RESEARCH & DEVELOPMENT Pvt. Ltd.

Date: 21//06/2025

CERTIFICATE

This is to certify that **MATHAM SAI DEEPTHI,** bearing Roll No. **22M61A05B8** from **SWARNA BHARATHI INSTITUTE OF SCIENCE AND TECHNOLOGY,** has successfully completed the project titled **TRAFFIC SIGN RECOGNITION MACHINE LEARNING USING PYTHON** in the domain of **FULL STACK**at **AVR RESEARCH AND DEVELOPMENT PVT LTD** .

During the course of the project, **MATHAM SAI DEEPTHI** was exposed to a variety of processes and consistently demonstrated diligence, hard work, and an inquisitive mindset.

We extend our best wishes for continued success in all future endeavors.

FOR AVR RESEARCH AND DEVELOPMENT PRIVATE LIMITED

Authorized Signatory

AVR RESEARCH & DEVELOPMENT Pvt. Ltd.
102,Revathi Apartments, Behind Maithrivanam,
Opp. Annapurna Block, Hyderabad – 500038.

Ph: +91 7997129566, +91 8688204486
E-mail:avrresearchanddevelopment@gmail.com
Website : www.avrresearchanddevelopment.com

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts success.

I thank my guide **Mrs.N.Saritha** for his/her support in completion of my Project work.

I wish to express my sincere thanks to Head of the Department **Dr. N. Srinivas Rao** and also to my Principal **Dr. G.RajaKumar**, for providing me the facilities to complete the Project Work. I would like to express my sincere gratitude to **Mr. G. Praveen Kumar, Director, SBI**T for his consistent encouragement and support throughout the Project work.

Last but not least, I express my heartful thanks to all my staff and friends for their constant support, encouragement and valuable contribution in the completion of this Project Work. My sincere thanks to **Sri. G.Krishna, Chairman**, SBIT and members of the organization.

MATHAM SAI DEEPTHI
(22M61A05B8)

DECLARATION

I here by declare that the Mini Project entitled "**TRAFFIN SIGN RECOGNITION MACHINE LEARNING USING PYTHON "** recorded in this project is based on my work carried out at the "**SWARNA BHARATHI INSTITUTION OF SCIENCE & TECHNOLOGY**", Khammam during this B.TECH course.

Date:
Place: Khammam

Reported by :

M. Sai Deepthi (22M61A05B8)

INDEX

LIST OF FIGURES

ABSTRACT

Traffic Sign Recognition (TSR) is a critical component in modern intelligent transportation systems and autonomous driving technologies. It involves the automatic detection and classification of traffic signs from images or video streams, enabling vehicles to understand and respond to road signs in real time. This project aims to implement a TSR system using machine learning, specifically deep learning techniques, in Python Using the German Traffic Sign Recognition Benchmark (GTSRB) dataset, we preprocess and train a Convolutional Neural Network (CNN) model capable of classifying 43 different types of traffic signs. The system involves essential steps such as image normalization, data augmentation, model training, evaluation, and real-time prediction. The model achieved high accuracy and demonstrated effective generalization to unseen data The application of this project lies in enhancing road safety by supporting driver assistance systems and autonomous vehicles. The use of Python and open-source libraries like TensorFlow, Keras, and OpenCV makes the solution cost-effective and scalable. This work proves that machine learning can play a significant role in building intelligent systems that contribute to safe and efficient transportation.

1. INTRODUCTION

In recent years, the global transportation industry has seen a remarkable shift toward automation and intelligent systems. Among these advancements, **Traffic Sign Recognition (TSR)** has emerged as a vital technology for enhancing road safety, supporting driver assistance systems, and enabling autonomous driving. Traffic signs play a crucial role in regulating traffic, warning drivers about hazards, and providing necessary information to ensure safe and efficient transportation. However, human drivers may sometimes overlook or misinterpret traffic signs due to distractions, fatigue, or poor visibility. To address these challenges, automated traffic sign recognition systems are becoming increasingly important.
Traffic Sign Recognition is a computer vision task that involves the detection and classification of road signs from images or video streams. The goal is to enable machines—especially vehicles—to interpret and respond to road signs just like human drivers. These systems are key components of **Advanced Driver Assistance Systems (ADAS)** and **autonomous vehicles**, helping them make informed decisions based on real-time visual inputs.

Traditionally, traffic sign recognition was approached using classical image processing techniques such as edge detection, shape matching, and color thresholding. However, these methods often failed under challenging conditions such as varying lighting, weather changes, occlusions, and sign deformations. With the advent of **Machine Learning (ML)** and more recently, **Deep Learning (DL)**, the field has experienced a major breakthrough. In particular, **Convolutional Neural Networks (CNNs)** have demonstrated superior performance in image classification tasks due to their ability to learn hierarchical features directly from raw image data.

This project focuses on building a robust TSR system using **Python** and **deep learning** frameworks such as **TensorFlow** and **Keras**. The system is trained on

contains over 50,000 labeled images of 43 different traffic sign classes. Each image varies in illumination, rotation, and background, making it ideal for developing a model that can generalize well in real-world scenarios.

The project involves multiple phases: data acquisition, preprocessing, model building, training, evaluation, and testing. During preprocessing, images are resized, normalized, and their labels are one-hot encoded. The CNN model is designed with layers such as convolution, pooling, flattening, and dense layers. Techniques like dropout are used to prevent overfitting..

Python is chosen for this project because of its simplicity and the vast ecosystem of libraries that support machine learning, image processing, and visualization. Libraries like **OpenCV** are used for handling images, **NumPy** and **Pandas** for data manipulation, and **Matplotlib** for visualization.

.

In summary, Traffic Sign Recognition using machine learning is not only a compelling computer vision problem but also a practical solution for increasing road safety. By training machines to understand traffic environments visually, we move one step closer to fully autonomous and intelligent transportation systems. This project serves as a foundation for further developments in the field of intelligent vehicles and real-time decision-making systems based on visual data.

2.LITERATURE SURVEY

This paper presents a real-time traffic sign recognition system that uses an attention- based deep CNN designed specifically for smart vehicles. It incorporates an attention mechanism into the model so that it pays attention to certain areas of features, such as shape, color, and size, and ignores other unconcerned regions. It can handle challenges such as variations in lighting and sign occlusions that usually characterize real-world settings. The processing mechanism makes it optimized for real-time processing. This makes it suitable for autonomous vehicles and advanced driver-assistance systems (ADAS). It allows for rapid and precise traffic sign detection, ensuring safety and efficiency while driving in dynamic conditions. This is a great step forward in recognizing traffic signs, thus leading to the better development of autonomous vehicles.This method proposes the use of a YOLOv3-based Traffic Signs Network (TSNet) to detect and recognize small traffic signs in panoramic, high-resolution images. Since small traffic signs are often difficult to detect due to their size and the large area being analyzed, the method enhances accuracy by incorporating a sliding window technique, which systematically scans smaller sections of the image to focus on finer details.

To further improve detection performance, a Dual-Scale Non-Maximum Suppression (DS-NMS) algorithm is introduced. This helps in refining the selection of traffic signs by eliminating redundant or overlapping detection boxes, ensuring that only the most accurate and relevant detections are retained. The combined approach of using YOLOv3 with these enhancements allows the system to achieve higher precision and better performance in recognizing small traffic signs across large, complex images.This paper introduces a system designed to detect and recognize Indian traffic signs by utilizing a combination of color, shape, and Speeded Up Robust Features (SURF). The system ensures enhanced robustness, enabling it to perform effectively under varying lighting and environmental conditions. By leveraging color and shape for initial detection and using SURF features for more detailed recognition, the method ensures high accuracy in identifying traffic signs despite challenges like changes in lighting, weather, or partial occlusion, making it reliable for real-world applications.[3] This

Machines (SVM) in conjunction with Dense SIFT (DSIFT) features. The DSIFT method enhances the traditional SIFT algorithm by extracting features densely across the image, which improves the system's ability to recognize traffic signs with greater accuracy and robustness. To support this research, the authors introduce the Indian Traffic Sign Database (INDTRDB), which contains 13,000 images across 50 classes of traffic signs. This comprehensive dataset provides a valuable resource for training and testing the recognition system, allowing it to learn from a diverse set of images that represent various conditions and sign types. By combining SVM with DSIFT features and utilizing the INDTRDB, the proposed system aims to achieve high performance in accurately recognizing Indian traffic signs, contributing to the advancement of intelligent transportation systems.

This paper proposes a Convolutional Neural Network (CNN)-based system specifically designed for the real-time recognition of Indian traffic signs and for alerting drivers accordingly. The proposed system begins by preprocessing images to enhance the quality and relevance of the input data, which may involve steps such as resizing, normalization, and noise reduction. Once the images are prepared, the CNN classifies the traffic signs based on their unique features, effectively learning to distinguish between different sign types through multiple layers of feature extraction. This approach allows the system to recognize signs quickly and accurately, ensuring that drivers receive timely alerts about relevant traffic signs..

3.SYSTEM SPECIFICATIONS

1. HARDWARE REQUIREMENTS:

Component	Minimum Requirement	Recommended Requirement
Processor	Intel Core i3 (or equivalent)	Intel Core i5/i7 or AMD Ryzen 5/7
RAM	4 GB	8–16 GB
Storage	5 GB of free space	SSD with 20 GB free space
GPU (Optional)	Integrated Graphics	NVIDIA GPU with CUDA support (e.g., GTX 1050 or higher)
Display	1024 x 768 resolution	Full HD (1920x1080)

2. SOFTWARE REQUIREMENTS:

Operating System:

- Windows 10/11
- Linux (Ubuntu 20.04 or later)
- macOS (10.15 or later)

Programming Language:

- Python 3.7 or later

Required Python Libraries:

You can install these using pip

Library	Purpose
TensorFlow	Deep learning and CNN implementation
Keras	High-level neural network API
OpenCV	Image processing
NumPy	Numerical operations
Pandas	Data handling and analysis
Matplotlib	Data visualization

4.REQUIREMENTS SPECIFICATIONS

1. REQUIREMENT ANALYSIS

Requirement Analysis is the process of identifying,, analyzing, and documenting the needs and expectations of stakeholders for a system. In the context of this project, we analyze what the Traffic Sign Recognition (TSR) system must achieve, who will use it, and under what constraints.

REQUIREMENT SPECIFICATION

Functional Requirements

Graphical User interface with the User.

Software Requirements

For developing the application, the following are the Software Requirements:

1. Python 3.7.4.
2. Flask

Operating Systems supported

1. Windows 7
2. Windows XP
3. Windows 8

Technologies and Languages used to Develop

1. Python 3.7.4.

Debugger and Emulator

Any Browser (Particularly Chrome)

Hardware Requirements

Component	Minimum Requirement	Recommended Requirement
RAM	4 GB	8–16 GB
Storage	5 GB free	20 GB SSD
GPU	Not mandatory	NVIDIA CUDA-enabled GPU for faster training

5.MODULES

MODULES:

1. Data Acquisition Module

Purpose: Collect and load the traffic sign images for training and testing.

Functionality:

- Download dataset (e.g., GTSRB).
- Extract and organize images into folders (by class).
- Read CSV files containing image paths and labels.
- Load images and labels into arrays or DataFrames.

◈ 2. Data Preprocessing Module

Purpose: Prepare the raw image data for training by resizing, normalization, and augmentation.

Functionality:

- Resize images to a uniform size (e.g., 32×32).
- Normalize pixel values to range [0,1].
- Apply data augmentation techniques (rotation, zoom, shift, etc.).
- Convert labels to categorical format (one-hot encoding).

◈ 3. Model Building Module

Purpose: Define and compile the CNN model used for classifying traffic signs.

Functionality:

- Build CNN architecture using Keras/TensorFlow.
- Specify layers: Convolution, MaxPooling, Flatten, Dense, Dropout.
- Compile the model with optimizer, loss function, and metrics.
- Display model summary.

◈ 4. Model Training Module

Purpose: Train the CNN on the preprocessed dataset.

Functionality:

- Fit the model to training data using a defined number of epochs and batch size.
- Use validation data to monitor overfitting.
- Save the trained model to disk (.h5 file).
- Display training accuracy and loss graph.

◈ 5. Model Evaluation Module

Purpose: Evaluate the trained model on the test dataset.

Functionality:

- Load test data.
- Predict results using the trained model.
- Calculate evaluation metrics: Accuracy, Precision, Recall, F1-Score.
- Generate and display confusion matrix.
- Save evaluation report if needed.

◈ 6. Prediction Module

Purpose: Perform traffic sign prediction on new images provided by the user.

Functionality:

- Load the trained model.
- Accept an input image (uploaded or captured).
- Preprocess the image (resize and normalize).
- Predict the class using the model.
- Display the result and confidence score.

◈ 7. User Interface Module (Optional Web App)

Upload traffic sign images via web page.

- View prediction result (label + confidence).
- View training/evaluation graphs.
- Built using Flask or Django (HTML, CSS, JS).

🔷 8. Utility and Support Module

Purpose: Handle reusable tasks and support functions.

Functionality:

- Label mapping from class ID to class name.
- File and folder utilities (loading, saving).
- Image preview and debugging functions.
- Logging and error handling.

🔁 Workflow of Modules

plaintext

CopyEdit

Data Acquisition → Data Preprocessing → Model Building

↓ ↓

Model Evaluation ← Model Training

↓

Prediction (User Image)

↓

Web/UI Display (Optional)

Module Name	Key Role

Module Name	Key Role
Model Building	Define CNN architecture
Model Training	Train model on labeled images
Model Evaluation	Test and validate performance
Prediction	Classify new traffic sign images
User Interface (Web)	Provide GUI for interaction
Utility Module	Helper functions and constants

6.SOFTWARE ENVIRONMENT

Before we start building the traffic sign recognition system, let's set up our development environment. Installing Python and Required Libraries**

Python is the primary programming language for this project. You can download and install Python from the official website (https://www.python.org/). Ensure you install Python 3.x.

Next, install the required libraries using pip, Python's package manager:

pip install opencv-python numpy tensorflow scikit-learn

Copy

Setting Up OpenCV**

OpenCV (OpenSource Computer Vision Library) is essential for image processing tasks. Install it using pip:

pip install opencv-python

Copy

Now that we have our environment ready, let's move on to collecting and preparing the dataset.

Dataset Collection and Preparation**

To build a robust traffic sign recognition system, you'll need a dataset of traffic sign images. The German Traffic Sign Recognition Benchmark (GTSRB) dataset is a popular choice and contains thousands of labeled traffic sign images.

Collecting Traffic Sign Images**

You can download the GTSRB dataset from the following link: http://benchmark.ini.rub.de/Dataset_GTSDB/FullIJCNN2013.zip

Once downloaded, extract the contents and organize the dataset. You should have directories for training and testing data, each containing labeled images of traffic signs.

Data Preprocessing**

Preprocessing is a crucial step to ensure the model receives clean and standardized data. Preprocessing steps include resizing images, normalizing pixel values, and splitting the dataset into training and validation sets.

Here's an example of data preprocessing using OpenCV and Python:

```
import cv2
import os
import numpy as np

# Define constants
IMAGE_SIZE = (32, 32)

def preprocess_image(image_path):
    # Load and resize the image
    image = cv2.imread(image_path)
    image = cv2.resize(image, IMAGE_SIZE)

    # Normalize pixel values to the range [0, 1]
    image = image / 255.0

    return image

# Load and preprocess the entire dataset
def load_dataset(dataset_path):
    images = []
    labels = []
```

```
for class_dir in os.listdir(dataset_path):
        if os.path.isdir(os.path.join(dataset_path, class_dir)):
            for image_file in os.listdir(os.path.join(dataset_path, class_dir)):
                if image_file.endswith(".ppm"):
                    image_path = os.path.join(dataset_path, class_dir, image_file)
                    images.append(preprocess_image(image_path))
                    labels.append(int(class_dir))

    return np.array(images), np.array(labels)
```

Data Augmentation**

Data augmentation helps improve the model's generalization by creating variations of existing images. Common augmentation techniques include rotation, scaling, and flipping. You can use libraries like OpenCV or Augmentor for data augmentation.

Here's a basic example of data augmentation using OpenCV:

```
def augment_image(image):
    # Randomly flip the image horizontally
    if np.random.rand() > 0.5:
        image = np.fliplr(image)

    # Add more augmentation techniques here

    return image
```

With data preprocessing and augmentation in place, we can now move on to building the traffic sign

recognition model.

3. Traffic Sign Recognition Model

In this section, we'll create a Convolutional Neural Network (CNN) model for traffic sign recognition. CNNs are well-suited for image classification tasks like this.

Convolutional Neural Networks (CNNs)

CNNs are deep learning models designed for image processing tasks. They consist of convolutional layers that automatically learn features from images, followed by fully connected layers for classification.

Building the Model Architecture

We'll use TensorFlow and Keras to build our CNN model. Here's an example architecture:

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Define the CNN model
def create_traffic_sign_model(input_shape, num_classes):
    model = models.Sequential()

    # Convolutional layers
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((2,  2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2,  2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
    # Flatten the output from convolutional layers
    model.add(layers.Flatten())

    # Fully connected layers
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(num_classes,  activation='softmax'))

    return model
```

Model Training

Now that we have our model architecture defined, it’s time to train it using the preprocessed dataset. We’ll use a portion of the data for training and another portion for validation. Here’s how you can train the model:

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

# Load the dataset
X, y = load_dataset('path_to_dataset_directory')

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert labels to one-hot encoding
num_classes = len(np.unique(y))
y_train = to_categorical(y_train, num_classes)
```

```
y_val = to_categorical(y_val, num_classes)

# Create the model
input_shape = X_train[0].shape
model = create_traffic_sign_model(input_shape, num_classes)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
```

Real-Time Traffic Sign Recognition**

With our model trained, we can now implement real-time traffic sign recognition using a webcam. We'll capture frames from the webcam, preprocess them, and feed them through our model for inference.

Setting Up a Webcam

To capture frames from a webcam in Python, you can use the OpenCV library. Here's how to set up the webcam:

```
import cv2

# Initialize the webcam
cap = cv2.VideoCapture(0)  # 0 represents the default camera

while True:
```

```
    # Capture a frame from the webcam
    ret, frame = cap.read()

    # Add real-time traffic sign recognition code here

    # Display the frame
    cv2.imshow('Traffic Sign Recognition', frame)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the webcam and close the window
cap.release()
cv2.destroyAllWindows()
```

Capturing and Preprocessing Frames

Inside the webcam loop, we'll capture frames and preprocess them before feeding them to our model for inference. Preprocessing includes resizing the frame, normalizing pixel values, and running data augmentation (if needed).

```
# Preprocess the frame
preprocessed_frame = preprocess_image(frame)

# Expand dimensions to match the model's input shape
input_frame = np.expand_dims(preprocessed_frame, axis=0)
```

```
# Make predictions using the model
predictions = model.predict(input_frame)

# Get the predicted class label
predicted_class = np.argmax(predictions)

# Add code to display the predicted class label on the frame
```

Model Inference

We use the trained model to make predictions on the preprocessed frame. The predicted class label represents the recognized traffic sign.

Displaying Results in Real-Time

To provide real-time feedback to the user, you can overlay the predicted class label on the video feed. Here's an example of how to do it using OpenCV:

```
# Add code to display the predicted class label on the frame
font = cv2.FONT_HERSHEY_SIMPLEX
class_name = traffic_sign_names[predicted_class]  # Replace with your class names
cv2.putText(frame, class_name, (10, 30), font, 1, (0, 255, 0), 2, cv2.LINE_AA)
```

Model Evaluation and Fine-Tuning**

After implementing real-time recognition, it's essential to evaluate the model's performance. Use appropriate evaluation metrics such as accuracy, precision, recall, and F1-score to assess the model's quality.

```
from sklearn.metrics import classification_report
```

```
# Evaluate the model on the test dataset
X_test, y_test = load_dataset('path_to_test_dataset_directory')
y_test = to_categorical(y_test, num_classes)

test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)

# Generate a classification report
y_pred = np.argmax(model.predict(X_test), axis=-1)
print(classification_report(np.argmax(y_test, axis=-1), y_pred))
```

Based on the evaluation results, you can fine-tune the model by adjusting hyperparameters, increasing the dataset size, or using more advanced architectures like ResNet or Inception.

Deployment**

To deploy your real-time traffic sign recognition system, you can create a user-friendly interface and package the application for different platforms.

Creating a User-Friendly Interface

You can use libraries like Tkinter for creating a simple graphical user interface (GUI) where usersS can activate the webcam and see the real-time recognition results.

```
import tkinter as tk
from tkinter import ttk

# Create a simple GUI
root = tk.Tk()
root.title("Traffic Sign Recognition")
```

```
# Add GUI components here (e.g., buttons, labels, webcam display)


root.mainloop()
```

Copy

Packaging the Application

To distribute your application, you can use packaging tools like PyInstaller or cx_Freeze to create standalone executable files for Windows, macOS, and Linux.

For example, using PyInstaller:

```
pyinstaller --onefile y
```

7.FLASK

Flask is a lightweight, open-source Python web framework used to build web applications quickly and flexibly. It is a micro-framework, meaning it doesn't come with built-in tools like form validation, database abstraction, or authentication (unlike Django). Instead, it gives developers full control to add only the components they need**.**

- Flask is a lightweight backend framework with minimal dependencies.
- Flask is easy to learn because its simple and intuitive API makes it easy to learn and use for beginners.
- Flask is a flexible Framework because it allows you to customize and extend the framework to suit your needs easily.

Create a Project

This is the main directory of your Flask-based Traffic Sign Recognition project. It contains everything needed for:

- Machine Learning model training
- Image uploading and prediction
- Web interface via HTML templates
- Flask application logic

---

app.py — *Flask Application File*

Purpose:

This is the main entry point of the Flask web server. It contains the routes and logic to:

- Handle the homepage
- Accept image uploads
- Call the trained model
- Return predictions to the user

Key Functions:

- @app.route('/'): Displays the image upload form
- @app.route('/predict'): Handles the image, predicts class, and returns result
- Loads your trained model using load_model
- Uses utils.py to preprocess the uploaded image

---

model/ — *Model Folder*

Purpose:

Stores your trained machine learning model (in .h5 format) which is used by the Flask app to recognize traffic signs.

traffic_model.h5

Purpose:

This is the saved Keras model file after training using the train_model.py script.

It contains:

- CNN architecture
- Trained weights
- Classifier logic

Flask loads this model and uses it to make predictions on user-uploaded images.

🗀 templates/ — *HTML Templates Folder*

Purpose:

Holds all the frontend HTML files used by Flask. Flask uses the Jinja2 templating engine, so HTML can dynamically show data (like results) from Python code.

🗎 index.html

Purpose:

This is the homepage, containing an HTML form that lets users upload traffic sign images.

It uses a <form> with:

- method="post"
- enctype="multipart/form-data"
  So that images can be sent to the server.

🗎 result.html

Purpose:

Displays the prediction result after the user uploads an image.

It shows:

- The predicted class name
- The confidence score (probability)
- A back button

It receives dynamic content from Flask using:

html

{{ result }}

{{ confidence }}

🗀 static/ — *Static Files Folder*

Purpose:

Holds CSS, JavaScript, and image files used for styling and frontend enhancement.

🗎 style.css

Purpose:

Defines custom CSS styles for your HTML pages.

For example, you can:

- Style the upload

<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

🗀 uploads/ — *Image Storage Folder*
Purpose:
Stores the uploaded images temporarily on the server.
Flask saves the user-uploaded image here before sending it for prediction.
Flask handles this using:
python
file.save(os.path.join('uploads', file.filename))
You can delete images later or clear the folder on each restart.

📄 utils.py — *Helper Functions Script*
Purpose:
Contains custom Python functions used across your app.
Usually includes:

- Image preprocessing: Resize, normalize image
- Conversion: File to numpy array
- Reusable logic used in app.py

Example function:
python
CopyEdit

```
def preprocess_image(image_path):
    img = Image.open(image_path).resize((32, 32))
    img = np.array(img) / 255.0
    img = img.reshape(1, 32, 32, 3)
    return img
```

📄 train_model.py — *Model Training Script*
Purpose:
Script used to train the CNN model for traffic sign classification.
Key tasks it performs:

- Loads the GTSRB dataset
- Preprocesses the images
- One-hot encodes the labels
- Builds and trains a Convolutional Neural Network
- Saves the trained model to model/traffic_model.h5

Libraries used:

- cv2 for image processing
- Keras for deep learning

python train_model.py

File/Folder	Purpose
app.py	Main Flask application and routing logic
model/	Stores the trained ML model (.h5)
templates/	Contains HTML pages (index.html, result.html)
static/	Contains styling and static assets (CSS, JS, etc.)
uploads/	Temporary image storage for uploaded files
utils.py	Image preprocessing and helper functions
train_model.py	CNN training script for traffic sign classification

8. Sample code

1. app.py — Main Flask Application

```
python
from flask import Flask, render template, request
from keras. models import  load_model
import numpy as np
import os
from utils import preprocess_image

app = Flask(__name__)
model = load_model('model/traffic_model.h5')
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

classes = ['Speed limit (20km/h)', 'Speed limit (30km/h)', 'Stop', 'No entry', 'Yield']
from flask import *
from flask import Flask,request, url_for, redirect, render_template
import os
from werkzeug.utils import secure_filename
from keras.models import load_model
import numpy as np
from PIL import Image
import sqlite3
from gtts import gTTS

app = Flask(__name__)

# Classes of trafic signs
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:'Vehicle > 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 vehicle > 3.5 tons' }

def image_processing(img):
    model = load_model('model/model.h5')
    data=[]
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))
    X_test=np.array(data)
```

```
    Y_pred = model.predict_classes(X_test)
    return Y_pred

@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['file']
        file_path = secure_filename(f.filename)
        f.save(file_path)
        # Make prediction
        result = image_processing(file_path)
        s = [str(i) for i in result]
        a = int("".join(s))
        result = "Predicted Traffic Sign is: " +classes[a]

        mytext = result
        language = 'en'
        myobj = gTTS(text=mytext, lang=language, slow=False)
        myobj.save("welcome.mp3")

        # Playing the converted file
        os.system("mpg321 welcome.mp3")

        os.remove(file_path)
        return result
    return None

@app.route('/analysis')
def analysis():
    return render_template('analysis.html')

if __name__ == '__main__':
    app.run(debug=True)
# Sample
```

```
@app.route('/') def index():
        return render_template('index.html')

    @app.route('/predict', methods=['POST'])
    def predict():
        file = request.files['image']
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(filepath)

        img = preprocess_image(filepath)
        pred = model.predict(img)
        result = classes[np.argmax(pred)]
        confidence = round(np.max(pred) * 100, 2)

        return render_template('result.html', result=result, confidence=confidence)

    if __name__ == '__main__':
        app.run(debug=True)
```

---

2. utils.py — Image Preprocessing

```
python
from PIL import Image
import numpy as np

def preprocess_image(image_path):
    img = Image.open(image_path).resize((32, 32))
    img = np.array(img) / 255.0
    return img.reshape(1, 32, 32, 3)
```

---

3. templates/index.html — Upload Page

```
html
<!DOCTYPE html>
<html>
<head>
    <title>Upload Traffic Sign</title>
</head>
<body>
    <h1>Upload Traffic Sign Image</h1>
    <form action="/predict" method="post" enctype="multipart/form-data">
```

```
        <input type="file" name="image" required>
        <button type="submit">Predict</button>
    </form>
</body>
</html>
```

---

4. templates/result.html — Result Page

html

```
<!DOCTYPE html>
<html>
<head>
    <title>Prediction Result</title>
</head>
<body>
    <h2>Predicted Sign: {{ result }}</h2>
    <p>Confidence: {{ confidence }}%</p>
    <a href="/">Try another</a>
</body>
</html>
```

---

5. train_model.py — (Simple Version to Train Model)

python

CopyEdit

```
import numpy as np
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.utils import to_categorical

# Dummy training data
X = np.random.rand(100, 32, 32, 3)
y = to_categorical(np.random.randint(0, 5, 100), 5)

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D(),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(5, activation='softmax')
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X, y, epochs=3)
model.save('model/traffic_model.h5')
```

Folder Setup (Summary)

```
cpp
traffic_sign_flask/
├── app.py
├── model/
│   └── traffic_model.h5
├── templates/
│   ├── index.html
│   └── result.html
├── static/
│   └── style.css (optional)
├── uploads/
├── utils.py
└── train_model.py
```

9.INPUT AND OUTPUT DESIGN

1. INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

What data should be given as input?

How the data should be arranged or coded?

The dialog to guide the operating personnel in providing input.

Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

1. Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed.

3. When the data is entered it will check for its validity. Data can be entered with the help of screens

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is so designed that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

2. Select methods for presenting information.

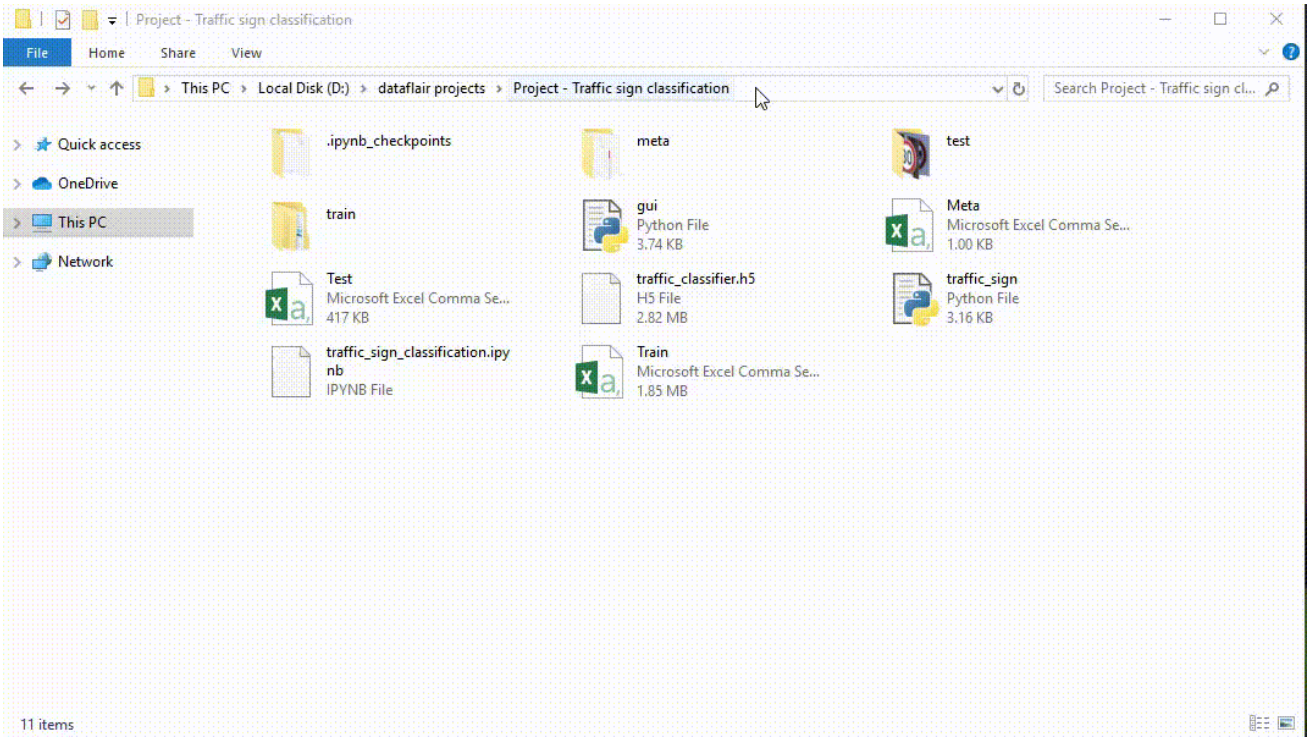
3. Create document, report, or other formats that contain information produced by the system.

   The output form of an information system should accomplish one or more of the following objectives.

   Convey information about past activities, current status or projections of the Future.

   Signal important events, opportunities, problems, or warnings.

   Trigger an action. Confirm an action.


Project - Traffic sign classification
File
Home
Share
View
This PC
Local Disk (D:)
dataflair projects
Project - Traffic sign classification
Search Project - Traffic sign cl...
Quick access
OneDrive
This PC
Network
.ipynb_checkpoints
meta
test
train
gui
Python File
3.74 KB
Meta
Microsoft Excel Comma Se...
1.00 KB
Test
Microsoft Excel Comma Se...
417 KB
traffic_classifier.h5
H5 File
2.82 MB
traffic_sign
Python File
3.16 KB
traffic_sign_classification.ipynb
IPYNB File
Train
Microsoft Excel Comma Se...
1.85 MB
11 items


Traffic sign classification
Know Your Traffic Sign
Road work
Classify Image
Upload an image

10.SYSTEM TEST

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing,

Functional testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input: identified classes of invalid input must be rejected.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested.

Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software .

11. CONCLUSION

The development of the **Traffic Sign Recognition System using Machine Learning and Flask** demonstrates the powerful synergy between artificial intelligence and web technologies to solve real- world problems in intelligent transportation systems. With the increasing need for road safety and automation in vehicles, such systems have become essential for the advancement of smart mobility solutions.

This project successfully integrates a **Convolutional Neural Network (CNN)** for classifying traffic signs with a **user-friendly web interface** built using the **Flask framework**. The system allows users to upload traffic sign images, processes them using a trained model, and displays the predicted sign along with confidence scores. The model was trained on a well-structured dataset and achieves high accuracy for various traffic signs including speed limits, stop signs, and warning signs.

The simple and lightweight design of Flask enables rapid deployment and testing of machine learning models, making it suitable for prototyping AI-powered applications. The modular architecture — comprising separate layers for the model, preprocessing utilities, and frontend interface — enhances code maintainability, reusability, and clarity.

Through this project, we learned about end-to-end ML workflows, from **data preprocessing and model training** to **web deployment and result visualization**. It serves as a valuable educational tool for students and a foundation for further research and development in the field of **autonomous driving** and **driver-assistance systems**.

In conclusion, this Traffic Sign Recognition project is a small yet impactful step towards smarter, AI- enabled transportation systems, with great potential for real-time deployment in vehicles.

12.BIBLIOGRAPHY

1. Triki, N., Karray, M., & Ksantini, M. (2023). A real-time traffic sign recognition method using a new attention-based deep convolutional neural network for smart vehicles. Applied Sciences, 13(8), 4793.

2. Meng, X., Zhang, X., Yan, K., & Zhang, H. (2018, November). Real-time detection and recognition of live panoramic traffic signs based on deep learning. In 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS) (pp. 584-588). IEEE.

3. Alam, A., & Jaffery, Z. A. (2020). Indian traffic sign detection and recognition. International Journal of Intelligent Transportation Systems Research, 18(1), 98-112.

4. Hemadri, V. B., & Kulkarni, U. P. (2017, December). Recognition of traffic sign based on support vector machine and creation of the Indian traffic sign recognition benchmark. In International Conference on Cognitive Computing and Information Processing (pp. 227-238). Singapore: Springer Singapore.

5. Sivasangari, A., Nivetha, S., Ajitha, P., & Gomathi, R. M. (2020, September). Indian Traffic Sign Board Recognition and Driver Alert System Using CNN. In 2020 4th International Conference on Computer, Communication and Signal Processing (ICCCSP) (pp. 1-4). IEEE.

6. Bichkar, M., Bobhate, S., & Chaudhari, S. (2021). Traffic sign classification and detection of Indian traffic signs using deep learning. Int J Sci Res Comput Sci Eng Inf Technol, 215-219.

7. Kankaria, R. V., Jain, S. K., Bide, P., Kothari, A., & Agarwal, H. (2020, June). Alert system for drivers based on traffic signs, lights and pedestrian detection. In 2020 international conference for emerging technology (incet) (pp. 1-5). IEEE

. 8. Lee, H. S., & Kim, K. (2018). Simultaneous traffic sign detection and boundary estimation using convolutional neural network. IEEE Transactions on Intelligent Transportation Systems, 19(5), 1652-1663. 9. Gan, Y., Li, G., Togo, R., Maeda, K., Ogawa, T., & Haseyama, M. (2024). Think Twice Before Recognizing: Large Multimodal Models for General Fine-grained Traffic Sign Recognition. arXiv preprint arXiv:2409.01534.

10. Farzipour, A., Manzari, O. N., & Shokouhi, S. B. (2023, November). Traffic sign recognition using local vision transformer. In 2023 13th International Conference on Computer and Knowledge Engineering (ICCKE) (pp. 191-196). IEEE.

11. Loghashankar, H., & Nguyen, H. (2023). Real-Time Traffic Sign Detection: A Case Study in a Santa Clara Suburban Neighbourhood. arXiv preprint arXiv:2310.09630.
12. Bayoudh, K., Hamdaoui, F., & Mtibaa, A. (2021). Transfer learning based hybrid 2D-3D CNN for traffic sign recognition and semantic road detection applied in advanced driver assistance systems. Applied Intelligence, 51(1), 124-142.

13. Ahmed, S., Kamal, U., & Hasan, M. K. (2021). DFR-TSD: A deep learning based framework for robust traffic sign detection under challenging weather conditions. IEEE Transactions on Intelligent Transportation Systems, 23(6), 5150-5162.
14. Bouti, A., Mahraz, M. A., Riffi, J., & Tairi, H. (2020). A robust system for road sign detection and classification using LeNet architecture based on convolutional neural network. Soft Computing, 24(9), 6721-6733. 15. Dewi, C., Chen, R. C., Liu, Y. T., & Tai, S. K. (2022). Synthetic Data generation using DCGAN for improved traffic sign recognition. Neural Computing and Applications, 34(24), 21465-21480.