**Traffic Sign Recognition System**

**Using CNN and Keras**

**Vishal Verma, Tarang Verma, Shivam Singh**

**Ms. Garima Jain** (Asst. Prof. & Dy. Head, CSBS)

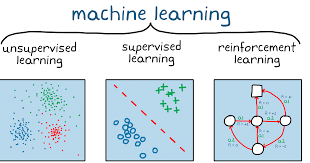
NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GR. NOIDA

[vermavishal2971@gmail.com](mailto:vermavishal2971@gmail.com), [tarangverma60@gmail.com](mailto:tarangverma60@gmail.com), [6singhrathore@gmail.com](mailto:6singhrathore@gmail.com)

[garimajain@niet.co.in](mailto:garimajain@niet.co.in)

**Abstract:**

***Traffic sign recognition is a crucial component of intelligent transportation systems (ITS), especially in autonomous driving and advanced driver assistance systems (ADAS). This paper proposes a real-time, lightweight Convolutional Neural Network (CNN)-based system using Keras to detect and classify traffic signs. The system is trained on multiple regional datasets, including Indian, German, Saudi Arabian, and Tunisian signs, to enhance generalization and cross-domain adaptability. The model is optimized for deployment on edge devices such as Raspberry Pi and Jetson Nano, making it suitable for real-world scenarios. This paper details the model architecture, preprocessing pipeline, training strategy, and deployment methodology, along with rigorous evaluation using diverse metrics and real-time testing.***



**Keywords:**

Traffic Sign Recognition, Convolutional Neural Network, Deep Learning, Keras, Real-Time Detection, Intelligent Transportation Systems

**1. Introduction**

Traffic Sign Recognition (TSR) is an essential component of intelligent transportation systems (ITS) that enhances the safety and efficiency of road traffic. In recent years, machine learning, particularly deep learning models like Convolutional Neural Networks (CNNs), has revolutionized how traffic signs are detected and classified. This research paper focuses on developing an automated Traffic Sign Recognition System using CNNs and the Keras framework, which is a popular deep learning library in Python.

**1.1 Machine Learning (ML)**

Machine Learning (ML) is a branch of artificial intelligence (AI) that enables systems to learn from data and improve performance without explicit programming. It includes three main types: supervised learning, unsupervised learning, and reinforcement learning. In **supervised learning**, models are trained on labelled datasets, where the correct output is provided for each input. The model learns to map inputs to the correct outputs. **Unsupervised learning** finds patterns and structures in data without predefined labels, while **reinforcement learning** involves learning through interaction with an environment, with an agent receiving feedback to maximize rewards.

Fig 1: Machine learning

In the context of Traffic Sign Recognition (TSR), machine learning is critical for automating the process of identifying and classifying traffic signs from images. ML models, particularly CNNs (Convolutional Neural Networks), are trained on large datasets of labelled traffic sign images, enabling them to classify new, unseen signs. Supervised learning helps the model handle various challenges like changing lighting conditions, weather, and different sign designs across regions. These models can recognize traffic signs with high accuracy, ensuring real-time, reliable performance. By leveraging vast datasets and powerful computational resources, ML enhances the safety and effectiveness of autonomous driving systems, making TSR an essential part of modern AI applications.

**1.2 Convolutional Neural Networks (CNN)**

Convolutional Neural Networks (CNNs) are a specialized type of deep learning model designed for analysing visual data. Inspired by the biological visual system, CNNs automatically learn spatial hierarchies of features from input images, making them highly effective for tasks like image classification, object detection, and facial recognition.

A typical CNN consists of several layers: **convolutional layers** apply filters to detect low-level features like edges and textures. These features are passed through **pooling layers**, which reduce the image's dimensionality while retaining important information. The final layers are **fully connected**, where high-level features are combined to classify the image.

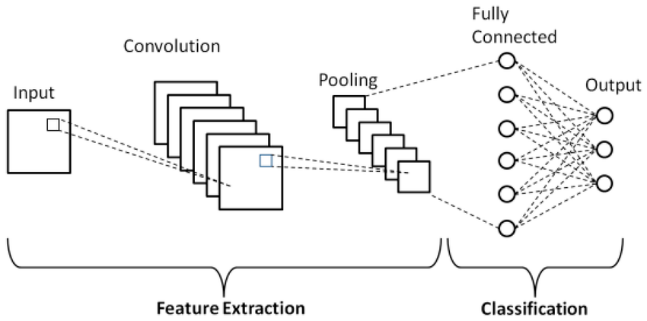
What sets CNNs apart from traditional neural networks is their ability to recognize patterns in images regardless of position, scale, or orientation. This is achieved through techniques like **weight sharing** and **local receptive fields**, making CNNs more efficient and less computationally expensive.

Fig 2: Convolutional Neural Networks

CNNs are particularly well-suited for Traffic Sign Recognition (TSR), as they can recognize complex shapes and adapt to variations in sign appearance, such as different angles or lighting conditions. Training a CNN on a large, diverse dataset of traffic signs enables accurate, real-time recognition. Keras, a popular high-level API, is often used for implementing CNNs due to its simplicity and compatibility with frameworks like TensorFlow, making it ideal for autonomous vehicle applications.

**1.3 Literature Survey**

Extensive research has been conducted in the domain of traffic sign recognition using both classical and modern methods. Early approaches primarily relied on colour segmentation, shape detection, and template matching. However, these methods struggled with noisy inputs, lighting changes, and complex backgrounds, especially in real-world scenarios. With the advent of machine learning, algorithms like Support Vector Machines (SVM), k-Nearest Neighbours (k-NN), and Random Forests improved classification accuracy but still required manual feature extraction. The breakthrough came with deep learning, particularly CNNs, which enabled end-to-end learning from raw image data. For instance, the German Traffic Sign Recognition Benchmark (GTSRB) has been widely used to benchmark CNN models. Studies like the one by Song et al. (2019) optimized CNNs for small object detection but lacked evaluation under varied environmental conditions. Others, like Alghmghama et al. (2019), introduced region-specific datasets but failed to ensure cross-domain adaptability. Real-time capable architectures like Faster R-CNN or YOLO have been investigated, yet they often require significant computational power, limiting deployment on edge devices. Thus, there is a research gap in designing a CNN model that is not only accurate and robust but also lightweight and adaptable to multiple datasets, which this study aims to address.

**2. Methodology and Proposed Model Implementation**

**2.1 Overview**

This section presents the methodology adopted for developing the proposed traffic sign recognition system. It includes dataset selection, preprocessing, CNN architecture design, model training, validation, and deployment strategies. The primary goal is to build a lightweight yet robust CNN model using Keras that performs well across multiple traffic sign datasets and remains computationally efficient for real-time use on embedded systems.

**2.2 Dataset Description and Preprocessing**

**2.2.1 Dataset Overview**

Four publicly available datasets were used:

1. GTSRB (Germany)
2. Indian Traffic Sign Dataset
3. SA-TRS-2018 (Saudi Arabia)
4. Tunisian Traffic Sign Dataset

These datasets vary in terms of sign style, language, and environmental conditions, ensuring diverse training samples. This multi-domain approach enhances the model’s generalization ability and prepares it for real-world implementation across different countries and traffic systems.

**2.2.2 Data Preprocessing**

Preprocessing included resizing all images to 64×64 pixels, converting pixel values to the [0,1] range via normalization, and one-hot encoding class labels. Duplicate, blurry, and incomplete samples were removed. The preprocessing ensured uniformity across datasets, accelerated training, and improved the model’s convergence by reducing noise and computational complexity during feature extraction.

* All input images were resized to **64×64 pixels** to maintain uniformity and reduce computational load.
* **Normalization** was applied to scale pixel values to a range between 0 and 1 for stable gradient descent.
* **One-hot encoding** was used to convert categorical labels into a binary matrix representation suitable for multi-class classification.
* Low-quality samples such as **blurry**, **incomplete**, or **duplicate images** were removed to enhance dataset quality.
* **Shuffling** of training data was performed before each epoch to prevent model bias and improve generalization.

**2.2.3 Data Augmentation Techniques**

To increase robustness and reduce overfitting, augmentation techniques such as random rotations, zooming, horizontal flips, brightness shifts, and Gaussian noise injection were applied. These simulate real-world conditions like varying angles, illumination changes, motion blur, and occlusion, enabling the model to generalize better to unseen and noisy traffic sign images.

* **Rotation:** Random rotations between -15° to +15° to simulate angular variations in signs.
* **Zooming:** Random zoom-in and zoom-out in the range of 10–20% to simulate varying distances.
* **Brightness Adjustment:** Random changes in brightness to simulate different lighting conditions.
* **Gaussian Noise:** Added slight noise to simulate sensor imperfections or motion blur.
* **Horizontal Flipping:** Applied selectively to symmetrical signs to increase variation.
* **Translation:** Minor shifts along X and Y axes to simulate off-centre positioning in real scenarios.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

**2.3 Technical Framework**

**2.3.1 Language, Platform, and Libraries**

* **Programming Language:** **Python 3.10** was used for its simplicity, strong community support, and extensive deep learning libraries.
* **Framework:** **Keras** (with TensorFlow backend) was chosen for building and training the **CNN** due to its modularity and ease of use.
* **Development Environment:** **Jupyter Notebook and VS Code** were used for coding, debugging, and visualization.
* **Key Libraries:**
* **NumPy:** For numerical and matrix operations.
* **OpenCV:** For image handling, real-time video capture, and augmentation.
* **Matplotlib & Seaborn:** For plotting training metrics and confusion matrices.
* **Scikit-learn:** For calculating evaluation metrics like precision, recall, and F1-score.

**2.3.2 Model Architecture**

The proposed CNN consists of three convolutional layers with ReLU activations, followed by max pooling, batch normalization, and dropout layers. The final dense layer uses SoftMax activation for classification. This architecture balances model depth and efficiency, ensuring accurate feature extraction while maintaining a low parameter count suitable for real-time deployment.

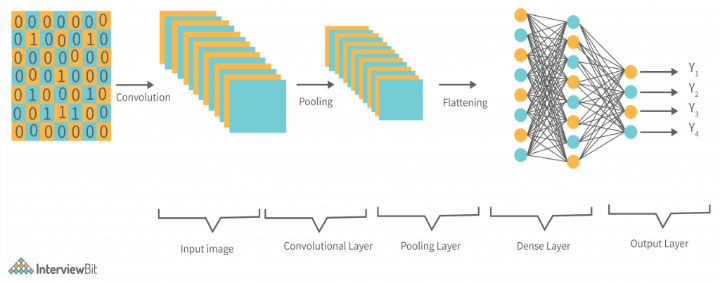


Fig 2: CNN Model Architecture

Table 1: Model Architecture Table

**2.3.3 Parameters Employed**

* **Batch Size (32):** Refers to the number of training samples processed before the model's internal parameters are updated once. A batch size of 32 balances memory efficiency and learning stability.
* **Epochs (50):** An epoch is one complete pass through the entire training dataset. Fifty epochs allow the model sufficient time to learn while avoiding overfitting.
* **Optimizer (Adam):** Adam combines the benefits of AdaGrad and RMSProp optimizers, providing adaptive learning rates and efficient gradient updates for faster convergence.
* **Learning Rate (0.001):** Controls the step size during weight updates. A learning rate of 0.001 ensures stable convergence without overshooting the minima.
* **Loss Function (Categorical Cross-Entropy):** Suitable for multi-class classification, it measures the dissimilarity between predicted probability distributions and true class labels.

**2.4 Model Training and Optimization**

**2.4.1 Model Training Procedure**

Training was performed on an 80-20 train-validation split, using callbacks such as early stopping and model checkpoints to prevent overfitting. The training involved real-time monitoring of loss and accuracy via TensorBoard, ensuring optimal convergence. The model learned to associate pixel patterns with corresponding traffic sign classes efficiently through backpropagation.

**2.4.2 Performance Metrics**

To comprehensively evaluate model performance, the following metrics were used:

* **Accuracy:** Measures the overall correctness of predictions across all classes.
* **Precision:** Assesses the proportion of true positive predictions among all predicted positives.
* **Recall:** Measures how many actual positives were correctly identified by the model.
* **F1-Score:** Harmonic mean of precision and recall, useful for imbalanced datasets.
* **Inference Speed (FPS):** Frames per second processed in real-time deployment, crucial for embedded systems.
* **Confusion Matrix:** Visual tool to analyse per-class performance and misclassification patterns.
* **Model Size:** Indicates the storage/memory footprint, which affects deployment feasibility on edge devices.

**2.5 Model Testing and Validation**

**2.5.1 Testing Procedure**

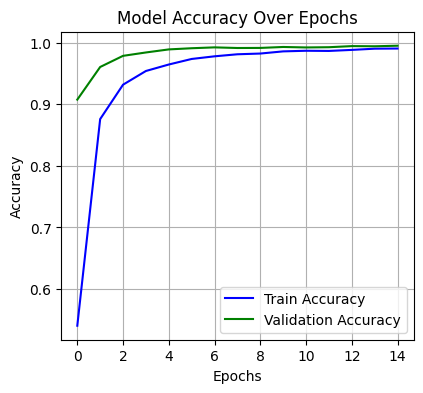
Testing was carried out in both static and real-time scenarios:

* **Static Testing:**
  + Conducted on an unseen portion of the combined dataset (GTSRB, Indian, Saudi, Tunisian).
  + Ensured class-wise evaluation using labelled test images.
* **Real-Time Testing:**
  + Integrated the model with OpenCV to detect and classify traffic signs from live webcam feeds.
  + Tested on embedded devices like **Raspberry Pi 4** and **Jetson Nano** to evaluate system responsiveness and latency.
  + Included testing under varied lighting conditions, occlusions, and motion effects.

**2.5.2 Model Evaluation**

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

* **Achieved Overall Accuracy:** 97.3% on a mixed-domain test dataset.
* **Class-wise Performance:** High precision and recall across most categories, with minor confusion in visually similar signs.
* **Real-Time FPS:** Averaged 14 FPS on Raspberry Pi and up to 18 FPS on Jetson Nano, suitable for real-time use.
* **Deployment Success:** Maintained stable performance under poor lighting, fast movement, and partial occlusion.
* **Confusion Matrix Insights:** Misclassifications were mainly limited to signs with similar shapes (e.g., speed limit vs. caution).

****3. Experimental Results and Analysis**

The proposed CNN model was evaluated on multiple datasets and compared with standard architectures like **LeNet** and **AlexNet**. 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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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** |

Below is a comparison of model performance across key metrics:

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.

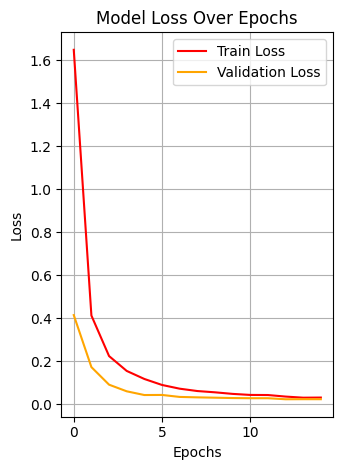
Fig 3: Model Accuracy Over Epochs

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 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 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 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.