

**A MINI - PROJECT REPORT ON**  
**Automated Traffic Sign Classification using Python & Deep**  
**Learning**

Submitted in partial fulfillment for the degree of  
Bachelor of Technology in  
Computer Science and Technology

Submitted by  
Kashish Singh (63)  
Gayatribala Balamurgan (64)  
Shreya Yadav(70)

Group No. 3  
Under the guidance of  
Prof. Prajakta Gotarne



**USHA MITTAL INSTITUTE OF TECHNOLOGY**

**S.N.D.T WOMEN'S UNIVERSITY**

**MUMBAI – 400049**

**2024 – 2025**

## **CERTIFICATE**

This is to certify that **Kashish Singh(63), Gayatribala Balamurgan(64), Shreya Yadav(70)** (Group No. 3), has successfully completed Mini-Project phase-1 project work on "**Automated Traffic Sign Classification using Python & Deep Learning**" in partial fulfillment of the B.Tech. degree in **Computer Science and Technology** during the year 2024-25 as prescribed by SNDT Women's University.

(Project Guide)

Prof. Prajakta Gotarne

(Head of Department)

Prof. Kumud Wasnik

Principal

Dr. Yogesh Nerkar

Examiner 1

Examiner 2

# Acknowledgement

I have a great pleasure to express my gratitude to all those who have contributed and motivated during my project work. We are highly indebted to Prof. Prajakta Gotarne for her guidance and constant supervision as well as for providing necessary information regarding the project also for her support in this project. Without her help, this project would not have been possible. We would also like to thank our Head of Department Prof.Kumud Wasnik for her insights in shaping the direction and content of this report. We would also like to extend our gratitude to our classmates who provided valuable feedback and encouragement throughout the process. Finally, we would like to express our appreciation to our families for their unwavering support and understanding during the long hours and late nights spent working on this project. Thank you all for your contributions, support, and encouragement.

Date:Sep 5,2024

Name of Candidates

Kashish Singh(63)

Gayatribala Balamurgan(64)

Shreya Yadav(70)

# Contents

<b>1 Problem statement</b>	<b>5</b>
<b>2 Literature Survey</b>	<b>6</b>
<b>3 Introduction</b>	<b>7</b>
<b>4 Existing System</b>	<b>8</b>
4.1 Feature Extraction & Learning . . . . .	8
4.2 Real-Time Performance . . . . .	8
4.3 Accuracy and Robustness . . . . .	8
4.4 Model Interpretability and Transparency . . . . .	9
4.5 Scalability and Adaptability . . . . .	9
<b>5 Proposed System</b>	<b>10</b>
5.1 Feature Extraction & Learning . . . . .	10
5.2 Real Time Performance . . . . .	10
5.3 Accuracy and Robustness . . . . .	10
5.4 Model Interpretability and Transparency . . . . .	11
5.5 Scalability and Adaptability . . . . .	11
<b>6 Architectural Overview</b>	<b>12</b>
<b>7 Hardware &amp; Software requirement</b>	<b>14</b>
7.1 Development Environment . . . . .	14
7.2 Deep Learning Libraries . . . . .	14
7.3 Data Handling and Preprocessing . . . . .	14
7.4 Visualization Tools . . . . .	15
7.5 Data Augmentation Tools . . . . .	15
7.6 Hardware Acceleration . . . . .	15
7.7 Version Control . . . . .	15
7.8 Deployment Tools (Optional) . . . . .	16

7.9	Operating System . . . . .	16
7.10	Dataset Management . . . . .	16
<b>8</b>	<b>Implementation Details</b>	<b>17</b>
<b>9</b>	<b>Applications</b>	<b>20</b>
<b>10</b>	<b>Scope/Future Work:</b>	<b>21</b>
<b>11</b>	<b>Conclusion:</b>	<b>23</b>
<b>12</b>	<b>References:</b>	<b>24</b>

## Abstract

This report presents the development and deployment of an automated traffic sign classification system utilizing Python and deep learning techniques, specifically convolutional neural networks (CNNs). Traffic sign recognition (TSR) is an essential component in modern road safety and autonomous driving technologies, as accurate and real-time recognition of traffic signs helps to reduce human error and prevent accidents. Leveraging the dataset, which comprises of 51,838 images of 43 different classes of traffic signs, we employ Convolutional Neural Networks (CNNs) to build a robust classification model. The model is designed to process and interpret visual information from traffic environments, enabling vehicles or driver assistance systems to make informed decisions based on the recognized signs. The dataset is preprocessed through image resizing and normalization, and then split into training and testing sets to enhance the model's accuracy and robustness. The core of the project is a CNN model built using TensorFlow and Keras, designed to handle various transformations and complexities within traffic sign images. The architecture includes layers of convolutional, pooling, and dropout functions that effectively capture features and reduce overfitting, respectively. After extensive training, the model achieved high accuracy, demonstrating its suitability for deployment in real-world applications. To make this model accessible and scalable, it is integrated into a Flask web application, allowing users to interact with the model via a simple interface. This application is then deployed on the cloud platform, enabling real-time traffic sign classification through a browser-based interface. Users can upload images of traffic signs to receive instant classification results, showcasing the model's potential in live scenarios.

# 1 Problem statement

Traffic sign recognition is a crucial task in the development of autonomous driving systems, advanced driver-assistance systems (ADAS), and road safety technologies. The ability to accurately identify and classify traffic signs in real-time enables vehicles to make informed decisions, follow regulations, and adapt to dynamic driving environments, thereby enhancing safety and driving efficiency. However, this task is inherently complex due to the diversity of traffic signs, environmental conditions, and the need for quick processing to match real-time requirements. These are the key challenges:

1. Diverse Sign Variations: Traffic signs vary significantly across countries and even regions, with differences in shape, colour, text, and pictorial representation. These variations create a need for models that can generalize across multiple types and designs of traffic signs.
2. Environmental Conditions: Traffic signs may appear in varied lighting conditions (e.g., daytime, nighttime, cloudy), weather conditions (e.g., rain, fog, snow), and road backgrounds (e.g., urban, rural). These factors can obscure sign features, reduce visibility, and affect colour recognition, making it challenging for models to identify signs accurately.
3. Occlusions and Partial Visibility: Traffic signs may be partially covered by obstacles like trees, poles, or other vehicles. This partial visibility requires the model to be robust enough to recognize a sign even when only part of it is visible.
4. Variety in Size and Distance: Traffic signs appear in different sizes and at varying distances, which impacts their perceived size and clarity in images. The classification model must be able to identify signs accurately regardless of these variations.

## 2 Literature Survey

Paper Title	Authors	Observations
TRAFFIC SIGN RECOGNITION USING CONVOLUTIONAL NEURAL NETWORKS,(Year-2018)	Ervin Miloš, Aliaksei Kolesau, Dmitrij Šešok	The paper shows that using CNNs with preprocessing, batch normalisation, dropout, and data augmentation significantly improves traffic sign recognition accuracy, achieving up to 99.24%. The proposed approach outperforms many standard models but suggests future work with spatial transformer layers for even better results.
Flexible, High Performance Convolutional Neural Networks for Image Classification	Dan C. Ciresan et al.	Observation 1: The paper focuses on using (CNNs) for image classification, which aligns well with our project's goal. Observation 2: The authors highlight the benefits of using GPUs for efficient CNN training. Observation 3: The paper emphasizes the importance of deep CNN architectures with multiple layers for achieving high accuracy in image classification tasks.
The German Traffic Sign Recognition Benchmark: A multi-class classification competition (Year-2011)	Johannes Stallkamp, Marc Schlipsing, Jan Salmen, Christian Igel	GTSRB is a valuable benchmark: Its large, diverse dataset and precomputed features make it a valuable resource for traffic sign recognition research. Class imbalance and baseline performance: Addressing class imbalance and comparing algorithms to baseline methods could provide deeper insights. Evolving field: Traffic sign recognition has likely progressed since 2011, and newer datasets might be available.

### Literature Survey

### 3 Introduction

In recent years, the rapid advancement of autonomous vehicles and advanced driver assistance systems (ADAS) has brought the need for reliable and efficient traffic sign recognition into sharp focus. Traffic signs play a crucial role in regulating, warning, and guiding drivers, and their accurate interpretation is essential for the safe operation of vehicles on the road. For autonomous vehicles, the ability to recognize and respond to traffic signs is a fundamental requirement, ensuring compliance with traffic laws and enhancing road safety. Traditional methods for traffic sign recognition have relied heavily on manual feature extraction and classical machine learning techniques, which often struggle with variability in sign appearance due to factors such as lighting conditions, weather changes, occlusions, and varying angles of view. These challenges necessitate the development of more robust and accurate recognition systems capable of operating in real-world conditions.

With the advent of deep learning, particularly Convolutional Neural Networks (CNNs), there has been a paradigm shift in image recognition tasks, including traffic sign recognition. CNNs have demonstrated superior performance in automatically learning hierarchical features from raw pixel data, leading to significant improvements in accuracy and robustness. This project aims to leverage the power of CNNs to develop a traffic sign recognition system that can classify traffic signs with high accuracy and operate effectively in real-time, even under challenging environmental conditions.

Using the dataset, which is widely recognized for its comprehensive collection of traffic sign images, this project will focus on designing, training, and evaluating a CNN-based model. The model will be trained to recognize 43 different types of traffic signs, encompassing a wide range of categories, including prohibitory, mandatory, and danger signs.

This project not only aims to build a high-performing traffic sign recognition model but also seeks to contribute to the broader field of autonomous driving technology by addressing key challenges such as real-time processing, model generalization, and deployment feasibility. Through this work, we hope to advance the capabilities of traffic sign recognition systems, paving the way for safer and more reliable autonomous vehicles on our roads.

## 4 Existing System

### 4.1 Feature Extraction & Learning

- Traditional Methods Rely on handcrafted feature extraction techniques like Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), and color or shape-based detection. These methods are limited by their dependency on manually selected features, which may not generalize well across different conditions.
- Deep Learning Methods Use Convolutional Neural Networks (CNNs) to automatically learn features from raw image data. These models are more robust than traditional methods but often require significant computational resources and large datasets for training.

### 4.2 Real-Time Performance

- Traditional methods typically struggle with real-time performance due to the separate detection and classification stages and the manual feature extraction process.
- Deep learning-based systems have improved real-time performance but can still be limited by the computational demands of deep networks, especially in resource-constrained environments.

### 4.3 Accuracy and Robustness

- Traditional systems may suffer from lower accuracy and robustness due to their reliance on predefined features that may not capture the complexity of real-world traffic sign variations.
- Deep learning models offer improved accuracy but can still face challenges with robustness, particularly when encountering uncommon sign appearances or adverse conditions.

## **4.4 Model Interpretability and Transparency**

- Traditional systems are more interpretable, as the feature extraction process is manual and well-understood, but this comes at the cost of limited flexibility and generalization.
- Deep learning systems, while powerful, are often considered "black boxes," making it difficult to understand the decision-making process.

## **4.5 Scalability and Adaptability**

- Traditional methods are less scalable as they require significant adjustments to handle new types of traffic signs or adapt to new environments.
- Deep learning models are more adaptable but may require retraining with new data to maintain high performance.

## 5 Proposed System

### 5.1 Feature Extraction & Learning

- The proposed system will employ an optimized deep learning approach using a tailored CNN architecture. The model will focus on balancing accuracy and computational efficiency, potentially incorporating transfer learning from pre-trained models or using lightweight architectures to reduce the computational burden without sacrificing accuracy.
- Enhanced data augmentation techniques and preprocessing will be applied to improve the model's ability to generalize under various real-world conditions.

### 5.2 Real Time Performance

- The proposed system will be designed with real-time deployment in mind, optimizing the model architecture for faster inference times. Techniques such as model quantization, pruning, or using more efficient layers (like depthwise separable convolutions) may be applied to reduce the computational load, allowing the system to perform recognition in real-time even on less powerful hardware.

### 5.3 Accuracy and Robustness

- The proposed system aims to achieve higher accuracy by using an advanced CNN architecture trained on a more extensive and diverse dataset, possibly supplemented with synthetic data to cover a broader range of scenarios.
- Additional strategies like ensemble learning, where multiple models are combined, or using attention mechanisms to focus on the most relevant parts of the image, can further enhance robustness against variations in lighting, angles, and occlusions.

## **5.4 Model Interpretability and Transparency**

- The proposed system will incorporate techniques to improve the interpretability of the deep learning model, such as using visualization tools like Grad-CAM to highlight which parts of the image the model is focusing on when making a decision. This can provide more transparency in how the model is recognizing and classifying traffic signs.

## **5.5 Scalability and Adaptability**

- The proposed system will be designed for scalability, with an architecture that can easily adapt to recognize additional types of traffic signs by fine-tuning or transfer learning, minimizing the need for extensive retraining. This will make the system more flexible and easier to update as new traffic signs or regulations are introduced.

## 6 Architectural Overview

1. User Interface: The front-end application (e.g., a web) where users can upload images of traffic signs. It communicates with the Flask server to send image data and receive classification results.

2. Flask: A lightweight web framework used to handle incoming requests from the user interface. It manages the communication between the user, the backend processing modules, and the cloud server if needed. Flask routes the image data for preprocessing and prediction.

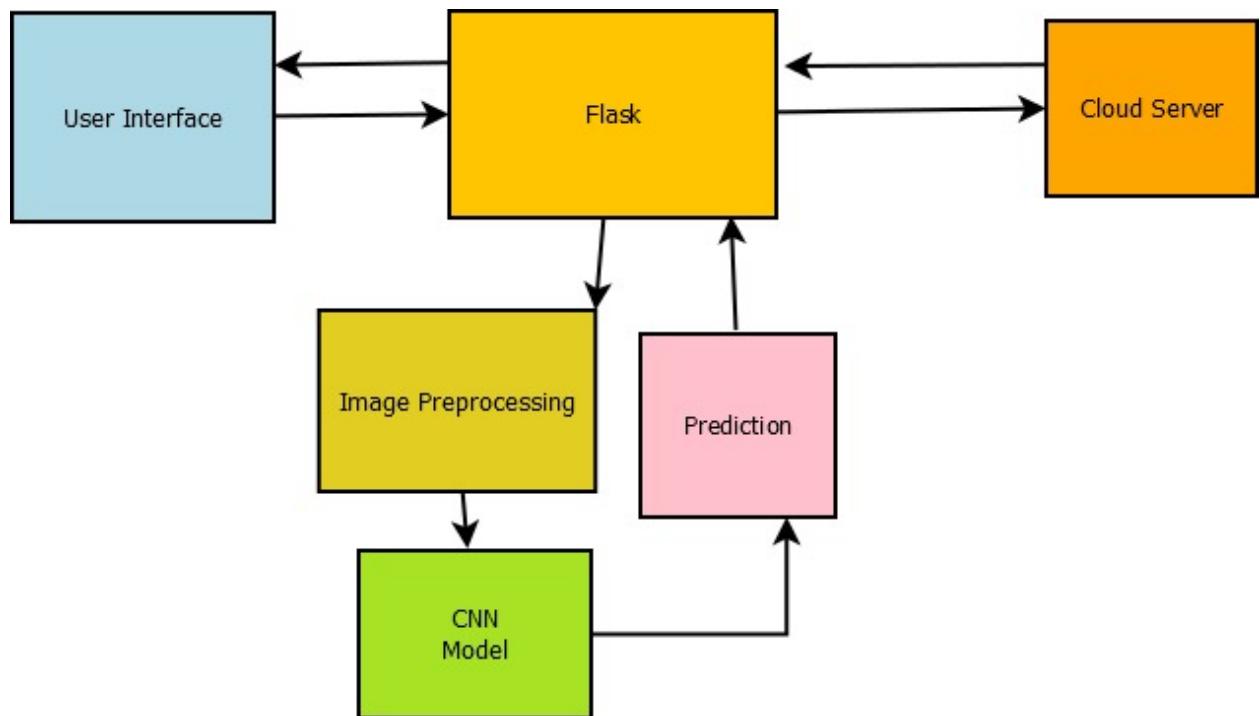
3. Image Preprocessing: This module handles preparing the input images for prediction. It includes tasks like resizing, normalization, and possibly augmentation to ensure the image is in the right format for the CNN model.

4. CNN Model: The Convolutional Neural Network (CNN) is the core of the system. It processes the preprocessed images to classify the traffic signs. The model is trained on a labeled dataset of traffic signs to recognize various types of signs accurately.

5. Prediction: Once the CNN model processes the image, this module interprets the model's output to generate a human-readable prediction (e.g., "Speed Limit 50," "Stop Sign"). The prediction result is sent back to the Flask server.

6. Cloud Server: This optional component can be used to store data, manage heavy computations, or serve as a backup for the CNN model. It can offload intensive tasks from the local server, making the system more scalable.

7. Feedback Loop to User Interface: The prediction result is sent back to the user interface, where users can view the recognized traffic sign in real-time or save the result for future reference.



Architectural Diagram

## 7 Hardware & Software requirement

### 7.1 Development Environment

- Python: The primary programming language for developing machine learning and deep learning models. Versions 3.6 or higher are recommended.
- Jupyter Notebook: An interactive development environment ideal for experimentation and debugging.
- Google Colab: An interactive development environment ideal.

### 7.2 Deep Learning Libraries

- TensorFlow/Keras: A widely used deep learning library that provides extensive tools for building and training neural networks. Keras is a high-level API that runs on top of TensorFlow.
- PyTorch: An alternative deep learning framework known for its flexibility and ease of use, particularly for research and experimentation.
- OpenCV: A computer vision library used for image processing tasks such as image resizing, augmentation, and real-time video processing.

### 7.3 Data Handling and Preprocessing

- NumPy: A library for numerical computing in Python, used for handling arrays and matrices, which are essential for image data processing.
- Pandas: A data manipulation library that can be used to handle datasets, especially for tasks like reading annotations or managing data labels.
- Scikit-learn: A machine learning library that provides tools for preprocessing, such as data normalization and splitting datasets into training and validation sets.

## 7.4 Visualization Tools

- Matplotlib/Seaborn: Libraries for creating static, animated, and interactive visualizations in Python, used for plotting loss curves, accuracy trends, and other metrics.
- TensorBoard: A tool for visualizing the training process, including metrics like loss and accuracy, which is integrated with TensorFlow.

## 7.5 Data Augmentation Tools

- Albumentations: A fast image augmentation library that provides various transformations to increase the diversity of the training dataset, improving the model's robustness.
- imgaug: Another augmentation library that integrates with NumPy arrays and can be used to apply a wide range of augmentations like rotation, scaling, and flipping.

## 7.6 Hardware Acceleration

- CUDA Toolkit (for NVIDIA GPUs): Required if you're using a GPU to accelerate deep learning training. CUDA provides the necessary libraries and drivers to harness GPU power.
- cuDNN: NVIDIA's deep learning library, which works with CUDA to optimize neural network computations, particularly on TensorFlow and PyTorch.

## 7.7 Version Control

- Git: A version control system to manage changes to your codebase, especially when working in a team or managing multiple versions of the model.
- GitHub/GitLab: Platforms for hosting Git repositories, useful for collaboration and version management.

## **7.8 Deployment Tools (Optional)**

- TensorFlow Lite: A lightweight solution for deploying TensorFlow models on mobile and embedded devices.
- ONNX (Open Neural Network Exchange): A format that allows models to be shared between different deep learning frameworks, enabling deployment in various environments.

## **7.9 Operating System**

- Windows: Is used, but it required additional configuration for GPU support.

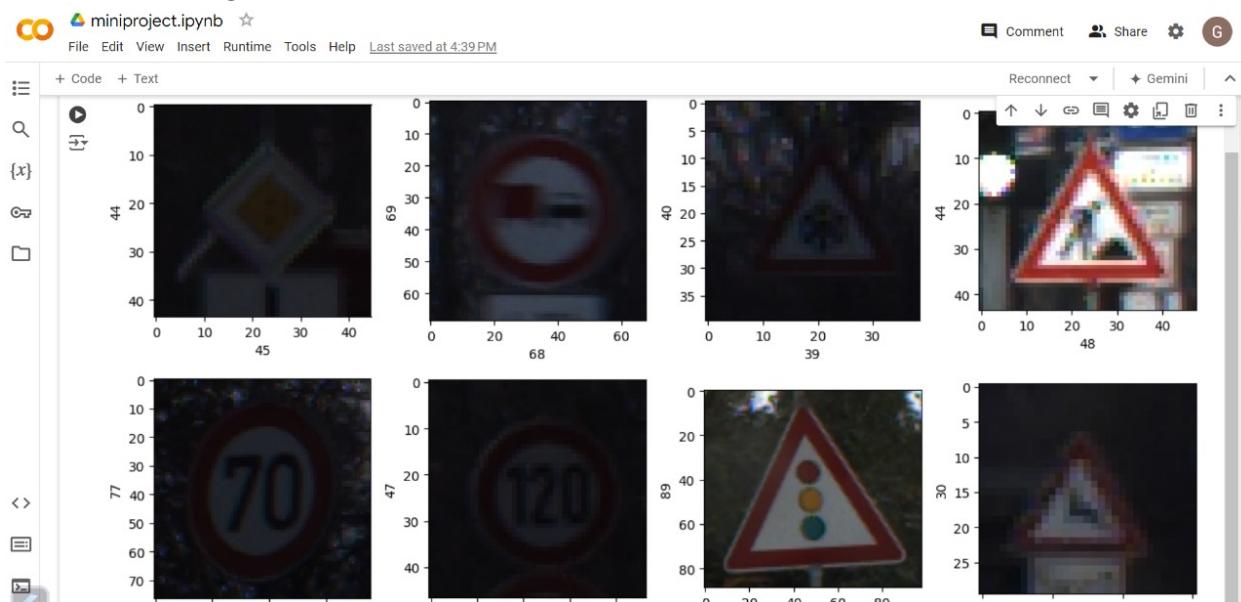
## **7.10 Dataset Management**

- Kaggle API: For easy access to datasets like the GTSRB (German Traffic Sign Recognition Benchmark).
- Google Drive/Google Colab: For storing large datasets and leveraging free GPU resources for training.

## 8 Implementation Details

Traffic sign recognition using deep learning in Python involves several steps:

1. Data Collection: Gathered a dataset of images of traffic signs-GTSRB (German Traffic Sign Recognition Benchmark) from kaggle.com .
2. Data Preprocessing: Resized images, converted to grayscale, and normalized pixel values.
3. Model Selection: Chosen a deep learning architecture like Convolutional Neural Networks (CNNs).
4. Training: Trained the model using our dataset, with a suitable optimizer and loss function.
5. Evaluation: Tested the model's performance using metrics like accuracy, precision, and recall.
6. Deployment: Integrate the trained model into an application to recognize traffic signs in real-time images or videos.



File Edit View Insert Runtime Tools Help Last saved at 3:32PM

Comment Share Gemini

+ Code + Text

Preprocess the images

```
[x] [ ] 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)
            data.append(image)
            labels.append(i)
        except Exception as e:
            print(e)
```

Converting lists into numpy arrays

```
[ ] data = np.array(data)
labels = np.array(labels)
```

Save Labels & Data for future use

```
[ ] np.save("/content/training/data",data)
np.save("/content/training/target",labels)
```

miniproject.ipynb

File Edit View Insert Runtime Tools Help Last saved at 4:39PM

Comment Share Gemini

+ Code + Text

```
{x} [ ] !unzip traffic_sign_dataset/gtsrb-german-traffic-sign.zip -d traffic_sign_dataset
!rm traffic_sign_dataset/gtsrb-german-traffic-sign.zip
!rm -rf traffic_sign_dataset/Meta
!rm -rf traffic_sign_dataset/meta
!rm -rf traffic_sign_dataset/test
!rm -rf traffic_sign_dataset/train
!rm traffic_sign_dataset/Meta.csv
```

↳ [ ] inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00017.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00018.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00019.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00020.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00021.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00022.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00023.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00024.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00025.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00026.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00027.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00028.png
inflating: traffic\_sign\_dataset/train/5/00005\_00054\_00029.png
inflating: traffic\_sign\_dataset/train/5/00005\_00055\_00000.png
inflating: traffic\_sign\_dataset/train/5/00005\_00055\_00001.png
inflating: traffic\_sign\_dataset/train/5/00005\_00055\_00002.png
inflating: traffic\_sign\_dataset/train/5/00005\_00055\_00003.png
inflating: traffic\_sign\_dataset/train/5/00005\_00055\_00004.png
inflating: traffic\_sign\_dataset/train/5/00005\_00055\_00005.png
inflating: traffic\_sign\_dataset/train/5/00005\_00055\_00006.png
inflating: traffic\_sign\_dataset/train/5/00005\_00055\_00007.png
inflating: traffic sign dataset/train/5/00005 00055 00008.png

miniproject.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share G

+ Code + Text ✓ RAM Disk Gemini ^

Load data & Labels {x}

```
[ ] data=np.load("/content/training/data.npy")
labels=np.load("/content/training/target.npy")
```

Print data shape

```
[ ] print(data.shape, labels.shape)
(39209, 30, 30, 3) (39209,)
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=0)
```

```
[ ] print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

Convert labels to onehot encoding

```
[ ] y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
```

0s completed at 10:08 PM

## 9 Applications

The applications of a Traffic Sign Recognition project using deep learning are numerous:

1. Autonomous Vehicles: Integrating traffic sign recognition into self-driving cars to improve navigation and safety.
2. Driver Assistance Systems: Enhancing existing driver assistance systems with real-time traffic sign recognition for improved safety.
3. Smart Traffic Management: Analyzing traffic sign recognition data to optimize traffic flow, reduce congestion, and improve traffic management.
4. Surveillance Systems: Integrating traffic sign recognition into surveillance systems for monitoring and enforcing traffic rules.
5. Navigation Apps: Improving navigation apps with real-time traffic sign recognition for more accurate routing and directions.
6. Road Maintenance: Identifying damaged or missing traffic signs using traffic sign recognition, enabling proactive road maintenance.
7. Accessibility: Assisting visually impaired drivers with traffic sign recognition and audio feedback.
8. Traffic Sign Inventory Management: Automating the process of tracking and managing traffic signs, reducing manual labor and costs.
9. Research and Development: Contributing to the development of more advanced AI models and computer vision techniques.
10. Public Safety: Enhancing public safety by detecting and alerting authorities to potential hazards, such as damaged or obscured traffic signs.

These applications can have a significant impact on road safety, efficiency, and accessibility, making the project a valuable contribution to the field of computer vision and deep learning.

## **10 Scope/Future Work:**

1. Model Enhancement and Optimization: Future work can focus on refining the neural network architecture to boost accuracy, reduce computation time, and lower the model's resource usage. Techniques like transfer learning, quantization, and model pruning can be employed to optimize performance on low-power devices like embedded systems and smartphones.
2. Real-time Processing and Deployment: Achieving real-time traffic sign recognition is crucial for applications in autonomous vehicles and driver assistance systems. Future improvements can involve leveraging hardware accelerators (e.g., GPUs, TPUs) and edge computing solutions to minimize latency. Deploying the model on platforms like Raspberry Pi or Jetson Nano can bring real-time capabilities to smart city infrastructure.
3. Dataset Expansion and Diversification: Expanding the dataset to include traffic signs from various countries, different lighting conditions, and weather scenarios can significantly improve the model's robustness and generalization. Collecting video data for temporal analysis and using synthetic data augmentation can further enhance performance in diverse environments.
4. Integration with Other Systems: Integrating traffic sign recognition systems with advanced driver-assistance systems (ADAS), navigation apps, and smart city infrastructure can offer holistic solutions for road safety. This integration can facilitate intelligent traffic management, automated driving decisions, and optimized route planning based on detected traffic signs.
5. User Interface and Experience Improvement: A user-centric approach can involve developing intuitive dashboards and mobile apps that provide real-time traffic sign alerts, history logs, and personalized driving tips. Enhancements could include voice alerts, haptic feedback, and augmented reality (AR) overlays for better situational awareness.
6. Compliance and Safety Features: Future versions can incorporate features that ensure compliance with traffic laws, like speed limit monitoring and stop sign adherence. This can be particularly useful for fleet management, where safety compliance is critical. Real-time alerts for traffic rule violations can significantly reduce accidents and enhance road safety.

7.Research and Development: Continuous RD efforts can explore emerging technologies like sensor fusion, where data from LiDAR, radar, and cameras are combined for more accurate traffic sign detection. Other areas include exploring unsupervised and self-supervised learning methods, improving the model's interpretability, and applying federated learning for privacy-preserving sign recognition models.

## 11 Conclusion:

## 12 References:

- [1] J. Stallkamp, M. Schlipsing, and J. Salmen, The German Traffic Sign Recognition Benchmark: A Multi-Class Classification Competition, 1st ed., Bochum, Germany: Institut für Neuroinformatik, Ruhr-Universität Bochum, 2011.
- [2] E. Miloš, A. Kolesau, and D. Šešok, "Traffic sign recognition using convolutional neural networks," in Science – Future of Lithuania, vol. 10, Vilnius: Vilnius Gediminas Technical University, 2018.
- [3] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark, 1st ed., Bochum, Germany: Institut für Neuroinformatik, Ruhr-Universität Bochum, 2013.
- [4] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, Flexible, High Performance Convolutional Neural Networks for Image Classification, 1st ed., Zurich, Switzerland: ETH Zurich, 2012.
- [5] B. Bayar and M. C. Stamm, Design Principles of Convolutional Neural Networks for Multimedia Forensics, 1st ed., New York, NY, USA: Springer, 2019.
- [6] S. Jung, U. Lee, J. Jung, and D. H. Shim, Real-time Traffic Sign Recognition System with Deep Convolutional Neural Network, 1st ed., Seoul, South Korea: Seoul National University, 2020.