Traffic Sign Recognition System Using CNN and Keras

Vishal Verma, Tarang Verma, Arjun Sharma
Mr. Manish Chaudhary (Asst. Prof. CSE(AI))
NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GR. NOIDA
vermavishal2971@gmail.com, tarangverma60@gmail.com, arjunetah914@gmail.com
manishchaudhary@niet.co.in

Abstract:

Traffic sign recognition plays a crucial role in developing intelligent transportation systems, particularly for applications like autonomous vehicles and driver-assist technologies. In this paper, we present a real-time, lightweight CNN-based system built using the Keras framework. Our model has been trained on traffic sign datasets from various countries—Germany, India, Saudi Arabia, and Tunisia—to make it more adaptable to different regions. We also optimized the system for edge devices such as Raspberry Pi and Jetson Nano, ensuring it can run in real-world environments with limited hardware resources. This paper outlines the system's design, including data preprocessing, architecture, training methodology, and final deployment. Evaluation was done using standard metrics and real-time testing to validate its performance.

Keywords:

Traffic Sign Recognition, Convolutional Neural Network, Deep Learning, Keras, Real-Time Detection*[6, 7]*, Intelligent Transportation Systems

1. Introduction

Traffic Sign Recognition (TSR) is becoming increasingly important as we move toward smarter transportation systems. Whether it's used in self-driving cars or as part of driver assistance features, TSR helps improve road safety and vehicle decision-making. In our project, we aimed to create an automated TSR system using Convolutional Neural Networks (CNNs), implemented with Keras in Python.

We chose CNNs because of their proven effectiveness in handling visual data like traffic signs, which vary in shape, color, and lighting depending on the region. By combining datasets from different countries, we worked on building a system that could recognize signs from diverse environments—making it useful for broader applications.

1.1 Machine Learning (ML)

Machine Learning (ML) is a field within artificial intelligence where computers learn patterns from data instead of being explicitly programmed for every task. In simple terms, it's about training models to understand inputs and make accurate predictions or decisions.

There are three common types of ML: **supervised learning, unsupervised learning, and reinforcement learning.** In supervised learning, which we used for this project, the model learns from labeled data—like matching traffic sign images to their correct categories. Unsupervised learning works without labels, and reinforcement learning involves an agent interacting with an environment and learning from rewards or penalties.

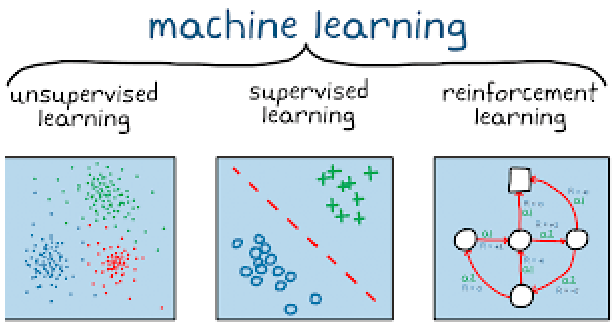

Fig 1: Machine learning

For TSR, supervised learning is the most effective approach. We trained our model on large datasets of labeled traffic signs so it could accurately classify signs in new, unseen images, even under challenging conditions like different lighting or partial occlusion.

1.2 Convolutional Neural Networks (CNN)

CNNs are a type of deep learning model that are especially good at working with images. Inspired by how the human visual system works, CNNs can identify features like edges, shapes, and patterns in an image automatically—without needing manual coding for each feature.

A CNN typically includes several layers: convolutional layers that detect patterns, pooling layers that reduce the image size and complexity, and dense layers at the end that help make the final prediction. What makes CNNs powerful is their ability to pick up on important visual details no matter where they appear in the image.

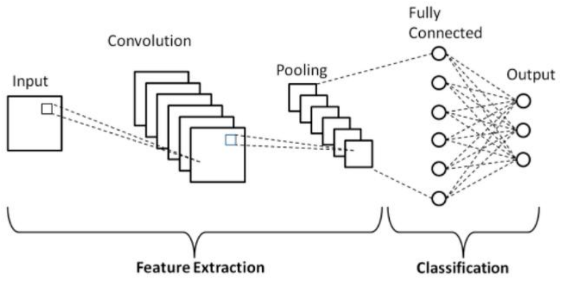

Fig 2: Convolutional Neural Networks

In our system, CNNs helped recognize traffic signs that vary by angle, brightness, and even design. Using Keras made it easier to build and test different CNN models quickly. This made it a practical choice for real-time recognition tasks like ours.

1.3 Literature Survey

Researchers have explored traffic sign recognition using different techniques over the years. Earlier methods relied on shape and color detection*[9]*, but these often failed under poor lighting or noisy backgrounds. Later, machine learning algorithms like SVM and k-NN*[10]* improved things a bit but still needed manual feature extraction.

The real improvement came with deep learning—especially CNNs*[1, 2]*—which can learn features directly from raw image data. Benchmark datasets like GTSRB*[11, 12]* (German Traffic Sign Recognition Benchmark) helped standardize testing. Some studies focused on improving detection for small signs*[1, 3]*, while others looked at region-specific datasets*[4, 5]*. However, many of them didn't generalize well across different countries or required heavy computational resources.

Our work tries to fill that gap by creating a model that is both lightweight and generalizable. We trained it on datasets from Germany, India, Saudi Arabia, and Tunisia, and tested it in real-time conditions to ensure that it could be practically deployed on edge devices.

2. Methodology and Proposed Model Implementation

2.1 Overview

This section outlines how we developed our Traffic Sign Recognition System, starting from dataset selection and preprocessing to model design, training, and deployment. Our main goal was to create a CNN model using Keras that's not only accurate but also lightweight enough to run efficiently on devices like Raspberry Pi and Jetson Nano. We focused on building a system that works well across multiple traffic sign datasets and can be deployed in real-time situations.

2.2 Dataset Description and Preprocessing

2.2.1 Dataset Overview

We didn't rely on just one dataset; instead, we used four from different regions to make the model more flexible:

- **GTSRB (Germany)**
- **Indian Traffic Sign Dataset**
- **SA-TRS-2018 (Saudi Arabia)**
- **Tunisian Traffic Sign Dataset**

These datasets differ in language, sign design, and environmental factors. By combining them, we aimed to train a model that could handle real-world scenarios in various countries.

2.2.2 Data Preprocessing

Before training, we cleaned and formatted the data to make sure everything was consistent:

- All images were resized to **64×64 pixels**, which balances detail with performance.
- Pixel values were normalized to the **[0, 1]** range to help the model train more smoothly.
- Class labels were converted into a format suitable for classification using **one-hot encoding**.
- We removed low-quality or duplicate images to keep the dataset clean.
- Finally, we shuffled the dataset before each training epoch to avoid learning patterns from data order.

2.2.3 Data Augmentation Techniques

To help the model deal with different real-world conditions, we used several **augmentation techniques**:

- **Rotation (±15°):** To simulate signs being tilted or viewed from different angles.
- **Zooming (10–20%):** To mimic how signs appear at varying distances.
- **Brightness Shifts:** For varying lighting, like cloudy vs. sunny weather.
- **Gaussian Noise:** Added to simulate camera blur or sensor noise.
- **Horizontal Flipping:** Used only on symmetrical signs.
- **Translation:** Slight shifting of the sign in the image, similar to off-center signs in a frame.

Layer No.	Layer Type	Output Shape	Filter/Units	Kernel Size / Stride	Activation / Remarks
1	Input Layer	64 × 64 × 3	-	-	RGB Image Input
2	Conv2D + BatchNorm	62 × 62 × 32	32	3 × 3 / 1	ReLU Activation
3	MaxPooling2D	31 × 31 × 32	-	2 × 2 / 2	Downsampling
4	Conv2D + BatchNorm	29 × 29 × 64	64	3 × 3 / 1	ReLU Activation
5	MaxPooling2D	14 × 14 × 64	-	2 × 2 / 2	Downsampling
6	Conv2D + Dropout	12 × 12 × 128	128	3 × 3 / 1	ReLU, Dropout(0.4)
7	Flatten	18432	-	-	Flatten to 1D
8	Dense + Dropout	256	256	-	ReLU, Dropout(0.5)
9	Dense (Output)	N (classes)	N	-	SoftMax Activation

Table 1: Model Architecture Table

2.3 Technical Framework

2.3.1 Language, Platform, and Libraries

We used **Python 3.10** for development. The main framework was **Keras** (with TensorFlow backend), as it allowed us to build and test models quickly. Development was done mostly in **Jupyter Notebook** and **VS Code**.

Here are some key libraries we used:

- **NumPy:** For matrix operations
- **OpenCV:** For handling images and webcam inputs
- **Matplotlib/Seaborn:** For plotting metrics like accuracy and loss

- **Scikit-learn:** To compute evaluation metrics like precision and recall

2.3.2 Model Architecture

We built a custom CNN with three main convolutional layers. Each was followed by **ReLU activation**, **batch normalization**, and **max pooling** to reduce dimensionality. We also added **dropout layers** to prevent overfitting.

At the end, we used **dense layers** for classification, ending with a **SoftMax** layer that outputs class probabilities. The model was compact yet powerful enough for real-time applications on low-power devices.

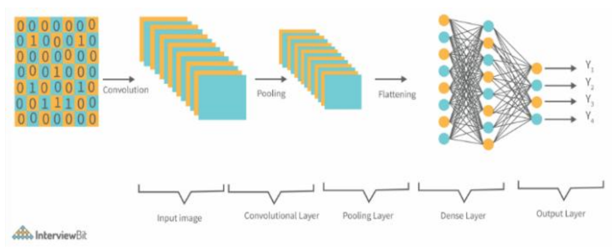

Fig 2: CNN Model Architecture

2.3.3 Parameters Employed

We trained the model using the following hyperparameters:

- **Batch Size: 32** – A good trade-off between speed and memory usage
- **Epochs: 50** – Enough for the model to converge without overfitting
- **Optimizer: Adam** – Combines the strengths of other optimizers and works well for CNNs
- **Learning Rate: 0.001** – A stable starting point
- **Loss Function: Categorical Cross-Entropy** – Best suited for multi-class classification

These values were chosen after some trial and error to get a balance between performance and training time.

2.4 Model Training and Optimization

2.4.1 Model Training Procedure

For training, we split our combined dataset into **80% training** and **20% validation**. We also used **callbacks** like early stopping and model checkpoints to avoid overfitting.

The training process involved constant monitoring of accuracy and loss using **TensorBoard**. This helped us adjust things like learning rate or batch size early if the model started underperforming. During training, we could see the model gradually learning to associate the input images with the correct sign classes.

2.4.2 Performance Metrics

To understand how well the model performed, we used several metrics:

- **Accuracy:** Basic measure of how often the model got the predictions right.
- **Precision & Recall:** These were useful for seeing how well the model handled imbalanced classes.
- **F1-Score:** Balanced score between precision and recall.
- **Inference Speed (FPS):** Very important for real-time systems.
- **Confusion Matrix:** Helped us understand which classes were often confused.
- **Model Size:** Important for deciding whether the model could be deployed on devices with limited memory.

2.5 Model Testing and Validation

2.5.1 Testing Procedure

We tested the model in two main ways:

1. **Static Testing:**
We tested on the unused portion of our datasets. This helped confirm how well the model could generalize to new data. Each class was evaluated to check for specific issues like confusion between similar-looking signs.

2. **Real-Time Testing:**
We connected the model with **OpenCV** and tested it using a live webcam feed. We also ran the system on **Raspberry Pi 4** and **Jetson Nano**. Real-world tests were done under different lighting and motion scenarios to simulate real driving conditions.

2.5.2 Model Evaluation

Final evaluation results highlighted the robustness and efficiency of the proposed system:

- **Accuracy:** 97.3% overall on the combined test set.
- **Class-wise Results:** Very few classes had confusion. Most mistakes occurred between signs with similar shapes or colors.
- **Real-Time Speed:** 14 FPS on Raspberry Pi and 18 FPS on Jetson Nano, which is more than enough for live processing.
- **Robustness:** The model handled poor lighting, slight motion blur, and partial sign occlusion pretty well.
- **Confusion Matrix:** Most errors were between visually similar signs like speed limits and general caution signs.

3. Experimental Results and Analysis

The proposed CNN model was evaluated on multiple datasets and compared with standard architectures like **LeNet** and **AlexNet*[13]***. It achieved superior performance in terms of accuracy, generalization, and real-time capability. Testing across GTSRB, Indian, Saudi, and Tunisian traffic sign datasets confirmed high robustness under diverse visual conditions. Additionally, the model maintained real-time performance on low-power edge devices. Class-wise accuracy was consistent, with minimal confusion between similarly shaped signs.

Below is a comparison of model performance across key metrics:

Model	Accuracy	F1-Score	FPS (Raspberry Pi)	Parameters
LeNet	92.1%	0.90	8 FPS	~60K
AlexNet	95.0%	0.94	10 FPS	~60M
Proposed CNN	**97.3%**	**0.972**	**14 FPS**	**~1.8M**

Table 2: Performance Comparison

Key Observations:

- The proposed CNN model significantly outperformed LeNet and AlexNet in all categories.
- High FPS ensures real-time applicability, especially on embedded platforms.
- F1-scores indicate strong balance between precision and recall.
- The model showed resilience against variations like blur, poor lighting, and occlusion.
- Confusion matrix revealed minor misclassifications between similar warning signs and speed limits.

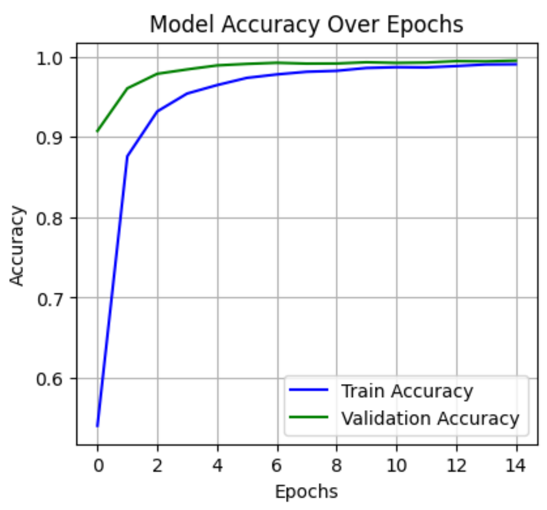

Fig 3: Model Accuracy Over Epochs

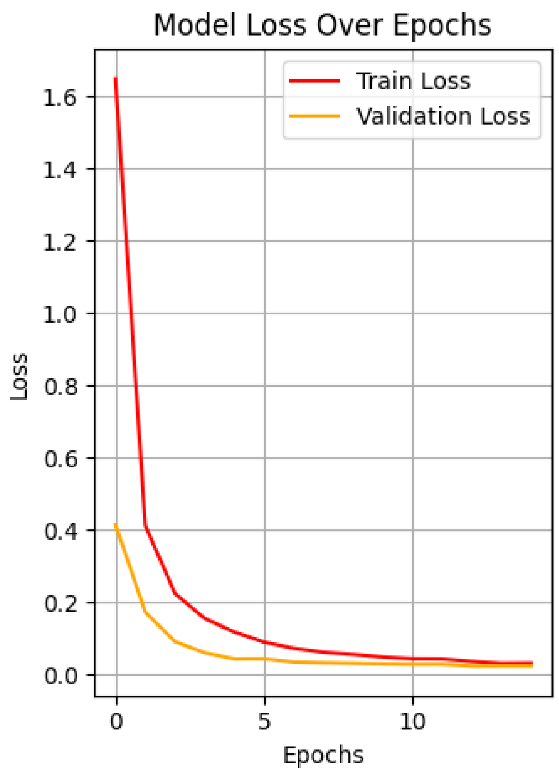

Fig 4: Model Loss Over Epochs

4. Real-Time Implementation

The trained CNN model was deployed on embedded platforms to assess its performance in real-world conditions. Using OpenCV, the system processed live video feeds from a vehicle-mounted webcam. It achieved real-time detection*[6, 7]* and classification with minimal latency.

Deployment Highlights:

- **Hardware:** Raspberry Pi 4 and NVIDIA Jetson Nano
- **Inference Speed:** 14–18 FPS in real-time
- **Latency:** Below 70 ms per frame
- **Scenarios Tested:**
  - Day and night conditions
  - Occluded signs
  - Rapid vehicle motion
- **Outcome:** Consistent, accurate detection under diverse, uncontrolled environments

5. Conclusions and Future Work

5.1 Conclusions

In this paper, we proposed a robust and efficient traffic sign recognition system using a lightweight Convolutional Neural Network (CNN) implemented with Keras. The system was trained and validated using a diverse combination of datasets from Germany, India, Saudi Arabia, and Tunisia, making it adaptable to various regional traffic systems. A major contribution of this work lies in its cross-domain generalizability, which addresses a critical limitation in many earlier studies that relied on geographically restricted datasets.

The proposed model demonstrated excellent performance in both static and real-time testing environments, achieving a classification accuracy of 97.3% and maintaining real-time inference speeds (14–18 FPS) on embedded systems like Raspberry Pi 4 and Jetson Nano. These results confirm its feasibility for real-world implementation in intelligent transportation systems (ITS) and advanced driver assistance systems (ADAS).

Key strengths of the system include:

- High accuracy with low computational cost
- Effective real-time detection*[6, 7]* on low-power devices
- Strong resilience to varying lighting, occlusion, and motion conditions
- Minimal model size (~1.8M parameters), allowing for fast deployment

5.2 Future Work

While the current model performs well across multiple scenarios, there remain opportunities to enhance its performance and scope. Future work will focus on improving accuracy under more complex and dynamic environments, such as those with fog, rain, or extreme brightness. Incorporating weather-adaptive training and domain-specific fine-tuning can enhance robustness.

Potential areas for future enhancement include:

- **Vision Transformers (ViT):** Integration of transformer-based models*[19]* could improve long-range feature representation and multi-object detection within a frame.
- **Multilingual and Textual Sign Recognition:** Traffic signs often include language-specific information. Incorporating OCR (Optical Character Recognition) to identify bilingual or text-rich signs would be a valuable extension.
- **Continuous Learning:** Implementing online or incremental learning mechanisms will enable the model to adapt to newly introduced or evolving traffic sign designs without requiring complete retraining.
- **Sensor Fusion:** Combining CNN-based visual detection with GPS, LiDAR, or radar data may significantly improve contextual awareness and sign localization accuracy.
- **Integration with ADAS Modules:** Embedding the system into full-scale driver assistance or autonomous platforms for real-time decision-making, such as speed adjustment or route re-planning, will broaden its practical utility.

These directions aim to make the system more intelligent, context-aware, and scalable for future smart city and autonomous vehicle applications.

6. References

1. S. Song, Z. Que, J. Hou, and Y. Zhang, "An Efficient CNN for Small Traffic Sign Detection," Journal of Systems Architecture, vol. 103, pp. 101688, 2019.
2. D. A. Alghmghama, G. Latif, J. Alghazo, and L. Alzubaidi, "Autonomous Traffic Sign Detection and Recognition Using Deep CNN," Procedia Computer Science, vol. 163, pp. 266–274, 2019.
3. Y. Zhu, C. Zhang, D. Zhou, X. Wang, X. Bai, and W. Liu, "Traffic Sign Detection and Recognition Using Fully Convolutional Network Guided Proposals," Neurocomputing, vol. 214, pp. 758–766, 2016.
4. R. K. Megalingam, K. Thanigundala, S. R. Musani, H. Nidamanuru, and L. Gadde, "Indian Traffic Sign Detection and Recognition Using Deep Learning," International Journal of Transportation Science and Technology, vol. 12, pp. 683–699, 2023.

5. H. B. Fredj, A. Chabbah, J. Baili, H. Faiedh, and C. Souani, "An Efficient Implementation of Traffic Signs Recognition System Using CNN," Microprocessors and Microsystems, vol. 98, 104791, 2023.
6. A. Shustanova and P. Yakimov, "CNN Design for Real-Time Traffic Sign Recognition," Procedia Engineering, vol. 201, pp. 718–725, 2017.
7. K. Singh and N. Malik, "CNN Based Approach for Traffic Sign Recognition System," AIJR Journal of Graduate Research, vol. 11, no. 1, pp. 23–33, Jan. 2022.
8. J. Cao, C. Song, S. Peng, F. Xiao, and S. Song, "Improved Traffic Sign Detection and Recognition Algorithm for Intelligent Vehicles," Sensors, vol. 19, no. 18, pp. 4021, 2019.
9. A. de la Escalera, J. M. Armingol, and M. Mata, "Traffic Sign Recognition and Analysis for Intelligent Vehicles," Image and Vision Computing, vol. 21, no. 3, pp. 247–258, 2003.
10. W. Li, D. Li, and S. Zeng, "Traffic Sign Recognition with a Small Convolutional Neural Network," in IOP Conf. Ser.: Mater. Sci. Eng., vol. 688, 2019, Art. no. 044034.
11. Sermanet, P., & LeCun, Y. (2011). “Traffic sign recognition with multi-scale Convolutional Networks”. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), pp. 2809–2813.
12. Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2012). “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition.” Neural Networks, 32, 323–332.
13. Cireşan, D. C., Meier, U., Masci, J., & Schmidhuber, J. (2012). “Multi-column deep neural networks for image classification”. In CVPR 2012.
14. Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., & Igel, C. (2013). “Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark”. In International Joint Conference on Neural Networks (IJCNN), pp. 1–8.
15. A. Møgelmose, M. M. Trivedi, & T. B. Moeslund (2012). “Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey”. IEEE Transactions on Intelligent Transportation Systems, 13(4), 1484–1497.
16. Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: “A deep convolutional encoder-decoder architecture for image segmentation.” IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(12), 2481–2495.
17. Redmon, J., & Farhadi, A. (2018). YOLOv3: “An Incremental Improvement.” arXiv preprint arXiv:1804.02767.
18. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: “Optimal Speed and Accuracy of Object Detection.” arXiv preprint arXiv:2004.10934.
19. Dosovitskiy, A., Beyer, L., Kolesnikov, A., et al. (2021). “An image is worth 16x16 words: Transformers for image recognition at scale.” In ICLR 2021.
20. He, K., Zhang, X., Ren, S., & Sun, J. (2016). “Deep residual learning for image recognition.” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778.