Traffic Sign Detection in Autonomous Vehicles

Project Exhibition Report

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology

In

Electronics and Communication (Specialization in AI & Cybernetics)

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

Submitted to

VIT BHOPAL UNIVERSITY (M. P)

Submitted by

JENISH MURDIA – 20BAC10004
SHAUN JACOB VARGHESE – 20BAC10022
VARUN RAM S – 20BAC10038

Under the Supervision of

Dr. ANIRBAN BHOWMICK

SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING
VIT BHOPAL UNIVERSITY

BHOPAL (M.P.)-466114

December - 2021

VIT BHOPAL UNIVERSITY, BHOPAL

SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING

DECLARATION

We hereby declare that the Dissertation entitled "*Traffic Sign Detection in Autonomous Vehicles*" is our own work conducted under the supervision of *Dr. Anirban Bhowmik, Assistant Professor, School of Electrical and Electronics Engineering* at VIT Bhopal University, Bhopal.

We further declare that to the best of our knowledge this report does not contain any part of work that has been submitted for the award of any degree either in this university or in other university / Deemed University without proper citation.

Varun Ram S (20BAC10038)

Shaun Jacob (20BAC10022)

Jenish Murdia (20BAC10004)

VIT BHOPAL UNIVERSITY, BHOPAL

SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING

CERTIFICATE

This is to certify that the work embodied in this project report entitled **“Traffic Sign Detection in Autonomous Vehicles”** has been satisfactorily completed by **Mr. Jenish Murdia (20BAC10004), Mr. Shaun Jacob Varghese (20BAC10022), & Mr. Varun Ram S (20BAC10038)** in the School of Electrical & Electronics Engineering at VIT Bhopal University, Bhopal. This work is a bonafide piece of work, carried out under my/our guidance in the School of Electrical and Electronics Engineering for the partial fulfilment of the degree of Bachelor of Technology.

Dr. Susant Kumar Panigrahi
Assistant Professor

Dr. Anirban Bhowmick
Assistant Professor

Forwarded by

Dr. Soumitra K Nayak
Program Chair

Approved by

Dr. Debashis Adhikari
Professor & Dean

Acknowledgement

In the first place we would like to record our gratitude to “**Dr. Anirban Bhowmick”** for his supervision, advice, and guidance from the very early stage of this thesis work as well as giving us extraordinary experiences throughout the work. Above all and the most needed, he provided us unflinching encouragement and support in various ways. His words have always inspired us to work in an efficient and comprehensive way. We would like to thank him for his constant encouragement that enabled us to grow as persons. His presence has definitely improved us as better human beings.

We thank our Co-Guide **“Dr. Susant Kumar Panigrahi**” who was always available to motivate and give suggestions and opinions during the periodic reviews of the Project that was conducted in the University.

We express our gratitude towards “**Dr. Debashis Adhikari”,** Professor and Dean of School of Electrical and Electronics Engineering Department, VIT Bhopal University, Bhopal, for providing us all the help and permitted us to work with complete enthusiasm,

We shall ever remain indebted to “**Dr. U Kamachi Mudali”,** Vice Chancellor, VIT Bhopal University, for providing us institutional and administrative facilities during our project work at VIT.

Varun Ram S

Shaun Jacob

Jenish Murdia

Table of Contents

Abstract

"As reported in the findings of an eight-year study conducted by the National Highway Traffic Safety Administration (NHTSA), there were on average 1,578 fatalities each year resulting from two-vehicle traffic crashes at intersections controlled by traffic signals. Approximately 51% of those fatal crashes were caused by drivers who ran red lights. Approximately 29% were caused by drivers who failed to yield the right-of-way at traffic signals." These lines taken from the NHTSA study shows the grim number of deaths that occur each year due to the misinterpretation of traffic signs ultimately caused due to human error. Although not completely, we believe that the deployment of machine learning-based models in cars to assist drives will help reduce the number of accidents occurring due to human error substantially.In this project, we present a unique thin yet deep convolutional neural network architecture for traffic sign identification that is both energy-efficient and robust. Each convolutional layer in the proposed design has fewer than 50 features, allowing us to train our convolutional neural network without necessarily using a graphics processing unit.

List of Figures

List of Tables

List of Symbols & Abbreviations

ADAS	**Advanced Driver Assistance System**
AI	**Artificial Intelligence**
CNN	**Convolutional Neural Network**
PC	**Personal Computer**
IDE	**Integrated Development Environment**
RGB	**Red-Green-Blue**

X

CHAPTER 1: INTROUDCTION

1.1 MOTIVATION

AI holds the power to make or break the autonomous vehicle industry. It is pivotal for Machine learning models to exist that are capable of doing tasks such as speed control, changing lanes, object detection etc with high accuracy and efficiency.

As students of the school of Electrical and Electronics engineering and Artificial intelligence with Cybernetics, we wanted to pursue a project which would allow us to exhibit and work on our knowledge on both these fields. We believe our project idea combines these two and can showcase a good example of an AI based control system.

1.2 PROJECT PREAMBLE

This interesting Project mainly focuses on how we minimize accidents in Autonomous Vehicles. We intend to work with a number of libraries in Python such as Keras and Tensorflow in order to design a model that will be able to detect traffic signs on roads and give the appropriate feedback to the user. In real life, such model can be added to Self-Driving cars which is getting popular day by day. This Project also intends to reduce the number of accidents on the roads, mostly focussing on the ones caused by Human error due to misinterpretation of traffic signs.

1.3 OBJECTIVE

- To create a Machine learning model that should clearly be able to detect a traffic sign via the camera
- Then After Detecting the traffic sign, our trained Convolutional neural network model should be able to recognize the traffic sign and display its meaning next to it

1.4 COMPONENTS REQUIRED

- A PC, preferably a laptop with Windows 7 or higher
- An IDE such as PyCharm or Jupyter Notebook
- A Hard copy of the Images of Traffic Signs

CHAPTER 2: LITERATURE SURVEY

- **Paper name** - *DeepThin: A novel lightweight CNN architecture for traffic sign recognition without GPU requirements*
  Authors name - Wasif Arman Haquea, Samin Arefinb, A.S.M. Shihavuddinc Muhammad Abul Hasan.
  Publisher: ScienceDirect
  Date of publishing - 9 December 2020
  Summary: The authors suggest a new energy-efficient Thin but Deep convolutional neural network architecture for traffic sign identification in this study. Two publicly accessible traffic sign datasets, the German Traffic Sign Recognition Benchmark and the Belgian Traffic Sign Classification dataset, are used to evaluate the proposed architecture's performance.

  They first use the enormous German Traffic Sign Recognition Benchmark dataset to train and assess the proposed architecture's performance. They next use transfer learning to retrain the network models on the more difficult Belgian Traffic Sign Classification dataset to evaluate test performance. With at least five times fewer parameters in the individual end-to-end network for training, the suggested architecture outperforms state-of-the-art traffic sign approaches.

- **Paper name** - *On circular traffic sign detection and recognition*
  Authors name - Selcan Kaplan Berkaya, Huseyin Gunduz, Ozgur Özşen, Cuneyt Akinlar, Serkan Gunal.
  Publisher: ScienceDirect
  Date of publishing - 2 December 2015
  Summary: A novel approach to traffic sign identification is suggested, which makes use of a recently developed circle detection algorithm and an RGB-based colour thresholding technique. Within a support vector machine classification framework, an ensemble of features such as histogram of directed gradients, local binary patterns, and Gabor features are used to classify traffic signs. The suggested detection and recognition algorithms are tested using datasets from the German Traffic Sign Detection and Recognition Benchmarks, respectively. The experimental findings show that both techniques provide equivalent or even greater performance than the best ones reported in the literature, and that they are also compatible with real-time operation.

CHAPTER 3: PROBLEM FORMULATION AND PROPOSED METHODOLOGY

3.1 PROBLEM FORMULATION

- **Problem Statement**
  - **Problem Definition:** Agent is in motion; agent arrives at a path with attached traffic sign. Agent will only be lawful if it abides the traffic sign and takes necessary action.
  - **Problem Limitation:** agent must be able to interpret and differentiate between various traffic signs and also increase or decrease its speed according to instruction displayed.

- **Problem Solution:** Detect and interpret traffic sign, and take necessary action to stay lawful

- **Solution Space:** There are multiple possibilities of traffic sign to choose from.

3.2 METHODOLOGY

Our approach to this project was with a simple methodology that would allow us to arrive at the best possible iteration of our work. Our main aim was to take data sets from different online sources and analyse them to extract useful parameters for our project. We would then train our machine learning model on data taken from these data sets and other sources and then test and finally we would try to validate our machine learning model. Here as we can see in the flowchart below (Fig 3.2), we would reach a loop statement where we would try to train a model and if the result was not desirable as expected, it meant that the accuracy was not high enough, hence we would go back to the previous step and re-evaluate our model and look for ways to train it better so that we may get a better accuracy score. Only if we passed the basic criteria of higher accuracy, we would finalize the model and go ahead with that iteration of the project.

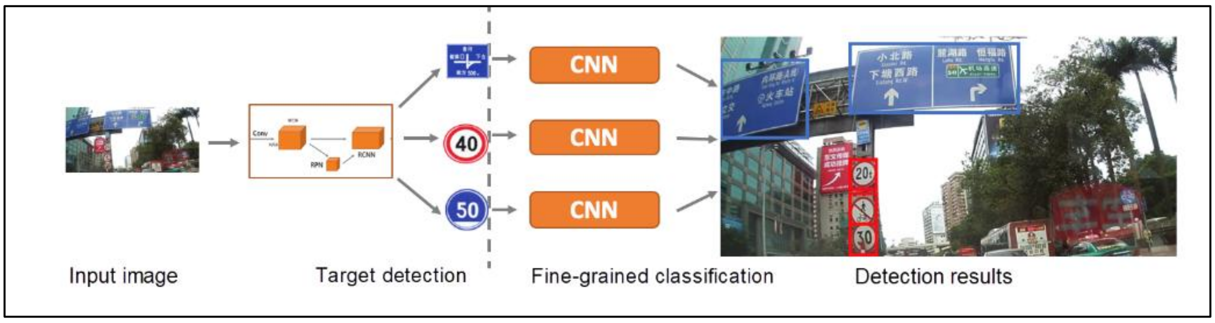


Fig 3.1 Depiction of how the Model intends to work

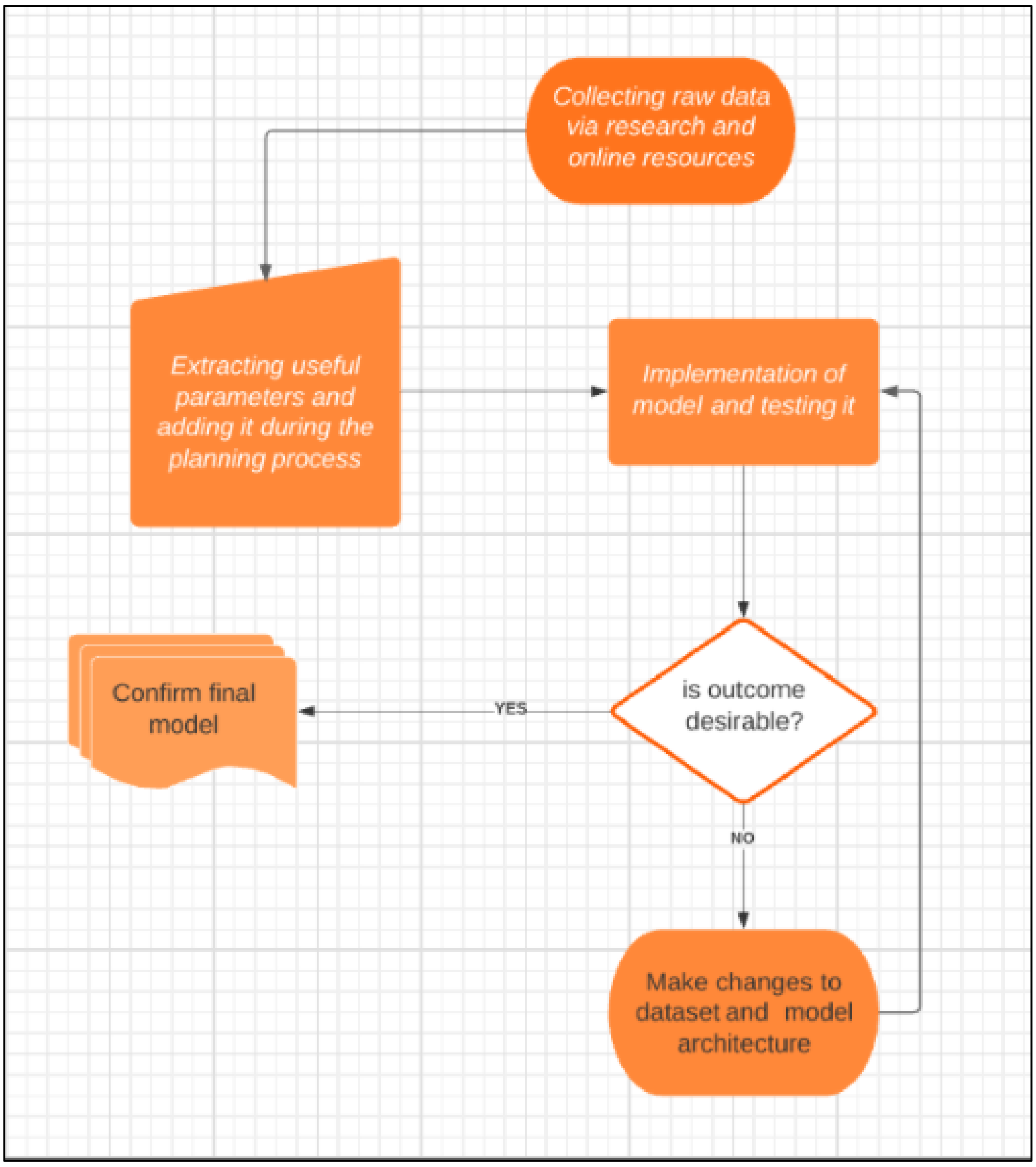

Collecting raw data via research and online resources
Extracting useful parameters and adding it during the planning process
Implementation of model and testing it
is outcome desirable?
YES
Confirm final model
NO
Make changes to dataset and model architecture

Fig 3.2. Methodology Flowchart

CHAPTER 4: TECHNICAL IMPLEMENTATION

4.1 PYTHON CODE FOR ML MODEL

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import cv2
from sklearn.model_selection import train_test_split
import os
import pandas as pd
import numpy as np
import random
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
from numpy import loadtxt



print("finished")

################# Parameters #####################

path = "myData"  # folder with all the class folders
labelFile = 'labels.csv'  # file with all names of classes
batch_size_val = 50

epochs_val = 30
imageDimesions = (32, 32, 3)
testRatio = 0.2
validationRatio = 0.2

###################################################

############################### Importing of the Images
count = 0
images = []
classNo = []
myList = os.listdir(path)
print("Total Classes Detected:",len(myList))
noOfClasses=len(myList)
print("Importing Classes.....")
for x in range (0,len(myList)):
    myPicList = os.listdir(path+"/"+str(count))
    for y in myPicList:
        curImg = cv2.imread(path+"/"+str(count)+"/"+y)
        images.append(curImg)
        classNo.append(count)
    print(count, end =" ")
    count +=1
print(" ")
images = np.array(images)
classNo = np.array(classNo)

################################ Split Data
X_train, X_test, y_train, y_test = train_test_split(images, classNo, test_size=testRatio)
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train,
test_size=validationRatio)
steps_per_epoch_val = len(X_train) // batch_size_val
# X_train = ARRAY OF IMAGES TO TRAIN
# y_train = CORRESPONDING CLASS ID

################################ TO CHECK IF NUMBER OF IMAGES MATCHES
TO NUMBER OF LABELS FOR EACH DATA SET
print("Data Shapes")
print("Train", end="");
print(X_train.shape, y_train.shape)
print("Validation", end="");
print(X_validation.shape, y_validation.shape)
print("Test", end="");
print(X_test.shape, y_test.shape)
assert (X_train.shape[0] == y_train.shape[
    0]), "The number of images in not equal to the number of lables in training set"
assert (X_validation.shape[0] == y_validation.shape[
    0]), "The number of images in not equal to the number of lables in validation set"
assert (X_test.shape[0] == y_test.shape[0]), "The number of images in not equal to the
number of lables in test set"
assert (X_train.shape[1:] == (imageDimesions)), " The dimesions of the Training images are
wrong "
assert (X_validation.shape[1:] == (imageDimesions)), " The dimesionas of the Validation
images are wrong "
assert (X_test.shape[1:] == (imageDimesions)), " The dimesionas of the Test images are
wrong"

################################ READ CSV FILE
data = pd.read_csv(labelFile)
print("data shape ", data.shape, type(data))

################################ DISPLAY SOME SAMPLES IMAGES  OF ALL
THE CLASSES
num_of_samples = []
cols = 5
num_classes = noOfClasses
fig, axs = plt.subplots(nrows=num_classes, ncols=cols, figsize=(5, 300))
```

```
fig.tight_layout()
for i in range(cols):
    for j, row in data.iterrows():
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(0, len(x_selected) - 1), :, :],
cmap=plt.get_cmap("gray"))
        axs[j][i].axis("off")
        if i == 2:
            axs[j][i].set_title(str(j) + "-" + row["Name"])
            num_of_samples.append(len(x_selected))

################################ DISPLAY A BAR CHART SHOWING NO OF
SAMPLES FOR EACH CATEGORY
print(num_of_samples)
plt.figure(figsize=(12, 4))
plt.bar(range(0, num_classes), num_of_samples)
plt.title("Distribution of the training dataset")
plt.xlabel("Class number")
plt.ylabel("Number of images")
plt.show()


################################ PREPROCESSING THE IMAGES

def grayscale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img


def equalize(img):
    img = cv2.equalizeHist(img)
    return img


def preprocessing(img):
    img = grayscale(img)  # CONVERT TO GRAYSCALE
    img = equalize(img)  # STANDARDIZE THE LIGHTING IN AN IMAGE
    img = img / 255  # TO NORMALIZE VALUES BETWEEN 0 AND 1 INSTEAD OF 0
TO 255
    return img


X_train = np.array(list(map(preprocessing, X_train)))  # TO IRETATE AND PREPROCESS
ALL IMAGES
X_validation = np.array(list(map(preprocessing, X_validation)))
X_test = np.array(list(map(preprocessing, X_test)))
cv2.imshow("GrayScale Images",X_train[random.randint(0, len(X_train) - 1)])  # TO
CHECK IF THE TRAINING IS DONE PROPERLY

################################ ADD A DEPTH OF 1
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
```

```
X_validation = X_validation.reshape(X_validation.shape[0], X_validation.shape[1],
X_validation.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)

################################ AUGMENTATAION OF IMAGES TO MAKE IT
MORE GENERIC
dataGen = ImageDataGenerator(width_shift_range=0.1,
                             # 0.1 = 10%     IF MORE THAN 1 E.G 10 THEN IT REFFERS TO NO.
OF  PIXELS EG 10 PIXELS
                             height_shift_range=0.1,
                             zoom_range=0.2,  # 0.2 MEANS CAN GO FROM 0.8 TO 1.2
                             shear_range=0.1,  # MAGNITUDE OF SHEAR ANGLE
                             rotation_range=10)  # DEGREES
dataGen.fit(X_train)
batches = dataGen.flow(X_train, y_train,
                       batch_size=20)  # REQUESTING DATA GENRATOR TO GENERATE
IMAGES  BATCH SIZE = NO. OF IMAGES CREAED EACH TIME ITS CALLED
X_batch, y_batch = next(batches)

# TO SHOW AGMENTED IMAGE SAMPLES
fig, axs = plt.subplots(1, 15, figsize=(20, 5))
fig.tight_layout()

for i in range(15):
    axs[i].imshow(X_batch[i].reshape(imageDimesions[0], imageDimesions[1]))
    axs[i].axis('off')
plt.show()

y_train = to_categorical(y_train, noOfClasses)
y_validation = to_categorical(y_validation, noOfClasses)
y_test = to_categorical(y_test, noOfClasses)


################################ CONVOLUTION NEURAL NETWORK MODEL
def myModel():
    no_Of_Filters = 60
    size_of_Filter = (5, 5)  # THIS IS THE KERNEL THAT MOVE AROUND THE IMAGE
TO GET THE FEATURES.
    # THIS WOULD REMOVE 2 PIXELS FROM EACH BORDER WHEN USING 32 32
IMAGE
    size_of_Filter2 = (3, 3)
    size_of_pool = (2, 2)  # SCALE DOWN ALL FEATURE MAP TO GERNALIZE MORE,
TO REDUCE OVERFITTING
    no_Of_Nodes = 500  # NO. OF NODES IN HIDDEN LAYERS
    model = Sequential()
    model.add((Conv2D(no_Of_Filters, size_of_Filter, input_shape=(imageDimesions[0],
imageDimesions[1], 1),
                      activation='relu')))  # ADDING MORE CONVOLUTION LAYERS = LESS
FEATURES BUT CAN CAUSE ACCURACY TO INCREASE
    model.add((Conv2D(no_Of_Filters, size_of_Filter, activation='relu')))
    model.add(MaxPooling2D(pool_size=size_of_pool))  # DOES NOT EFFECT THE
DEPTH/NO OF FILTERS

    model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
    model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
    model.add(MaxPooling2D(pool_size=size_of_pool))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(no_Of_Nodes, activation='relu'))
    model.add(Dropout(0.5))  # INPUTS NODES TO DROP WITH EACH UPDATE 1 ALL
0 NONE
    model.add(Dense(noOfClasses, activation='softmax'))  # OUTPUT LAYER

    # COMPILE MODEL

    model.compile(Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
    return model

#training
model = myModel()
model.build(input_shape=(22271, 32, 32, 1))
model.summary()

history = model.fit(dataGen.flow(X_train, y_train, batch_size=batch_size_val),
steps_per_epoch=steps_per_epoch_val,
                    epochs=epochs_val, validation_data=(X_validation, y_validation), shuffle=1)

################################## PLOT
plt.figure(1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training','validation'])
plt.title('loss')
plt.xlabel('epoch')
plt.figure(2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title('Acurracy')
plt.xlabel('epoch')
plt.show()
score =model.evaluate(X_test,y_test,verbose=0)
print('Test Score:',score[0])
print('Test Accuracy:',score[1])


# STORE THE MODEL AS An OBJECT
model.save("modelfinal.h5")
```

4.2 PYTHON CODE FOR WEBCAM CONTROL & IMAGE PRE-PROCESSING

```
import numpy as np
import cv2
from tensorflow import keras

threshold = 0.75  # THRESHOLD
font = cv2.FONT_HERSHEY_SIMPLEX
model = keras.models.load_model('modelfinal.h5') #load model

#preprocsing
def preprocess_img(imgBGR, erode_dilate=True):  # pre-processing to detect signs
in  image.
    rows, cols, _ = imgBGR.shape
    imgHSV = cv2.cvtColor(imgBGR, cv2.COLOR_BGR2HSV)
    Bmin = np.array([100, 43, 46])
    Bmax = np.array([124, 255, 255])
    img_Bbin = cv2.inRange(imgHSV, Bmin, Bmax)

    Rmin1 = np.array([0, 43, 46])
    Rmax1 = np.array([10, 255, 255])
    img_Rbin1 = cv2.inRange(imgHSV, Rmin1, Rmax1)

    Rmin2 = np.array([156, 43, 46])
    Rmax2 = np.array([180, 255, 255])
    img_Rbin2 = cv2.inRange(imgHSV, Rmin2, Rmax2)
    img_Rbin = np.maximum(img_Rbin1, img_Rbin2)
    img_bin = np.maximum(img_Bbin, img_Rbin)

    if erode_dilate is True:
        kernelErosion = np.ones((3, 3), np.uint8)
        kernelDilation = np.ones((3, 3), np.uint8)
        img_bin = cv2.erode(img_bin, kernelErosion, iterations=2)
        img_bin = cv2.dilate(img_bin, kernelDilation, iterations=2)

    return img_bin

def contour_detect(img_bin, min_area, max_area=-1, wh_ratio=2.0) :
    rects = []
    contours, _ = cv2.findContours(img_bin.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
    if len(contours) == 0:
        return rects

    max_area = img_bin.shape[0] * img_bin.shape[1] if max_area < 0 else max_area
    for contour in contours:
        area = cv2.contourArea(contour)
        if area >= min_area and area <= max_area:
            x, y, w, h = cv2.boundingRect(contour)
            if 1.0 * w / h < wh_ratio and 1.0 * h / w < wh_ratio:
                rects.append([x, y, w, h])
    return rects

def grayscale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img
def equalize(img):
    img = cv2.equalizeHist(img)
    return img


def preprocessing(img):
    img = grayscale(img)
    img = equalize(img)
    img = img / 255
    return img

def getCalssName(classNo):
    if classNo == 0:
        return 'Speed Limit 20 km/h'
    elif classNo == 1:
        return 'Speed Limit 30 km/h'
    elif classNo == 2:
        return 'Speed Limit 50 km/h'
    elif classNo == 3:
        return 'Speed Limit 60 km/h'
    elif classNo == 4:
        return 'Speed Limit 70 km/h'
    elif classNo == 5:
        return 'Speed Limit 80 km/h'
    elif classNo == 6:
        return 'End of Speed Limit 80 km/h'
    elif classNo == 7:
        return 'Speed Limit 100 km/h'
    elif classNo == 8:
        return 'Speed Limit 120 km/h'
    elif classNo == 9:
        return 'No passing'
    elif classNo == 10:
        return 'No passing for vehicles over 3.5 metric tons'
    elif classNo == 11:
        return 'Right-of-way at the next intersection'
    elif classNo == 12:
        return 'Priority road'
    elif classNo == 13:
        return 'Yield'
    elif classNo == 14:
        return 'Stop'
    elif classNo == 15:
        return 'No vehicles'
    elif classNo == 16:
        return 'Vehicles over 3.5 metric tons prohibited'
    elif classNo == 17:
        return 'No entry'
    elif classNo == 18:
        return 'General caution'
    elif classNo == 19:
        return 'Dangerous curve to the left'
    elif classNo == 20:
        return 'Dangerous curve to the right'
    elif classNo == 21:
        return 'Double curve'
    elif classNo == 22:
        return 'Bumpy road'
    elif classNo == 23:
        return 'Slippery road'
    elif classNo == 24:
        return 'Road narrows on the right'
    elif classNo == 25:
        return 'Road work'
    elif classNo == 26:
        return 'Traffic signals'
    elif classNo == 27:
        return 'Pedestrians'
    elif classNo == 28:
        return 'Children crossing'
    elif classNo == 29:
        return 'Bicycles crossing'
    elif classNo == 30:
        return 'Beware of ice/snow'
    elif classNo == 31:
        return 'Wild animals crossing'
    elif classNo == 32:
        return 'End of all speed and passing limits'
    elif classNo == 33:
        return 'Turn right ahead'
    elif classNo == 34:
        return 'Turn left ahead'
    elif classNo == 35:
        return 'Ahead only'
    elif classNo == 36:
        return 'Go straight or right'
    elif classNo == 37:
        return 'Go straight or left'
    elif classNo == 38:
        return 'Keep right'
    elif classNo == 39:
        return 'Keep left'
    elif classNo == 40:
        return 'Roundabout mandatory'
    elif classNo == 41:
        return 'End of no passing'
    elif classNo == 42:
        return 'End of no passing by vehicles over 3.5 metric tons'

if __name__ == "__main__":
    cap = cv2.VideoCapture(0)
    cols = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    rows = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    while (1):
        ret, img = cap.read()
        img_bin = preprocess_img(img, False)
        cv2.imshow("bin image", img_bin)
        min_area = img_bin.shape[0] * img.shape[1] / (25 * 25)
        rects = contour_detect(img_bin, min_area=min_area)   # get x,y,h and w.
        img_bbx = img.copy()
        for rect in rects:
            xc = int(rect[0] + rect[2] / 2)
            yc = int(rect[1] + rect[3] / 2)

            size = max(rect[2], rect[3])
            x1 = max(0, int(xc - size / 2))
            y1 = max(0, int(yc - size / 2))
            x2 = min(cols, int(xc + size / 2))
            y2 = min(rows, int(yc + size / 2))

            # rect[2] is width and rect[3] for height
            if rect[2] > 100 and rect[3] > 100:            #only detect those signs whose height and
width >100
                cv2.rectangle(img_bbx, (rect[0], rect[1]), (rect[0] + rect[2], rect[1] + rect[3]), (0, 0,
255), 2)
            crop_img = np.asarray(img[y1:y2, x1:x2])
            crop_img = cv2.resize(crop_img, (32, 32))
            crop_img = preprocessing(crop_img)
            cv2.imshow("afterprocessing", crop_img)
            crop_img = crop_img.reshape(1, 32, 32, 1)       # (1,32,32) after reshape it become
(1,32,32,1)
            predictions = model.predict(crop_img)          # make predicion
            classIndex = np.argmax(predictions, axis=1)
            probabilityValue = np.amax(predictions)
            if probabilityValue > threshold:
                #write class name on the output screen
                cv2.putText(img_bbx, str(classIndex) + " " + str(getCalssName(classIndex)),
(rect[0], rect[1] - 10),
                            font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
                # write probability value on the output screen
                cv2.putText(img_bbx, str(round(probabilityValue * 100, 2)) + "%", (rect[0], rect[1]
- 40), font, 0.75,
                            (0, 0, 255), 2, cv2.LINE_AA)

        cv2.imshow("detect result", img_bbx)
        if cv2.waitKey(1) & 0xFF == ord('q'):          # q for quit
            break
cap.release()
cv2.destroyAllWindows()
```

CHAPTER 5: RESULTS AND DISCUSSION

Given below is the exact output we see once we run the two python files in our project file IDE respectively.

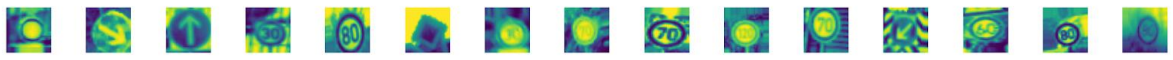
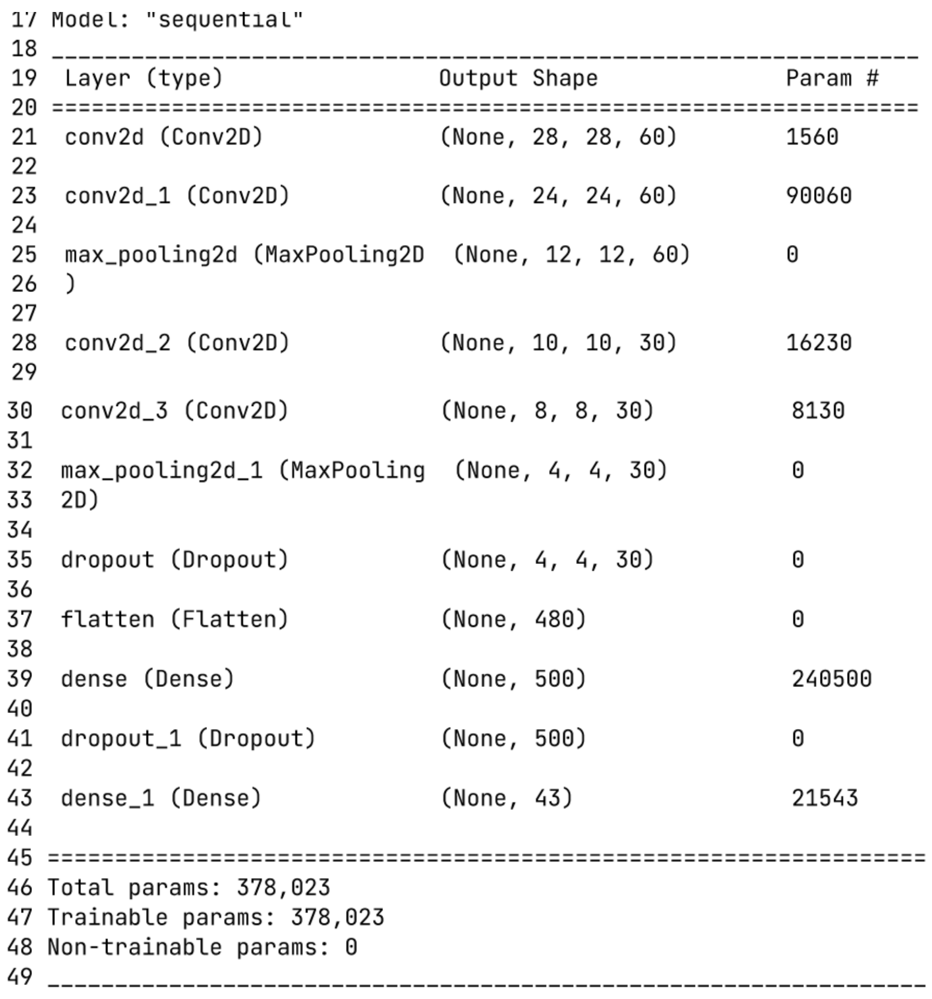


Fig 5.1. Pre-Processed Images


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 60)        1560

 conv2d_1 (Conv2D)           (None, 24, 24, 60)        90060

 max_pooling2d (MaxPooling2D  (None, 12, 12, 60)       0
 )

 conv2d_2 (Conv2D)           (None, 10, 10, 30)        16230

 conv2d_3 (Conv2D)           (None, 8, 8, 30)          8130

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 30)         0
 2D)

 dropout (Dropout)           (None, 4, 4, 30)          0

 flatten (Flatten)           (None, 480)               0

 dense (Dense)               (None, 500)               240500

 dropout_1 (Dropout)         (None, 500)               0

 dense_1 (Dense)             (None, 43)                21543

=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
_________________________________________________________________
```

Table 5.1. Model Layers and the Output Shape

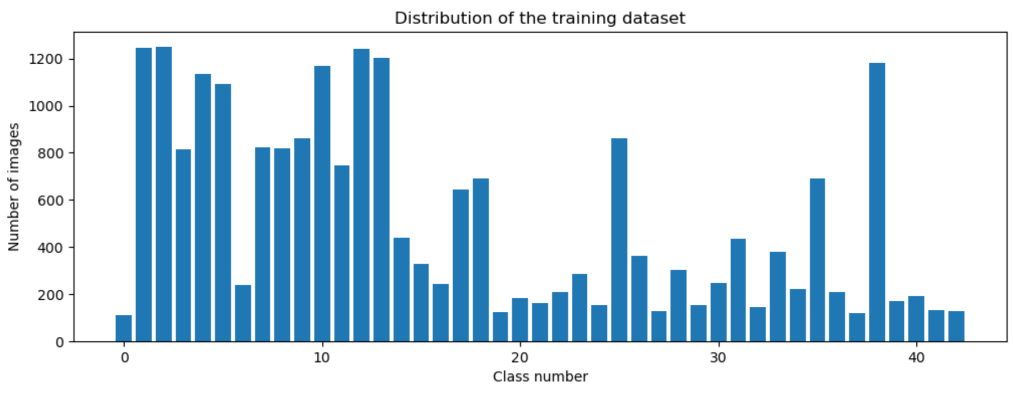
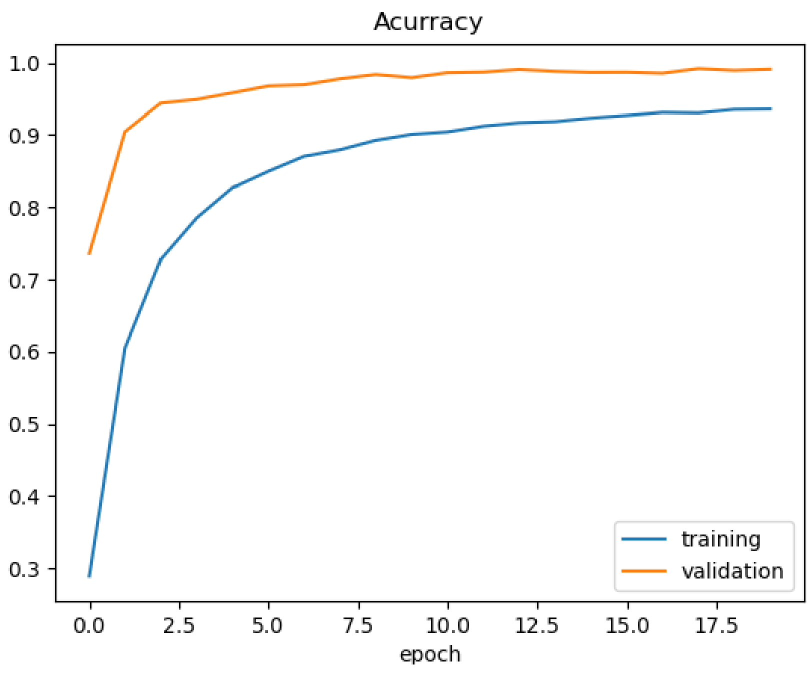


Fig 5.2. Distribution of Training Data Set

Taking the main.py file first into consideration, this is our machine learning model file and the output is relatively standard. When the code is executed, the program imports all the libraries required and also imports the images from the data set. It also displays the shape and the relative size of the data we are dealing with and then proceeds to split the data into training and testing sets. After this we decide to pre-process the images which helps the computer more easily to understand and process the images we are about to train and validate it on. Then we write an argument to the images to make them look more generic which helps train the data for more real-life scenarios. Next, we come to building the convolutional neural network model itself here we use a combination of various convolutional and max pooling layers and with fewer dropout layers of 0.5 sprinkled in between. Finally, we compile it into a flatten layer and we have a model then we come to the main part of training the model. As we can see from the graph(s) (Fig 5.3 & Fig. 5.4) provided below in both loss testing and accuracy testing has been favourable and no underfitting and overfitting has occurred so it is safe to proceed with this model. Our test scores and test accuracy for the model are as follows:



Fi 5.3. Graph showing Accuracy after training

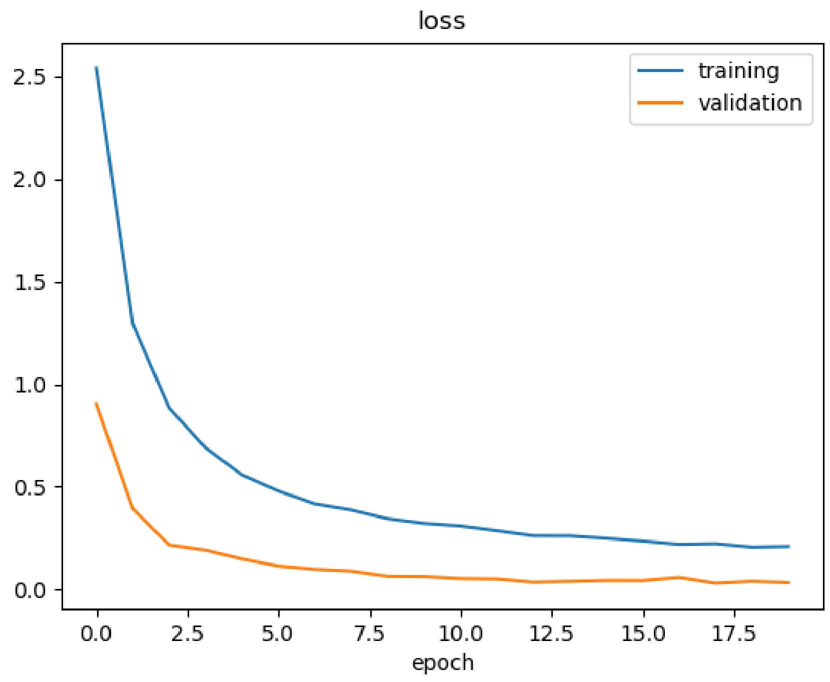
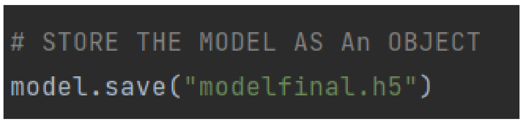


Fig. 5.4. Graph showing loss after training

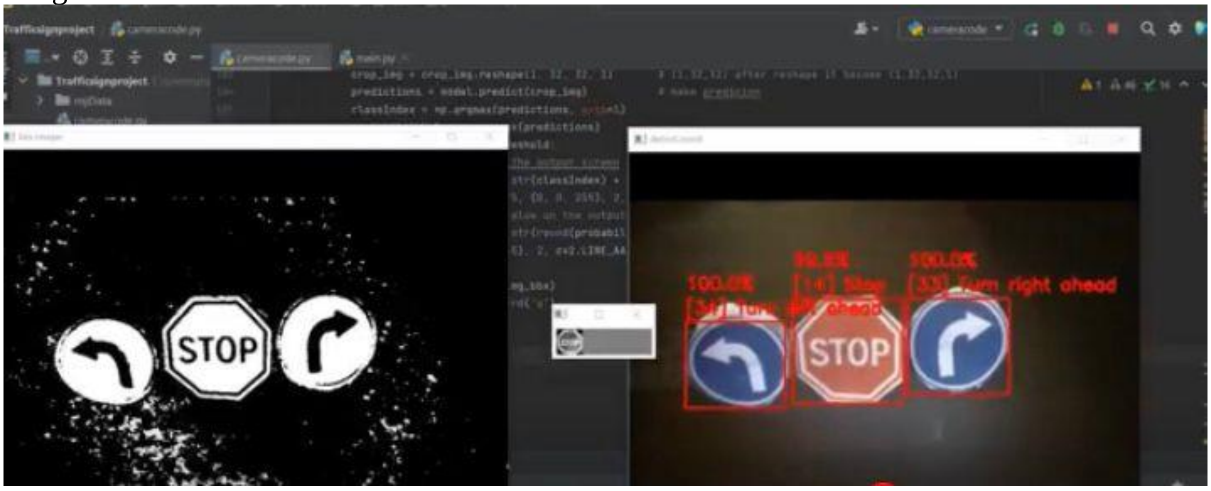
We now save the model as a h5 object and we name this "modelfinal.h5".



```
# STORE THE MODEL AS An OBJECT
model.save("modelfinal.h5")
```

Fig 5.5. Saving Model as h5 object

Next coming to the cameracode.py file, we import only 3 modules that is required in this code namely being numpy, cv2 and tensorflow for keras, as first using the keras function "keras.model" we load the model that we have saved and give code to further process the images that will be shown in front of the web camera.



```
crop_img = crop_img.reshape(1, 32, 32, 1)
predictions = model.predict(crop_img)
classIndex = np.argmax(predictions, axis=1)
```

Fig. 5.6. Output of cameracode.py

CHAPTER 6: CONCLUSION AND FUTURE SCOPE

The main conclusion that this project gives us is the fact that we were able to design a model that is to be able to detect traffic signs with a greater accuracy than initially expected. The training and testing, both were successful as per our expectations and the requirements of the Problem Statement were solved.

Coming to the future scope of this implementation/model, we believe that this project can be upgraded to a major one with more training of images for traffic signs and also, training of other models for detecting objects on the roads such as Pedestrians, other vehicles and identifying them appropriately as a truck, car, bus etc., and even creating a model to detect lanes on the road. Many tech giants and startups are still on the experimental phase when it comes to autonomous cars. The technology is still new and budding. Indian roads are a bit more challenging as compared to roads in the western countries.

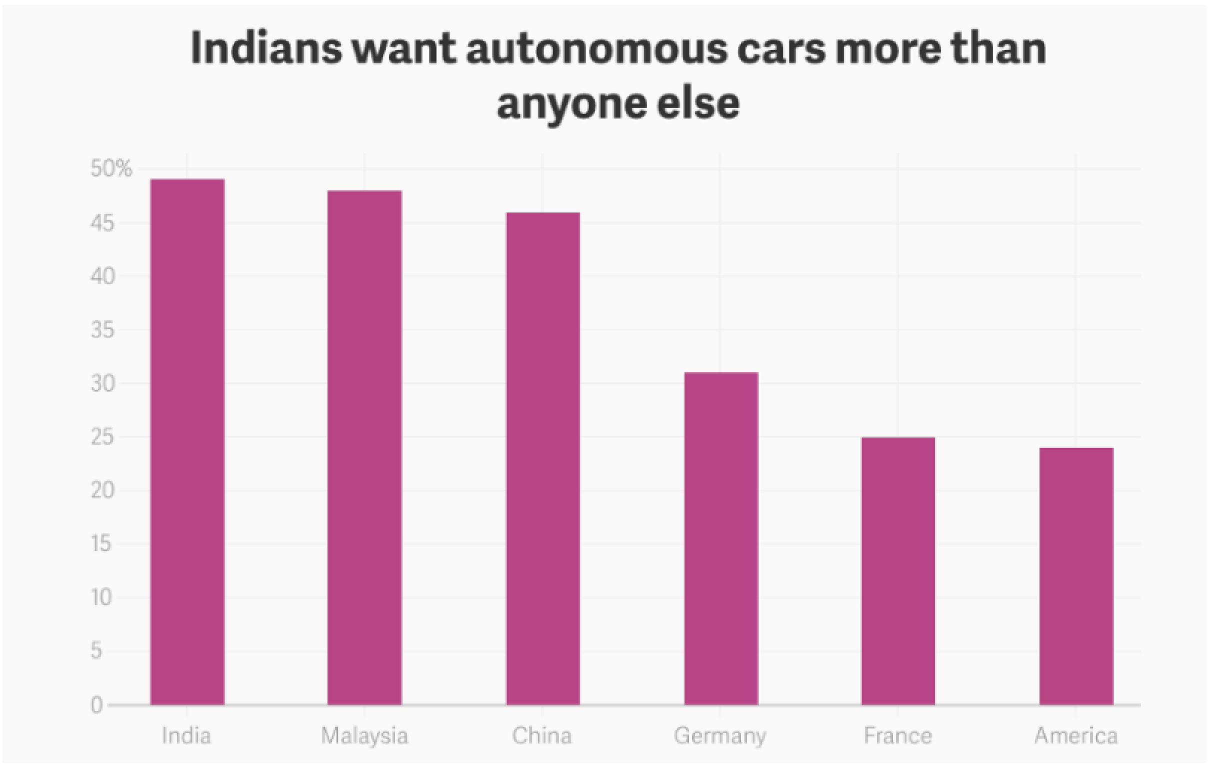
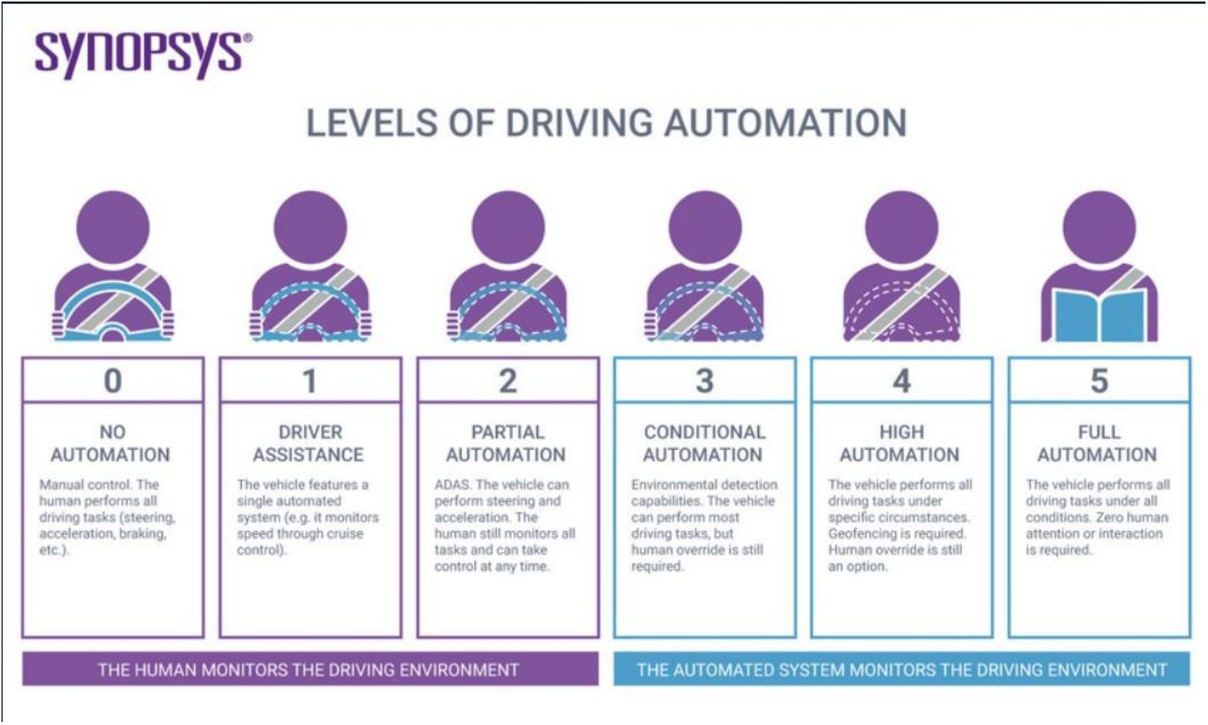


Fig 6.1 Graph Showing Demand for Self-Driving cars in different countries

First and foremost, we must comprehend India's autonomous vehicle requirements. Indian roadways are notorious for being clogged with traffic, resulting in extremely congested roads. The frequency of vehicle accidents and the number of people killed in them is increasing at an alarming rate. These are the few primary problem areas where autonomous vehicles can make a significant difference.

A good way to introduce Self Driving cars into the Indian market would be through ADAS (Advanced Driver Assistance System). Application of ADAS includes Pedestrian detection/avoidance, Lane departure warning/correction, Traffic sign recognition, Automatic emergency braking, Blind spot detection. These lifesaving systems are key to ensuring the success of ADAS applications, incorporating the latest interface standards and running multiple vision-based algorithms to support real-time multimedia, vision co-processing, and sensor fusion subsystems.


SYNOPSYS®
LEVELS OF DRIVING AUTOMATION
0
NO AUTOMATION
1
DRIVER ASSISTANCE
2
PARTIAL AUTOMATION
3
CONDITIONAL AUTOMATION
4
HIGH AUTOMATION
5
FULL AUTOMATION
THE HUMAN MONITORS THE DRIVING ENVIRONMENT
THE AUTOMATED SYSTEM MONITORS THE DRIVING ENVIRONMENT

Manual control. The human performs all driving tasks (steering, acceleration, braking, etc.).

The vehicle features a single automated system (e.g. it monitors speed through cruise control).

ADAS. The vehicle can perform steering and acceleration. The human still monitors all tasks and can take control at any time.

Environmental detection capabilities. The vehicle can perform most driving tasks, but human override is still required.

The vehicle performs all driving tasks under specific circumstances. Geofencing is required. Human override is still an option.

The vehicle performs all driving tasks under all conditions. Zero human attention or interaction is required.

Fig 6.2 ADAS system features

References

- Ajay More. Press Release. *Autonomous Vehicle Market Share 2021 Global Trend, Segmentation, Size, Business Growth, Top Key Players Analysis Industry, Opportunities and Forecast to 2030.* Weblog. [Online] Available from: https://www.marketwatch.com/press-release/autonomous-vehicle-market-share-2021-global-trend-segmentation-size-business-growth-top-key-players-analysis-industry-opportunities-and-forecast-to-2030-2021-07-21-5197440 [Accessed from: 21st July 2021].

- Soumya Ramasubhramanian. Technology. *India is Second-Least Prepared Country for driverless vehicles, report says.* Weblog. [Online] Available from: https://www.thehindu.com/sci-tech/technology/india-is-second-least-prepared-country-for-driverless-vehicles-report-says/article32395940.ece [Accessed from: 19th August 2020]

- Connor Hoffman. *How Capable Is Tesla's Autopilot Driver-Assist System? We Put It to the Test.* Weblog.[Online] Available from: https://www.caranddriver.com/news/a35839385/tesla-autopilot-full-self-driving-autonomous-capabilities-tested-explained/ [Accessed from: 30th April 2021]

- Anton Hristozov. Technical Article. *The role of Artificial Intelligence in Autonomous Vehicles*. Weblog. [Online] Available from: https://www.embedded.com/the-role-of-artificial-intelligence-in-autonomous-vehicles/ [Accessed from: 15th July 2020]

- Synopsys. Automotive. *What is ADADS?* Weblog. [Online] Available from: https://www.synopsys.com/automotive/what-is-adas.html [Accessed from: N/A]

- Nancy Cohen. Automotive. *BMW puts traffic light recognition to the test*. Weblog. [Online] Available from: https://techxplore.com/news/2019-06-bmw-traffic-recognition.html [Accessed from: 27th June 2019]

- Dr. Lance B Eliot. AI Trends Insider on Autonomy. Self-Driving Cars. *Framework of AI Driverless Cars: The Big Picture.* Weblog. [Online] Available from: https://www.aitrends.com/ai-insider/framework-ai-self-driving-driverless-cars-big-picture/ [Accessed from: 24th October 2017]

- Data Set: https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/published-archive.html

APPENDIX-I

Libraries used in Python

1) **NumPy** - NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

2) **cv2** - OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source Apache 2 License.

3) **Matplotlib** - Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

4) **pandas** - pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

5) **Keras** - Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML.

6) **os** - The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The os and os.path modules include many functions to interact with the file system.

7) **Scikit-learn** - it is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines

8) **Tensorflow** - TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on the training and inference of deep neural networks.