A
Project Phase-II Report
On
Traffic Sign Board Recognition And Voice Alert Using CNN

By

Nehete Vinit Y.
Patil Yadnesh B.
Chavan Gaurav M.
Sonar Mihir K.

Department of Computer Engineering

The Shirpur Education Society's
R. C. Patel Institute of Technology, Shirpur - 425405

[2022-23]

A
Project Phase-II Report
On
Traffic Sign Board Recognition And Voice Alert Using CNN

In partial fulfillment of requirement for the degree of

Bachelor of Engineering

In

Computer Engineering

Submitted By

Nehete Vinit Y.
Patil Yadnesh B.
Chavan Gaurav M.
Sonar Mihir K.

Under the Guidance of

Prof. Ms. P. A. Agrawal

The Shirpur Education Society's
R. C. Patel Institute of Technology, Shirpur - 425405

[2022-23]

The Shirpur Education Society's

R. C. Patel Institute of Technology, Shirpur, Dist. Dhule (M.S)

Department of Computer Engineering

CERTIFICATE

This is to certify that the project entitled **"Traffic Sign Board Recognition And Voice Alert Using CNN"** has been carried out by team:

Nehete Vinit Y.(1951721245065)
Patil Yadnesh B.(1951721245060)
Chavan Gaurav M.(1951721245015)
Sonar Mihir K. (19517212450028)

under the guidance of **Prof Ms. P. A. Agrawal** in partial fulfillment of the requirement for the degree of Bachelor of Engineering in Computer Engineering of Dr. Babasaheb Ambedkar Technological University,Lonere during the academic year 2022-23 (Semester-VIII).

Date:

Place: Shirpur

Guide
Prof. Ms. P. A. Agrawal

Project Phase-II Coordinator
Prof. Dr. S. S. Sonawane

H.O.D
Prof. Dr. Nitin N. Patil

External Examiner

Director
Prof. Dr. J. B. Patil

Acknowledgment

No volume of words is enough to express our gratitude towards our guide, **Prof. Ms. P. A. Agrawal**, Associate Professor in Computer Engineering Department, who has been very concerned and have aided for all the material essential for the preparation of this work. She has helped us to explore this vast topic in an organized manner and provided us with all the ideas on how to work towards a research-oriented venture.

We express our gratitude to **Prof. Dr. N. N. Patil**, Head of Department of Computer & Engineering for his constant encouragement, co-operation, and support.

Lastly, We would like to express our gratitude to the Director of R.C. Patel Institute Of Technology , **Prof. Dr. J. B. Patil**, for their leadership and commitment

Nehete Vinit Y.
Patil Yadnesh B.
Chavan Gaurav M.
Sonar Mihir K.

Abstract

Road signs are essential for ensuring a safe and smooth flow of traffic. Negligence in viewing traffic signs and incorrectly interpreting them is a major cause of road accidents. The proposed system is trained using a Convolutional Neural Network (CNN), which aids in the recognition and classification of traffic signs. To improve accuracy, a set of classes is defined and trained on a specific dataset. Driving could be an advanced, continuous, and multitask method that involves the driver's noesis, perception, and motor movements. The approach to road traffic signs and vehicle data is displayed impacts powerfully driver's attention with enlarged mental work resulting in safety considerations. Drivers should keep their eyes on the road, however, will continually use some help in maintaining their awareness and guiding their attention to potential rising hazards. Experiments results show that deep convolutional neural networks show that joint learning greatly enhances the potential of detection and still retains its real-time performance. Investigations on vision-based TSDR have received substantial interest within the analysis community that is especially impelled by 3 factors, that area unit detection, trailing, and classification.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

There have been a lot of technological advancements and cars with auto-pilot modes have come up. Autonomous vehicles have come into existence. There has been a boom in the self-driving car industry. However, these features are available only in some cars which are not affordable to the masses. We wanted to devise a system that helps in easing the job of driving to some extent. In conducting a survey, we found that the magnitude of road accidents in India is alarming. Reports suggest that every hour there are about 53 mishaps taking place on the roads. Moreover, every hour more than 16 deaths occur due to these mishaps. When someone neglects to obey traffic signs while driving, they are putting their lives as well as the life of the other drivers, their passengers, and those on the road at risk. Hence, we came up with this system in which traffic signs are automatically detected and are read out aloud to the driver who may then take the required decision. In this era of a fast-paced life, people generally tend to miss out on recognizing the traffic sign and hence break the rules. A lot of research has been done in this domain in order to reduce the number of accidents. Researchers have used a variety of classification algorithms and a number of CNN architectures to classify traffic signs and alert the driver. Our system aims to optimize the process of recognition and at the same time provide other benefits such

as an early alert to the driver The detection of traffic signs has been done in a variety of techniques in numerous studies signs are then categorized as hexagonal, triangular, or circular in shape and transmitted for template matching after these operations.

1.2 Motivation

In the past and recent times, there have been many road accidents and the main reason for these is inadequate knowledge of road and traffic signs. Even though speed is one of the key issues for the cause of such atrocities, in a survey, it was found that the second most heard reason was an individual not knowing what a particular traffic sign meant. We strongly believe that the research that we have done would help individuals learn these signs intuitively, especially the adolescence of the 21st century, who also stay and live around technology, which is growing faster than ever. Our research focuses on detecting traffic signs when provided an image to it through CNN.

1.3 Problem Statement

There is a huge number of increasing road accidents across the world day by day. These may be due to ignorance of Traffic signs. The main goal of the system is to produce voice alerts to the driver, so it becomes useful for the safety of drivers as well as pedestrians.

1.4 Objectives

1. To reduce the number of accidents.
2. To produce a voice alert to the driver and then give a sign of upcoming traffic signs.

3. To detect the images and predict the signal.

4. The focused will be placed on designing a system that will accurately monitor the traffic sign for upcoming signal .

5. The objective is develop a system for traffic sign recognition .

1.5 Scope

The creation of different classes for each Traffic sign has helped in increasing the accuracy of the model. A voice message is sent after recognition of the sign which alerts the driver. A map is displayed on which the signs in the vicinity of the driver are displayed thus helping him/her take appropriate decisions. This paper is a significant advancement in the field of driving as it would ease the job of the driver without compromising on the safety aspect. Also, this system can easily be implemented without the need for much hardware thus increasing its reach.

Chapter 2

Literature Survey

2.1 Review of Existing System

In this era of fast-paced life, people generally tend to miss out on recognizing traffic signs and hence break the rules. A lot of research has been done in this domain to reduce the number of accidents. Researchers have used a variety of classification algorithms and several CNN architectures to classify traffic signs and alert the driver. Our system aims to optimize the process of recognition and, at the same time, provide other benefits such as an early alert to the driver.

Yadav, Shubham, Patwa, Anuj, Rane, Saiprasad, Narvekar, Chhaya (2019) proposed an Indian Traffic Sign Board Recognition and Driver Alert System Using Machine Learning. The authors employed the Support Vector Machine technique with a dataset divided into 90/10 for training and testing purposes. The process involved phases like Color Segmentation, Shape Classification, and Recognition [1].

Chuanwei Zhang et al. conducted a study on Traffic Sign Recognition by Optimized Lenet-5 Algorithm. They suggested a traffic sign recognition method based on an improved Lenet-5 network. The improved Lenet-5 classifier outperformed other classifiers in terms of accuracy and real-time performance [9].

Y. Yuan, Z. Xiong, Q. Wang (2019) proposed VSSA-NET, a Vertical Spatial Sequence Attention Network for Traffic Sign Detection. Their method utilized a multi-resolution feature fusion network architecture and a vertical spatial sequence attention module for improved detection. They also incorporated GPS-based tracking

and Augmented Reality technology in mobile apps [4].

Saha S., Islam M.S., Khaled M.A.B., Tairin S. (2019) presented an efficient traffic sign recognition approach using a novel deep neural network selection architecture. They utilized the coordinates of a user's smartphone as a pointer to assist in locating possible resources based on the user's camera view [6].

2.2 Limitation of Existing System

One of the limitations of the CNN model is that it requires images with the same dimensions for training. Therefore, it is mandatory to have images in the dataset with similar dimensions. It is essential to check the dimension of all the images in the dataset to process them into having uniform dimensions.

Chapter 3

Requirement Analysis

3.1 Method Used For Requirement Analysis

3.1.1 Convolutional Neural Network

Convolutional neural networks(CNN) are one of the most popular models used today. This neural network computational model uses a variation of multilayer perceptron's and contains one or more convolutional layers that can be either entirely connected or pooled. These convolutional layers create feature maps that record a region of the image which is ultimately broken into rectangles and sent out for nonlinear processing.

Advantages:

1. Very High accuracy in image recognition problems.
2. Automatically detects important features without any human supervision.
3. Reduction in road accidents will be one of the main advantages.

Disadvantages:

1. CNN does not encode the position and orientation of the object.

2. Lack of ability to be spatially invariant to the input data.

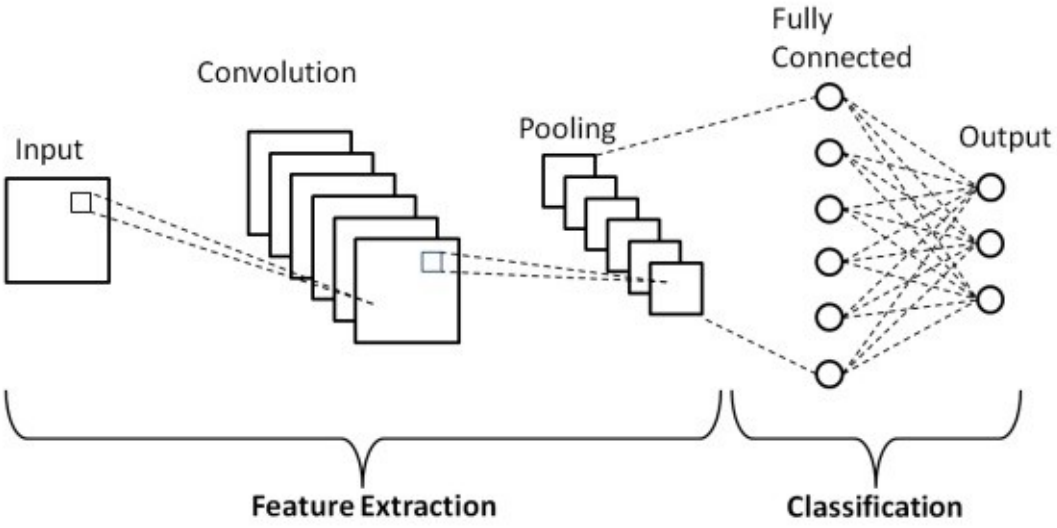

Figure 3.1: Schematic diagram of a basic Convolutional neural network (CNN) architecture

3.2 Data Requirements

In the proposed system, the German Traffic Sign Benchmarks (GTSRB) Dataset is used. Fig. 1 shows the 43 different traffic signs that are considered to train the model. It has 51,900 single images distributed among the 43 classes including the training and the test dataset. The count of the number of photos per class is shown in Fig. 2. There is no ambiguity as the images are just focused on the traffic signs and each of them is unique. The training dataset has different folders for each of the present classes.

Figure 3.2: Traffic Signs Taken into consideration

3.3 Functional Requirements

There is a huge number of increasing road accidents across the world day by day. These may be due to ignorance of Traffic signs. The main goal of the system is to produce voice alerts to the driver, so it becomes useful for the safety of drivers as well as pedestrians [11] .

1. The system dataset is imported for training the dataset using the TensorFlow library for training the model by Installing its version suitable for the model.
2. The 'NumPy' is used for converting Images Pinto array for processing. This array are we to train and test purpose.
3. Created an object for different signals sign .
4. Correctly able to detect signals png / jpeg format images The system must be able. to detect the actual images with the given images.
5. They will be viewed by showing the accuracy after detecting images properly.

3.4 System Requirements

Hardware Requirements:

- System: Intel i5 processor
- Hard Disk: - 500 GB
- Monitor: 15.6 inch
- RAM - 4GB
- Input Devices: - Keyboard, Mouse

Software Requirements:

- Operating System: - Windows 7 onward
- Coding language: - Python
- Web Framework - Flask

Chapter 4

Planning And Scheduling

.

4.1 Project Planning

We have Planned the machine learning Model Outline and the user requirements and user Expectations.And we implemented the planned model we divided the modules into proper schedules in order to meet the deadline.

- Goal: The goal of this system is to ensure the safety of the vehicle's driver, passengers, and pedestrians.
- Final Deadline:
- Steps/Task :
  - Collection of Data sets
  - Model Exploration
  - Model refinement
  - Testing and Evaluation
  - Model Deployment
- Task for Team Members:
  - Collection of Data Set

- Software Building (Coding)
- Information Collection about Project

4.2 Project Scheduling(Cost Efforts)

As it is a Machine learning project cost, we estimated electricity and internet Connection. Also, we don't have to pay for the dataset because it is available on the internet. We planned to spend hours per day to discuss the implementation of project to complete it before the actual deadline

4.3 Gantt Chart

The Gantt chart provides an overview of the project timeline and the associated tasks. The chart is divided into several sections, each representing a specific task or milestone.

The project includes the following steps/tasks:

- Collection of Data sets (Start: [22/10/2022], End: [29/10/2022]): This involves gathering relevant datasets for training the model.
- Model Exploration (Start: [5/11/2022], End: [15/11/2022]): This step focuses on researching and exploring different models and algorithms suitable for traffic light recognition.
- Model refinement (Start: [21/11/2022], End: [15/01/2023]): Once a model is selected, it will be refined and optimized for better performance.
- Testing and Evaluation (Start: [8/04/2023]], End: [29/04/2023]): The model will be tested using various datasets to assess its accuracy and effectiveness.
- Model Deployment (Start: [1/05/2023], End: [13/05/2023]): After successful testing, the trained model will be deployed to a production environment for real-time traffic light recognition.

The team members are assigned specific tasks:

- Collection of Data Set (Start: [22/10/2022], End: [29/10/2022]): Team member A will be responsible for gathering the necessary datasets.
- Software Building (Coding) (Start: [5/11/2022], End: [29/04/2023]): Team member B will handle the implementation of the machine learning model using Python and CNN.
- Information Collection about Project (Start: [5/05/2023], End: [10/05/2023]): Team member C will gather relevant information and research materials related to the project.

The Gantt chart provides a visual representation of the project timeline, allowing the team to track progress and ensure timely completion of each task.

Chapter 5

Software Requirements Specification

5.1 Design Details

For image detection, first, we need to build a model using a Machine learning CNN algorithm. For building this model, we have taken a dataset called the German Traffic Sign Benchmarks (GTSRB) dataset. This dataset has 51,900 single images distributed in 43 classes. After taking this dataset, we will train the dataset by using CNN techniques. Before performing some preprocessing like changing the size of the image pixels, etc. After preprocessing, we train the dataset.

5.2 Data Flow Diagrams (Levels 0, 1, 2)

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both. It shows how data enters and leaves the system, what changes the information, and where data is stored. A graphical tool for defining and analysing minute data with an active or automated system, including process, data stores, and system delays. Data Flow Data is a key and basic tool for the architecture of all other objects. Bubble-bubble or data flow graph is another name for DFD. DFDs are a model of the

proposed system. They should indicate the requirements on which the new system should be built in a clear and direct manner. This is used as a basis for building chart structure plans over time during the design process. DFD is presented in hierarchical fashion that is rest data flow model represents the system as a whole subsequent DFD the context diagram (Level 0 DFD), providing increasing details with each subsequent level.

5.2.1 Data Flow Diagram Level 0

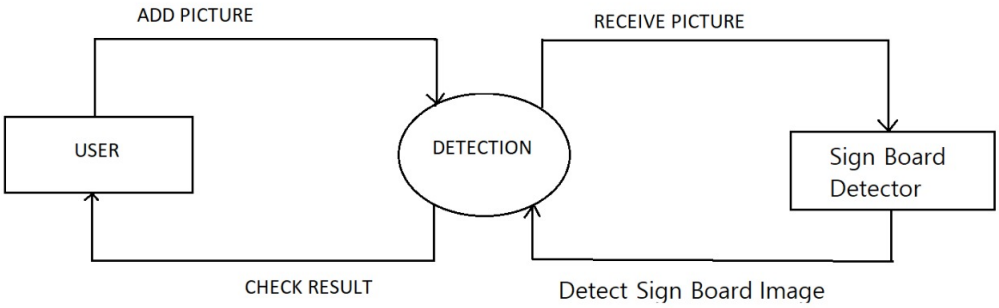

Figure 5.1: Data Flow Diagram Level 0 For Traffic sign recognition and voice alert using CNN.

5.2.2 Data Flow Diagram Level 1

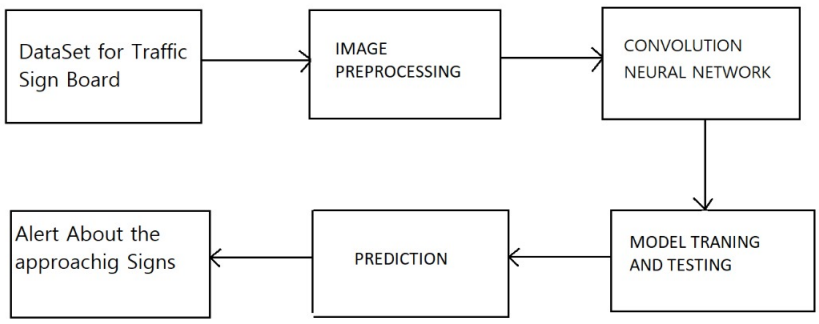

Figure 5.2: Data Flow Diagram Level 1 For Traffic sign recognition and voice alert using CNN.

5.2.3 Data Flow Diagram Level 2

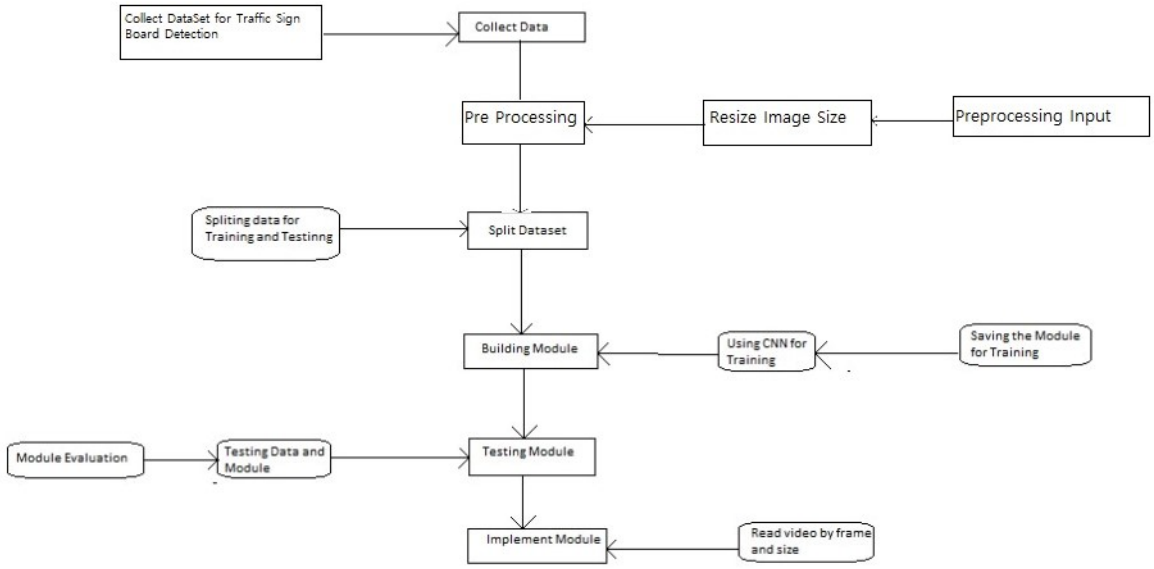

Figure 5.3: Data Flow Diagram Level 2 For Traffic sign recognition and voice alert using CNN.

Chapter 6

System Modeling- Need Of System Modeling

6.1 UML Diagram

6.1.1 Use Case Diagrams

A use case defines the behavioral features of a system. Each use case is named using a verb phrase that expresses a goal of the system. A use case diagram shows a set of use cases and actors &their relationships. Use case diagrams address the static use case view of a system. These diagrams are especially important in organizing and modeling the behaviors of a system. It shows the graphical overview of the functionality provided by the system intents actor.

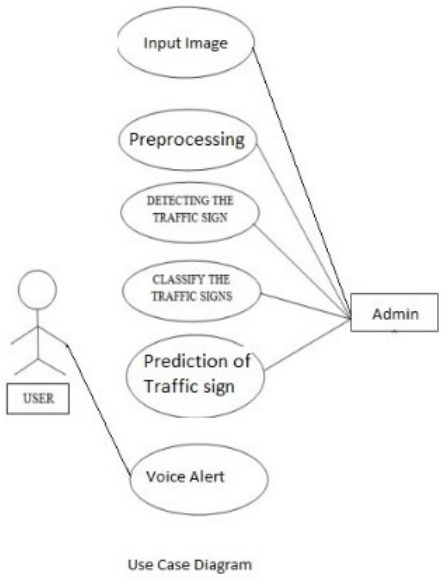

Figure 6.1: Use Case Diagram for Traffic sign recognition and voice alert using CNN.

6.1.2 Activity Diagram

An activity diagram of a special kind of state chart diagram that shows the flow from activity within a system. An activity addresses the dynamic view of a system. The activity diagram is often seen as part of the functional view of a system because it describes logical processes or functions. Each process describes a sequence of tasks and the decisions that govern when and they are performed. The flow in an activity diagram is driven by the completion of an action.

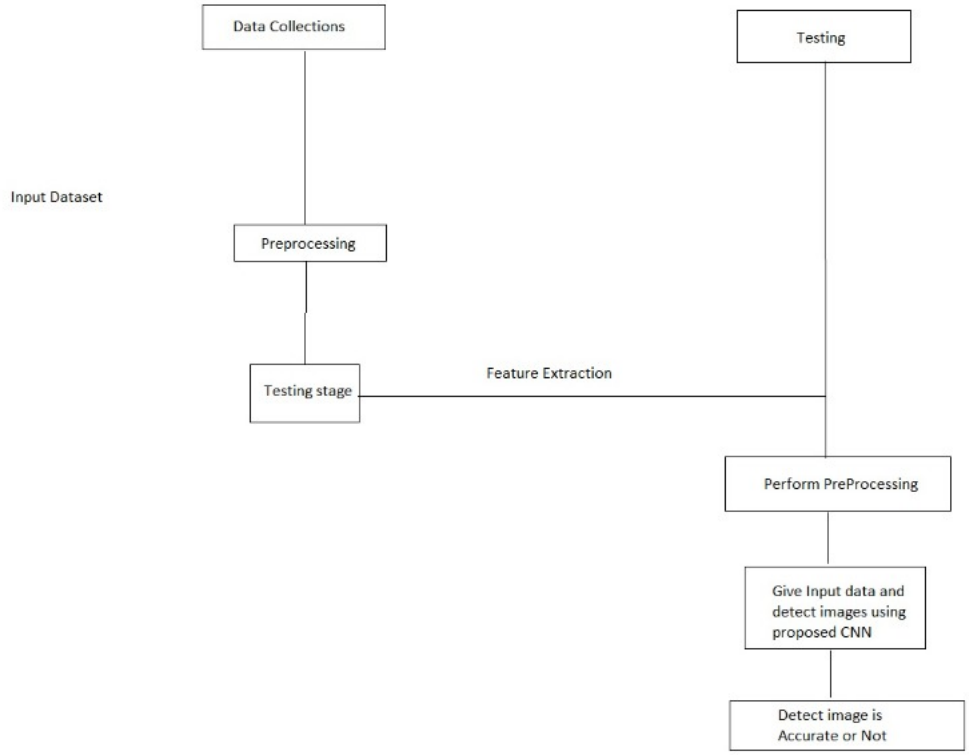

Figure 6.2: Activity Diagram for Traffic sign recognition and voice alert using CNN.

6.1.3 Sequence Diagram

A sequence diagram is a type of diagram that shows the interactions between objects or components in a system, along with the sequence of messages that are exchanged between them. Sequence diagrams are often used in software engineering to visualize the flow of messages and events within a system, and to help designers and developers understand how the different components of a system interact with each other.

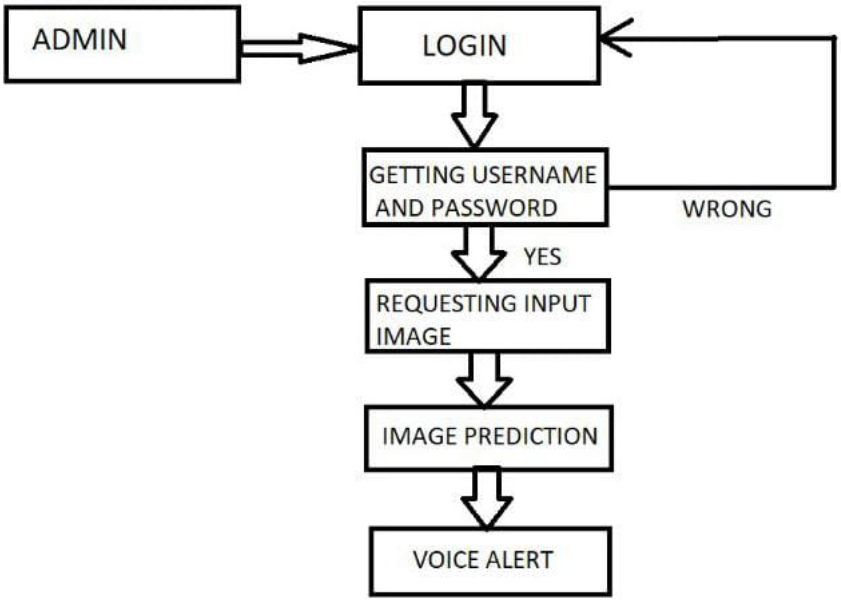

Figure 6.3: Sequence Diagram for Traffic sign recognition and voice alert using CNN.

6.1.4 Class Diagram

A class diagram shows a set of classes, interfaces, and collaborations and their relationship. These diagrams are the most common diagram found in modeling object-oriented systems. Class diagrams addressed the static design view of a system.

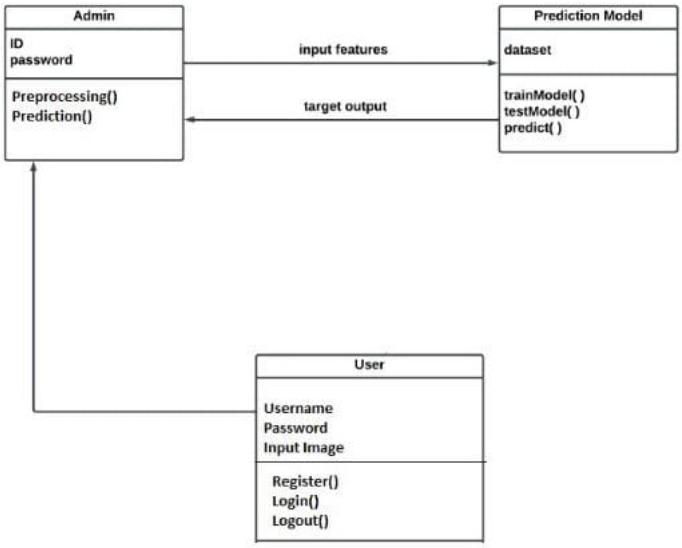

Figure 6.4: Class Diagram for Traffic sign recognition and voice alert using CNN.

Chapter 7

Implementation

7.1 Hardware Specification

For the development of this system we needd a laptop or a desktop with following specifications

- Processor: Intel i5 processor
- Hard Disk: - 500 GB
- Monitor: 15.6 inch
- RAM :- 4GB
- Internet Access: - For updation and activation
- Graphic card:- 2Gb

7.2 Platform

During the development of process we are gone through various platform for each of the phase , for research purpose we use the browser. For the code implementation we

use juypter . For backend we use the python version 3.9.7 . And for the frontend we use python flask webframework.

7.3 Programming Languagae used

For the frontend implementation we use the python webframework flask , which allow to give the user interface. With flask we route the html pages to connect the backend. The backend implementation we use python 3.9.7 . Python is used to train and predict the output.

7.4 System Implementation

1. **Dataset Preparation** :
    - Gather a dataset of traffic sign images with corresponding labels.
    - Preprocess the images by resizing them to a consistent size and applying any necessary normalization or augmentation techniques.
    - Model refinement
    - Testing and Evaluation
    - Model Deployment
2. **CNN Model Training :**
    - Implement a Convolutional Neural Network (CNN) model using Python and a deep learning library such as TensorFlow.
    - Split the dataset into training and validation sets.
    - Train the CNN model on the CNN architecture.

3. **Voice Alert System :**

   - Install required libraries like SpeechRecognition to handle voice recognition and synthesis.
   - Design and implement a voice alert system that can generate voice alerts for different traffic sign classes.
   - Create a mapping between traffic sign labels and corresponding voice alerts to be generated.

4. **BackEnd Development :**

   - Set up a Python Flask server as the backend for your application.
   - Create an API endpoint that receives an image for traffic sign recognition.
   - Use the trained CNN model to predict the traffic sign from the received image.
   - Generate the corresponding voice alert based on the predicted traffic sign class.
   - Return the predicted traffic sign label and voice alert as a response from the API.

5. **FrontEnd Development :**

   - Design a user interface using HTML, CSS, and JavaScript to interact with the backend.
   - Create a form or file input element to allow users to upload images of traffic signs.
   - Use the trained CNN model to predict the traffic sign from the received image.
   - Use JavaScript to send the uploaded image to the Flask API endpoint for prediction.
   - Display the predicted traffic sign label and play the generated voice alert to the user.

7.5 Coding

```
from google.colab import drive
drive.mount('/content/drive')
Mounted at /content/drive
import os
Root = "/content/drive/MyDrive/Colab
Notebooks/TSF/Traffic_sign_classification"
os.chdir(Root)
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score
np.random.seed(42)
from matplotlib import style
style.use('fivethirtyeight')

data_dir = "/content/drive/MyDrive/Colab
Notebooks/TSF/Traffic_sign_classification"
train_path = "/content/drive/MyDrive/Colab
Notebooks/TSF/Traffic_sign_classification/Train/"
test_path = "/content/drive/MyDrive/Colab
Notebooks/TSF/Traffic_sign_classification/Test/"
```

```
IMG_HEIGHT = 30
IMG_WIDTH = 30
channels = 3

NUM_CATEGORIES = len(os.listdir(train_path))
NUM_CATEGORIES
43

classes = { 0:'Speed limit (20km/h)',
1:'Speed limit (30km/h)',
2:'Speed limit (50km/h)',
3:'Speed limit (60km/h)',
4:'Speed limit (70km/h)',
5:'Speed limit (80km/h)',
6:'End of speed limit (80km/h)',
7:'Speed limit (100km/h)',
8:'Speed limit (120km/h)',
9:'No passing',
10:'No passing veh over 3.5 tons',
11:'Right-of-way at intersection',
12:'Priority road',
13:'Yield',
14:'Stop',
15:'No vehicles',
16:'Veh > 3.5 tons prohibited',
17:'No entry',
18:'General caution',
19:'Dangerous curve left',
20:'Dangerous curve right',
21:'Double curve',
22:'Bumpy road',
```

```
23:'Slippery road',
24:'Road narrows on the right',
25:'Road work',
26:'Traffic signals',
27:'Pedestrians',
28:'Children crossing',
29:'Bicycles crossing',
30:'Beware of ice/snow',
31:'Wild animals crossing',
32:'End speed + passing limits',
33:'Turn right ahead',
34:'Turn left ahead',
35:'Ahead only',
36:'Go straight or right',
37:'Go straight or left',
38:'Keep right',
39:'Keep left',
40:'Roundabout mandatory',
41:'End of no passing',
42:'End no passing veh > 3.5 tons' }

folders = os.listdir(train_path)
train_number = []
class_num = []

for folder in folders:
train_files = os.listdir(train_path + '/' + folder)
train_number.append(len(train_files))
class_num.append(classes[int(folder)])

# Sorting the dataset on the basis of number of images in each class
zipped_lists = zip(train_number, class_num)
```

```
sorted_pairs = sorted(zipped_lists)
tuples = zip(*sorted_pairs)
train_number, class_num = [ list(tuple) for tuple in tuples]

image_data = []
image_labels = []

for i in range(NUM_CATEGORIES):
path = data_dir + '/Train/' + str(i)
images = os.listdir(path)
for img in images:
try:
image = cv2.imread(path + '/' + img)
image_fromarray = Image.fromarray(image, 'RGB')
resize_image = image_fromarray.resize((IMG_HEIGHT,
IMG_WIDTH))
image_data.append(np.array(resize_image))
image_labels.append(i)
except:
print("Error in " + img)

# Changing the list to numpy array
image_data = np.array(image_data)
image_labels = np.array(image_labels)
print(image_data.shape, image_labels.shape)
(39605, 30, 30, 3) (39605,)
shuffle_indexes = np.arange(image_data.shape[0])
np.random.shuffle(shuffle_indexes)
image_data = image_data[shuffle_indexes]
image_labels = image_labels[shuffle_indexes]
X_train, X_val, y_train, y_val = train_test_split(image_data,
image_labels, test_size=0.3, random_state=42, shuffle=True)
```

```
X_train = X_train/255
X_val = X_val/255
print("X_train.shape", X_train.shape)
print("X_valid.shape", X_val.shape)
print("y_train.shape", y_train.shape)
print("y_valid.shape", y_val.shape)
X_train.shape (27723, 30, 30, 3)
X_valid.shape (11882, 30, 30, 3)
y_train.shape (27723,)
y_valid.shape (11882,)
y_train = keras.utils.to_categorical(y_train, NUM_CATEGORIES)
y_val = keras.utils.to_categorical(y_val, NUM_CATEGORIES)
print(y_train.shape)
print(y_val.shape)
(27723, 43)
(11882, 43)

from tensorflow.keras import layers, models, optimizers
model = models.Sequential()

# First convolutional block
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3),
padding='same', activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH,
channels)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))

# Second convolutional block
```

```
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))

# Third convolutional block
model.add(layers.Conv2D(filters=128, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=128, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))

# Flatten the output and feed it into a fully connected layer
model.add(layers.Flatten())
model.add(layers.Dense(units=512, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(rate=0.5))

# Output layer
model.add(layers.Dense(units=NUM_CATEGORIES, activation='softmax'))
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
lr = 0.001
epochs = 30
opt = Adam(lr=lr, decay=lr / (epochs * 0.5))
```

```
model.compile(loss='categorical_crossentropy', optimizer=opt,
metrics=['accuracy'])
aug = ImageDataGenerator(
rotation_range=10,
zoom_range=0.15,
width_shift_range=0.1,
height_shift_range=0.1,
shear_range=0.15,
horizontal_flip=False,
vertical_flip=False,
fill_mode="nearest")
history = model.fit(aug.flow(X_train, y_train, batch_size=32),
epochs=epochs, validation_data=(X_val, y_val))

model.save("tsf.h5")
test = pd.read_csv(data_dir + '/Test.csv')
labels = test["ClassId"].values
imgs = test["Path"].values
data = []
for img in imgs:
try:
image = cv2.imread(data_dir + '/' + img)
image_fromarray = Image.fromarray(image, 'RGB')
resize_image = image_fromarray.resize((IMG_HEIGHT, IMG_WIDTH))
data.append(np.array(resize_image))
except:
print("Error in " + img)
X_test = np.array(data)
X_test = X_test / 255
pred = model.predict(X_test)
pred_classes = np.argmax(pred, axis=1)
```

```
#Accuracy with the test data
print('Test Data accuracy: ', accuracy_score(labels, pred_classes) *
100)
395/395 [==============================] - 18s 44ms/step
Test Data accuracy: 98.83610451306414
```

Chapter 8

Testing

8.1 How Software Defects Arise:

Software defects in Traffic Sign Recognition systems can occur due to various reasons. Here are some common causes:

1. **Inadequate Training Data**: If the data used to teach the system is not enough or biased, it may make the system less accurate. The data should include different situations like lighting changes, and various types of signs.

2. **Improper Preprocessing**: ISometimes, the way the data is prepared before training can introduce errors. Incorrect resizing, normalization, or other processing steps can harm the system's performance. Preprocessing should handle challenges like partial signs coverage or different signs sizes.

3. **Wrong Algorithm Choice**: Selecting an unsuitable Traffic Sign Recognition-algorithm can lead to poor results. Different algorithms have different strengths and limitations, so it's important to choose the right one for the task at hand.

4. **Model Issues**: Overfitting occurs when the model is too focused on the training data, while underfitting happens when the model fails to learn properly. Both situations can lead to inaccurate predictions. Proper training and validation are needed to avoid these issues.

5. **Environmental Factors**: Traffic Sign Recognition systems can be sensitive to factors like lighting, background noise, or camera angles. If the system doesn't handle these variations well, it can result in false detections or reduced accuracy.

6. **Compatibility Problems**: Sometimes, the system may not work properly on certain devices or software configurations. Compatibility issues can cause unexpected behavior or errors if not properly tested.

7. **Insufficient Testing**:Inadequate testing can mean that some defects go unnoticed. Not testing enough scenarios or not using enough data for evaluation can lead to missed issues.

To prevent software defects in systems, it's important to use diverse and representative data, choose appropriate algorithms, consider environmental factors, and conduct thorough testing and validation. Regular updates and monitoring can help improve the system over time.

8.2 Inability to Find All Faults

happens because there are certain challenges:

Challenges in Detection: Detecting a sign can be difficult due to different types of signs. The system may not always detect signs correctly in these situations.

1. **Environmental Factors**: The system's performance can be affected by things like lighting, background noise, or camera angles. In some environments, it may struggle to detect signs reliably.

2. **Accuracy and Sensitivity**: The detection algorithm used in the system may have limitations in terms of accuracy or sensitivity. It may mistakenly detect signs when they are not present or fail to detect signs when they are.

3. **Limited Dataset**: The training data used to develop the system plays a big role in its performance. If the data does not represent real-life situations well or lacks variety, the system may struggle to detect signs accurately.

4. **Real-Time Constraints**: In real-time applications, the system needs to work quickly and efficiently. This can create limitations in terms of processing power or speed, which may affect the system's ability to find all faults accurately.

To improve the system's fault detection, it is important to make regular updates, refine algorithms, and use more diverse training data. Testing the system in different environments and gathering user feedback can also help in finding and addressing faults.

8.3 When Testing is Carried Out

1. **Algorithm Development**: Testing the accuracy and performance of Traffic Sign Recognition algorithms using known datasets.

2. **Model Training and Validation**: Testing the trained model's performance on separate datasets to measure accuracy and identify any issues.

3. **Integration Testing**: Testing how different system components interact to ensure smooth functionality and data flow.

4. **System Testing**: Testing the system's ability to detect signs and classify signs accurately in various scenarios and conditions.

5. **User Acceptance Testing**: : Gathering feedback from users to ensure the system meets their requirements and provides a satisfactory experience.

6. **Performance and Load Testing**:Testing the system's behavior under different workloads to identify performance issues and scalability

7. **Regression Testing**:: Testing for new defects or impact on existing functionality when making updates or changes.

Testing is done iteratively throughout development, deployment, and maintenance to ensure the system works well in real-world situations.

8.4 Finding Faults Early in the Process

Finding faults early in the process of developing a system is crucial to ensure a reliable and accurate system. Here are some key strategies for identifying faults early in the process:

1. **Requirements Analysis**: Conduct a thorough analysis of the system requirements, including the desired accuracy, performance, and environmental considerations. This helps to identify any potential faults or gaps in the requirements early on.

2. **Prototyping**:Create prototypes of the system to visualize and test the user interface, workflow, and system behavior. This allows for early identification of design flaws or usability issues that could lead to faults in the final system.

3. **Code Reviews and Pair Programming**: Implement a process of code reviews and pair programming, where multiple developers review and collaborate on the code. This helps to catch errors, identify potential faults, and improve code quality before it becomes integrated into the system.

4. **Unit Testing**: : Implement unit testing, where individual components or modules of the traffic signs detection system are tested in isolation. This early testing helps to identify defects and faults in the specific components, allowing for prompt resolution.

5. **Continuous Integration and Testing**:Employ continuous integration practices, where code changes are frequently integrated into a shared repository and automatically tested. This approach helps to detect integration issues and faults early on, allowing for rapid identification and resolution.

6. **Alpha and Beta Testing**:Conduct alpha and beta testing phases with real users or selected test groups. This allows for gathering valuable feedback and identifying faults, usability issues, or unexpected behaviors before the system is released to a wider audience.

By following these strategies, faults and issues in the Traffic Sign Recognition system can be identified and addressed early in the development process. This helps

to minimize risks, improve system quality, and ensure a more reliable and accurate final product.

8.5 Measuring Software Testing

1. **Test Coverage**: Evaluate how much of the system has been tested, including code, functionalities, and requirements.
2. **Accuracy Metrics**: Measure the correctness of signs detection using metrics like precision, recall, and F1 score.
3. **Defect Detection and Fixing**:Keep track of the number and severity of defects found during testing and how quickly they are fixed.
4. **Test Execution Time**: Monitor the time taken to run the test suite to identify potential efficiency improvements.
5. **Test Case Effectiveness**: Assess the ability of test cases to detect faults using metrics like success rate and fault detection rate.
6. **Regression Test Coverage**: Measure the coverage of regression testing to ensure changes don't introduce new faults.
7. **User Satisfaction**:Gather user feedback to understand their satisfaction with the system's performance and usability.

By considering these measures, the testing process can be improved, leading to a more reliable and accurate sign.

8.6 Static and Dynamic Testing

Static Testing : Static testing involves examining the system's code, design, and documentation without executing the software. In the case of traffic sign recognition, static testing may include:

1. **Code Review**: Analyzing the code to identify potential coding errors, vulnerabilities, or inconsistencies.

2. **Design Review**: Assessing the system's architecture, algorithms, and data flow to ensure they align with best practices and meet the desired requirements.

3. **Documentation Review**:Verifying that the system's documentation, including user manuals and technical specifications, is accurate, complete, and up to date.

Static testing helps identify defects and issues early in the development process, reducing the likelihood of errors and improving the overall quality of the system.

Dynamic Testing : Dynamic testing involves executing the software and analyzing its behavior during runtime. In the case of traffic sign recognition, dynamic testing may include:

1. **Functional Testing**: Verifying that the Traffic Sign Recognition system performs as intended, detecting signs accurately and producing correct results.

2. **Performance Testing**:Assessing the system's performance under different workloads to ensure it meets performance requirements, such as response time and resource utilization.

3. **Usability Testing** : Evaluating the user interface and user experience of the Traffic Sign Recognition system to ensure it is intuitive, easy to use, and meets user expectations.

4. **Compatibility Testing** : Testing the system's compatibility with different platforms, operating systems, and devices to ensure it works as expected across various environments.

Dynamic testing focuses on assessing the system's functionality, performance, and usability in real-world scenarios, ensuring its effectiveness and reliability.

By combining static and dynamic testing techniques, a traffic sign recognition system can be thoroughly evaluated, defects can be identified and resolved early, and the overall quality of the system can be improved.

8.7 Software Testing Measurement

To measure software testing in a traffic sign recognition system, focus on key metrics. Test coverage assesses the extent of testing, including code, functionality, and requirements. Defect detection measures the number and severity of identified issues. Test execution time evaluates efficiency. Accuracy metrics, like precision and recall, assess correct sign detection. Regression testing coverage ensures changes don't introduce new defects. User satisfaction gathers feedback for user experience evaluation. Test case effectiveness assesses fault detection. These measurements help evaluate testing quality, accuracy, and reliability while identifying areas for improvement.

8.8 Test Plan and Method

Objective: The objective of this test plan is to ensure the accuracy and reliability of the traffic sign detection system.

1. **Test Scope**: The test will focus on evaluating the performance of the Traffic Sign Recognition algorithm or system. It will cover the following areas:

   (a) **Detection accuracy**: The ability to correctly identify whether a sign is traffic sign or not a sign.

   (b) **Voice alert generation**: The accuracy and appropriateness of the voice alerts generated for different traffic sign classes.

   (c) **Speed**: The processing time required for the detection and classification of traffic signs.

   (d) **Test Environment**: The tests will be conducted in a controlled environment using a dataset of labeled traffic sign images. The system will be tested on different hardware configurations and operating systems.

   (e) **Test Data**: A diverse dataset of traffic sign images will be used for testing, including different traffic sign types, sizes, and variations. The dataset will also include challenging scenarios such as occlusions and low-quality images.

2. **Test Environment**:The test will be conducted in a controlled frontend environment with the following specifications:

   (a) **Frontend application**:The traffic sign recognition system with voice alerts will be tested within the frontend application interface.

   (b) **Input data**: A diverse set of traffic sign images will be used as input for testing, including various shapes, colors, and sizes, commonly encountered on roads.

   (c) **Simulated user interactions**: The frontend will simulate user interactions by feeding the traffic sign images to the prediction algorithm and capturing the voice alerts generated by the system.

3. **Test Methodology**: The following steps will be followed during the testing process:

   (a) **Test Data Collection**:

      i. Collect a dataset of images containing different traffic signs.

      ii. Ensure the dataset represents a wide range of scenarios, including different images.

      iii. Annotate the dataset with ground truth labels indicating whether a sign is traffic sign or not.

   (b) **Test Scenario Definition**:

      i. Define various test scenarios, such as different traffic signs for blur images.

   (c) **Test Execution**:

      i. Run the traffic signdetection algorithm or system on each test scenario.

      ii. Measure and record the following metrics:

         A. **Detection accuracy**: Calculate the percentage of correctly identified signs.

         B. **False positive/negative rate**: Calculate the percentage of incorrectly identified signs.

C. **Processing time** : Measure the time taken for detection in each scenario.

D. **Robustness**: Observe the system's performance under different images conditions.

(d) **Test Analysis**:

i. Analyze the collected data and metrics to evaluate the performance of the traffic sign detection system.

ii. Identify any issues, such as false positives/negatives, performance bottlenecks, or limitations.

iii. Determine if the system meets the required accuracy, speed, and robustness criteria

4. **Test Reporting**:

(a) Prepare a comprehensive report documenting the test plan, methodology, and results.

(b) Include an analysis of the system's performance, highlighting strengths and areas for improvement.

(c) Provide recommendations for enhancements or optimizations if necessary.

8.9 Runtime Snapshots

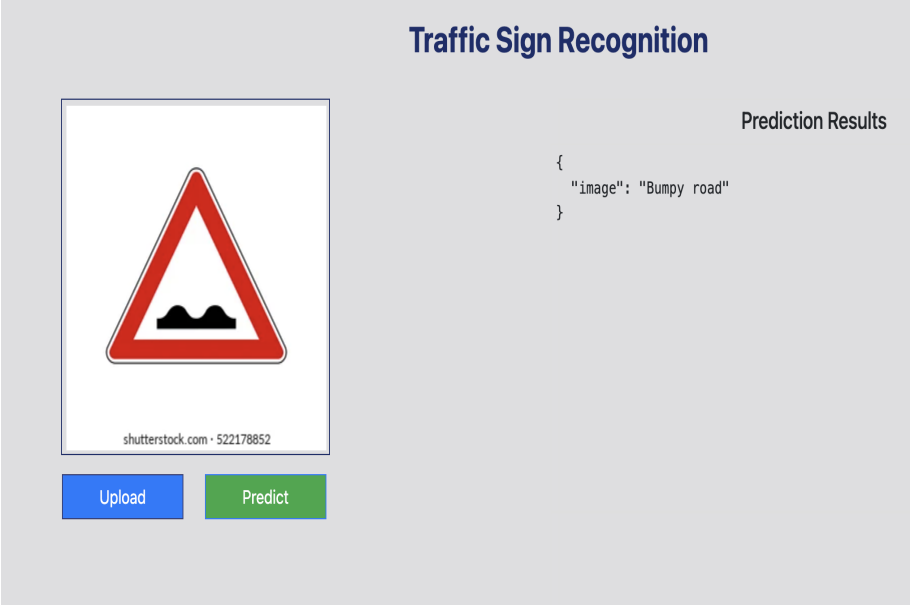

Figure 8.1: Runtime Snapshot 1

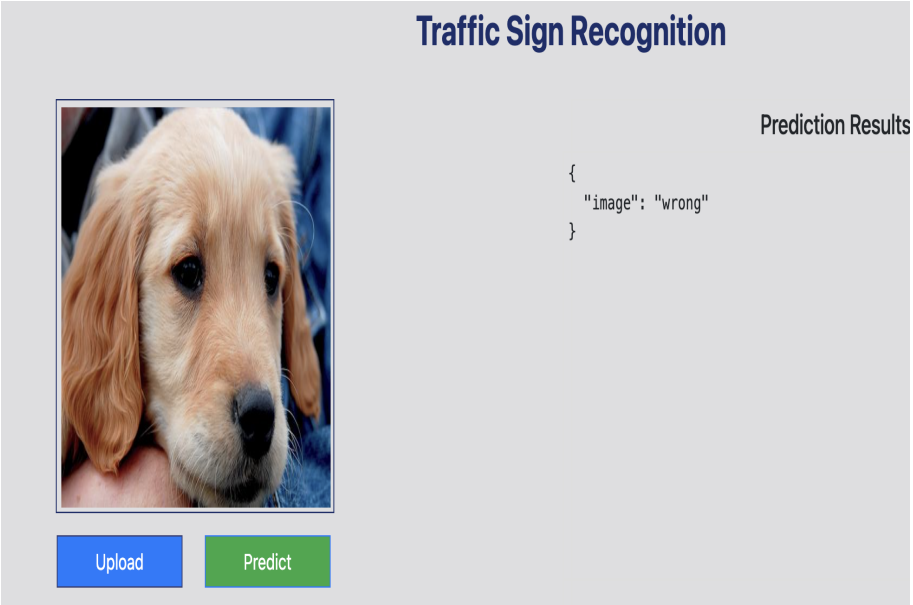

Figure 8.2: Runtime Snapshot 2

Figure 8.3: Runtime Snapshot 3

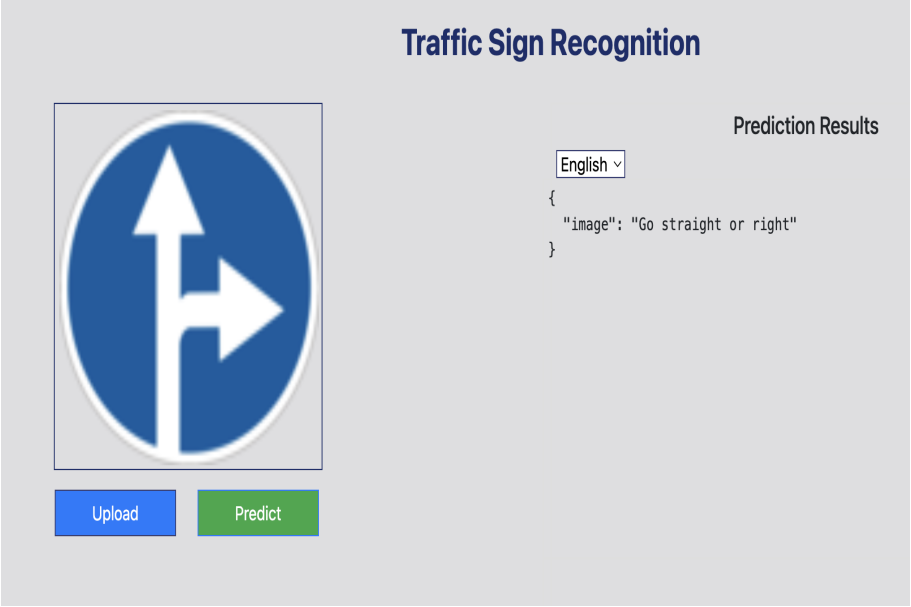

Figure 8.4: Runtime Snapshot 4

Conclusion

The Traffic Sign Board Detection and Voice Alert System is implemented using Convolutional Neural Network. Various models under the CNN heading were studied and the one with the highest accuracy on the GTSRB dataset was implemented. The creation of different classes for each Traffic sign has helped in increasing the accuracy of the model. A voice message is sent after recognition of the sign, which alerts the driver. The accuracy for this model is 98.8%.

Bibliography

[1] Yadav, Shubham Patwa, Anuj Rane, Saiprasad Narvekar, Chhaya. Indian Traffic Sign Board Recognition and Driver Alert System Using Machine Learning. International Journal of Applied Sciences and Smart Technologies. 1. 1-10. 10.24071/ijasst. v1i1.1843 *2*019 .

[2] Anushree.A., S., Kumar, H., Iram, I., Divyam, K. Automatic Signboard Detection System by the Vehicles. S. Harini, V. Abhiram, R. Hegde, B. D. D. Samarth, S. A. Shreyas and K. H. Gowranga, "A smart driver alert system for vehicle traffic using image detection and recognition technique," 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT), Bangalore, India, 2017, pp. 1540-1543, doi: 10.1109/RTEICT.2017.8256856 *2*019 .

[3] Wang, "Research and Application of Traffic Sign Detection and Recognition Based on Deep Learning," 2018 International Conference on Robots Intelligent System (ICRIS), Changsha, China, 2018, pp. 150-152, doi: 10.1109/ICRIS.2018.00047. M A Muchtar et al 2017 J. Phys.: Conf. Ser. 801 012010 *2*018.

[4] Yuan, Z. Xiong and Q. Wang, "VSSA-NET: Vertical Spatial Sequence Attention Network for Traffic Sign Detection," in IEEE Transactions on Image Processing, vol. 28, no. 7, pp. 3423-3434, July 2019, doi: 10.1109/TIP.2019.2896952.

[5] A. Pon, O. Adrienko, A. Harakeh and S. L. Waslander, "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), Toronto, ON, Canada, 2018, pp. 102-109, doi: 10.1109/CRV.2018.00024.

[6] Saha S., Islam M.S., Khaled M.A.B., Tairin S. (2019) An Efficient Traffic Sign Recognition Approach Using a Novel Deep Neural Network Selection Architecture. In: Abraham A., Dutta P., Mandal J., Bhattacharya A., Dutta S. (eds) Emerging Technologies in Data Mining and Information Security. Advances in Intelligent Systems and Computing, vol 814. Springer, Singapore. https://doi.org/10.1007/978-981-13-1501-574

[7] A. Welzel, A. Auerswald and G. Wanielik, "Accurate camera-based traffic sign localization," 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Qingdao, China, 2014, pp. 445-450, doi: 10.1109/ITSC.2014.6957730.

[8] M. Karaduman and H. Eren, "Deep learning based traffic direction sign detection and determining driving style," 2017 International Conference on Computer Science and Engineering (UBMK), Antalya, Turkey, 2017, pp. 1046-1050, doi: 10.1109/UBMK.2017.8093453.

[9] Chuanwei Zhang et al., Study on Traffic Sign Recognition by Optimized Lenet-5 Algorithm, International Journal of Pattern Recognition and Artificial Intelligence, doi:0.1142/S0218001420550034

[10] Pal R, Ghosh A, Kumar R, et al. Public health crisis of road traffic accidents in India: Risk factor assessment and recommendations on prevention on the behalf of the Academy of Family Physicians of India. J Family Med Prim Care. 2019;8(3):775-783. doi:10.4103/jfmpc.jfmpc21418

[11] Krish Sukhani , Dr Radha Shankarmani , Jay Shah , Krushna Shah Traffic Sign Board Recognition and Voice Alert System using Convolutional Neural Network .2021 2nd International Conference for Emerging Technology (INCET) doi:10.1109/INCET51464.2021.9456302