.histogram(data['steering'], num_bins) center
= bins[:-1] + bins[1:] * 0.5 # center the bins to 0

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

```
Out[14]: []
```




```
print('Total data: {0}'.format(len(data)))
## Make list of indices to remove
remove_list = []
for j in range(num_bins):
  list_ = []
  for i in range(len(data['steering'])):
    steering_angle = data['steering'][i]
    if steering_angle >= bins[j] and steering_angle <= bins[j+1]:
          list_.append(i)
  list_ = shuffle(list_)
  list_ = list_[samples_per_bin:]
  remove_list.extend(list_)
```

```
## Remove from extras from list
data.drop(data.index[remove_list], inplace=True)
print('Removed: {0}'.format(len(remove_list)))
print('Remaining: {0}'.format(len(data)))

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

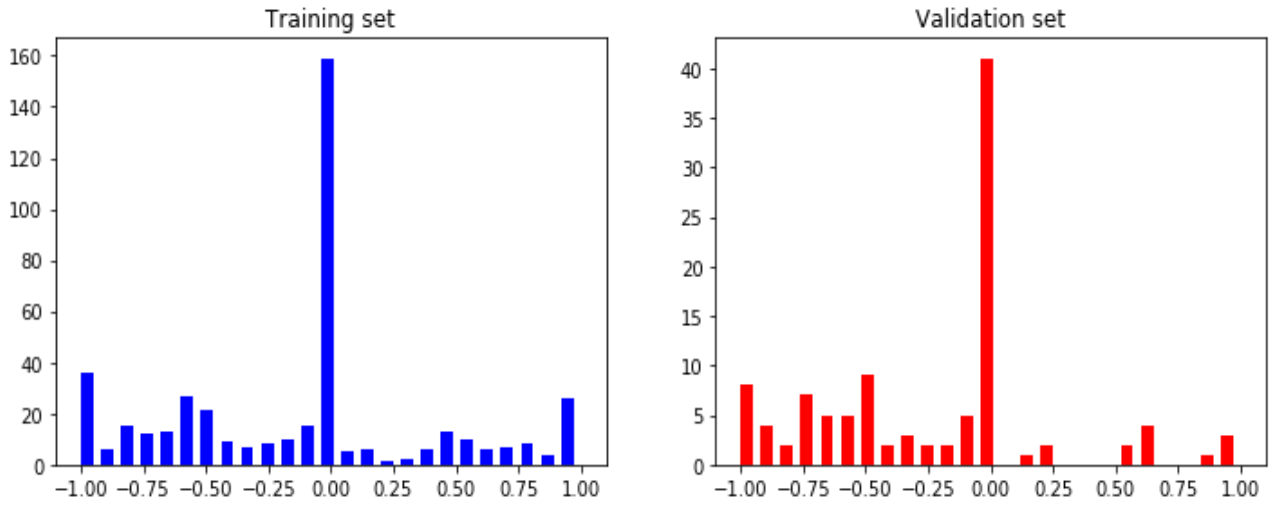



```
def load_img_steering(datadir, df):
  ## Get img and steering data into arrays
  image_path = []
  steering = []
  for i in range(len(data)):
    indexed_data = data.iloc[i]
    center, left, right = indexed_data[0], indexed_data[1], indexed_data[2]
    image_path.append(os.path.join(datadir, center.strip()))
    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)
```

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

Check that data is valid

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')



def img_preprocess(img):

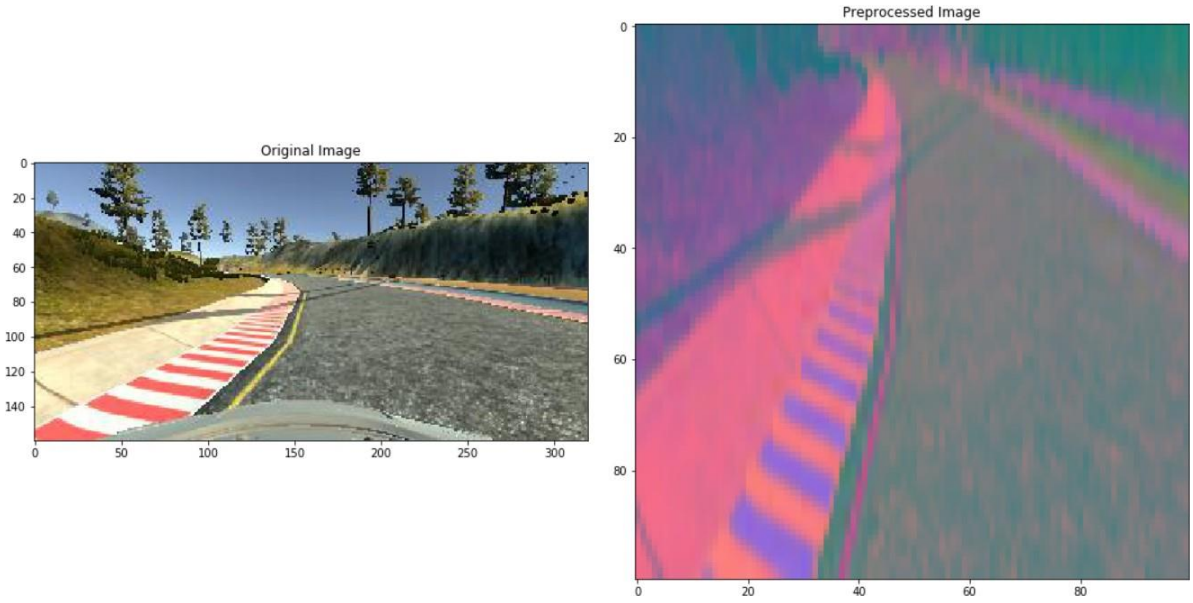
##Take in path of img, returns preprocessed image

img = npimg.imread(img)

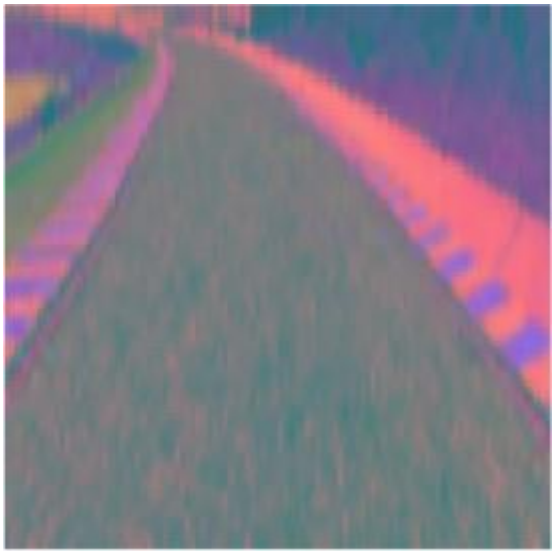
Crop image to remove unnecessary features

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

## Get any image
image = image_paths[100] original_image
= npimg.imread(image)
preprocessed_image  =  img_preprocess(image)

## Compare original and preprocessed 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')
```




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

plt.imshow(X_train[random.randint(0, len(X_train)-1)])
plt.axis('off')
 print(X_train.shape)
```



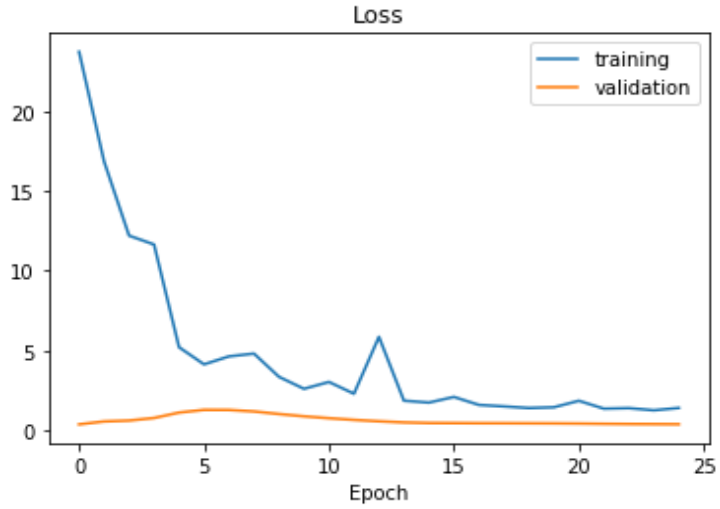
```
from keras.applications import ResNet50
#Load the ResNet50 model
resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(100, 100,3))

# Freeze the layers except the last 4 layers
for layer in resnet.layers[:-4]:
    layer.trainable = False

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

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())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 4, 4, 2048)        23587712
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 2048)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 32768)             0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               3276900
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dropout_4 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 26,870,183
Trainable params: 4,337,191
Non-trainable params: 22,532,992
_________________________________________________________________
None
```

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

```
Train on 432 samples, validate on 108 samples
Epoch 1/25
432/432 [==============================] - 1s 1ms/step - loss: 1.4304 - acc: 0.1597 - val_loss: 0.4176 - val_acc: 0.3796
Epoch 21/25
432/432 [==============================] - 1s 1ms/step - loss: 1.8383 - acc: 0.1736 - val_loss: 0.4077 - val_acc: 0.3796
Epoch 22/25
432/432 [==============================] - 1s 1ms/step - loss: 1.3433 - acc: 0.1620 - val_loss: 0.3917 - val_acc: 0.3796
Epoch 23/25
432/432 [==============================] - 1s 1ms/step - loss: 1.3764 - acc: 0.1551 - val_loss: 0.3795 - val_acc: 0.3796
Epoch 24/25
432/432 [==============================] - 1s 1ms/step - loss: 1.2427 - acc: 0.1435 - val_loss: 0.3730 - val_acc: 0.3796
Epoch 25/25
432/432 [==============================] - 1s 1ms/step - loss: 1.3871 - acc: 0.1875 - val_loss: 0.3651 - val_acc: 0.3796
```

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



Result:

Conclusion

A prototype of a self-driving car model is designed and developed. With the help of Image Processing and Machine Learning techniques a successful model was developed which worked as per our expectation. Image processing techniques like calculating region of interest (ROI), conversion of images from RGB into grayscale, applying Gaussian filter for removing noise, canny edge detection for extracting lane edges are successfully performed for detection of lanes as well as machine learning model is successfully trained in HAAR cascade software by using positive and negative samples of Stop Sign, Red Light, Green light and Obstacles for traffic sign detection. At last, the car is able to follow the lane efficiently and the traffic signs are detected properly as well as followed and decisions are made accurately. In conclusion, self-driving cars are a promising technology that have the potential to transform transportation and improve safety, efficiency, and accessibility. The development of self-driving cars relies on a combination of advanced technologies, including machine learning, computer vision, and robotics, and requires significant research and development.

Bibliography

- Deep Learning for Autonomous Driving by Markus Wulfmeier, Abbas Abdolmaleki, and Roland Hafner. This book covers the use of deep learning techniques in autonomous driving systems. It includes topics such as perception, prediction, planning, and control, and provides practical examples and code snippets.

- End-to-End Learning for Self-Driving Cars by Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xiaoyu Zhang, and Jake Zhao. This paper describes a deep learning approach for end-to-end driving, which means that the system learns to map raw sensor inputs directly to steering commands. It achieved promising results in simulation and real-world tests.

- Machine Learning Techniques for Autonomous Vehicles Perception: A Survey" by Yanan Liu, Yuyue Wang, and Wenjie Wang. This paper provides an overview of machine learning techniques that are used in perception tasks for autonomous driving, such as object detection and tracking, semantic segmentation, and depth estimation. It discusses the advantages and limitations of different methods and provides insights into future directions.

- Reinforcement Learning for Autonomous Driving: A Survey by Jiaoyang Li, Jiaqi Ma, and Wenbo Hu. This paper reviews the use of reinforcement learning (RL) in autonomous driving systems, which involves training an agent to take actions that maximize a reward signal. It covers RL algorithms, training environments, and evaluation metrics, as well as challenges and opportunities in applying RL to autonomous driving.

- Artificial Intelligence and Autonomous Vehicles: 2020 Edition by the Stanford Artificial Intelligence Laboratory. This report provides an overview of the current state of AI and autonomous vehicles, as well as trends and challenges. It covers topics such as perception, prediction, planning, control, and human factors, and discusses the ethical and legal implications of autonomous driving.