Recognition and Mapping of Traffic Signs

A PROJECT REPORT

for

Artificial Intelligence (ITE2010)

in

B.Tech (Information Technology)

by

Vinay Ravindra Maske (20BIT0128)
Devansh Kumar (20BIT0141)
Sharon Dhawan (20BIT0168)
Ishaan Chhipa (20BIT0179)
Aakanksha Satish Bagal (20BIT0190)

Sem 5, Third year

Under the guidance of

Prof. Dr. Harshita Patel

Associate Professor (Senior), SITE

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

November, 2022

DECLARATION BY THE CANDIDATE

We here by declare that the project report entitled **"Recognition and Mapping of Traffic Signs"** submitted by us to Vellore Institute of Technology University, Vellore in partial fulfilment of the requirement for the award of the course **Artificial Intelligence (ITE2010)** is a record of bonafide project work carried out by us under the guidance of **Prof. Dr. HARSHITA PATEL.** We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other course.

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology & Engineering [SITE]

CERTIFICATE

This is to certify that the project report entitled **"Recognition and Mapping of Traffic Signs"** submitted by **Vinay Ravindra Maske (20BIT0128), Devansh Kumar (20BIT0141), Sharon Dhawan (20BIT0168), Ishaan Chhipa (20BIT0179), Aakanksha Satish Bagal (20BIT0190)** to Vellore Institute of Technology University, Vellore in partial fulfillment of the requirement for the award of the course **Artificial Intelligence (ITE2010)** is a record of bonafide work carried out by them under my guidance.

Prof. Dr. HARSHITA PATEL

GUIDE

Associate Professor (Senior), SITE

Index

Recognition and Mapping of Traffic Signs

Vinay Ravindra Maske[1], Devansh Kumar[2], Sharon Dhawan[3], Ishaan Chhipa[4], Aakanksha S. Bagal[5]

[1,2,3,4,5]**Department of Information Technology, VIT University, Vellore, Tamil Nadu, India**

Abstract

Our project aims at detecting and recognizing traffic sign board in varied condition, states of lighting condition, orientation. Then storing it with their particular co-ordinates so that we can monitor any updates or changes in traffic sign on that particular place or route. Recognition of Traffic sign have various machinery application in the likes of which would be Intelligent Transport Systems like automated cars. This would be crucial and hence it is important that the right Traffic Sign be recognized with best accuracy, in such a case as it can cause serious accidents otherwise. In a real time, environment, for detection of Traffic signs we have implemented it using YOLO object detection Algorithm to draw a bounding box around the detected road sign. However, in a more ideal case for solely recognition of the Traffic Sign, where we have an image of only the traffic sign with very little background, we have used a CNN approach. We can use the part of the image inside the bounding box obtained by using YOLO algorithm as the input image for the CNN.

Keywords – YOLO, CNN, Character segmentation, Preprocessing, RGB to YCbCr, Feature extraction

Introduction

Traffic-sign recognition is a technology by which a vehicle is able to recognize the traffic signs put on the road e.g., speed limit, children turn ahead, etc. The technology is being developed by a variety of automotive suppliers. It uses image processing techniques to detect the traffic signs. The detection methods can be generally divided into color based, shape based and learning based methods.

Traffic-sign recognition first appeared, in the form of speed limit sign recognition, in 2008 for the 2009 Vauxhall Insignia[1]. Later in 2009 they appeared on the new BMW 7 Series, and the following year on the Mercedes-Benz S-Class. At that time, these systems only detected the round speed limit signs found all across Europe[2]. Traffic signs can be analyzed using forward-facing cameras in many modern cars, vehicles and trucks. One of the basic use cases of a traffic-sign recognition system is for speed limits. Most of the GPS data would procure speed information, but additional speed limit traffic signs can also be used to extract information and display it in the dashboard of the car to alert the driver about the road sign. This is an advanced driver-assistance feature available in most high-end cars, mainly in European vehicles[3].

We are proposing that our model will first recognize that traffic sign and then it will store that in excel file along with the coordinate of where that sign was present so that if we or anyone else go to that same route again he/she will know which sign are going to appear and when and prepare themself for that and in self driving this method is very useful, system will know in advance and could react accordingly, And if there is any change it will report to main server to change at that particular place or particular sign.

Literature review

Munoz-Organero et al.[1] Proposed a novel mechanism that focuses on the automatic detection of street elements such as traffic lights, street crossings and roundabouts which could be used to generate street maps and populate them with traffic influencing infrastructural elements such as traffic lights. In order to minimize the system requirements and simplify the data collection from many users with minimal impact for them, only traces of GPS data from a mobile device while driving is used. The proposed algorithm achieves a combined recall of 0.89 and a combined precision of 0.88 for classification.

Arinaldi, Ahmad et al.[2] The core of such system is the detection and classification of vehicles in traffic videos. We implement two models for this purpose, first is a MoG + SVM system and the second is based on Faster RCNN, a recently popular deep learning architecture for detection of objects in images. We show in our experiments that Faster RCNN outperforms MoG in detection of vehicles that are static, overlapping or in night time conditions. Faster RCNN also outperforms SVM for the task of classifying vehicle types based on appearances. Faster RCNN outperforms the SVM classifier.

Kurniawan, Jason et al.[3] We use a deep learning architecture, convolutional neural network (CNN) which is currently the state-of-the art for image processing method. We only do minimal image pre-processing steps on the small size image, where the conventional methods require a high quality, handcrafted features need to do manual calculation. The CNN model is trained to do binary classification about road traffic condition using 1000 CCTV monitoring image feeds with balance distribution. CNN basic architecture that trained on small grayscale images has an average classification accuracy of 89.50%.

Yu, Changhe et al.[4] They propose an architecture which combines deep packet detection and semi-supervised machine learning of multi-classifier in SDN. This architecture can classify flows into different QoS categories. Based on this, network can achieve fine-grained adaptive QoS traffic engineering. Moreover, through deep packet detection techniques, network can maintain a dynamic flow database. Classifier can adapt to the rapid emergence of network application and fickle traffic characteristics of current network by periodically re-training with the dynamic flow database. Experimental results show high accuracy for classification.

Kryvinska, Natalia et al.[5] Describes an approach towards road sign detection and recognition system. The model consists of two parts: localization and recognition. First one is responsible for the identification of region of interest by creating rectangular boxes marking off areas where signs are located. Recognition part takes these rectangular areas and uses data from those regions to tell which road traffic sign is located there. Correct classification is dependent on both parts working and communicating quickly, accurately and efficiently. It also shows the techniques used for the recognition and classification of the road signs in scientific environments and it describes the technology and the database of images that are widely used. Accuracy of 63%.

Sheikh, Muhammad Sameer et al.[6] proposed the estimation and detection of traffic incidents based on independent component analysis (ICA) and hybrid observer (HO)-generalized likelihood ratio (GLR) techniques. The combined HO-GLR method can produce better incident detection, improve traffic safety, and enhance traffic management systems. The performance metrics used to evaluate the performance of the proposed method includes detection rate, false alarm rate, classification rate, mean time to detection and the area under receiving operating characteristics curve. The proposed method obtained a better DR value of 97.58% and 97.80%, lower FAR value of 2.30% and 0.23%, a higher CR value of 94.83% and 97.18%, lower MTTD value of 2.20 min and 3.75 min, and a higher AUC value of 94.82% and 95.62% on the AYE and the I-880 datasets.

Ali, MD Hazrat, Syuhei Kurokawa et al.[7] proposed an efficient traffic control system by detecting and counting the vehicle numbers at various times and locations. At present, one of the biggest problems in the main cities in many countries are the traffic jam during office hour and office break hour. This can be handled by adjusting TSL timing proposed by the developed ARSS. High accuracy with less false detection.

Laguna, Rubén et al.[8] First, an image pre-processing step and the detection of regions of interest (ROIs), which involves a series of steps that include transforming the image to grayscale and applying edge detection by the Laplacian of Gaussian (LOG) filter. The potential traffic signs detection, Then, a recognition stage using a cross-correlation algorithm, where each potential traffic sign, if validated, is classified according to the data-base of traffic signs. Finally, the previous stages can be managed and controlled by a graphical user interface. The percentages of recognized signs for this application are high.

Tabernik, Domen et al.[9] proposed convolutional neural network (CNN) approach, the Mask R-CNN, to address the full pipeline of detection and recognition with automatic end-to-end learning and also several improvements that are evaluated on the detection of traffic signs and result in an improved overall performance. This approach is applied to detection of 200 traffic-sign categories represented in our novel dataset. Below 3% error rates.

Pon, Alex et al.[10] proposed a deep hierarchical architecture in conjunction with a mini-batch proposal selection mechanism that allows a network to detect both traffic lights and signs from training on separate traffic light and sign datasets. In this paper they focused on autonomous car deployment and show our network is more suitable than others because of its low memory footprint and real-time image processing time.

Latif, Ghazanfar et al.[11] proposed Deep Convolutional Neural Network (CNN) to develop an autonomous traffic and road sign (ATRS) detection and recognition system. The proposed system works in real time detecting and recognizing traffic sign images. The overall accuracy of the combined Multilanguage database was 99.26% with a precision of 99.29% on average.

Wan, Haifeng et al.[12] proposed Traffic Sign Yolo (TS-Yolo) based on the convolutional neural network to improve the detection and recognition accuracy of traffic signs, especially under low visibility and extremely restricted vision conditions. the mixed depth-wise convolution (MixConv) was employed to mix different kernel sizes in a single convolution operation, so that different patterns with various resolutions can be captured. Furthermore, the

attentional feature fusion (AFF) module was integrated to fuse the features based on attention from same-layer to cross-layer scenarios, including short and long skip connections, and even performing the initial fusion with itself.
The precision was 74.53 and 2.61 higher than that with data augmentation.

Ellahyani, Ayoub et al.[13] reviews the popular traffic sign detection methods (TSD) prevalent in recent literature. It enhances the safety by informing the drivers about the current state of traffic signs and offering valuable information about precautions. The methods are divided into color-based, shape-based, and machine learning based ones. Color space, segmentation method, features, and shape detection method are the terms considered in the review of the detection module.
We obtained competitive results, with an area under the precision-recall curve(AUC) of 99.73% in the category "Danger", and an AUC of 97.62% in the category "Mandatory".

Wu, Yihui et al.[14] proposed e an approach for traffic sign detection based on Convolutional Neural Networks (CNN). First transform the original image into the gray scale image by using support vector machines, then use convolutional neural networks with fixed and learnable layers for detection and recognition. The fixed layer can reduce the amount of interest areas to detect, and crop the boundaries very close to the borders of traffic signs. The learnable layers can increase the accuracy of detection significantly. Result of 99.72% in the category "Danger", and 97.62% in the category "Mandatory".

Houben, S., et al.[15] This paper proposes the design and analysis of the "German Traffic Sign Recognition Benchmark" dataset and competition. The results of the competition show that state of-the-art machine learning algorithms perform very good in the challenging task of traffic sign recognition.
Very high performance of up to 98.98% correct recognition rate which is similar to human performance on this dataset.

Yao, Yingbiao, et al.[16] improves YOLOv4-Tiny's feature fusion method and proposes an adaptive feature pyramid network (AFPN), which aims to adaptively fuse the two feature layers with different scales. Secondly, two receptive field blocks (RFB) are added after the two feature layers of the backbone network. These two RFBs are composed of multi-branch structures and dilated convolution with different dilation rates, which can enhance the feature extraction ability of the backbone network. The CCTSDB and GTSDB datasets are used to evaluate the effectiveness of the improved method. The experimental results show that our proposed network is superior to the original network in the precision, recall rate, and mAP. The proposed network improves the precision, recall rate, and mAP by 2.62%, 2.17%, and 1.34%, respectively.

Rodríguez, Rúben Castruita, et al.[17] propose a methodology for detecting and classifying Mexican traffic signs using deep learning. The methodology consists of the creation of a new Mexican traffic sign data set, the training, testing, and comparing of two sign detectors (the Region-based Convolutional Neural Network (R-CNN) and the You Only Look Once (YOLO v3)), and the use of a modified Residual Neural Network (ResNet-50) for classification. According to the detection results, the combination R-CNN/ResNet-50 yielded a mean Average Precision (mAP) of 95.33%, while the YOLO v3/ResNet-50 yielded 90.33%. The

overall classification accuracy was 99.00%. Our results are competitive to those presented in the literature. We demonstrated the robustness of our proposal by conducting a test to classify images that do not contain traffic signs. The accuracy for the R-CNN/ResNet-50 was 99.5% and 99.77% for the YOLO v3/ResNet-50.

Kassani, Peyman Hosseinzadeh et al.[18] Traffic sign recognition (TSR) is an integrated part of driver assistance systems and it remains an active research topic in computer vision today. This paper proposes a solution for TSR problem which composed of robust traffic sign image descriptor and sparse classifiers. Specifically, we outline a variant of histogram of oriented gradients (HOG), namely Soft HOG (SHOG) which exploits the symmetry shape of traffic sign images to find the optimal locations of the cell of histogram for SHOG computation. We show that our compact SHOG feature is more discriminative than HOG. With smaller feature size the proposed model can get higher classification accuracy.

Ouyang, Zhenchao et al.[19] Traffic light detection is a key module in the autonomous driving system to enhance the interactions between drivers and unmanned vehicles. In recent studies, deep neural networks are widely used for traffic light detection and resource/power consumption is a major concern for model deployment in vehicular edge devices. This paper proposes a novel light-weight deep CNN model that integrates the multi-backbone of state-of-the-art architectures for the self-driving traffic light detection. The MBBNet (Multi-BackBone Network) consists of three common convolutional backbones, i.e., the normal, residual and highway (DenseNet) convolutional modules. Simple ensemble of those backbones may incur high computational load. Therefore, channel compression is adopted to control the model parameters, while guaranteeing the accuracy for mobile and embedded hardware. It achieves higher accuracy ($accuracy > 0.94$).

Jiang, Feifeng et al.[20] Traffic crash detection is a major component of intelligent transportation systems. It can explore inner relationships between traffic conditions and crash risk, prevent potential crashes, and improve road safety. However, there exist some limitations in current studies on crash detection: (1) The commonly used machine learning methods cannot simulate the evolving transitions of traffic conditions before crash occurrences; (2) Current models collected traffic data of only one temporal resolution, which cannot fully represent traffic trends in different time intervals. Therefore, this study proposes a Long short-term memory (LSTM) based framework considering traffic data of different temporal resolutions (LSTMDTR) for crash detection. LSTM is an effective deep learning method to capture the long-term dependency and dynamic transitions of pre-crash conditions. The dropout technique can reduce overfitting and improve the generalization ability of the LSTMDTR model, Increasing crash accuracy from 64.49 % to 70.43 %.

Wang, Shuihua et al.[21] A computer vision-based wayfinding and navigation aid can improve the mobility of blind and visually impaired people to travel independently. In this paper, we develop a new framework to detect and recognize stairs, pedestrian crosswalks, and traffic signals based on RGB-D (Red, Green, Blue, and Depth) images. Since both stairs and pedestrian crosswalks are featured by a group of parallel lines, we first apply Hough transform to extract the concurrent parallel lines based on the RGB (Red, Green, and Blue) channels. Then, the Depth channel is employed to recognize pedestrian crosswalks and stairs. The detected stairs are further identified as stairs going up (upstairs) and stairs going down (downstairs). The distance between the camera and stairs is also estimated for blind users.

Furthermore, the traffic signs of pedestrian crosswalks are recognized. High effectiveness and accuracy.

Arcos-Garcia et al.[22] This paper presents an efficient two-stage traffic sign recognition system. First, 3D point cloud data is acquired by a LINX Mobile Mapper system and processed to automatically detect traffic signs based on their retro-reflective material. Then, classification is carried out over the point cloud projection on RGB images applying a Deep Neural Network which comprises convolutional and spatial transformer layers. Accuracy of 99.71%.

Shustanov et al.[23] Nowadays, more and more object recognition tasks are being solved with Convolutional Neural Networks (CNN). Due to its high recognition rate and fast execution, the convolutional neural networks have enhanced most of computer vision tasks, both existing and new ones. In this article, we propose an implementation of traffic signs recognition algorithm using a convolution neural network. The entire procedure for traffic sign detection and recognition is executed in real time on a mobile GPU. The experimental results confirmed high efficiency of the developed computer vision system. Accuracy of 99.94%.

Liu, Huaping, Yulong Liu et al.[24] Recognizing traffic signs is a challenging problem; and it has captured the attention of the computer vision community for several decades. Essentially, traffic sign recognition is a multi-class classification problem that has become a real challenge for computer vision and machine learning techniques. Although many machine learning approaches are used for traffic sign recognition, they are primarily used for classification, not feature design. Identifying rich features using modern machine learning methods has recently attracted attention and has achieved success in many benchmarks. First, we introduce a new feature learning approach using group sparse coding. The primary goal is to exploit the intrinsic structure of the pre-learned visual codebook. Second, we use a non-uniform quantization approach based on log-polar mapping. Using the log-polar mapping of the traffic sign image, rotated and scaled patterns are converted into shifted patterns in the new space. Influence of image noise can be reduced.

Megalingam, Rajesh Kannan et al.[25] This paper presents a deep-learning-based autonomous scheme for cognizance of traffic signs in India. The automatic traffic sign detection and recognition was conceived on a Convolutional Neural Network (CNN)- Refined Mask R-CNN (RM R-CNN)-based end-to-end learning. The proffered concept was appraised via an innovative dataset comprised of 6480 images that constituted 7056 instances of Indian traffic signs grouped into 87 categories. We present several refinements to the Mask R-CNN model both in architecture and data augmentation. We have considered highly challenging Indian traffic sign categories which are not yet reported in previous works. The evaluation results indicate lower than 3% error, proposed model achieves precision of 97.08%. Accuracy of the proposed RMR-CNN model is 97.08%.

Proposed algorithms

We propose a Traffic sign Detection and Recognition system that on recognition further identifies and notes the location of the traffic sign on the road. The algorithms that we are using in our model are Character segmentation, Image Pre-processing (Thresholding, Filtering, Edge Detection, RGB to YCbCr), CNN, YOLO, .

We use various image pre-processing techniques for the initial stages of detection. The first stage known as Pre-processing involves Thresholding, Filtering and Edge detection.

- Thresholding[4] : An image processing method that creates a bitonal (aka binary) image based on setting a threshold value on the pixel intensity of the original image. While most commonly applied to grayscale images, it can also be applied to colour images.

- Filtering[5] : Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

- Edge detection[6]: Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

The next stage involves conversion of the pre-processed image from RGB to YCbCr format. The advantage of YCbCr colour space is that it can separate luminance from chrominance more effectively compare to RGB colour space. This helps with the easier classification of the different colours present in the image. The next step involves DWT conversion which reduces the size of the image without compromising on its quality, and hence the resolution increases, increasing the efficiency of identification.
After processing the image according to our requirements, we perform Feature Extraction.

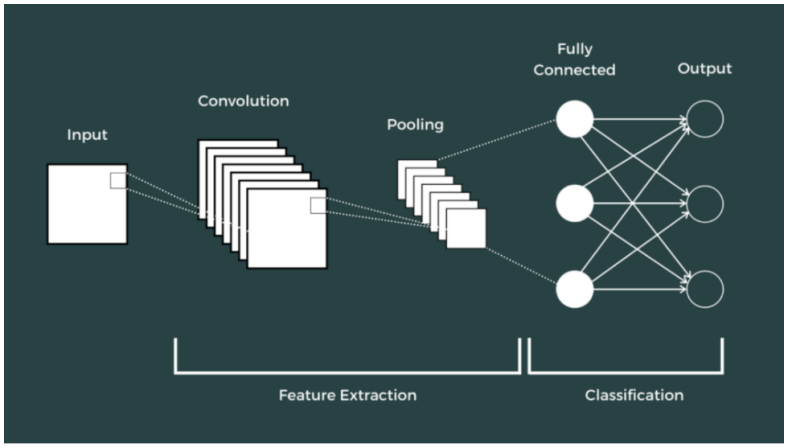


Figure: Explains the working of feature extraction within CNN
Image Source

Feature extraction is a part of the CNN algorithm. It refers to the process of transforming raw data into numerical features that can be processed while preserving the information in the original dataset[7].

The algorithms (CNN and YOLO) analyse the extracted features and compare the images to the ones in the dataset and identify the traffic sign shown.

CNN[8] :

A Convolutional Neural Network is a deep learning algorithm. It is most commonly used for classification and computer vision tasks. It takes an image as the input and assigns appropriate weights to the objects in the image to make them differentiable. So, two images that can be represented in the form of a matrix are multiplied with each other to produce an output which is further used to extract features from the image, here the role of the CNN is to reduce the images into a form that is easier to process and that with minimum loss of features to get a good prediction.

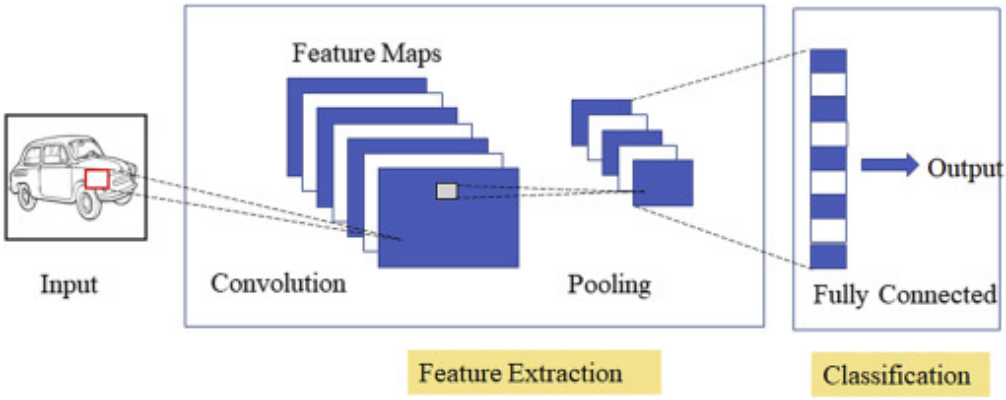


Figure: CNN structure
Image Source

YOLO[9] :

YOLO stands for "You Only Look Once". This algorithm is primarily used for object detection and recognition in a picture and in real time. Object detection in YOLO is done as a regression problem and it provides the class probabilities of the detected images. YOLO make use of CNN to perform object detection in real time. It only requires a single forward propagation through CNN to detect objects.

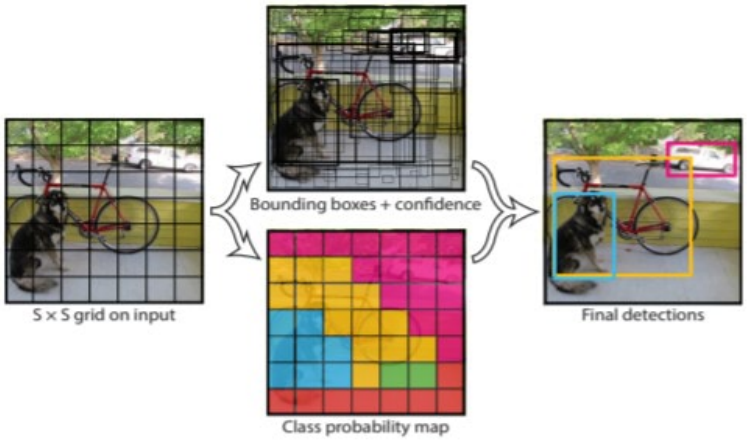


Figure : YOLO algorithm
Image Source

Powershell commands to access the location API:

The used script uses the Windows Location provider and the Google Geocoding API to retrieve the geographical location of the system. Implementing the following system in our model allows us to identify the location of the sign.

The database contains the identified sign along with the location at which it was observed by the system.

Figure: a representation of the location marks of the traffic signs on a map
Image source

Datasets Description and Sample Data

a) Dataset Information

Here is the dataset for classifying the different classes of traffic signs. There are around 58 classes and each class has around 120 images. the labels.csv file has the respective description of the traffic sign class. You can change the assignment of these classIDs with descriptions. We can use the basic CNN model to get decent accuracy. We have around 2000 files for testing.

b) Attribute Information

All files in the dataset are converted in .png format.

c) Sample Dataset

Link: https://www.kaggle.com/datasets/ahemateja19bec1025/traffic-sign-dataset-classification

The following dataset consists around 58 classes and each class has around 120 images and around 2000 files for testing.

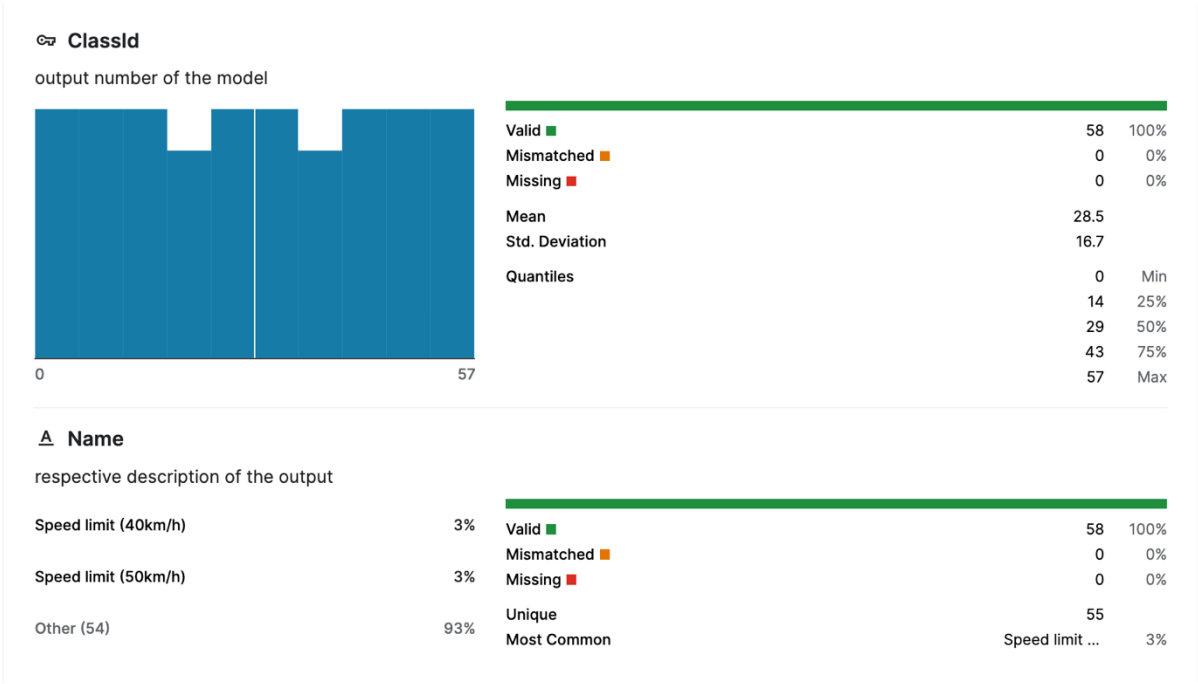


Figure: Description of the dataset

The above algorithms allow the model to identify the traffic sign displayed.
Further we aim to also locate the exact coordinates at which the sign was location. Hence we use PowerShell commands using API keys provided by Google themselves to mark the locations on a map.

Flowchart:

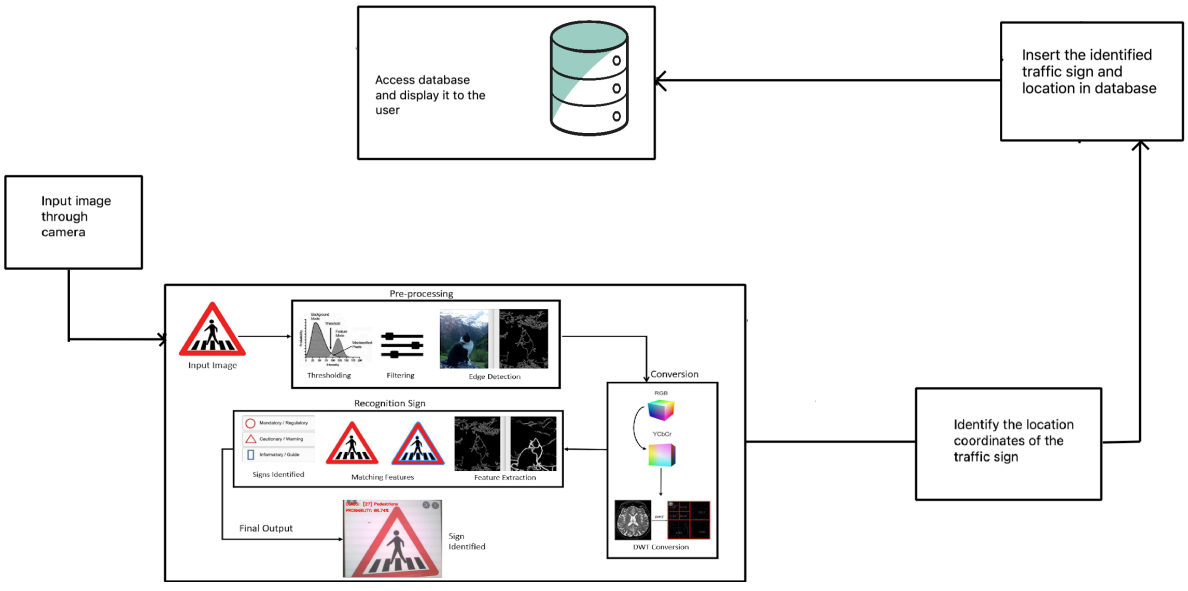


Figure: A flowchart of our working model

Identification of Performance measure

We will be using confusion matrix for evaluating the performance of our model.

Confusion matrix
Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with the predicted values. This tells us about how is our model is performing and what kind of errors is it making.

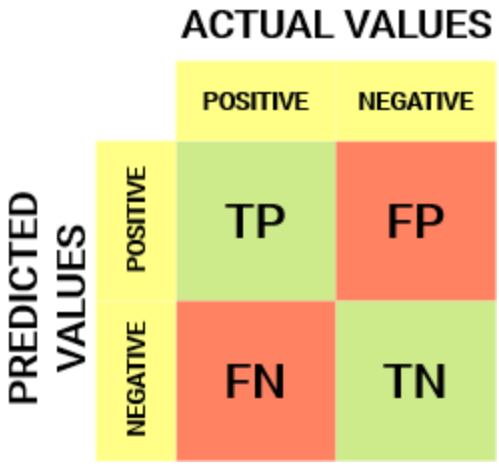


Figure: 2 x 2 Confusion Matrix (Confusion Matrix for Machine Learning, Analytics Vidhya, 2020)
Image Source

Accuracy from confusion matrix is calculated as follows

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision is how many correctly predicted cases turned out to be positive. This determines whether our model is reliable or not.

$$Precision = \frac{TP}{TP + FP}$$

Recall is how many of the actual positive cases were predicted correctly.

$$Recall = \frac{TP}{TP + FN}$$

The performance is measured by the accuracy of the model and probability that the sign will be correctly detected.
The accuracy versus epoch data will be visualized using matplotlib library, the model will be evaluated, and the loss and accuracy will be determined.

Code:

Trafficsign_main.ipynm (model_main)

```
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from 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 pickle
import os
import pandas as pd
import random
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

```
import os

path = "C:\\Users\\Ishaan\\AI project model (3)\\Dataset2\\traffic_Data\\DATA"
# path = "C:\\Users\\Ishaan\\AI project model (3)\\Dataset3\\Images"
# path = "C:\\Users\\Ishaan\\AI project model (3)\\Dataset5\\train"

i=0
# r=root, d=directories, f = files
for r, d, f in os.walk(path):
    for file in f:
#         if file.endswith('.png'):
#             pat=os.path.join(r, file)
#             with Image.open(pat) as im:
#                 if im.size!=(32, 32):
#                     im=im.resize((32, 32),Image.LANCZOS)
#                 im.save(pat.replace(".png",".jpg"))
#             os.remove(pat)
#             i+=1
#             print(i,end='\r')

        if file.endswith('.jpg'):
            pat=os.path.join(r, file)
            with Image.open(pat) as im:
                if im.size!=(32, 32):
                    im=im.resize((32, 32),Image.LANCZOS)
```

```
#                     im.save(pat)
                im.save(pat.replace(".jpg",".png"))
                os.remove(pat)
                i+=1
                print(i,end='\r')

        elif file.endswith('.ppm'):
            pat=os.path.join(r, file)
            with Image.open(pat) as im:
                im.save(pat.replace(".ppm",".png"))
            os.remove(pat)
            i+=1
            print(i,end='\r')

        elif file.endswith('.csv'):
            pat=os.path.join(r, file)
            os.remove(pat)
```

```
################ Parameters ####################

path = "myData" # folder with all the class folders
labelFile = 'labels.csv' # file with all names of classes
batch_size_val=50  # how many to process together
steps_per_epoch_val=2000
epochs_val=20
imageDimesions = (32,32,3)
testRatio = 0.2    # if 1000 images split will 200 for testing
validationRatio = 0.2 # if 1000 images 20% of remaining 800 will be 160 for
validation
################################################
```

```
############################## 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)
```

```
Total Classes Detected: 43
Importing Classes.....
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
```

```
############################### 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)

# 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"
```

```
Data Shapes
Train(22271, 32, 32, 3) (22271,)
Validation(5568, 32, 32, 3) (5568,)
Test(6960, 32, 32, 3) (6960,)
```

```
############################### 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))
```

Figure: images from dataset extracted to train our model

Experimental results

model_main:

We trained our model and using the given dataset we get the following results:

```
In [7]: ############################### 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()
```

```
[110, 1266, 1299, 791, 1113, 999, 215, 839, 808, 864, 1153, 769, 1182, 1213, 451, 344, 219, 6
47, 699, 110, 191, 170, 226, 292, 148, 849, 362, 138, 311, 169, 250, 436, 133, 367, 232, 700,
215, 122, 1221, 186, 205, 134, 123]
```

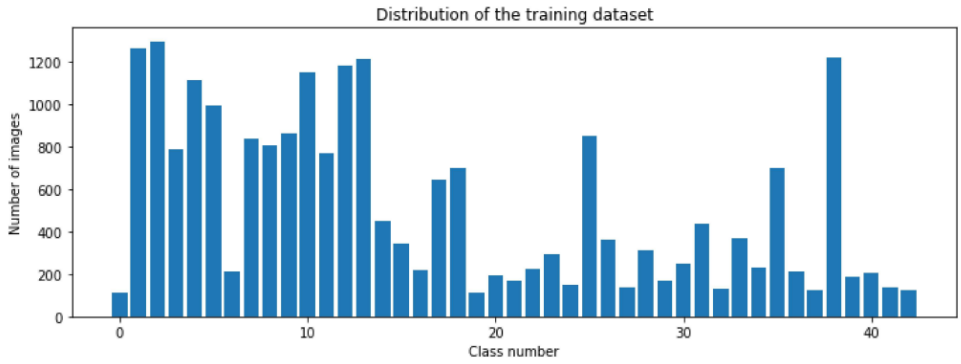


Figure: Distribution of the training Dataset

```
############################### 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_val
idation.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 MAKEIT 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)
```

```
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],image
Dimesions[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
```

```
############################### TRAIN
model = myModel()
print(model.summary())
history=model.fit_generator(dataGen.flow(X_train,y_train,batch_size=batch_size_val)
,steps_per_epoch=len(X_train)//batch_size_val,epochs=epochs_val,validation_data=(X_
validation,y_validation),shuffle=1)
#
history=model.fit_generator(dataGen.flow(X_train,y_train,batch_size=batch_size_val)
,steps_per_epoch=steps_per_epoch_val,epochs=epochs_val,validation_data=(X_validatio
n,y_validation),shuffle=1)
```

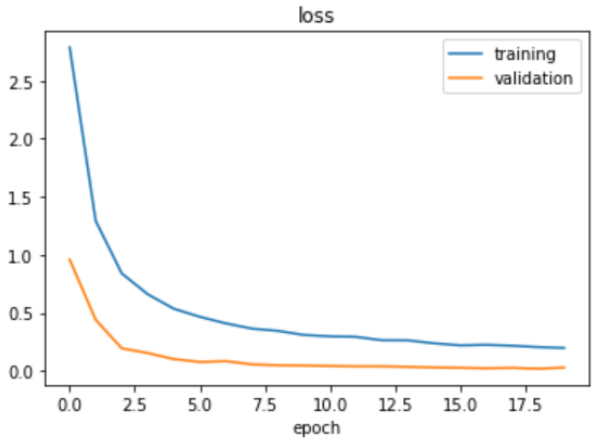
Model summary →

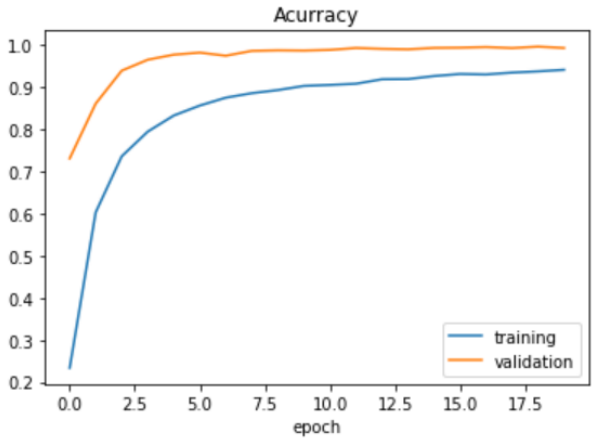
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 60)        1560

 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
_________________________________________________________________
```

Epochs →

```
Epoch 1/20
445/445 [==============================] - 113s 246ms/step - loss: 2.7867 - accuracy: 0.2336 - val_loss: 0.9591 - val_accuracy: 0.7302
Epoch 2/20
445/445 [==============================] - 106s 239ms/step - loss: 1.2922 - accuracy: 0.6025 - val_loss: 0.4400 - val_accuracy: 0.8603
Epoch 3/20
445/445 [==============================] - 107s 241ms/step - loss: 0.8398 - accuracy: 0.7353 - val_loss: 0.1946 - val_accuracy: 0.9384
Epoch 4/20
445/445 [==============================] - 108s 243ms/step - loss: 0.6599 - accuracy: 0.7938 - val_loss: 0.1530 - val_accuracy: 0.9643
Epoch 5/20
445/445 [==============================] - 107s 239ms/step - loss: 0.5352 - accuracy: 0.8320 - val_loss: 0.1013 - val_accuracy: 0.9763
Epoch 6/20
445/445 [==============================] - 107s 241ms/step - loss: 0.4657 - accuracy: 0.8556 - val_loss: 0.0764 - val_accuracy: 0.9811
Epoch 7/20
445/445 [==============================] - 108s 242ms/step - loss: 0.4091 - accuracy: 0.8743 - val_loss: 0.0835 - val_accuracy: 0.9738
Epoch 8/20
445/445 [==============================] - 106s 239ms/step - loss: 0.3639 - accuracy: 0.8850 - val_loss: 0.0559 - val_accuracy: 0.9853
Epoch 9/20
445/445 [==============================] - 107s 241ms/step - loss: 0.3444 - accuracy: 0.8924 - val_loss: 0.0480 - val_accuracy: 0.9865
Epoch 10/20
445/445 [==============================] - 107s 240ms/step - loss: 0.3094 - accuracy: 0.9022 - val_loss: 0.0458 - val_accuracy: 0.9860
Epoch 11/20
445/445 [==============================] - 107s 241ms/step - loss: 0.2973 - accuracy: 0.9045 - val_loss: 0.0417 - val_accuracy: 0.9876
Epoch 12/20
445/445 [==============================] - 108s 243ms/step - loss: 0.2928 - accuracy: 0.9073 - val_loss: 0.0391 - val_accuracy: 0.9921
Epoch 13/20
445/445 [==============================] - 107s 241ms/step - loss: 0.2636 - accuracy: 0.9180 - val_loss: 0.0388 - val_accuracy: 0.9901
Epoch 14/20
445/445 [==============================] - 107s 240ms/step - loss: 0.2627 - accuracy: 0.9185 - val_loss: 0.0346 - val_accuracy: 0.9890
Epoch 15/20
445/445 [==============================] - 106s 238ms/step - loss: 0.2371 - accuracy: 0.9256 - val_loss: 0.0289 - val_accuracy: 0.9923
Epoch 16/20
445/445 [==============================] - 77s 174ms/step - loss: 0.2197 - accuracy: 0.9305 - val_loss: 0.0268 - val_accuracy: 0.9928
Epoch 17/20
445/445 [==============================] - 74s 165ms/step - loss: 0.2240 - accuracy: 0.9293 - val_loss: 0.0220 - val_accuracy: 0.9943
Epoch 18/20
445/445 [==============================] - 72s 163ms/step - loss: 0.2164 - accuracy: 0.9338 - val_loss: 0.0259 - val_accuracy: 0.9919
Epoch 19/20
445/445 [==============================] - 70s 158ms/step - loss: 0.2046 - accuracy: 0.9367 - val_loss: 0.0188 - val_accuracy: 0.9953
Epoch 20/20
445/445 [==============================] - 72s 162ms/step - loss: 0.1972 - accuracy: 0.9404 - val_loss: 0.0280 - val_accuracy: 0.9921
```

```
############################## 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])

print('\nPrecision: ',precision)
print('Recall: ',recall)
```






```
Test Score: 0.030351312831044197
Test Accuracy: 0.991235613822937

Precision:  0.9920844064768418
Recall:  0.9918302368831072
```

Output

To test the proposed model →

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


import numpy as np
import cv2
import pickle
import os
import keras
import tensorflow as tf
from keras.models import load_model

import random
import string

import subprocess as sp
import re
import time

from tempfile import NamedTemporaryFile
import shutil
import csv

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

filename = 'my.csv'
fields = ['sign', 'latitude', 'longitude', 'radius']
tempfile = NamedTemporaryFile(mode='w', delete=False)


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

frameWidth= 640         # CAMERA RESOLUTION
frameHeight = 480
brightness = 180
threshold = 0.75         # PROBABLITY THRESHOLD
font = cv2.FONT_HERSHEY_SIMPLEX

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

# SET THE COUNTDOWN TIMER
# for simplicity we set it to 3
# We can also take this as input
TIMER = int(20)
```

```
# SETUP THE VIDEO CAMERA
cap = cv2.VideoCapture(0)
cap.set(3, frameWidth)
cap.set(4, frameHeight)
cap.set(10, brightness)

# IMPORT THE TRANNIED MODEL

# try:
#     pickle_in=open("model_trained.sav","rb")  ## rb = READ BYTE
#     model=pickle.load(pickle_in)
# except EOFError:
#     data=list()

model = load_model('model_d0_3.h5')

def loca():
    wt = 5 # Wait time
    accuracy = 100

    # while True:
    # time.sleep(wt)
    pshellcomm = ['powershell']
    pshellcomm.append('add-type -assemblyname system.device; '
'$loc = new-object system.device.location.geocoordinatewatcher;'
'$loc.start(); '                        'while(($loc.status -ne "Ready") -and
($loc.permission -ne "Denied")) '                          '{start-sleep -
milliseconds 100}; '                          '$acc = %d; '
'while($loc.position.location.horizontalaccuracy -gt $acc) '
'{start-sleep -milliseconds 100; $acc = [math]::Round($acc*1.5)}; '
'$loc.position.location.latitude; '
'$loc.position.location.longitude; '
'$loc.position.location.horizontalaccuracy; '                          '$loc.stop()'
%(accuracy))

    p = sp.Popen(pshellcomm, stdin = sp.PIPE, stdout = sp.PIPE, stderr = sp.STDOUT,
text=True)
    (out, err) = p.communicate()
    out = re.split('\n', out)

    lat = float(out[0])
    long = float(out[1])
    radius = int(out[2])
    return lat,long,radius

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 vechiles 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 vechiles'
    elif classNo == 16: return 'Vechiles 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 vechiles over 3.5 metric tons'
```

```
while True:

    # READ IMAGE
    success, imgOrignal = cap.read()

    # PROCESS IMAGE
    img = np.asarray(imgOrignal)
    img = cv2.resize(img, (32, 32))
    img = preprocessing(img)
    cv2.imshow("Processed Image", img)
    img = img.reshape(1, 32, 32, 1)
    cv2.putText(imgOrignal, "CLASS: " , (20, 35), font, 0.75, (0, 0, 255), 2,
cv2.LINE_AA)
    cv2.putText(imgOrignal, "PROBABILITY: ", (20, 75), font, 0.75, (0, 0, 255), 2,
cv2.LINE_AA)

    # PREDICT IMAGE
    predictions = model.predict(img)
    #classIndex = model.predict_classes(img)
    classIndex = np.argmax(model.predict(img), axis=-1)
    probabilityValue =np.amax(predictions)
    if probabilityValue > threshold:
        #print(getCalssName(classIndex))
        cv2.putText(imgOrignal,str(classIndex)+" "+str(getCalssName(classIndex)),
(120, 35), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
        cv2.putText(imgOrignal, str(round(probabilityValue*100,2) )+"%", (180, 75),
font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
    cv2.imshow("Result", imgOrignal)

    # check for the key pressed
    o = cv2.waitKey(125)

    if o == ord('s'):

        lat,long,radius = loca()
        op = open("my.csv", "r")
        dt = csv.DictReader(op)
        up_dt = []
        for row in dt:
            r = {'sign': row['sign'], 'longitude': row['longitude'], 'latitude':
row['latitude'], 'radius': row['radius']}
            if row['sign'] != str(getCalssName(classIndex)) and row['longitude'] ==
str(long) and row['latitude'] == str(lat) and row['radius'] == str(radius) :
                r = {'sign': str(getCalssName(classIndex)), 'longitude':
row['longitude'], 'latitude': row['latitude'], 'radius': row['radius']}
            up_dt.append(r)

        op.close()

        op = open("my.csv", "w", newline='')
```

```
        headers = ['sign', 'longitude', 'latitude', 'radius']
        data = csv.DictWriter(op, delimiter=',', fieldnames=headers)
        data.writerow(dict((heads, heads) for heads in headers))
        data.writerows(up_dt)

        op.close()

    if o == ord('a'):

        lat,long,radius = loca()

        op = open("my.csv", "r")
        dt = csv.DictReader(op)
        up_dt = []

        for row in dt:
            r = {'sign': row['sign'], 'longitude': row['longitude'], 'latitude':
row['latitude'], 'radius': row['radius']}
            up_dt.append(r)
        op.close()

        si, lo, la, ra = str(getCalssName(classIndex)), str(long), str(lat),
str(radius)
        r = {'sign': si, 'longitude': lo, 'latitude': la, 'radius': ra}
        up_dt.append(r)

        op = open("my.csv", "w", newline='')
        headers = ['sign', 'longitude', 'latitude', 'radius']
        data = csv.DictWriter(op, delimiter=',', fieldnames=headers)
        data.writerow(dict((heads, heads) for heads in headers))
        data.writerows(up_dt)

        op.close()

    if o == 27:
        break



# close the camera
cap.release()

# close all the opened windows
cv2.destroyAllWindows()


# In[ ]:
```

We get the following results :

Figure: Identified “STOP” sign

Figure: Identified “Road work” sign

Figure: Identified “Speed limit 60km/h” sign

Along with the identification of the traffic signs, it also identifies the location of the device which is stored in a database corresponding to its sign.

	A	B	C	D
1	sign	longitude	latitude	radius
2	Stop	79.16166	12.9725	59
3	Road work	79.16166	12.9725	58
4	Speed Limit 60 km/h	79.16395	12.97069	55
5				

On addition to the database, the .csv file is read and used to update locate the traffic sign coordinates on a map

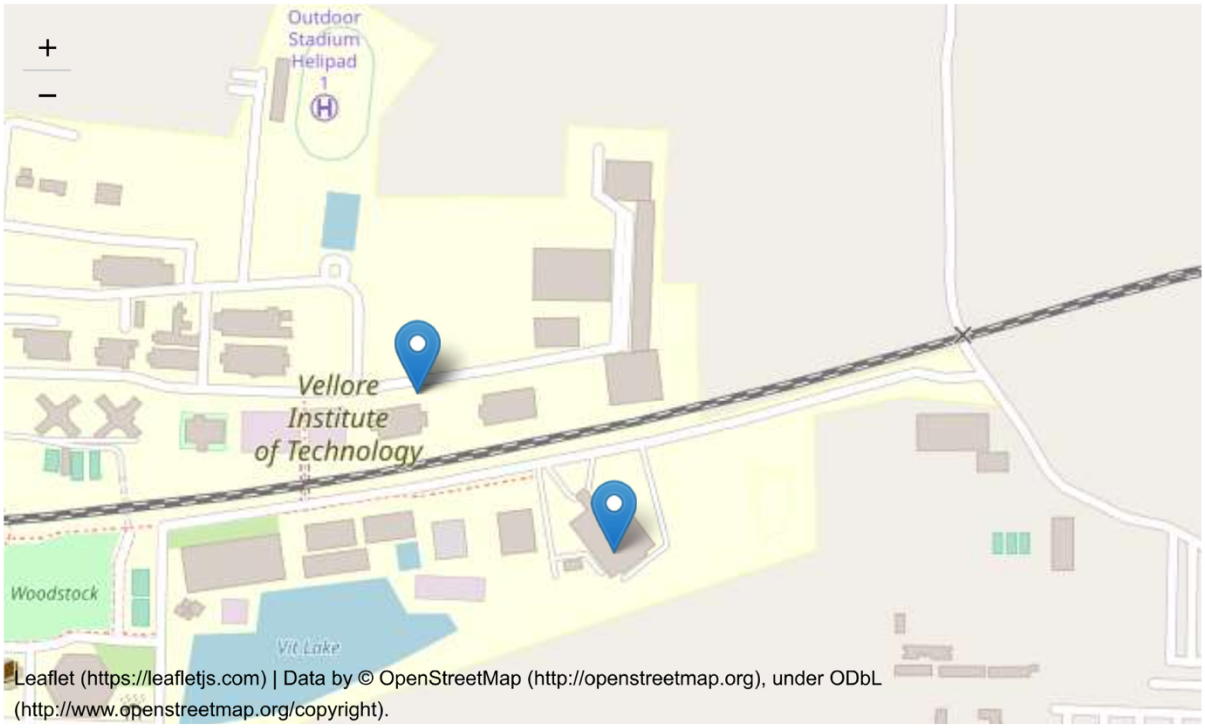
In [14]:
```
import folium
import pandas as pd
import re
```

In [15]:
```
signs = pd.read_csv('my.csv')
signs.head(3)
```

Out[15]:

	sign	longitude	latitude	radius
0	Stop	79.161659	12.972503	59
1	Road work	79.161664	12.972501	58
2	Speed Limit 60 km/h	79.163951	12.970685	55

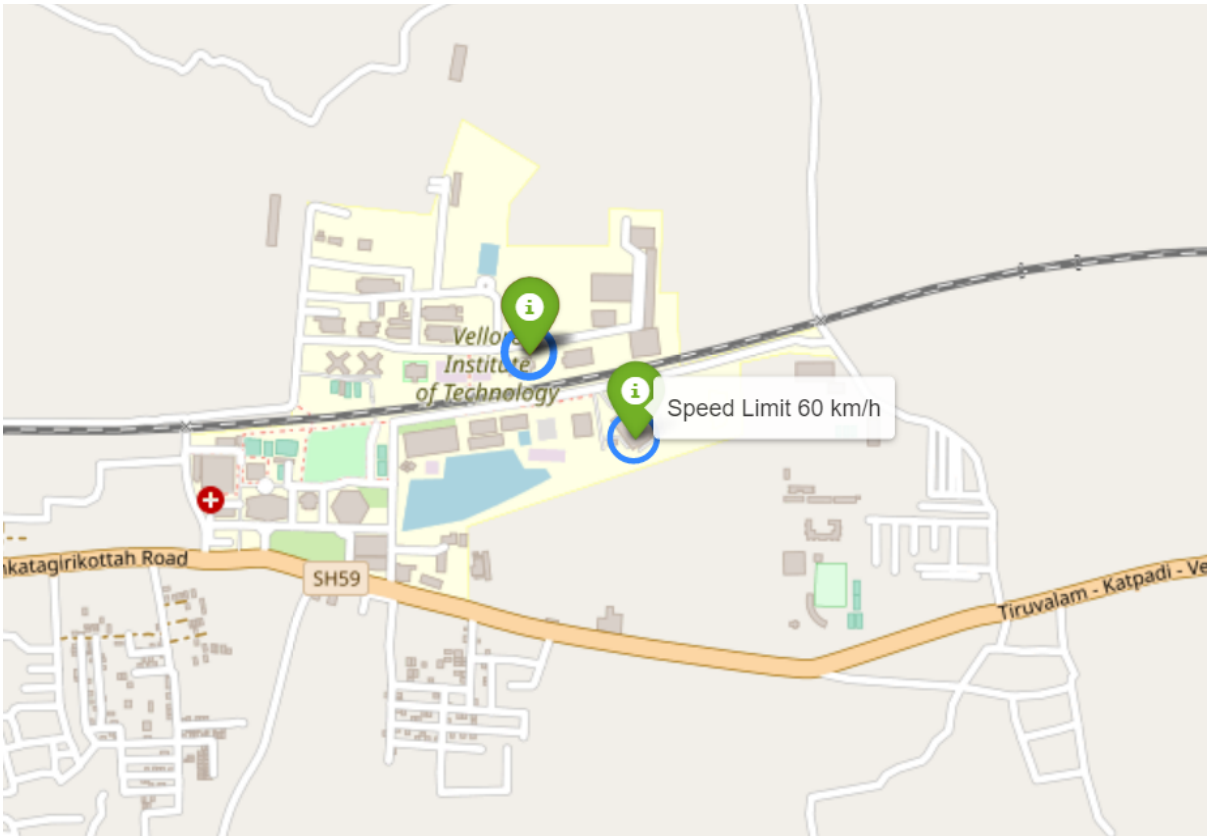
In [16]:
```
my_map = folium.Map(
    location = [12.9692, 79.15599],
    zoom_start=15
)

for _, sign in signs.iterrows():
    folium.Marker(
        location = [sign['latitude'], sign['longitude']],
        popup= sign['sign'],
        tooltip = sign['sign']
    ).add_to(my_map)

my_map
```



In [17]:
```
def select_marker_color(row):
    if row['sign'] == 'Stop':
        return 'red'
    elif row['sign'] == '^Speed limit':
        return 'blue'
    return 'green'
```

In [18]:
```
signs['color'] = signs.apply(select_marker_color, axis=1)
```

In [19]:
```
my_map = folium.Map(
    location = [12.9692, 79.15599],
    zoom_start=15
)

for _, sign in signs.iterrows():
    folium.Marker(
        location = [sign['latitude'], sign['longitude']],
        popup= sign['sign'],
        tooltip = sign['sign'],
        icon = folium.Icon(color=sign['color'])
    ).add_to(my_map)
    folium.Circle([sign['latitude'], sign['longitude']], radius = sign['radius']).add_to(my_m

my_map
```



Implementation of existing models for comparison

model_1[10]:

We use our main dataset to implement another model created by Gerry, Retired Director Satcom at General Dynamics**,** Scottsdale, Arizona, United States

Classification Report:

folders	precision	recall	f1-score	support
0	1.00	1.00	1.00	12
1	1.00	1.00	1.00	4
10	1.00	1.00	1.00	7
11	1.00	1.00	1.00	14
12	1.00	1.00	1.00	9
13	1.00	1.00	1.00	4
14	1.00	1.00	1.00	13
15	1.00	1.00	1.00	2
16	1.00	1.00	1.00	14
17	1.00	1.00	1.00	13
2	1.00	1.00	1.00	8
20	1.00	1.00	1.00	2
21	1.00	1.00	1.00	1
22	1.00	1.00	1.00	2
23	1.00	1.00	1.00	2
24	1.00	1.00	1.00	10
26	1.00	1.00	1.00	13
27	1.00	1.00	1.00	3
28	1.00	1.00	1.00	44
29	1.00	1.00	1.00	4
3	1.00	1.00	1.00	26
30	1.00	1.00	1.00	15
31	1.00	1.00	1.00	4
32	1.00	1.00	1.00	2
34	1.00	1.00	1.00	2
35	1.00	1.00	1.00	15
36	1.00	1.00	1.00	4
37	1.00	1.00	1.00	6
38	1.00	0.67	0.80	3
39	0.75	1.00	0.86	3
4	1.00	1.00	1.00	10
40	1.00	1.00	1.00	3
41	1.00	1.00	1.00	2
42	1.00	1.00	1.00	3
43	1.00	1.00	1.00	8
44	1.00	1.00	1.00	3
45	1.00	1.00	1.00	2

46	1.00	1.00	1.00	2
47	1.00	1.00	1.00	1
48	1.00	1.00	1.00	1
49	1.00	1.00	1.00	4
5	1.00	1.00	1.00	19
50	1.00	1.00	1.00	6
52	1.00	1.00	1.00	4
54	1.00	1.00	1.00	33
55	1.00	1.00	1.00	17
56	1.00	1.00	1.00	11
6	1.00	1.00	1.00	8
7	1.00	1.00	1.00	15

And the confusion matrix is found to be →

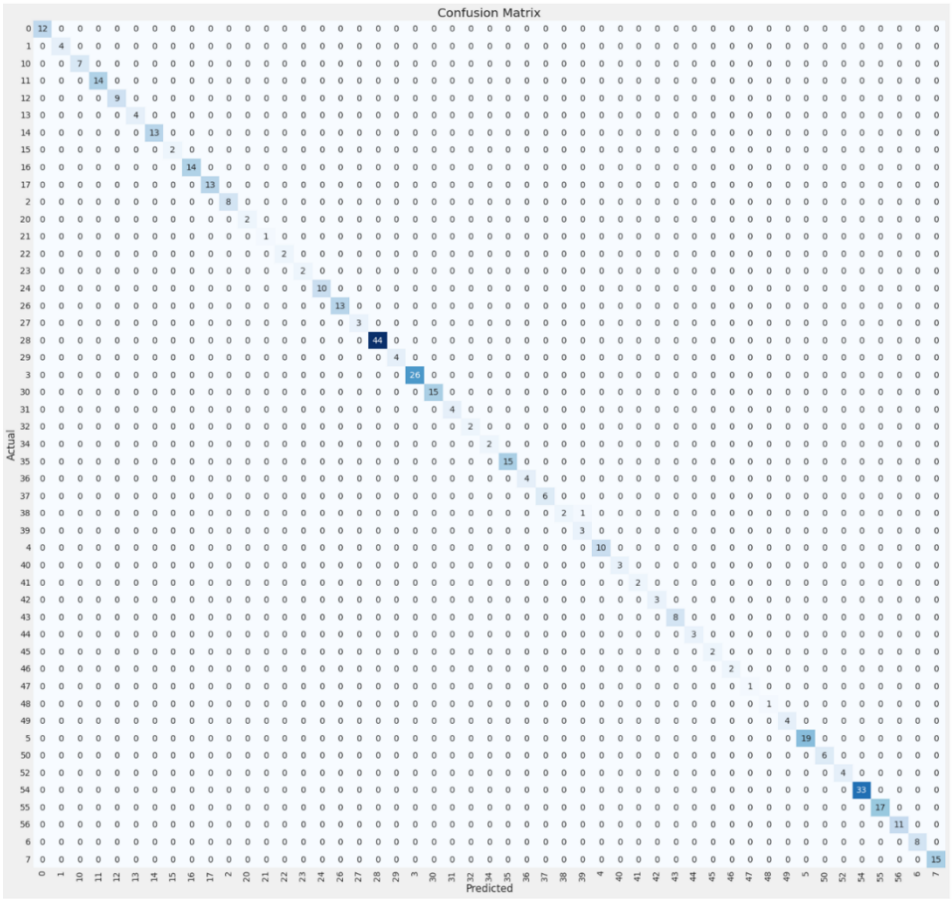


Figure: Confusion matrix for model_1

model_2[11]:

The second implementation of our dataset is on a model created by Hannah Morgan.

Folders	precision	recall	f1-score	support
0	1.00	1.00	1.00	59
1	1.00	1.00	1.00	20
2	1.00	1.00	1.00	35
3	1.00	1.00	1.00	69
4	1.00	1.00	1.00	48
5	1.00	1.00	1.00	18
6	1.00	0.98	0.99	64
7	1.00	1.00	1.00	11
8	1.00	1.00	1.00	71
9	1.00	1.00	1.00	65
10	1.00	1.00	1.00	4
11	1.00	1.00	1.00	2
12	1.00	1.00	1.00	40
13	1.00	1.00	1.00	9
14	1.00	1.00	1.00	6
15	1.00	1.00	1.00	9
16	1.00	1.00	1.00	7
17	1.00	1.00	1.00	50
18	1.00	1.00	1.00	1
19	1.00	1.00	1.00	63
20	1.00	1.00	1.00	14
21	1.00	1.00	1.00	223
22	1.00	1.00	1.00	22
23	1.00	0.99	1.00	130
24	1.00	1.00	1.00	75
25	1.00	1.00	1.00	21
26	1.00	1.00	1.00	7
27	1.00	1.00	1.00	2
28	1.00	1.00	1.00	13
29	1.00	1.00	1.00	78
30	1.00	1.00	1.00	20
31	1.00	1.00	1.00	29
32	0.94	1.00	0.97	15
33	1.00	0.94	0.97	17
34	1.00	1.00	1.00	49
35	1.00	1.00	1.00	16
36	1.00	1.00	1.00	9
37	1.00	1.00	1.00	16
38	0.95	1.00	0.98	41
39	1.00	0.87	0.93	15
40	1.00	1.00	1.00	12
41	1.00	1.00	1.00	9
42	1.00	1.00	1.00	6

43	1.00	1.00	1.00	5
44	1.00	1.00	1.00	21
45	1.00	0.67	0.80	97
46	1.00	1.00	1.00	28
47	1.00	1.00	1.00	4
48	1.00	1.00	1.00	18
49	1.00	1.00	1.00	1
50	1.00	1.00	1.00	162
51	1.00	1.00	1.00	81
52	1.00	1.00	1.00	55
53	1.00	1.00	1.00	3
54	1.00	1.00	1.00	39
55	0.69	1.00	0.82	76
56	1.00	1.00	1.00	4
57	1.00	1.00	1.00	1

The confusion matrix is found to be →

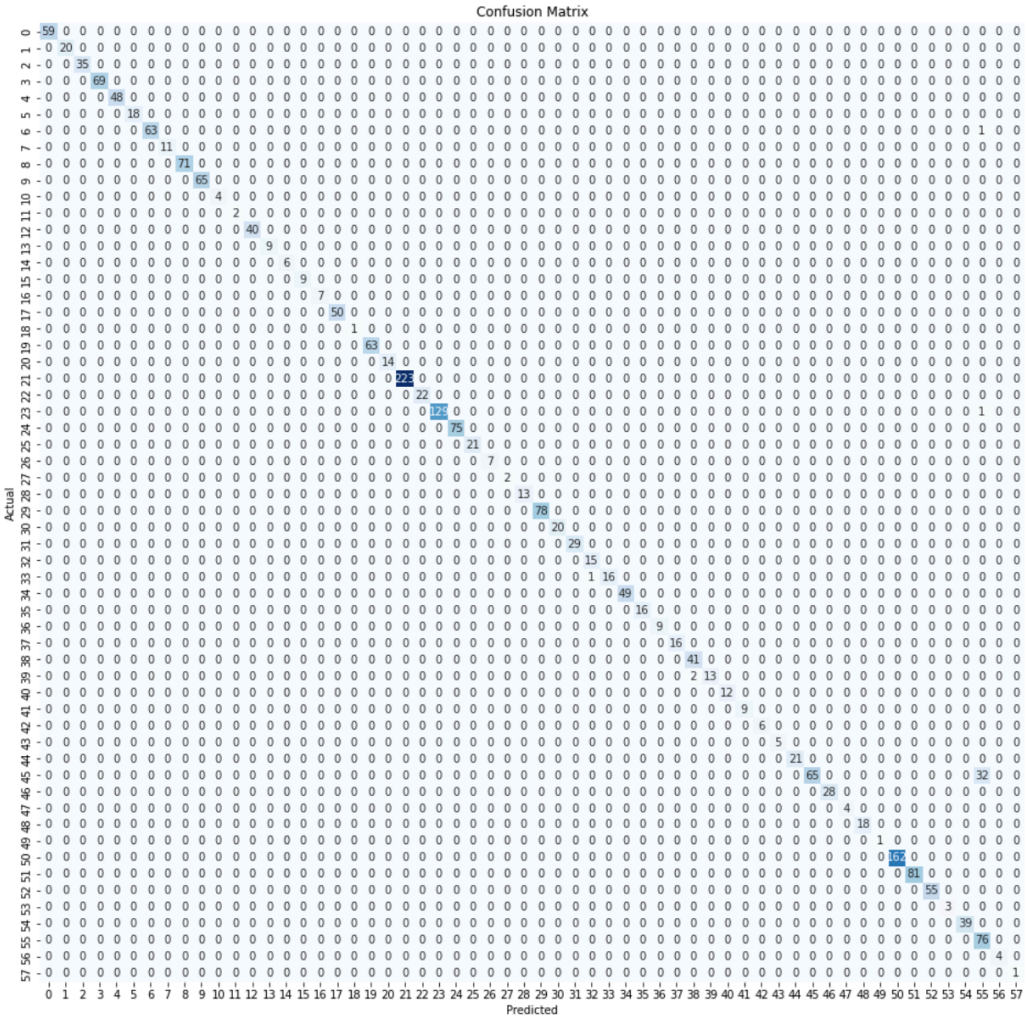


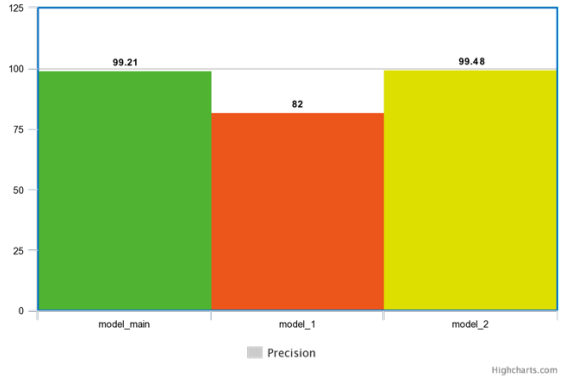
Figure: Confusion matrix for model_2

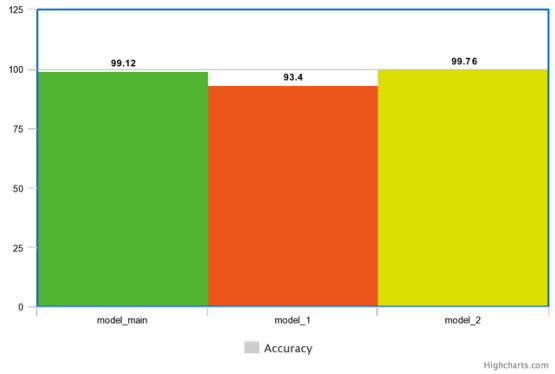
Results

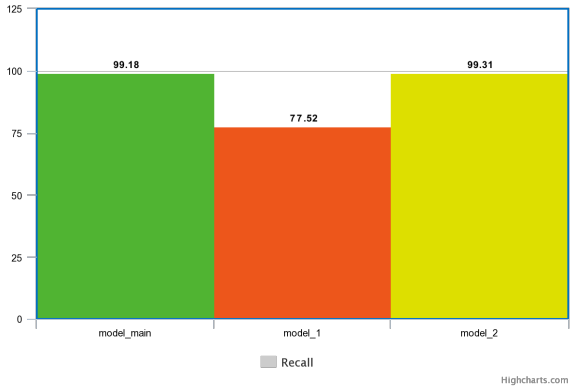
We implemented two existing models for the purpose of comparing them to our proposed model. The below Table1 compares our proposed model (model_main) to the existing models- model_1 and model_2, using our chosen dataset – Traffic Sign Dataset – Classification[12]

Table1:-
Comparison of Different models on our dataset (Dataset_main):

Dataset_main	Precision (%)	Accuracy (%)	Recall (%)
model_main	99.21	99.12	99.18
model_1	82.00	93.40	77.52
model_2	99.48	99.76	99.31





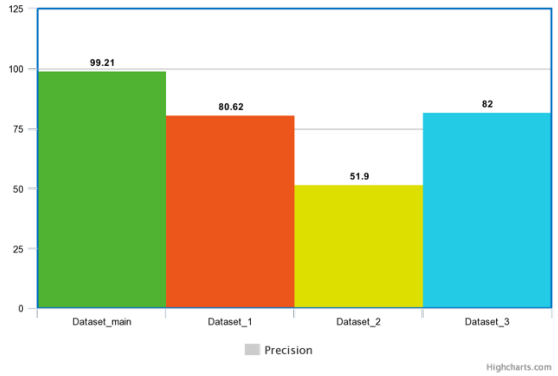


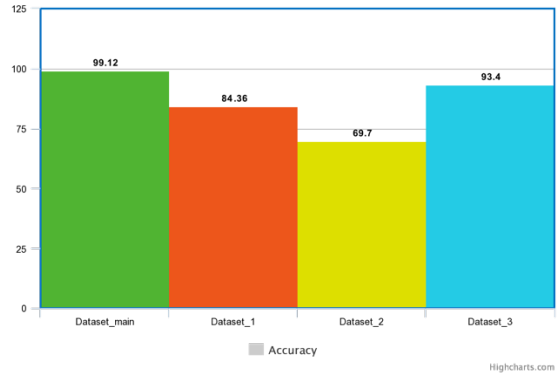
Now using our proposed model, we implement different datasets of different sizes to compare the results and the difference it makes on its precision, accuracy and recall.
Dataset_main: – Traffic Sign Dataset – Classification[12]
Dataset_1: Indian Traffic sign Dataset[13]
Dataset_2: Traffic Sign classification[14]
Dataset_3: German Traffic Sign Recognition Benchmark GTSRB[15]

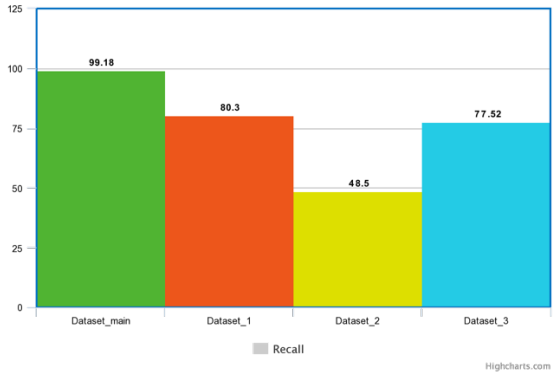
The below table2 compares the given conditions based on performance measures- precision, accuracy and recall

Table2:-
Comparison of different datasets on our Model (model_main):

model_main	Precision (%)	Accuracy (%)	Recall (%)
Dataset_main	99.21	99.12	99.18
Dataset_1	80.62	84.36	80.30
Dataset_2	51.90	69.70	48.50
Dataset_3	82.00	93.40	77.52







Conclusion and Future work

This study aimed in determining the causes of road accidents because of lack of awareness in terms of looking for important traffic signs and not knowing what traffic sign means. This study also provided an efficient solution to casualties and deaths happening due to road accidents. The proposed model, which is a model based on CNN, is implemented on German traffic sign dataset, Indian traffic sign dataset, and hybrid dataset, which results in an accuracy of (need to be changed 99.12%, 93.40%, 99.06%) respectively. The proposed model notably achieves higher accuracy on all datasets as compared to any other implemented models. The proposed model is robust since high accuracy was achieved by the model, and 102 traffic sign dataset was used that enabled us to accommodate wide range. Model epoch vs. accuracy/loss for hybrid dataset. of signs, which made the system reliable for a user. The proposed model takes us one step closer to significantly lesser accidents and making roads safer to drive.

In the future we would like to streamline and automate the process of the region of the image in the bounding box to be passed to the CNN for classification, instead of doing it manually. We would also train the CNN for more epochs so as to improve the accuracy of the model and eliminate many more false predictions. Also, we could use autoencoders as attention networks before passing the input images through the YOLO model. Autoencoders are unsupervised neural networks, whose objective is to learn a set of weights, with target value as the input

itself. It can be used to learn structures in inputs. This property of autoencoders can be useful to us as we can train this network to learn and recognize just the traffic signs. The entire image with the background can be passed through this trained autoencoder and hope that it would give a differential response towards the traffic signs when compared to other objects in the background. This would help in recognizing the required objects and leave out the others (e.g. cars in the background etc.) This output of the autoencoder could be given as the input to the YOLO network, instead of just the image and it has shown that this helps.

References

- Munoz-Organero, Mario, Ramona Ruiz-Blaquez, and Luis Sánchez-Fernández. "Automatic detection of traffic lights, street crossings and urban roundabouts combining outlier detection and deep learning classification techniques based on GPS traces while driving." *Computers, Environment and Urban Systems* 68 (2018): 1-8. https://www.sciencedirect.com/science/article/pii/S0198971517301096

- Arinaldi, Ahmad, Jaka Arya Pradana, and Arlan Arventa Gurusinga. "Detection and classification of vehicles for traffic video analytics." *Procedia computer science* 144 (2018): 259-268. https://www.sciencedirect.com/science/article/pii/S1877050918322361

- Kurniawan, Jason, Sensa GS Syahra, and Chandra K. Dewa. "Traffic congestion detection: learning from CCTV monitoring images using convolutional neural network." *Procedia computer science* 144 (2018): 291-297. https://www.sciencedirect.com/science/article/pii/S1877050918322397

- Yu, Changhe, et al. "QoS-aware traffic classification architecture using machine learning and deep packet inspection in SDNs." *Procedia computer science* 131 (2018): 1209-1216. https://www.sciencedirect.com/science/article/pii/S1877050918307129

- Kryvinska, Natalia, Aneta Poniszewska-Maranda, and Michal Gregus. "An approach towards service system building for road traffic signs detection and recognition." *Procedia Computer Science* 141 (2018): 64-71. https://www.sciencedirect.com/science/article/pii/S1877050918318003

- Sheikh, Muhammad Sameer, and Amelia Regan. "A complex network analysis approach for estimation and detection of traffic incidents based on independent component analysis." *Physica A: Statistical Mechanics and its Applications* 586 (2022): 126504. https://www.sciencedirect.com/science/article/pii/S0378437121007779

- Ali, MD Hazrat, Syuhei Kurokawa, and A. A. Shafie. "Autonomous road surveillance system: A proposed model for vehicle detection and traffic signal control." *Procedia Computer Science* 19 (2013): 963-970. https://www.sciencedirect.com/science/article/pii/S1877050913007424

- Laguna, Rubén, et al. "Traffic sign recognition application based on image processing techniques." *IFAC Proceedings Volumes* 47.3 (2014): 104-109. https://www.sciencedirect.com/science/article/pii/S1474667016416009

- Tabernik, Domen, and Danijel Skočaj. "Deep learning for large-scale traffic-sign detection and recognition." *IEEE transactions on intelligent transportation systems* 21.4 (2019): 1427-1440.

https://www.researchgate.net/publication/332140267_Deep_Learning_for_Large-Scale_Traffic-Sign_Detection_and_Recognition

- Pon, Alex, et al. "A hierarchical deep architecture and mini-batch selection method for joint traffic sign and light detection." *2018 15th Conference on Computer and Robot Vision (CRV)*. IEEE, 2018. https://www.researchgate.net/publication/329747574_A_Hierarchical_Deep_Architecture_and_Mini-batch_Selection_Method_for_Joint_Traffic_Sign_and_Light_Detection

- Latif, Ghazanfar, et al. "Deep convolutional neural network for recognition of unified multi-language handwritten numerals." *2018 IEEE 2nd International workshop on Arabic and derived script analysis and recognition (ASAR)*. IEEE, 2018. https://www.researchgate.net/publication/328082730_Deep_Convolutional_Neural_Network_for_Recognition_of_Unified_Multi-Language_Handwritten_Numerals

- Wan, Haifeng, et al. "A novel neural network model for traffic sign detection and recognition under extreme conditions." *Journal of Sensors* 2021 (2021). https://www.researchgate.net/publication/353173547_A_Novel_Neural_Network_Model_for_Traffic_Sign_Detection_and_Recognition_under_Extreme_Conditions

- Ellahyani, Ayoub, Ilyas El Jaafari, and Said Charfi. "Traffic sign detection for intelligent transportation systems: a survey." *E3S web of conferences*. Vol. 229. EDP Sciences, 2021. https://www.researchgate.net/publication/348749082_Traffic_Sign_Detection_for_Intelligent_Transportation_Systems_A_Survey

- Wu, Yihui, et al. "Traffic sign detection based on convolutional neural networks." *The 2013 international joint conference on neural networks (IJCNN)*. IEEE, 2013. https://www.researchgate.net/publication/261148216_Traffic_sign_detection_based_on_convolutional_neural_networks

- Houben, S., et al. "Proposal for IJCNN 2013 competition: the German traffic sign detection benchmark." *Proceedings of the IEEE Intelligent Vehicles Symposium*. 2013. https://www.researchgate.net/publication/242346620_Proposal_for_IJCNN_2013_competition_The_German_Traffic_Sign_Detection_Benchmark

- Yao, Yingbiao, et al. "Traffic sign detection algorithm based on improved YOLOv4-Tiny." *Signal Processing: Image Communication* 107 (2022): 116783. https://www.sciencedirect.com/science/article/abs/pii/S0923596522000960

- Rodríguez, Rúben Castruita, et al. "Mexican traffic sign detection and classification using deep learning." *Expert Systems with Applications* 202 (2022): 117247. https://www.sciencedirect.com/science/article/abs/pii/S0957417422006236

- Kassani, Peyman Hosseinzadeh, and Andrew Beng Jin Teoh. "A new sparse model for traffic sign classification using soft histogram of oriented gradients." *Applied Soft Computing* 52 (2017): 231-246 https://www.sciencedirect.com/science/article/abs/pii/S1568494616306627#!
.
- Ouyang, Zhenchao, et al. "MBBNet: An edge IoT computing-based traffic light detection solution for autonomous bus." *Journal of Systems Architecture* 109 (2020): 101835. https://www.sciencedirect.com/science/article/abs/pii/S1383762120301272

- Jiang, Feifeng, Kwok Kit Richard Yuen, and Eric Wai Ming Lee. "A long short-term memory-based framework for crash detection on freeways with traffic data of different temporal resolutions." *Accident Analysis & Prevention* 141 (2020): 105520.

https://www.sciencedirect.com/science/article/pii/S0001457519317713

- Wang, Shuihua, et al. "RGB-D image-based detection of stairs, pedestrian crosswalks and traffic signs." *Journal of Visual Communication and Image Representation* 25.2 (2014): 263-272. https://www.sciencedirect.com/science/article/abs/pii/S1047320313001934

- Arcos-Garcia, Alvaro, et al. "Exploiting synergies of mobile mapping sensors and deep learning for traffic sign recognition systems." *Expert Systems with Applications* 89 (2017): 286-295. https://www.sciencedirect.com/science/article/abs/pii/S0957417417305195

- Shustanov, Alexander, and Pavel Yakimov. "CNN design for real-time traffic sign recognition." *Procedia engineering* 201 (2017): 718-725. https://www.sciencedirect.com/science/article/pii/S1877705817341231

- Liu, Huaping, Yulong Liu, and Fuchun Sun. "Traffic sign recognition using group sparse coding." *Information Sciences* 266 (2014): 75-89. https://www.sciencedirect.com/science/article/abs/pii/S002002551400022X

- Megalingam, Rajesh Kannan, et al. "Indian traffic sign detection and recognition using deep learning." *International Journal of Transportation Science and Technology* (2022). https://www.sciencedirect.com/science/article/pii/S2046043022000557

Dataset_1 : https://www.kaggle.com/datasets/neelpratiksha/indian-traffic-sign-dataset
Dataset_2 : https://www.kaggle.com/datasets/srthk5/traffic-sign-classification
Dataset_3: https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/published-archive.html