ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH

(Formerly College of Engineering and Technology) Odisha

Techno Campus, Ghatikia, Mahalaxmi Vihar, Bhubaneswar-751029

MINOR PROJECT REPORT ON

"Self-Driving Car System"

Submitted by:

Nishikanta Parida

Regd. No.: **2124100015**

Under the Guidance of

Dr. Debasis Gountia

(Associate Professor in CSA Department)

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATION

Year 2021 - 2023

. . . .

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH BHUBANESWAR

(Techno Campus, PO-Ghatikia, Mahalaxmi Vihar, Bhubaneswar, 751003)

CERTIFICATE

This is to certify that the work embodied in the project work entitled "**SELF-DRIVING CAR System**" being submitted by **Mr. NISHIKANTA PARIDA** in partial fulfilment for the award of the Degree of Master of Computer Science and Application to the Odisha University of Technology & Research is a record of bonafide work carried out by him under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Signature of Internal Guide

Name: **Dr. Debasis Gountia**
Designation: Associate Professor

Signature of Head of the department

Name: **Dr. Jibitesh Mishra**
Designation: Associate Professor

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH BHUBANESWAR

(Techno Campus, PO-Ghatikia, Mahalaxmi Vihar, Bhubaneswar, 751029)

DECLARATION

I **Nishikanta Parida** bearing Regd. no.: **2124100015,** a bonafide student of Odisha University of Technology & Research, would like to declare that the Project work entitled "**SELF-DRIVING CAR System**", is a record of an original work done by me under the esteemed guidance of **Dr. Debasis Gountia**, Associate Professor in Computer Science & Application Department. This project work is submitted in the partial fulfilment of the requirements for Master's degree & not submitted for the award of my degree.

Nishikanta Parida

Regd. No.: 2124100015

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH BHUBANESWAR

(Techno Campus, PO-Ghatikia, Mahalaxmi Vihar, Bhubaneswar, 751029)

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my advisor **Dr. Debasis Gountia**, Associate Professor, Department of Computer Science and Application, whose knowledge and guidance has motivated me to achieve goals I never thought possible. She has consistently been a source of motivation, encouragement, and inspiration.

I would also like to convey my deep regards to all other faculty members of Department of Computer Science and Application, who have bestowed their great effort and guidance at appropriate times without which it would have been very difficult on my part to finish this work. Finally, I would also like to thank my friends for their suggestions and cooperation.

What I write or mention in this sheet will hardly be adequate in return for the amount of help and cooperation I have received from all the people who have contributed to make this project a reality. I am grateful to all for their constant support.

Nishikanta Parida

Date: ***Regd. No.: 2124100015***

ABSTRACT

Most of the current self-driving cars make use of multiple algorithms to drive. Furthermore, most of the approaches use supervised learning to train a model to drive the car autonomously. **The objective of this project is to build an end-to-end self-driving (level-1) system**. We implement the Deep *Q*-Learning algorithm to control a simulated car, end-to-end, autonomously. The algorithm is based on reinforcement learning which teaches machines what to do through interactions with the environment. The application of reinforcement learning for driving is of high relevance as it is highly dependent on interactions with the environment. Our model incorporates CNN as the deep *Q* network. The system was tested on an open-source car-racing simulator. The Deep *Q*-Learning approach allows the system to be more efficient and robust than a system that has been trained solely through supervised training. Our simulation results show that the system is able to drive autonomously and maneuver complex curves.

In this we used Deep learning, a type of machine learning that imitates the way humans gain certain types of knowledge. It is an important element of data science, which includes statistics and predictive modelling. This is used to take the decisions and to classify the different elements in the environment like the lanes, road signs etc.

We also used Reinforcement learning, an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior it should take in a specific situation. This system is used to reward the model for successfully driving the cars and to build upon the previously acquired knowledge through repeated simulations.

INDEX TERMS:

Self-driving car (level-1), Object detection, Reinforcement Learning, Deep Learning, Deep Q-Learning.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

INTRODUCTION:

The CNNs [1] have revolutionized pattern recognition [2]. Prior to the widespread adoption of CNNs, most pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images. Also, by using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations.

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.

CHAPTER 2

SOFTWARE & HARDWARE REQUIREMENTS

SOFTWARE & HARDWARE REQUIREMENTS:

2.1. SOFTWARE REQUIREMENTS

2.1.1. Anaconda interpreter (to run deployed project i.e., Python file)

2.1.2. Jupyter Notebook (For Model building, training and testing)

2.1.3. Visual Studio Code

2.1.4. Python Libraries

- **os:** provides functions for creating and removing a directory (folder), fetching its contents etc.
- cv2
- Numpy
- Pandas
- Matplotlib
- TensorFlow
- TensorFlow Object Detection API
- OpenCV
- PyTorch

2.2. HARDWARE REQUIREMENTS

2.2.1. Windows 10- 64 Bit

2.2.2. AMD Ryzen 5 Gen 3

2.2.3. Graphics card GeForce GTX 1650 4GB

2.2.4. RAM 8 GB

2.2.5. SSD: 512 GB (50 GB minimum)

CHAPTER 3

Literature Survey

3. Literature Survey

Most of the current self-driving cars make use of multiple algorithms to drive. Furthermore, most of the approaches use supervised learning to train a model to drive the car autonomously. This approach leads to human bias being incorporated into the model. We implement the Deep Q-Learning algorithm to control a simulated car, end-to-end, autonomously. The algorithm is based on reinforcement learning which teaches machines what to do through interactions with the environment, The application of reinforcement learning for driving is of high relevance as it is highly dependent on interactions with the environment. Our model incorporates a CNN as the deep Q network.

While CNNs with learned features have been in commercial use for over twenty years [3], their adoption has exploded in the last few years because of two recent developments. First, large, labeled data sets such as the Large-Scale Visual Recognition Challenge (ILSVRC) [4] have become available for training and validation. Second, CNN learning algorithms have been implemented on the massively parallel graphics processing units (GPUs) which tremendously accelerate learning and inference.

In this, we describe a CNN that goes beyond pattern recognition. It learns the entire processing pipeline needed to steer an automobile. The DARPA Autonomous Vehicle (DAVE) [5] in which a sub-scale radio control (RC) car drove through a alley way. DAVE was trained on hours of human driving in similar, but not identical environments. The training data included video from two cameras coupled with left and right steering commands from a human operator.

In many ways, DAVE-2 was inspired by the pioneering work of Pomerleau [6] who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. It demonstrated that an end-to-end trained neural network can indeed steer a car on public roads.

While DAVE demonstrated the potential of end-to-end learning, and indeed was used to justify starting the DARPA Learning Applied to Ground Robots (LAGR) program [7], DAVE's performance was not sufficiently reliable to provide a full alternative to more modular approaches to off-road driving. DAVE's mean distance between crashes was about 20 meters in complex environments.

The primary motivation for this work is to avoid the need to recognize specific human-designated features, such as lane markings, guard rails, or other cars, and to avoid having to create a collection of "if, then, else" rules.

Meanwhile, DeepMind's work [10] on deep reinforcement learning opened up new research avenues in problems where the system has to interact with the environment. This deep reinforcement learning proved to give superhuman performance in tasks that did not even perform as good as a human, using previous strategies.

CHAPTER 4

SOFTWARE REQUIREMENT ANALYSIS

SOFTWARE REQUIREMENT ANALYSIS:

4.1 PROBLEM STATEMENT

A SELF DRIVING CAR system is a key technique in most of road transport related applications and is an active research topic in the AI and image processing domain.

But due to the varying characteristics of the road, weather conditions, road signs, pedestrian Behaviour, and road safety rules further research is still needed in this area to develop a safe fully autonomous car.

4.2 PURPOSED SOLUTION

Our proposed solution is to implement the SELF DRIVING CAR system by using **a convolutional neural network (CNN)** model to map raw pixels from a single front-facing camera directly to steering commands.

And another model to classify traffic lights and road signs using **deep convolutional neural network** to perform detection and recognition in one model with high accuracy which can used to identify each signal light and signs can be used by the car to take proper decision.

4.3 DATASET DESCRIPTION:

The data has been obtained from the source:

4.3.1 Udacity's Self-Driving Car Simulator:

This simulator was built for Udacity's Self-Driving Car Nanodegree, to teach students how to train cars how to navigate road courses using deep learning. We used this simulator to capture images from three angles to train the CNN model.

4.3.2 LISA Traffic Light Dataset:

The database consists of continuous test and training video sequences, totaling 43,007 frames and 113,888 annotated traffic lights. The sequences are captured by a stereo camera mounted on the roof of a vehicle driving under both night- and daytime with varying light and weather conditions. It was used to train the model to detect and classify traffic light.

4.3.3 COCO Dataset:

The MS COCO dataset is a large-scale object detection, segmentation, and captioning dataset published by Microsoft. Machine Learning and Computer Vision engineers popularly use the COCO dataset for various computer vision projects. The **yolo5** model is trained on this dataset.

4.4 SOFTWARE DESCRIPTION:

4.4.1 Python:

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming. Many other paradigms are supported via extensions, including design by contract and logic programming. Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

4.4.2 Anaconda:

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and MacOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, both of which are not free. Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source package can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigato**r** as a graphical alternative to the command line interface (CLI).

13

The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

4.4.3 Jupyter Notebook:

The Jupyter Notebook is an opensource web application that you can use to create and share documents that contain live code, equations, visualizations, and text. In other words, Jupyter Notebook is an open-source, web-based IDE with deep cross language integration that allows you to create and share documents containing live code, equations, visualizations, and narrative text. Data scientists and engineers use Jupyter for data cleaning and transformation, statistical modeling, visualization, machine learning, deep learning, and much more. Jupyter Notebook's format (ipynb) has become an industry standard and can be rendered in multiple IDEs, GitHub, and other places. Jupyter has support for over 40 programming languages, including Python, R, Julia, and Scala. Notebooks can be shared easily with others, and your code can produce rich, interactive output, including HTML, images, videos, and custom MIME types. It allows you to leverage big data tools such as Spark and explore that same data with pandas, scikit-learn, TensorFlow, and ggplot2.

4.4.4 Numpy:

Numpy is a general-purpose array-processing package. It provides a highperformance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficientmulti dimensional container of generic data. At the core of the NumPy package, is the ndarray
object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

14

4.4.5 Pandas:

Pandas is a fast, powerful, flexible and easy to use opensource data analysis and manipulation tool, built on top of the Python programming language. Pandas is a Python package providing fast, flexible and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis is python.

4.4.6 Matplotlib:

Matplotlib is quite possibly the simplest way to plot data in Python. It is similar to plotting in MATLAB, allowing users full control over fonts, line styles, colors, and axes properties. This allows for complete customization and fine control over the aesthetics of each plot, albeit with a lot of additional lines of code. Plotly is another great Python visualization tool that's capable of handling geographical, scientific, statistical, and financial data. Plotly has several advantages over matplotlib. One of the main advantages is that only a few lines of codes are necessary to create aesthetically pleasing, interactive plots. The interactivity also offers a number of advantages over static matplotlib plots.

4.4.7 TensorFlow:

Tensorflow is an open-source library for numerical computation and large-scale machine learning that ease Google Brain TensorFlow, the process of acquiring data, training models, serving predictions, and refining future results.

Tensorflow bundles together Machine Learning and Deep Learning models and algorithms. It uses Python as a convenient front-end and runs it efficiently in optimized C++.Tensorflow allows developers to create a graph of computations to perform. Each node in the graph represents a mathematical operation and each connection represents data. Hence, instead of dealing with low-details like figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application.

4.4.8 TensorFlow Object Detection API:

The TensorFlow object detection API is the framework for creating a deep learning network that solves object detection problems. There are already pretrained models in their framework which they refer to as Model Zoo. This includes a collection of pretrained models trained on the COCO dataset, the KITTI dataset, and the Open Images Dataset. These models can be used for inference if we are interested in categories only in this dataset. They are also useful for initializing the models when training on the novel dataset.

4.4.9 OpenCV:

OpenCV (Open-Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel. OpenCV features GPU acceleration for real-time operations. is an open source computer vision and machine learning software library.OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception.

4.4.10 PyTorch:

PyTorch is an open-source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Meta AI. A number of pieces of deep learning software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers,

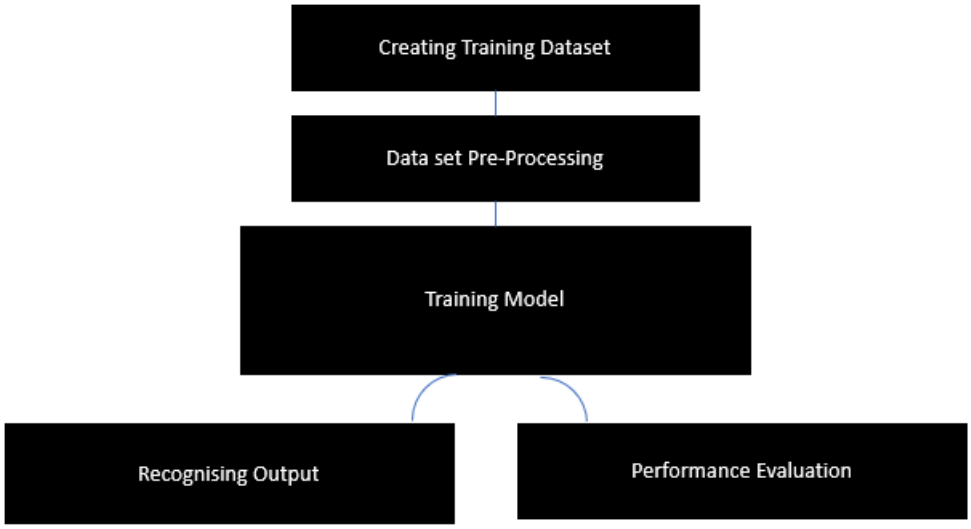
PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU).
- Deep neural networks built on a tape-based automatic differentiation system.

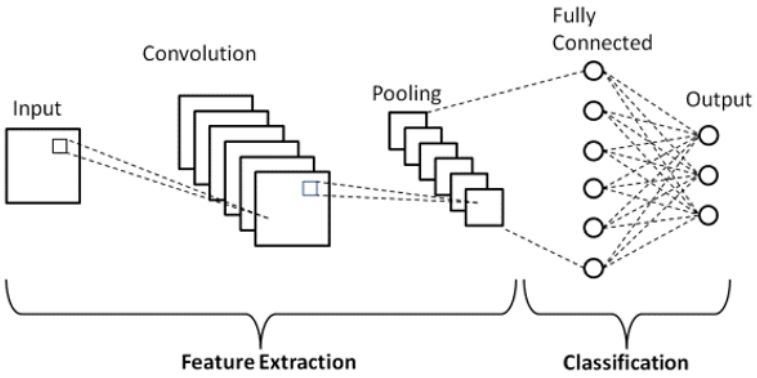
CHAPTER 5

SOFTWARE DESIGN

5.1 METHODOLOGY:



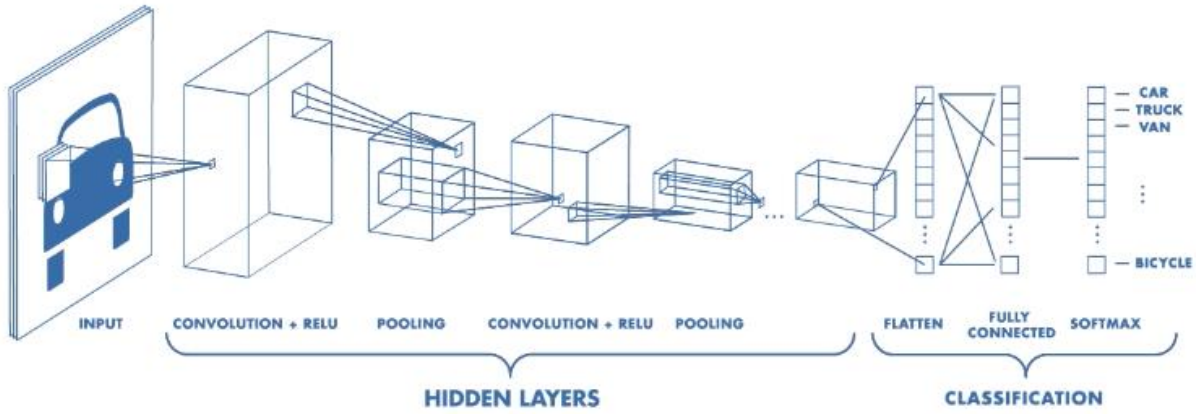
5.2 Convolutional Neural Network Model:



Convolutional Neural Network (CNN) Algorithm:

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what colour each pixel should be.

The human brain processes a huge amount of information the second we see an image. Each neuron works in its own receptive field and is connected to other neurons in a way that they cover the entire visual field. Just as each neuron responds to stimuli only in the restricted region of the visual field called the receptive field in the biological vision system, each neuron in a CNN processes data only in its receptive field as well. The layers are arranged in such a way so that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along. By using a CNN, one can enable sight to computers.



5.2.1 Convolution Layer

The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load. This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

5.2.2 Pooling Layer

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

5.2.3 Fully Connected Layer

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect. The Fully Connected layer helps to map the representation between the input and the output.

5.2.4 Non-Linearity Layers

Since convolution is a linear operation and images are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map.

There are several types of non-linear operations, the popular ones being:

a. Sigmoid

The sigmoid non-linearity has the mathematical form $\sigma(\kappa) = 1/(1+e^{-}\kappa)$. It takes a real-valued number and "squashes" it into a range between 0 and 1.

b. Tanh

Tanh squashes a real-valued number to the range [-1, 1]. Like sigmoid, the activation saturates, but — unlike the sigmoid neurons — its output is zero centered.

c. ReLU

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function $f(\kappa)=\max(0, \kappa)$. In other words, the activation is simply threshold at zero.

5.3 Training the Neural Network:

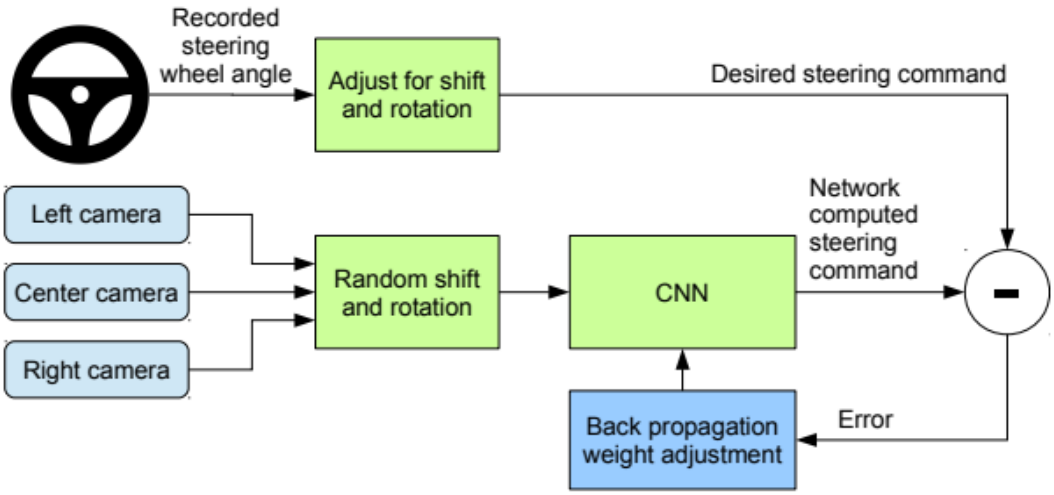
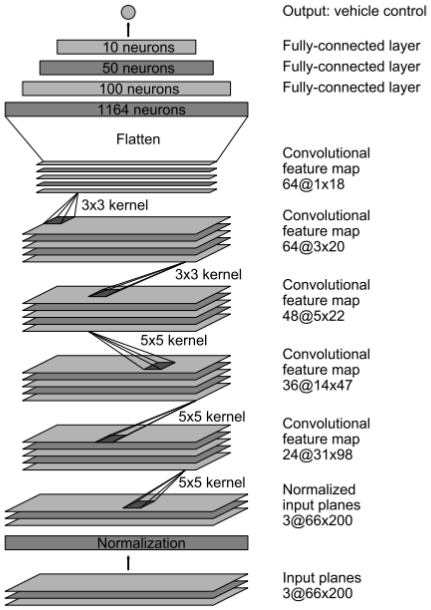


Figure 2: Training the neural network.

5.4 CNN ARCHITECTURE:



CHAPTER 6

CODE TEMPLATES AND RESULTS

6.1 -The program to create, train and test the CNN Model (Driving):

6.1.1 LoadData.py

```
 import cv2
import numpy as np
import csv
import pickle
import matplotlib.pyplot as plt


features_directory = './data/'
labels_file = './data/driving_log.csv'



def preprocess(img):
   resized = cv2.resize((cv2.cvtColor(img, cv2.COLOR_RGB2HSV))[:, :, 1], (40, 40))
   return resized



def data_loading(delta):
```

```
    logs = []
    features = []
    labels = []
    with open(labels_file, 'rt') as f:
        reader = csv.reader(f)
        for line in reader:
            logs.append(line)
        log_labels = logs.pop(0)

    for i in range(len(logs)):
        for j in range(3):
            img_path = logs[i][j]
            img_path = features_directory + 'IMG' +
(img_path.split('IMG')[1]).strip()
            img = plt.imread(img_path)
            features.append(preprocess(img))
            if j == 0:
                labels.append(float(logs[i][3]))
            elif j == 1:
                labels.append(float(logs[i][3]) + delta)
            else:
                labels.append(float(logs[i][3]) - delta)
    return features, labels
delta = 0.2
features, labels = data_loading(delta)
```

```
features = np.array(features).astype('float32')
labels = np.array(labels).astype('float32')

with open("features_40", "wb") as f:
    pickle.dump(features, f, protocol=4)
with open("labels", "wb") as f:
    pickle.dump(labels, f, protocol=4)
```

6.1.2 TrainModel.py

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from keras.layers import Input, Dense, Activation, Flatten, Conv2D, Lambda
from keras.layers import MaxPooling2D, Dropout
from keras.utils import print_summary
import tensorflow as tf
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
import pickle
from keras.optimizers import Adam


def keras_model():
    model = Sequential()
```

```
model.add(Lambda(lambda x: x / 127.5 - 1., input_shape=(40, 40, 1)))

model.add(Conv2D(32, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), padding='valid'))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), padding='valid'))

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), padding='valid'))

model.add(Flatten())
model.add(Dropout(0.5))

model.add(Dense(128))

model.add(Dense(64))
model.add(Dense(1))

model.compile(optimizer=Adam(lr=0.0001), loss="mse")
filepath = "Autopilot.h5"
```

```
    checkpoint1 = ModelCheckpoint(filepath, verbose=1,
save_best_only=True)
    callbacks_list = [checkpoint1]

    return model, callbacks_list



def loadFromPickle():
    with open("features_40", "rb") as f:
        features = np.array(pickle.load(f))
    with open("labels", "rb") as f:
        labels = np.array(pickle.load(f))

    return features, labels



def augmentData(features, labels):
    features = np.append(features, features[:, :, ::-1], axis=0)
    labels = np.append(labels, -labels, axis=0)
    return features, labels



def main():
    features, labels = loadFromPickle()
    features, labels = augmentData(features, labels)
```

```
    features, labels = shuffle(features, labels)
    train_x, test_x, train_y, test_y = train_test_split(features, labels,
random_state=0,
                                                        test_size=0.1)
    train_x = train_x.reshape(train_x.shape[0], 40, 40, 1)
    test_x = test_x.reshape(test_x.shape[0], 40, 40, 1)
    model, callbacks_list = keras_model()
    model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=5,
batch_size=64,
              callbacks=callbacks_list)
    print_summary(model)
    model.save('Autopilot.h5')


main()
```

6.1.3 DriveApp.py

```
import numpy as np
import cv2
from keras.models import load_model

model = load_model('models/Autopilot.h5')

def keras_predict(model, image):
    processed = keras_process_image(image)
```

```
    steering_angle = float(model.predict(processed, batch_size=1))
    steering_angle = steering_angle * 100
    return steering_angle


def keras_process_image(img):
    image_x = 40
    image_y = 40
    img = cv2.resize(img, (image_x, image_y))
    img = np.array(img, dtype=np.float32)
    img = np.reshape(img, (-1, image_x, image_y, 1))
    return img


steer = cv2.imread('resources/steering_wheel_image.jpg', 0)
rows, cols = steer.shape
smoothed_angle = 0

cap = cv2.VideoCapture('resources/run.mp4')
while (cap.isOpened()):
    ret, frame = cap.read()
    gray = cv2.resize((cv2.cvtColor(frame, cv2.COLOR_RGB2HSV))[:, :, 1], (40,
40))
    steering_angle = keras_predict(model, gray)
    print(steering_angle)
```

```
    cv2.imshow('frame', cv2.resize(frame, (500, 300),
interpolation=cv2.INTER_AREA))
    smoothed_angle += 0.2 * pow(abs((steering_angle - smoothed_angle)),
2.0 / 3.0) * (
        steering_angle - smoothed_angle) / abs(
        steering_angle - smoothed_angle)
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), -smoothed_angle, 1)
    dst = cv2.warpAffine(steer, M, (cols, rows))
    cv2.imshow("steering wheel", dst)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

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

6.2 Detection and Classification of traffic lights:

```
!pip install tensorflow-object-detection-api


import numpy as np
import matplotlib.pyplot as plt
import os
import tensorflow as tf
from glob import glob
import sys
import sklearn.metrics as metrics
import torch


import torchvision
import cv2
import numpy as np
import pandas as pd
import PIL
import io
import html
```

```
import time
import matplotlib.pyplot as plt
from object_detection.utils import visualization_utils as viz_utils


# to display the images.
%matplotlib inline
```

Detection

```
MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'


# Path to frozen detection graph. Model that is used for the object detection.
model_path = "./"
PATH_TO_CKPT = model_path + MODEL_NAME + '/frozen_inference_graph.pb'



def download_model():
    import six.moves.urllib as urllib
    import tarfile


    opener = urllib.request.URLopener()
```

```
    opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
    tar_file = tarfile.open(MODEL_FILE)
    for file in tar_file.getmembers():
        file_name = os.path.basename(file.name)
        if 'frozen_inference_graph.pb' in file_name:
            tar_file.extract(file, os.getcwd())
def load_graph():
    if not os.path.exists(PATH_TO_CKPT):
        download_model()

    detection_graph = tf.Graph()
    with detection_graph.as_default():
        od_graph_def = tf.GraphDef()
        with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')

    return detection_graph

def select_boxes(boxes, classes, scores, score_threshold=0,
target_class=10):
    """

    :param boxes:
```

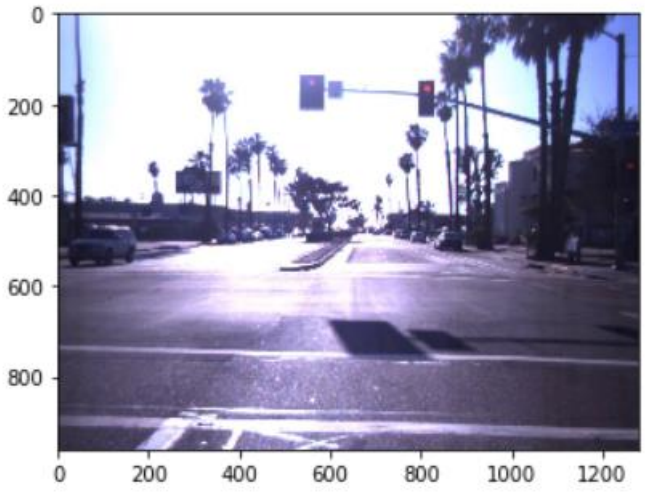
```
    :param classes:
    :param scores:
    :param target_class: default traffic light id in COCO dataset is 10
    :return:
    """

    sq_scores = np.squeeze(scores)
    sq_classes = np.squeeze(classes)
    sq_boxes = np.squeeze(boxes)

    sel_id = np.logical_and(sq_classes == target_class, sq_scores >
score_threshold)

    return sq_boxes[sel_id]

class TLClassifier(object):
    def __init__(self):

        self.detection_graph = load_graph()
        self.extract_graph_components()
        self.sess = tf.Session(graph=self.detection_graph)

        # run the first session to "warm up"
        dummy_image = np.zeros((100, 100, 3))
        self.detect_multi_object(dummy_image,0.1)
```

```
        self.traffic_light_box = None
        self.classified_index = 0


    def extract_graph_components(self):
        # Definite input and output Tensors for detection_graph
        self.image_tensor =
self.detection_graph.get_tensor_by_name('image_tensor:0')
        # Each box represents a part of the image where a particular object
was detected.
        self.detection_boxes =
self.detection_graph.get_tensor_by_name('detection_boxes:0')
        # Each score represent how level of confidence for each of the objects.
        # Score is shown on the result image, together with the class label.
        self.detection_scores =
self.detection_graph.get_tensor_by_name('detection_scores:0')
        self.detection_classes =
self.detection_graph.get_tensor_by_name('detection_classes:0')
        self.num_detections =
self.detection_graph.get_tensor_by_name('num_detections:0')
    def detect_multi_object(self, image_np, score_threshold):
        """
        Return detection boxes in a image


        :param image_np:
        :param score_threshold:
        :return:
        """
```

```
        # Expand dimensions since the model expects images to have shape:
[1, None, None, 3]
        image_np_expanded = np.expand_dims(image_np, axis=0)
        # Actual detection.


        (boxes, scores, classes, num) = self.sess.run(
            [self.detection_boxes, self.detection_scores, self.detection_classes,
self.num_detections],
            feed_dict={self.image_tensor: image_np_expanded})


        sel_boxes = select_boxes(boxes=boxes, classes=classes, scores=scores,
                                 score_threshold=score_threshold, target_class=10)


        return sel_boxes
tlc=TLClassifier()
def crop_roi_image(image_np, sel_box):
    im_height, im_width, _ = image_np.shape
    (left, right, top, bottom) = (sel_box[1] * im_width, sel_box[3] * im_width,
                                  sel_box[0] * im_height, sel_box[2] * im_height)
    cropped_image = image_np[int(top):int(bottom), int(left):int(right), :]
    return cropped_image
category_index ={'name':'dummyname','id':1}
%time
test_file = "../input/lisa-traffic-light-
dataset/daySequence1/daySequence1/frames/daySequence1--02001.jpg"
```

```
# test_file = "/kaggle/input/lisa-traffic-light-dataset/sample-dayClip6/sample-dayClip6/frames/dayClip6--00332.jpg"

from PIL import Image

im = Image.open(test_file)

image_np = np.asarray(im)

plt.imshow(image_np)
```

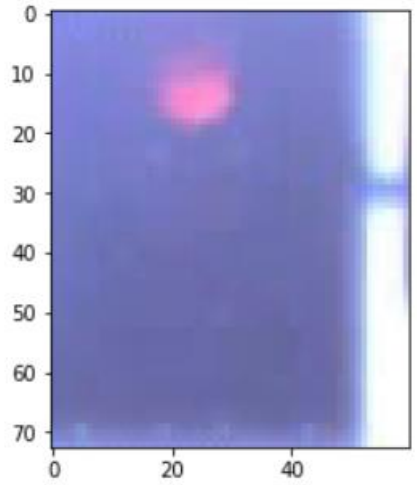
```
CPU times: user 2 µs, sys: 1e+03 ns, total: 3 µs
Wall time: 6.68 µs
<matplotlib.image.AxesImage at 0x7f0f64944d68>
```




```
%time

boxes=tlc.detect_multi_object(image_np,score_threshold=0.2)

if len(boxes)>0:

    cropped_image=crop_roi_image(image_np,boxes[0])

#     imgb=getWithBbox(image_np,boxes[0])

#     plt.imshow(imgb)

    plt.imshow(cropped_image)

    im = Image.fromarray(cropped_image)

    im.save('./temp.jpg')

else:
```

print('No traffic light was detected or light is off')

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.68 µs
```



CLASSIFICATION

```
from pathlib import Path

import tensorflow.keras

from tensorflow.keras.models import Sequential, Model, load_model

from tensorflow.keras.applications.vgg16 import VGG16,preprocess_input

from tensorflow.keras.preprocessing.image import
ImageDataGenerator,load_img, img_to_array

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense,
Dropout, Input, Flatten, Activation

from tensorflow.keras.optimizers import Adam, SGD, RMSprop

from tensorflow.keras.callbacks import Callback, EarlyStopping

from tensorflow.keras.utils import to_categorical

from sklearn.metrics import confusion_matrix

from tensorflow.keras import backend as K
```

```
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten

from tensorflow.keras.applications.inception_v3 import InceptionV3

from tensorflow.keras.applications.inception_v3 import preprocess_input

from tensorflow.keras.preprocessing import image

IMAGE_SIZE = [224, 224]

inception = InceptionV3(input_shape=IMAGE_SIZE + [3],
weights='imagenet', include_top=False)

for layer in inception.layers:

   layer.trainable = False

x = Flatten()(inception.output)

train_path = '../input/cropped-lisa-traffic-light-
dataset/cropped_lisa_1/train_1'

valid_path = '../input/cropped-lisa-traffic-light-
dataset/cropped_lisa_1/val_1'

folders = glob(train_path+'/*')

prediction = Dense(len(folders), activation='softmax')(x)

model = Model(inputs=inception.input, outputs=prediction)

model.compile(

 loss='categorical_crossentropy',

 optimizer='adam',

 metrics=['accuracy']

)

from tensorflow.keras.preprocessing.image import ImageDataGenerator


train_datagen = ImageDataGenerator(rescale = 1./255,

                    shear_range = 0.2,
```

```
                    zoom_range = 0.2,
                    horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
training_set = train_datagen.flow_from_directory(train_path,
                                target_size = (224, 224),
                                batch_size = 32,
                                class_mode = 'categorical',
                                shuffle = False)
val_set = test_datagen.flow_from_directory(valid_path,
                                target_size = (224, 224),
                                batch_size = 32,
                                class_mode = 'categorical',
                                shuffle = False)
print(training_set.class_indices)
r = model.fit_generator(
  training_set,
  validation_data=val_set,
  epochs=10,
  steps_per_epoch=len(training_set),
  validation_steps=len(val_set)
)
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
```

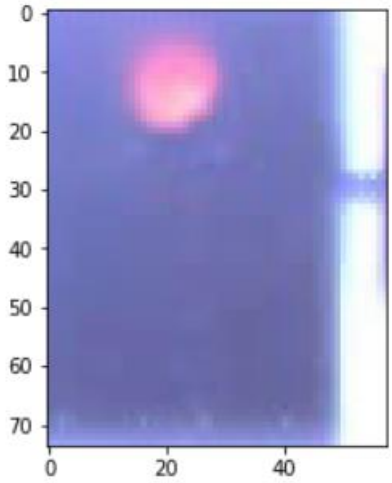
```
plt.show()
plt.savefig('LossVal_loss')


# plot the accuracy
plt.plot(r.history['acc'], label='train acc')
plt.plot(r.history['val_acc'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix
import numpy as np
batch_size = 32
def true_and_predicted_labels(dataset):
   labels = np.zeros((dataset.n,7))
   preds = np.zeros_like(labels)
   for i in range(len(dataset)):
      sys.stdout.write('evaluating batch {}\r'.format(i))
      sys.stdout.flush()
      batch = dataset[i]
      batch_images = batch[0]
      batch_labels = batch[1]
      batch_preds = model.predict(batch_images)
      start = i*batch_size
      labels[start:start+batch_size] = batch_labels
```

```
        preds[start:start+batch_size] = batch_preds
    return labels, preds

val_labels, val_preds = true_and_predicted_labels(val_set)
a = val_preds
idx = np.argmax(a, axis=-1)
a = np.zeros( a.shape )
a[ np.arange(a.shape[0]), idx] = 1
print(classification_report(val_labels, a))
print(confusion_matrix(val_labels.argmax(axis=1), a.argmax(axis=1)))
test_img = './temp.jpg'

image = tf.keras.preprocessing.image.load_img(test_img)
im1 = image.resize((224, 224), Image.ANTIALIAS)
input_arr = tf.keras.preprocessing.image.img_to_array(im1)
input_arr = np.array([input_arr])  # Convert single image to a batch.
predictions = model.predict(input_arr)
pos_pred = predictions.argmax()
key_list = list(all_class_names.keys())
val_list = list(all_class_names.values())
position = val_list.index(pos_pred)
print(key_list[position])
plt.imshow(cropped_image)
```

```
stop
```

:

```
<matplotlib.image.AxesImage at 0x7fa51e1a14e0>
```



6.3 Vehicles and Pedestrian detection:

```
%%bash

pip install -qr
https://raw.githubusercontent.com/ultralytics/yolov5/master/requirements.txt
# install dependencies

import torch


# Model

model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)


# Images

imgs = ['https://images.unsplash.com/photo-1546052594-
bb4efa212d4f?ixlib=rb-
1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1yZWxhdGVkfDE1fHx8ZW5
8MHx8fHw%3D&w=1000&q=80']  # batch of images


# Inference

results = model(imgs)


# Results

results.print()

results.save()

from IPython.display import Image, clear_output
```

Image('/content/runs/detect/exp3/photo-1546052594-bb4efa212d4f?ixlib=rb-1.2.jpg')

CHAPTER 7

MODEL METRICS

7.1 METRIC

We are considering precision ,recall and F1-Score for the object detection model and accuracy for the CNN model .

We estimate what percentage of the time the network could drive the car (autonomy). The metric is determined by counting simulated human interventions (see Section 6). These interventions occur when the simulated vehicle departs from the center line by more than one meter. We assume that in real life an actual intervention would require a total of six seconds: this is the time required for a human to retake control of the vehicle, re-center it, and then restart the self-steering mode. We calculate the percentage autonomy by counting the number of interventions, multiplying by 6 seconds, dividing by the elapsed time of the simulated test, and then subtracting the result from 1: autonomy = (1 − (number of interventions) · 6 seconds elapsed time [seconds]) · 100

7.2 CNN model for driving

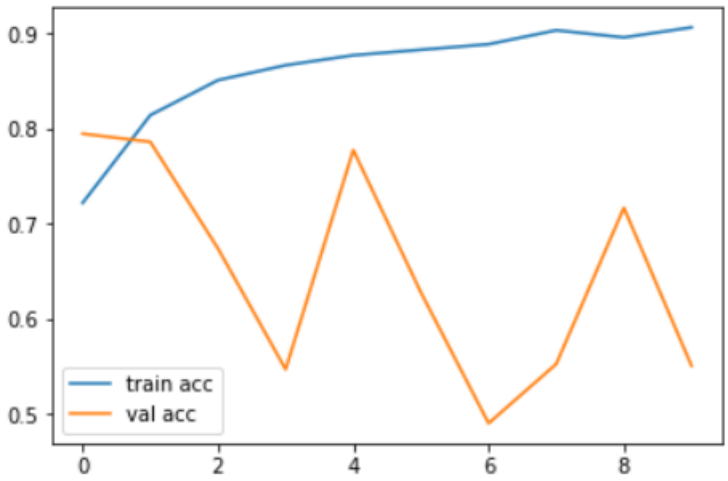
ACCURACY METRICS:

Mean Squared error: 0.2

Autonomy score: 90%

7.3 TensorFlow Model for Traffic light detection and classification

```
1025/1025 [==============================] - 379s 370ms/step - loss: 2.5609 - acc: 0.89
60 - val_loss: 10.6410 - val_acc: 0.7169
Epoch 10/10
```



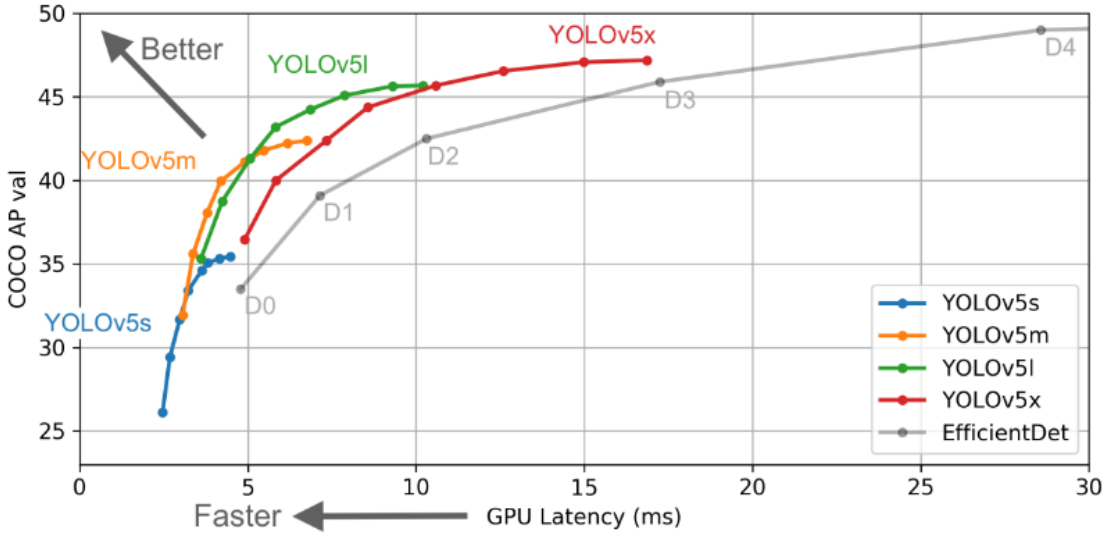
7.4 YOLO5

Precision: 95.55%

Recall: 95.84%,

F1-Score: 96.54%

mAP: 94.63%.



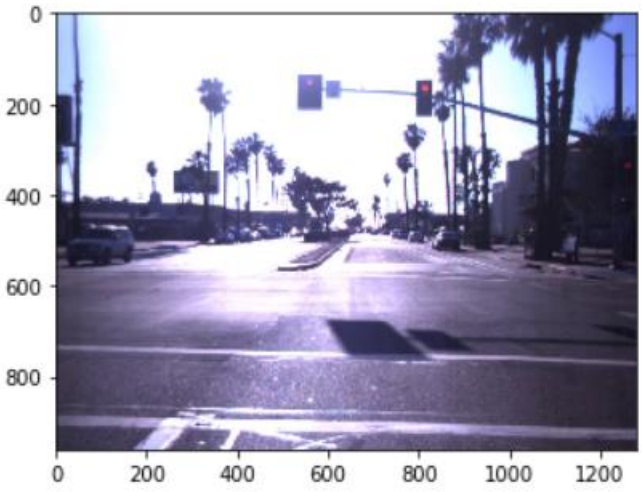
CHAPTER 8

TESTING

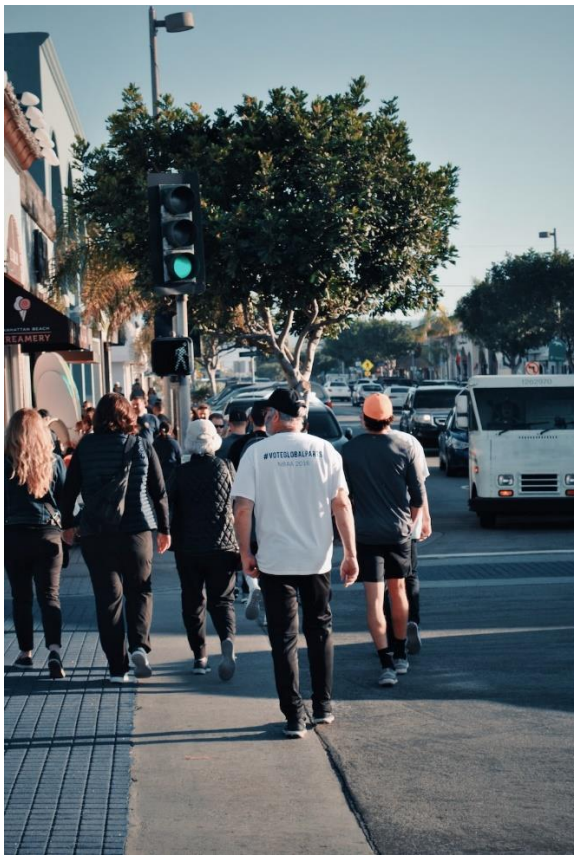
8.1 INPUT:

8.1.1: Driving simulator

8.1.2: Traffic light classification



8.1.3: Vehicles and pedestrian

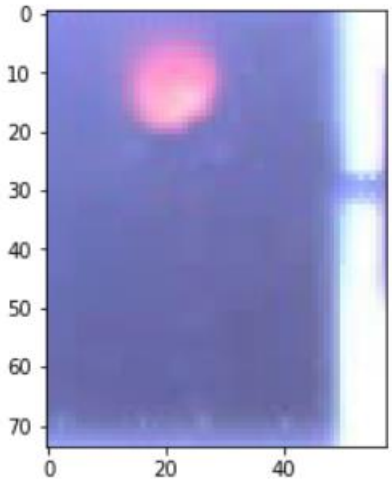


8.2 OUTPUT:

8.2.1 Self Driving

8.2.2 Traffic light classification

```
stop

<matplotlib.image.AxesI
```



8.2.3 Vehicles and pedestrian detection

CHAPTER 9

CONCLUSION

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.

The traffic light classification and road sign detection as we as pedestrian/vehicle detection and avoidance systems can be incorporated with the CNN model to build a robust Self Driving Car System.

CHAPTER 10

REFERENCES/ BIBILIOGRAPHY

REFERENCES:

1. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4):541–551, Winter 1989. URL: http://yann.lecun.org/exdb/publis/pdf/lecun-89e.pdf.
2. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012. URL: http://papers.nips.cc/paper/ 4824-imagenet-classification-with-deep-convolutional-neural-networks. pdf.
3. L. D. Jackel, D. Sharman, Stenard C. E., Strom B. I., , and D Zuckert. Optical character recognition for self-service banking. AT&T Technical Journal, 74(1):16–24, 1995.
4. Large scale visual recognition challenge (ILSVRC). URL: http://www.image-net.org/ challenges/LSVRC/.
5. Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004. Final technical report. URL: http://net-scale.com/doc/net-scale-dave-report.pdf.
6. Dean A. Pomerleau. ALVINN, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, 1989. URL: http://repository.cmu.edu/cgi/viewcontent. cgi?article=2874&context=compsci.
7. Wikipedia.org. DARPA LAGR program. http://en.wikipedia.org/wiki/DARPA_LAGR_ Program.
8. Danwei Wang and Feng Qi. Trajectory planning for a four-wheel-steering vehicle. In Proceedings of the 2001 IEEE International Conference on Robotics & Automation, May 21–26 2001. URL: http: //www.ntu.edu.sg/home/edwwang/confpapers/wdwicar01.pdf.
9. DAVE 2 driving a lincoln. URL: https://drive.google.com/open?id= 0B9raQzOpizn1TkRIa241ZnBEcjQ
10. Deep Reinforcement Learning (deepmind.com)