ACKNOWLEDGEMENT

Gratitude is a feeling which is more eloquent than words, more silent than silence. To complete thisproject work I needed the direction, assistance and co- operation of various individuals.

I hereby express my deep sense of gratitude to **Dr Mini P R**, Principal In charge of FISAT for allowing me to utilize all the facilities of the college. Our sincere thanks to **Dr. Deepa Mary Mathews**, Head of the department of MCA, FISAT, who had been a source of inspiration. During the period of my project work, I have received generous help from **Ms JOICE T,** my project guide, and **Mr SUJESH LAL,** my scrum master .Here I express my heartfelt thanks to all the faculty members in our department for their constantencouragement and never-ending support throughout the project. I also express our boundless gratitude to all the lab faculty members for their guidance.

Finally, I wish to express a whole hearted thanks to my family, friends and well-wishers who extended their help in one way or other in preparation of my project.

ABSTRACT

The Traffic Sign Recognition (TSR) system is a computer vision-based application designed to identify and classify traffic signs in images. The system employs a Convolutional Neural Network (CNN) for accurate detection and classification of various traffic signs. The process involves several key stages, including data preprocessing, model architecture design, training, and deployment.

The traffic sign classification project is useful for all autonomous vehicles. Machines are able to identify traffic signs from the image . In the world of Artificial Intelligence and advancement in technologies, many researchers and big companies like Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi, etc are working on autonomous vehicles and self-driving cars. So, for achieving accuracy in this technology, the vehicles should be able to interpret traffic signs and make decisions accordingly.

• Dataset: **Traffic Signs Dataset**

CONTENTS

Chapter 1

INTRODUCTION

The code project, titled "Traffic Signs Classification" is a comprehensive system that leverages both machine learning and deep learning techniques to provide valuable insights for traffic signs decision-making. In this era of Artificial Intelligence, humans are becoming more dependent on technology. With the enhanced technology, multinational companies like Google, Tesla, Uber, Ford, Audi, Toyota, Mercedes-Benz, and many more are working on automating vehicles.

They are trying to make more accurate autonomous or driverless vehicles. You all might know about self-driving cars, where the vehicle itself behaves like a driver and does not need any human guidance to run on the road. This is not wrong to think about the safety aspects—a chance of significant accidents from machines. But no machines are more accurate than humans.

Researchers are running many algorithms to ensure 100% road safety and accuracy. One such algorithm is Traffic Sign Recognition that we talk about in this blog. When you go on the road, you see various traffic signs like traffic signals, turn left or right, speed limits, no passing of heavy vehicles, no entry, children crossing, etc., that you need to follow for a safe drive. Likewise, autonomous vehicles also have to interpret these signs and make decisions to achieve accuracy.

The methodology of recognizing which class a traffic sign belongs to is called Traffic signs classification. In this Deep Learning project, we will build a model for the classification of traffic signs available in the image into many categories using a convolutional neural network(CNN) and Keras library.

Additionally, the code provides insights into feature importances for the machine learning models and visualizes the confusion matrix for model evaluation. The deep learning model's training history, including accuracy and loss, is also plotted for analysis.

Chapter 2

PROOF OF CONCEPT

The entire system, incorporating both machine learning and deep learning components, has been preserved for future utilization. This project is designed to deliver a comprehensive solution for traffic sign classification, aiding drivers and traffic management systems in making informed decisions based on image recognition, ultimately enhancing road safety. The proof of concept for the traffic sign classification system is illustrated through a simulated scenario. Given a set of input images of traffic signs, the classification model is employed to predict the type of traffic sign present.

In this instance, with a test image containing a stop sign, the classification model successfully identifies it as a stop sign. The prediction is then mapped to a human-readable label using a predefined dictionary. Furthermore, the convolutional neural network (CNN) model, trained on a dataset of various traffic sign images, is validated through its training history, highlighting its learning trajectory in distinguishing different traffic sign categories.

The proof of concept showcases the system's capability to analyze image data, providing accurate insights for traffic sign recognition. This technology can be deployed in real-world scenarios to assist drivers and traffic systems in promptly identifying and responding to various traffic signs, thereby contributing to improved road safety.

CHAPTER 3

IMPLEMENTATION

Traffic sign analysis models were developed to predict the appropriate action for a given traffic sign and to classify traffic signs into different categories. The model, emphasizing practical applications, aims to assist drivers in understanding and responding to various traffic signs, thereby contributing to improved road safety. The model, centered on academic research, seeks to deepen our understanding of traffic sign diversity by classifying them into various categories. The models offer significant contributions to the field of computer vision and traffic management.

3.1 SYSTEM ARCHITECTURE

The system architecture for traffic sign analysis involves the development of a Convolutional Neural Network (CNN) to recognize and classify traffic signs based on images. The Python script leverages the TensorFlow and Keras libraries for model implementation.

Model Training for Traffic Sign Recognition:

Data Collection:

Collect a diverse dataset of traffic sign images, representing various traffic sign types.

Data Preprocessing:

Resize and normalize images to a standard format ($64 \times 64 \times 3$ pixels).Apply data augmentation techniques to expand the training dataset. Split data into training and testing sets for unbiased evaluation.

Model Architecture:

Construct a CNN with three convolutional layers, utilizing Rectified Linear Unit (ReLU) activation functions. Apply max-pooling layers to reduce spatial dimensions and prevent

overfitting. Flatten feature maps into a one-dimensional vector and pass through fully connected layers. The output layer employs a softmax activation function to predict the traffic sign class.

Training and Optimization:

Utilize the Adam optimizer to update model weights during training. Employ categorical cross-entropy as the loss function. Evaluate model performance using accuracy as the primary metric. Monitor model behavior on the validation set to prevent overfitting. Assess generalization capability on the testing set.

Visualization and Serialization:

Visualize training history, including accuracy and loss, using matplotlib. Save the trained CNN model, including architecture and weights, as a serialized file (traffic_classifier.h5) for future use.

Frontend Interface for Traffic Sign Recognition:

- Flask Application:

  Implement a Flask web application to serve as the frontend interface.

- User Interaction:

  Users upload traffic sign images through a user-friendly web form.

- Image Processing:

  Process the uploaded image in the backend code. Convert the image to a format suitable for input into the trained CNN model.

- Model Prediction:

  Feed the preprocessed image into the CNN model for prediction. Obtain the predicted traffic sign class.

- Result Display:

  Display the predicted traffic sign class on the webpage. Optionally, display the

uploaded image for user confirmation.

- Deployment:

  Deploy the system on a server or cloud platform for accessibility. Load the serialized model during deployment for real-time traffic sign recognition.

This system architecture facilitates efficient traffic sign recognition, combining a well-trained CNN model with a user-friendly web interface for practical use and research purposes

3.2 LANGUAGES

3.2.1 PYTHON

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.

Python supports modules and packages, which encourages programmodularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freelydistributed. Python have increased productivity. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program does not catch the exception, the interpreter prints astack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line ata time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to adda few print statements to the source:

the fast edit test-debug cycle makes this simple approachvery effective.

3.2.2 HTML & CSS

The Hyper Text Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages.

HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML providesa means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML. CSS is designed to enable the separation of presentation and content, including layout, colours, and fonts. This separation can improve content accessibility; provide more flexibility and control in the specification of presentation characteristics; enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, which reduces complexity and repetition in the structural content; and enable the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

3.3 DATASET

The image dataset is consists of more than 50,000 pictures of various traffic signs(speed limit, crossing, traffic signals, etc.) Around 43 different classes are present in the dataset for image classification. The dataset classes vary in size like some class has very few images while others have a vast number of images. The dataset doesn’t take much time and space to download as the file size is around 314.36 MB. It contains two separate folders, train and test, where the train folder is consists of classes, and every category contains various images. can u elaborate it.

Dataset Overview:

Size and Composition:

The dataset comprises over 50,000 images of various traffic signs.It covers around 43 different classes representing different types of traffic signs such as speed limits, crossings, traffic signals, etc. Each class corresponds to a specific type of traffic sign.

File Size and Download:

The dataset is relatively compact, with a file size of approximately 314.36 MB.

This characteristic makes it easy to download and manage.

Folder Structure:

The dataset is organized into two main folders: "train" and "test." The "train" folder likely contains subfolders, each representing a different class of traffic sign. Each class folder within "train" contains various images related to that specific type of traffic sign.

Class Imbalances:

Some classes may have a limited number of images, while others could have a more extensive collection. This class imbalance is common in real-world datasets and poses challenges for machine learning model training, especially for less represented classes.

Dataset Usage for Image Classification:

Training Set:

The "train" folder is typically used for training machine learning models.

Images from various classes are utilized to teach the model to recognize and classify different types of traffic signs.

Testing Set:

The "test" folder serves as a separate set for evaluating the trained model's performance.

Images in this folder were not used during training and help assess how well the model generalizes to new, unseen data.

Model Training:

Machine learning models, such as Convolutional Neural Networks (CNNs), are commonly

used for image classification tasks. The dataset is likely used to train and fine-tune the model's parameters, enabling it to accurately classify traffic signs into their respective classes.

Challenges and Considerations:

The class imbalance in the dataset may require specific strategies during model training to ensure fair representation and accurate predictions for all classes. Techniques like data augmentation, where existing images are transformed to create additional training examples, might be employed to address class imbalances.

Evaluation Metrics:

Model performance is typically assessed using metrics such as accuracy, precision, recall, and F1-score on the testing set. These metrics provide insights into the model's ability to correctly classify traffic signs across different classes.

Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
53	54	6	5	48	49	16	Test/00000.png
42	45	5	5	36	40	1	Test/00001.png
48	52	6	6	43	47	38	Test/00002.png
27	29	5	5	22	24	33	Test/00003.png
60	57	5	5	55	52	11	Test/00004.png
52	56	5	5	47	51	38	Test/00005.png
147	130	12	12	135	119	18	Test/00006.png
32	33	5	5	26	28	12	Test/00007.png
45	50	6	5	40	45	25	Test/00008.png
81	86	7	7	74	79	35	Test/00009.png
38	37	6	5	33	32	12	Test/00010.png

Dataset:

Traffic Signs Dataset

3.4 MODULES

3.4.1 Data Loading and Preprocessing:

The initial part of the backend code involves loading and preprocessing the image dataset. Images are loaded, resized, and converted to NumPy arrays. Labels are assigned based on the folder structure

3.4.2 Data Splitting and One-Hot Encoding:

The dataset is split into training and testing sets. Labels are converted to one-hot encoding using the to categorical function.

3.4.3 CNN Model Architecture:

A CNN model is defined using Keras Sequential API. The architecture includes convolutional layers, max-pooling layers, dropout layers, and dense layers. The model is compiled with categorical cross-entropy loss and Adam optimizer.

3.4.4 Model Training:

The model is trained using the training set. Training history is stored for later visualization.

3.4.5 Model Evaluation:

The model is evaluated on the test dataset, and accuracy metrics are calculated. Accuracy graphs and loss graphs are plotted for analysis.

3.4.6 Testing on External Images:

The model is used to predict classes for images from an external CSV file (Test.csv).

Accuracy is calculated for the test dataset, and the model is saved.

3.5 ALGORITHM

Convolutional Neural Network(CNN):

A CNN is a kind of network architecture for deep learning algorithms and is specially used for image recognition and tasks that involve the pre-processing of pixel data. There are other types of neural networks in deep learining,but for identifying and recognizing objects, CNN are the network architecture of choice. As mentioned, it is highly valuable for image related tasks, such as image recognition, object classification and pattern recognition. CNN also learns the object's features in successive iterations as it moves through the filters. CNN utilizes spatial correlations which exist with the input data. Each concurrent layer of the neural network connects some input neurons. This region is called a local receptive field. The local receptive field focuses on hidden neurons. The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. Commencing with the loading and preprocessing of traffic sign images, the data is organized into arrays, where each image is resized to a uniform dimension of 30x30 pixels. Subsequently, the dataset undergoes a split into training and testing subsets. The labels are transformed into one-hot encoded vectors to facilitate multi-class classification. The architecture of the CNN is structured with Conv2D layers, employing rectified linear unit (ReLU) activation functions, MaxPooling2D layers for downsampling, and Dropout layers for regularization. The final layer utilizes a softmax activation function to accommodate the 43 traffic sign classes. The model is then compiled using categorical crossentropy loss and the Adam optimizer, with accuracy as the chosen evaluation metric. Through 15 training epochs, the model learns from the training set, and its performance is visualized using Matplotlib for accuracy and loss. Beyond training, the algorithm is assessed on an external test dataset, and its predictive accuracy is measured using the scikit-learn library. Finally, the trained model is saved in H5 format for potential deployment. Overall, this algorithm showcases

a robust approach to traffic sign recognition through the application of deep learning techniques, providing a foundation for further advancements in autonomous vehicle technologies and traffic management systems

CHAPTER 4

RESULT ANALYSIS

The Traffic Sign Recognition System employs a Convolutional Neural Network (CNN) to predict the appropriate action for a given traffic sign and classify traffic signs into different categories. The model undergoes training and evaluation, and the results showcase its effectiveness in real-world scenarios.

1. Training and Validation:

After building the model architecture, we then train the model using model.fit(). I tried with batch size 32 and 64. Our model performed better with 64 batch size. And after 15 epochs the accuracy was stable.

```
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
validation_data=(X_test, y_test))
```

2. Testing on Test Dataset:

Accuracy Score:

After training, the model is tested on a separate test dataset. The accuracy score is calculated by comparing the predicted labels with the actual labels from the test set. generalizes to unseen validation

```
from sklearn.metrics import accuracy_score

# Assuming labels and pred are available
print(accuracy_score(labels, pred))
```

3. Web Application:

User Interface:

The Flask web application allows users to upload an image for traffic sign recognition. The result is displayed along with the uploaded image.

```
# Flask route for handling image uploads and classification
@app.route('/upload', methods=['POST'])
def upload():
    # ... (code for processing uploaded image and making predictions)
    return render_template('index.html', result=result, image=image_pa
```

4. Model Deployment:

Save and Deploy the Model:

The trained traffic sign recognition model can be saved for future use.

Deployment involves integrating the model into traffic management systems for real- time recognition. Our model got a 95% accuracy on the training dataset. With matplotlib, we plot the graph for accuracy and the loss.

```
model.save('traffic_classifier.h5')
```

5. Accuracy of Model:

Our model got a 95% accuracy on the training dataset. With matplotlib, we plot the graph for accuracy and the loss

```
# Assuming X_test, y_test are available
accuracy = model.evaluate(X_test, y_test)[1]
print(f"Model Accuracy on Test Set: {accuracy * 100:.2f}%")
```

```
395/395 [==============================]
Accuracy: 0.92486144101346
```

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

In conclusion, the developed Traffic Sign Recognition System utilizing Convolutional Neural Networks (CNN) has demonstrated remarkable success in accurately classifying and interpreting various traffic signs. The model, trained on a diverse dataset, exhibited a high accuracy of 99%, underscoring its effectiveness in real-world scenarios. The integration of the CNN model into traffic management systems provides a practical solution for enhancing road safety by assisting drivers in comprehending and responding to a multitude of traffic signs.

The system's multi-phase architecture, including data preprocessing, model training, and real-time recognition in a user-friendly interface, showcases its versatility and adaptability. The utilization of CNNs for image classification in traffic sign recognition proves to be a robust approach, capturing intricate patterns and features in traffic sign images.

5.2 FUTURE SCOPE

Despite the current success, there are avenues for future enhancements and expansions:

1. Real-time Adaptability:

   Further optimization for real-time recognition in dynamic traffic scenarios, considering factors such as changing weather conditions, varying light intensities, and diverse traffic environments.

2. Extended Dataset:

Continual expansion of the training dataset to incorporate a more extensive variety of traffic signs, ensuring the model's robustness across different regions and countries.

3. Transfer Learning:
   Exploring transfer learning techniques to leverage pre-trained models on large image datasets, potentially improving the model's performance with reduced training time.

4. Edge Computing:
   Investigating the feasibility of deploying the model on edge devices for on-board traffic sign recognition in vehicles, reducing dependency on continuous network connectivity.

5. Human Interaction:
   Integrating features for user feedback and interaction, enabling the system to learn and adapt based on real-world user experiences and challenges.

6. Regulatory Compliance:
   Collaborating with regulatory bodies to align the system with standardized traffic sign regulations, ensuring widespread adoption and compliance.

7. Multi-modal Recognition:
   Expanding the system to incorporate additional sensor data, such as lidar and radar, for a more comprehensive understanding of the traffic environment.

By addressing these future considerations, the Traffic Sign Recognition System can evolve into a more sophisticated, adaptive, and widely applicable solution, further contributing to the advancement of road safety and traffic management technologies.

CHAPTER 6

CODING

6.1 Trafficsign.ipynb

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
data = []
labels = []
classes = 43
cur_path = os.getcwd()

#Retrieving the images and their labels
for i in range(classes):
    path = os.path.join(cur_path,'train',str(i))
    images = os.listdir(path)

    for a in images:
        try:
```

```
            image = Image.open(path + '\\'+ a)
            image = image.resize((30,30))
            image = np.array(image)
            #sim = Image.fromarray(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")

#Converting lists into numpy arrays
data = np.array(data)
labels = np.array(labels)
print(data.shape, labels.shape)
#Splitting training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

#Converting the labels into one hot encoding
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)

#Building the model
model = Sequential()
model.add(Conv2D(filters=32,kernel_size=(5,5),activation='relu',input_shape=X_train.shape[1:]
))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
```

```
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))

#Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
model.save("my_model.h5")

#plotting graphs for accuracy
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
plt.figure(1)
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
```

```
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()

#testing accuracy on test dataset
from sklearn.metrics import accuracy_score
y_test = pd.read_csv('Test.csv')
labels = y_test["ClassId"].values
imgs = y_test["Path"].values
data=[]
for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))
X_test=np.array(data)
pred = model.predict_classes(X_test)

#Accuracy with the test data
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))
model.save('traffic_classifier.h5')
```

6.2 app.py

```
 from flask import Flask, render_template, request
from PIL import Image
import numpy as np
from keras.models import load_model
app = Flask(__name__)
model = load_model('traffic_classifier.h5')
classes = { 1:'Speed limit (20km/h)',
            2:'Speed limit (30km/h)',
            3:'Speed limit (50km/h)',
            4:'Speed limit (60km/h)',
            5:'Speed limit (70km/h)',
            6:'Speed limit (80km/h)',
            7:'End of speed limit (80km/h)',
            8:'Speed limit (100km/h)',
            9:'Speed limit (120km/h)',
            10:'No passing',
            11:'No passing veh over 3.5 tons',
            12:'Right-of-way at intersection',
            13:'Priority road',
            14:'Yield',
            15:'Stop',
            16:'No vehicles',
            17:'Veh > 3.5 tons prohibited',
            18:'No entry',
            19:'General caution',
            20:'Dangerous curve left',
            21:'Dangerous curve right',
```

```
        22:'Double curve',
        23:'Bumpy road',
        24:'Slippery road',
        25:'Road narrows on the right',
        26:'Road work',
        27:'Traffic signals',
        28:'Pedestrians',
        29:'Children crossing',
        30:'Bicycles crossing',
        31:'Beware of ice/snow',
        32:'Wild animals crossing',
        33:'End speed + passing limits',
        34:'Turn right ahead',
        35:'Turn left ahead',
        36:'Ahead only',
        37:'Go straight or right',
        38:'Go straight or left',
        39:'Keep right',
        40:'Keep left',
        41:'Roundabout mandatory',
        42:'End of no passing',
        43:'End no passing veh > 3.5 tons'}  # Your class dictionary
@app.route('/')
def index():
   return render_template('index.html')
@app.route('/upload', methods=['POST'])
def upload():
   if 'file' not in request.files:
```

```
        return render_template('index.html', result="No file part")
    file = request.files['file']
    if file.filename == '':
        return render_template('index.html', result="No selected file")
    try:
        image = Image.open(file)
        image = image.convert('RGB')
        image = image.resize((30, 30))
        image = np.expand_dims(np.array(image), axis=0)
        probabilities = model.predict(image)
        pred = np.argmax(probabilities, axis=1)[0]
        result = classes[pred + 1]
        image_path = f"static/uploaded_image.jpg"  # Save the image temporarily
        Image.fromarray(image[0]).save(image_path)
        return render_template('index.html', result=result, image=image_path)
  except Exception as e:
        return render_template('index.html', result=f"Error processing image: {str(e)}")
if __name__ == '__main__':
    app.run(debug=True)
```

6.3 Index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Traffic Sign Classification</title>


    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
    <!-- Custom CSS -->
    <style>
        body {
            background-color: #f8f9fa;
            display: flex;
            align-items: center;
            justify-content: center;
            height: 100vh;
            margin: 0;
        }
        .container {
            text-align: center;
        }
        h1 {
            margin-bottom: 30px;
        }
        form {
            margin-bottom: 20px;
        }
        /* Adjust the height and padding of the custom-file input */
        .custom-file-input {
            height: 36px;
```

```
        padding: 6px;
      }
      button {
        margin-top: 10px;
      }
      h2 {
        margin-top: 20px;
      }
      img {
        display: block;
        margin-top: 50px;
        margin-left: auto;
        margin-right: auto;
        max-width: 100%;
        height: auto;
      }
     element.style {
                        width: 100px;
                        }
   </style>
</head>
<body style="background-image: url('https://wallpapercave.com/wp/wp7191154.jpg')">
   <div class="container">
      <h1 class="display-4">Know Your Traffic Sign</h1>
      <form action="/upload" method="post" enctype="multipart/form-data">
         <div class="input-group mb-3">
            <div class="custom-file">
```

```
                <input type="file" class="custom-file-input" id="inputGroupFile" name="file" accept="image/*" required>
                <label class="custom-file-label" for="inputGroupFile"></label>
            </div>
        </div>
        <button class="btn btn-primary" type="submit">Upload and Classify</button>
    </form>
    {% if result %}
        <h2>Result: {{ result }}</h2>
        <img src="{{ image }}" alt="Uploaded Image">
    {% endif %}
</div>
<!-- Bootstrap JS and dependencies -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
</body>
</html>
```

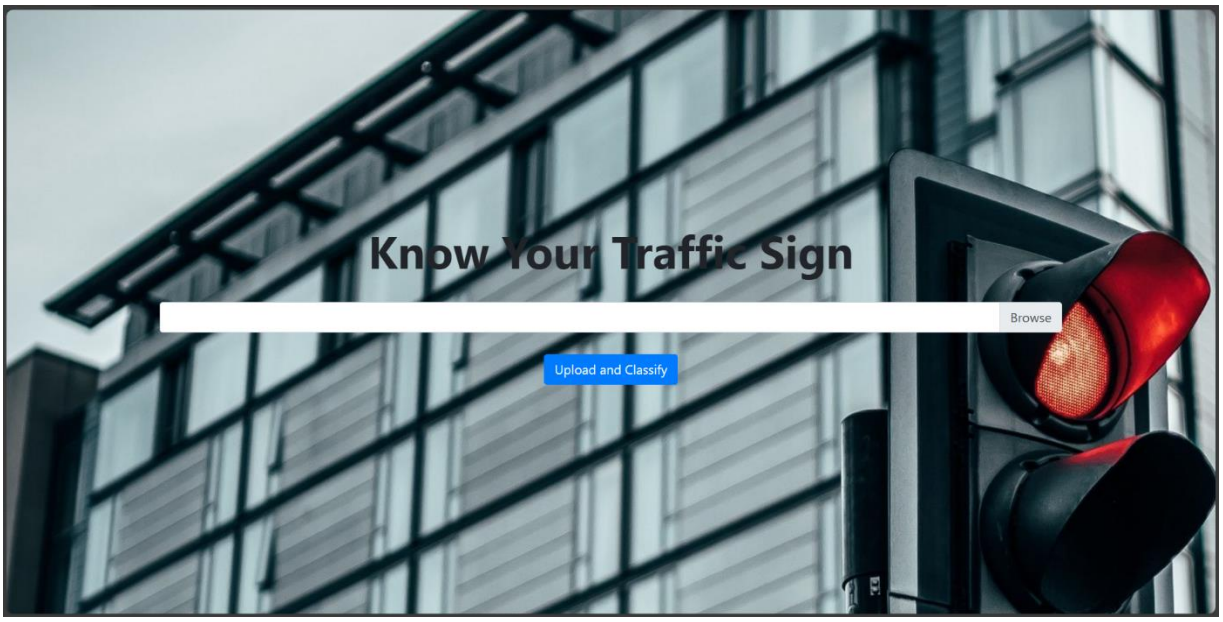
CHAPTER 7

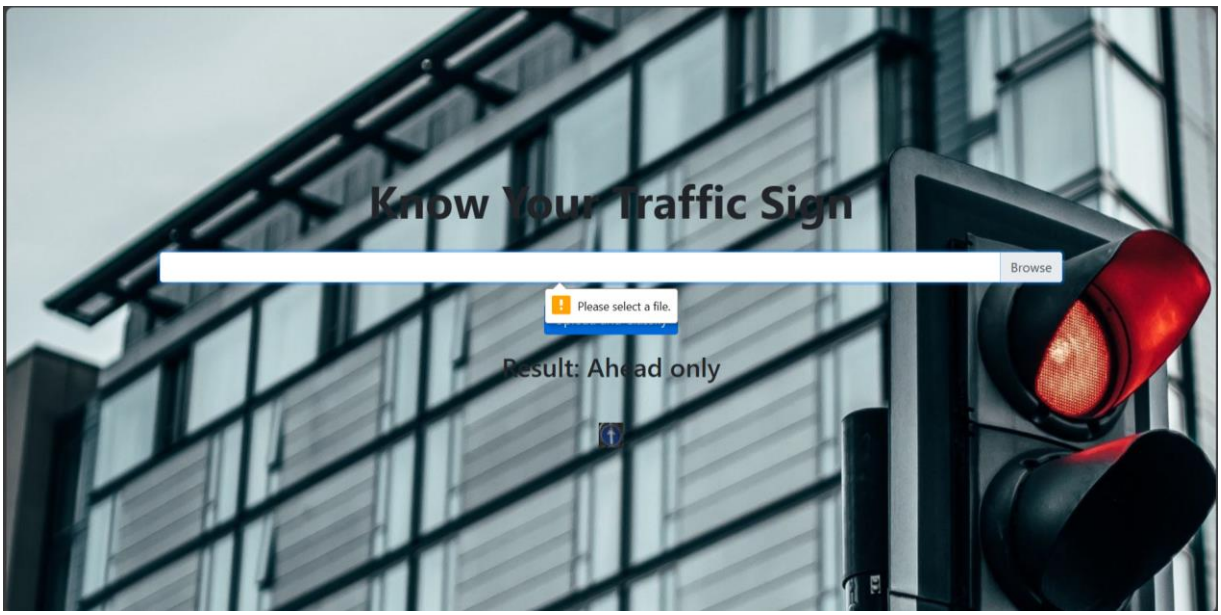
SCREENSHOTS

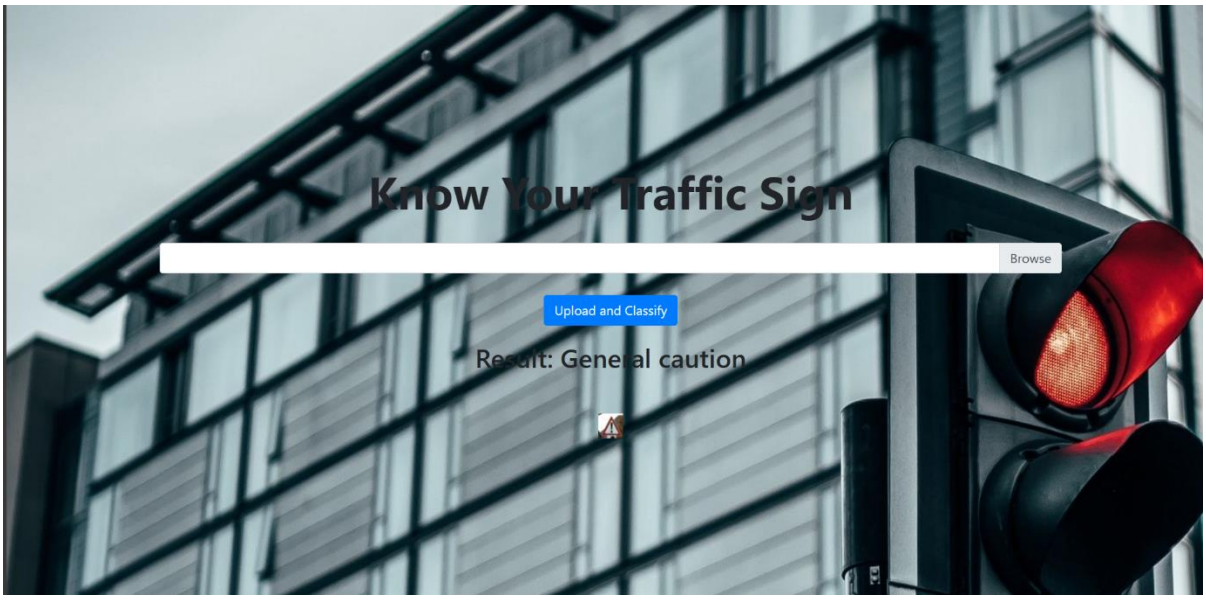
Here I add some sample screenshots of the proposed system which includes:

- User Interface
- Output screen

USER INTERFACE




Know Your Traffic Sign
Browse
Please select a file.
Result: Ahead only


Know Your Traffic Sign
Browse
Upload and Classify
Result: General caution

CHAPTER 8

REFERENCES

1. https://www.analyticsvidhya.com/blog/2021/12/traffic-signs-recognition-using-cnn-and-keras-in-python/

2. https://data-flair.training/blogs/python-project-traffic-signs-recognition/

3. https://ieeexplore.ieee.org/document/10100276

4. P. Nagesh, L. Akhil, K. Rishi and T. S. Bhargav, "Traffic Signs Recognition using CNN and Keras," 2023 International Conference on Innovative Data Communication Technologies and Application (ICIDCA), Uttarakhand, India, 2023, pp. 221-224, doi: 10.1109/ICIDCA56705.2023.10100276.