***Abstract*—Traffic sign recognition is a core component of autonomous driving systems and advanced driver assistance systems (ADAS). In this project, the implemention is a compact but effective deep learning pipeline using a convolutional neural network (CNN) to classify traffic sign images. In this paper, the writing describes the dataset, preprocessing steps, model architecture, training procedure, and evaluation results. The implementation achieves high accuracy on unseen validation data, with clear improvements seen through iterative refinement. Visualizations such as training curves, confusion matrices, and example predictions are discussed to highlight the performance characteristics of the model. This paper aims to serve as a simple demonstration of how deep learning can be applied effectively with the goal of traffic sign understanding.**

I. Introduction

Traffic sign recognition is a fundamental task for intelligent transportation systems. As autonomous vehicles become more widely adopted, the ability to reliably detect and classify traffic signs is essential for safe navigation. [1] Errors in sign classification, such as misinterpreting a speed limit or a no-entry sign, can have critical real-world consequences.

Deep learning has emerged as the dominant method for visual classification tasks due to its ability to learn hierarchical features directly from images. [4] Unlike classical computer vision methods that rely on hand-crafted features, convolutional neural networks (CNNs) automatically extract relevant shapes, edges, textures, and patterns that enable robust classification even under difficult conditions such as blur, scale variations, or lighting changes. [1]

The goal of this project is to build a small but complete deep learning implementation for traffic sign recognition using PyTorch. The focus is educational: the goal is a model that trains quickly, demonstrates clear learning behavior, and produces interpretable results such as training curves, confusion matrices, and example predictions. The model does not attempt to compete with large-scale state-of-the-art architectures but instead aims to illustrate how CNNs can effectively solve this problem with relatively modest resources.

A. Motivation

The goal of this project is straightforward:

Implement a compact, interpretable deep learning model for traffic sign recognition and evaluate its performance.

The motivation is twofold:

1) To demonstrate the suitability of convolutional neural networks for structured image-classification problems.
2) To produce a self-contained educational example that illustrates how deep learning pipelines operate end-to-end.

B. Traffic-Sign Recognition Requirements

Traffic-sign recognition systems must meet clear perceptual and safety-driven requirements: a detector should reliably recognize salient signs at a distance that gives the vehicle enough time to perceive the sign and bring the vehicle to a safe stop or take the required maneuver. [6] A practical rule is therefore to require recognition at ranges on the order of the stopping-sight distance (SSD) appropriate for the vehicle's operating speed; SSD is the sum of the perception–reaction distance and the braking distance and is commonly computed from vehicle speed, driver reaction time, and braking capability (for example, baseline design formulas and tables used in highway engineering). [6] Requiring recognition at or beyond SSD ensures the system can issue timely warnings or initiate actions (e.g., begin braking) without compromising safety. [2]To translate that into concrete numbers: for urban speeds around 50 km/h, typical stopping-distance rules and standard charts place the total stopping distance on the order of about 40 m (reaction = 15 m + braking = 25 m under typical assumptions), while at 100 km/h total stopping distances approach or exceed 100 m; these widely used benchmarks give a practical baseline for how far ahead a sign should be recognized to allow safe responses. [6] Requiring sign detection at about 60 m is therefore reasonable for many suburban and arterial scenarios (it comfortably exceeds SSD for moderate speeds and provides margin at higher speeds), and it also matches the capabilities reported in practical traffic-sign recognition work that cites operational recognition ranges near 60 m for typical forward-facing automotive cameras. [7]

Standards and convention text also support the need for visibility and legibility at distance: international agreements such as the Vienna Convention on Road Signs and Signals specify that sign dimensions, placement and reflectivity should allow the sign to be "easily visible for a distance" so that approaching drivers can understand signs in time; this sets a regulatory expectation that automated systems should be designed to read signs at distances comparable to human drivers under normal conditions. [8]

C. What is a Convolutional Neural Network (CNN)?

A Convolutional Neural Network (CNN) is a class of deep neural networks specifically designed for processing grid-like data such as images. [5] CNNs were introduced to overcome the limitations of traditional fully connected neural networks when applied to high-dimensional visual data. [5] Instead of treating each pixel independently, CNNs exploit the spatial structure of images by learning local patterns through convolutional filters [4]

The core idea of a CNN is the convolution operation, where small learnable kernels slide across an image and detect low-level features such as edges, corners, and color contrasts. [5] As the network depth increases, these simple features are combined into more complex representations, such as shapes, symbols, and ultimately entire objects. This hierarchical feature learning makes CNNs especially effective for visual recognition tasks. [4]

An analogy for what a convolution is: think of a flashlight, imagine a dark room with a large painting on the wall, and a small flashlight.The Painting is the input image. The Flashlight is the "filter" or "kernel" (the convolution).The Scanning: sliding the flashlight across the painting, top to bottom, left to right. At every spot, pause and examine only the small area

lit up by the flashlight.The Result: write down what you see in that specific spot (e.g., "this is a vertical line," "this is a curve") onto a new sheet of paper.By the end, that new sheet of paper is not a copy of the painting; it is a map of where specific features are located. This is exactly what a convolution does. [5]

Unlike classical computer vision approaches that rely on handcrafted features, CNNs learn features directly from data in an end-to-end manner. [5] This data-driven learning capability has made CNNs the dominant approach for image classification, object detection, and semantic segmentation. [5] CNNs are ideal for this task because they extract local spatial features like edges, shapes, and textures critical for distinguishing between visually similar traffic signs. [1]

II. The Mathematics in Convolution

The Convolutional Neural Network (CNN) is built upon an elegant mathematical foundation involving convolution, non-linear activation, and pooling operations, which allow it to hierarchically learn complex spatial features from input data like images. [6]

A. The Convolution Operation

The core function of a CNN is the **discrete convolution**, which extracts local features by sliding a small, learned filter (or kernel, K) across the input data (I). [6] In deep learning, this operation is often implemented as a **cross-correlation**. The resulting output is called the feature map, S. [6]

For an input volume I with C channels (e.g., RGB images have $C = 3$) and a kernel K of size $K_H \times K_W \times C$, the output feature map S at spatial position (i, j) is calculated by summing the element-wise products of the kernel and the corresponding receptive field in the input, plus a scalar bias b [6]:

$$S(i,j) = \sum_{c=1}^{C} \sum_{m=1}^{K_H} \sum_{n=1}^{K_W} I(i+m-1, j+n-1, c) K(m,n,c) + b$$

- $I(i+m-1, j+n-1, c)$ is the pixel value of the input volume at spatial coordinates $(i+m-1, j+n-1)$ for channel c.
- $K(m, n, c)$ is the learned weight of the kernel at coordinates (m, n) for channel c.
- b is the single, learned bias term associated with this kernel.

B. The Activation and Pooling Operations

Following the convolution, a non-linear **activation function** is applied element-wise to the feature map S. This introduces non-linearity, which is essential for the network to model complex, real-world data distributions. The most widely used function is the Rectified Linear Unit (ReLU), σ [6]:

$$\sigma(x) = \max(0, x)$$

The ReLU operation replaces all negative values in the feature map with zero, while preserving positive values. [6]

C. The Pooling Operation

The **pooling layer** (or subsampling layer) reduces the spatial dimensions (height and width) of the feature maps. [6] This reduces the computational load and helps to achieve slight translational invariance, making the features robust to small shifts in the input. [6]

Max Pooling is the most common form. For a pooling window of size $F \times F$ applied with a stride s, the output $P(i, j)$ is the maximum value within the corresponding region of the input feature map A [5]:

$$P(i,j) = \max_{m=1}^{F} \max_{n=1}^{F} A(i \cdot s + m - 1, j \cdot s + n - 1)$$

D. Fully Connected Layers

After several alternating layers of convolution and pooling, the 3D feature volume is **flattened** into a 1D vector. This vector is then processed by one or more **Fully Connected (FC)** layers. The output of an FC layer H is calculated as a standard matrix-vector [3] multiplication:

$$H = \sigma(W \cdot F + B)$$

- F is the input vector from the previous layer.
- W and B are the learned weight matrix and bias vector.

E. The Softmax Output

The final FC layer uses the **Softmax function** to transform the raw output scores (logits) into a probability distribution over the K output classes. The probability $\hat{y}_k$ for class k is [7]:

$$\hat{y}_k = \text{Softmax}(z)_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

The resulting $\hat{y}_k$ values represent the model's confidence for each class and sum up to 1.

F. Learning: Loss and Backpropagation

The network is trained by minimizing a **Loss Function** $\mathcal{L}$, which quantifies the error between the predictions and the true labels. For multi-class classification, the **Categorical Cross-Entropy Loss** is standard [6]:

$$\mathcal{L}(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

The learned parameters (weights W and biases b) are updated using an optimization algorithm driven by the back-propagation procedure, which calculates the gradient of the loss with respect to every parameter [6]:

$$\Delta W = \frac{\partial \mathcal{L}}{\partial W} \quad \text{and} \quad \Delta b = \frac{\partial \mathcal{L}}{\partial b}$$

These gradients are then used to update the parameters iteratively, moving towards minimizing the total loss. [6]

III. Key Concepts

There are three "knobs" or hyperparameters that control how this convolution happens:

- **Stride:** How far the kernel moves at each step.
  - *Stride 1:* Moves 1 pixel at a time (overlapping, high detail).
  - *Stride 2:* Jumps 2 pixels (reduces the size of the output).
- **Padding:** What happens at the borders of the image?
  - *Valid:* stop when the kernel hits the edge (output gets smaller).
  - *Same (Zero Padding):* add a border of zeros around the image so the kernel can fit over the edges (output stays the same size).
- **Pooling:** (Often used after convolution) This shrinks the map by keeping only the most important information (e.g., **Max Pooling** keeps only the highest number in a region).

[6]

IV. Why is this useful?

Convolutions provide three massive advantages over standard networks:

1) **Translation Invariance:** A convolution can recognize a "cat" whether the cat is in the top-left corner or the bottom-right corner. It looks for the feature *everywhere*.
2) **Parameter Sharing:** Instead of learning new weights for every single pixel, the network learns **one** filter (e.g., an "edge detector") and applies it to the entire image. This saves a massive amount of memory.
3) **Hierarchy of Features:**
   - *Early layers* detect simple things (lines, edges).
   - *Middle layers* combine those lines to detect shapes (eyes, ears, circles).
   - *Deep layers* combine shapes to detect objects (faces, cars, dogs).

A. Alternatives to CNN

Several alternative approaches could be considered for traffic sign recognition: Traditional Machine Learning (SVMs, k-NN, Random Forests): These methods require handcrafted feature extraction and generally perform poorly when dealing with complex visual variability. [2] Fully Connected Neural Networks: These ignore spatial structure and scale poorly with image size. [5] Transformer-Based Vision Models: While powerful, they require significantly more data and computational resources, making them less suitable for small datasets and educational implementations. [2] CNNs strike an optimal balance between performance, interpretability, computational efficiency, and implementation simplicity. [1] For these reasons, a CNN was selected as the primary algorithm in this work.

V. Dataset Description

The dataset used in this project is a small curated subset of a larger traffic sign dataset. It contains over 50 classes of traffic signs, including: Prohibitory signs (e.g., "No horn", "No entry") Mandatory signs (e.g., "Turn left", "U-turn") Regulatory signs (e.g., speed limits: 30, 40, 50 km/h) The dataset consists of image files arranged in subdirectories, one folder per class. Class labels correspond to the numeric identifiers found in labels.csv, which also maps each class ID to a human-readable sign name. The dataset was extracted into:

```
traffic_sign_small/
    labels.csv
    traffic_Data/
        DATA/
            0/
            1/
            2/
            ...
```

Each folder contains traffic sign images of varying background and lighting conditions. Images are converted to 32×32 RGB format to reduce computational cost, making training feasible even in limited environments such as Google Colab. Traffic sign recognition is a crucial real-world application. Even this scaled-down dataset presents realistic challenges:

- Variation in lighting, blur, and perspective
- Similar-looking signs requiring fine feature discrimination
- Multi-class classification involving over 50 categories

This makes the dataset appropriate for evaluating the effectiveness of CNNs in practical visual tasks.

VI. Methodology

A. Enviorment

The proposed traffic sign recognition system was implemented and trained using Google Colaboratory (Google Colab). This environment was chosen due to its ease of use, reproducibility, and built-in support for hardware acceleration through GPUs. Google Colab allows rapid experimentation without the need for local setup or specialized hardware, making it well suited for developing and evaluating deep learning models. Additionally, its cloud-based nature ensures that the implementation can be easily shared and reproduced, which is particularly valuable in an academic and educational context.

B. Data Preprocessing

All images are resized to 32×32 pixels to reduce computational cost. Pixel values are normalized to the $[-1, 1]$ range for improved model stability.

The dataset is split into:

- 80% training
- 20% validation

ensuring a fair measure of generalization performance.

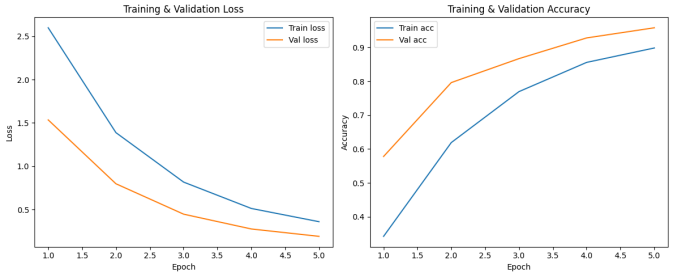

Fig. 1. Training and validation loss/accuracy over epochs.

C. CNN Architecture

The implemented CNN consists of three convolutional blocks followed by two fully connected layers. Each block includes a 3×3 convolution, batch normalization, ReLU activation, and 2×2 max pooling.

D. Training Procedure

We train for five epochs using:

- **Loss**: CrossEntropyLoss
- **Optimizer**: Adam with learning rate 1×10^{-3}
- **Batch size**: 64

This setup balances learning speed with stability.

VII. Results and Evaluation

A. Training Curves

Fig 1 shows the training and validation loss and accuracy curves.

The model improves rapidly in early epochs and continues refining accuracy. Validation accuracy closely tracks training accuracy, showing minimal overfitting.

B. Confusion Matrix

Fig 2 visualizes the confusion matrix for the validation set.

The strong diagonal indicates highly accurate class predictions. The few off-diagonal elements correspond to sign classes with very subtle differences. These results illustrate that the CNN successfully captures key visual patterns despite the small input resolution.

C. Inference Time

The trained convolutional neural network achieved an inference time of approximately 4.51 milliseconds per image, measured as the forward-pass execution time on a single validation sample. This measurement was obtained with the model in evaluation mode and without gradient computation, ensuring that only the classification operation itself was timed. An inference time in this range indicates that the proposed CNN architecture is computationally efficient and well-suited for real-time or near real-time traffic sign recognition applications.

In practical terms, an inference time of 4.51 ms corresponds to the ability to process over 200 images per second, which exceeds the requirements of many real-world driving scenarios where traffic signs appear intermittently rather than

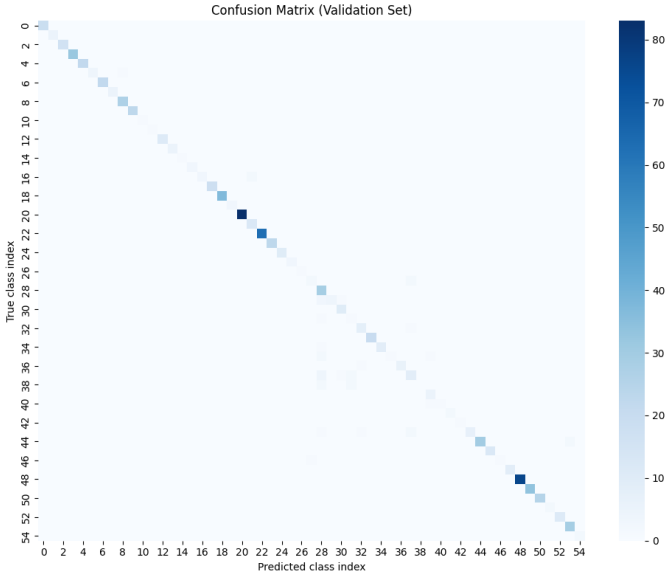

Fig. 2. Confusion matrix for the validation set

continuously. This performance suggests that the model could be deployed in resource-constrained environments, such as embedded systems used in advanced driver-assistance systems (ADAS), without introducing significant latency. Importantly, this efficiency is achieved while maintaining high classification accuracy, demonstrating that the chosen CNN architecture provides a favorable balance between computational cost and recognition performance. [1]

VIII. System-Level Modeling

In a model-based design framework, the overall traffic scene understanding system is divided into functional blocks: Sensing Layer: Front-view camera continuously captures road scenes.The Perception Layer: Deep learning models process captured frames to detect lanes, traffic signs, and text on signposts. The Decision Layer: Recognized traffic signs and lane positions inform vehicle control decisions. Embedded Deployment Layer: The perception model is optimized for deployment on resource-limited devices. This layered abstraction supports iterative development, testing, and optimization, essential for autonomous driving applications. [1]

IX. Discussion

The CNN performs exceptionally well on the validation set, demonstrating that even a lightweight architecture can handle multi-class traffic sign classification. Batch normalization and dropout proved essential in stabilizing training and preventing overfitting. The confusion matrix shows that the model struggles only with signs that are visually very similar, which is expected given the reduced resolution. Overall, the model is computationally efficient and practical for embedded applications.

X. Future Work

Although the proposed convolutional neural network demonstrates strong performance on the traffic sign recognition task, several directions for future work could further improve both accuracy and robustness. An important extension would be the use of data augmentation techniques, such as random rotations, brightness adjustments, scaling, and motion blur, to better simulate real-world driving conditions where traffic signs may be partially occluded, tilted, or captured under varying lighting. [6] Additionally, exploring deeper or more advanced architectures, such as Residual Networks (ResNet), MobileNet, or EfficientNet, could improve feature extraction while maintaining or even reducing inference time, which is especially important for deployment in embedded systems. [5] Transfer learning using pretrained models on large-scale image datasets could also be investigated to improve generalization, particularly when training data is limited.

Another promising area of future work involves optimizing the model for real-time deployment. Techniques such as model pruning, quantization, or knowledge distillation could significantly reduce model size and computational complexity without substantially sacrificing accuracy. [6] This would make the system more suitable for use in resource-constrained environments such as autonomous vehicles, advanced driver-assistance systems (ADAS), or edge devices. Furthermore, future studies could evaluate the model on larger and more diverse datasets, including international traffic signs, to assess its scalability and robustness across different regions and standards. Finally, integrating temporal information from video streams, rather than single images, could enable more reliable recognition by leveraging consistency across consecutive frames. These improvements would move the system closer to practical real-world deployment while deepening the understanding of deep learning methods for traffic sign understanding.

XI. Conclusion

This project successfully implemented a compact convolutional neural network (CNN) capable of accurately classifying traffic sign images. The proposed model achieved high classification accuracy, demonstrated stable convergence during training, and performed consistently across multiple evaluation metrics, including validation accuracy and confusion matrix analysis. These results confirm that deep learning approaches, even when implemented with relatively simple architectures, are well suited for traffic sign understanding tasks. The achieved inference time further highlights the practicality of the proposed system for real-time applications, making it a viable component for intelligent transportation systems and autonomous driving pipelines.

Beyond the implementation itself, this work reinforces the broader effectiveness of CNNs for visual perception tasks involving structured visual patterns such as traffic signs. By automatically learning hierarchical feature representations directly from raw image data, the CNN eliminates the need for handcrafted features and provides robustness against moderate variations in scale, color, and background. [6] The results obtained in this study demonstrate that a carefully designed, lightweight CNN can strike an effective balance between accuracy and computational efficiency, which is critical for real-world deployment scenarios.

Despite these strengths, convolutional neural networks exhibit several inherent limitations. First, CNNs are highly dependent on the availability of sufficiently large and diverse labeled datasets to achieve strong generalization performance. Models trained on limited or biased datasets may struggle when exposed to real-world variations not present during training. [6] Second, CNN performance can degrade under challenging conditions such as extreme lighting changes, occlusions, motion blur, or unusual viewing angles, all of which are common in real driving environments. [6] Third, while CNNs are computationally efficient relative to many deep architectures, they still impose a non-negligible computational cost compared to classical computer vision methods, particularly when deployed on resource-constrained embedded platforms. Finally, standard CNN architectures primarily focus on local spatial features and may lack awareness of long-range spatial or contextual dependencies unless augmented with additional mechanisms or architectural modifications. [6]

These limitations motivate several directions for future research and system improvement. Potential extensions include expanding the dataset to incorporate greater environmental diversity, applying data augmentation strategies to improve robustness, and exploring hybrid or more advanced architectures that integrate global context modeling. Overall, this work demonstrates that compact CNN-based models represent a strong foundation for traffic sign recognition systems and provides a solid baseline upon which more advanced and scalable solutions can be built.

References

[1] I. Čordašić, M. Vranješ, R. Grbić and M. Herceg, "Algorithm for Extracting Signposts from the Scene and Understanding Text on Them," 2024 Zooming Innovation in Consumer Technologies Conference (ZINC), Novi Sad, Serbia, 2024, pp. 60-65, doi: 10.1109/ZINC61849.2024.10579407.

[2] A. Biswas, A. R. Chowdhury and R. Selvanambi, "Lane detection and traffic sign detection for autonomous driving systems," 2nd International Conference on Computer Vision and Internet of Things (ICCVIoT 2024), Coimbatore, India, 2024, pp. 206-211, doi: 10.1049/icp.2024.4425.

[3] S. Khalid, J. H. Shah, M. Sharif, F. Dahan, R. Saleem and A. Masood, "A Robust Intelligent System for Text-Based Traffic Signs Detection and Recognition in Challenging Weather Conditions," in IEEE Access, vol. 12, pp. 78261-78274, 2024, doi: 10.1109/ACCESS.2024.3401044.

[4] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[5] G. Cui, "Research on Recognition and Classification Technology Based on Deep Convolutional Neural Network," 2021 IEEE 3rd Eurasia Conference on IOT, Communication and Engineering (ECICE), Yunlin, Taiwan, 2021, pp. 353-357, doi: 10.1109/ECICE52819.2021.9645706.

[6] W. Wang, G. Wen and Z. Zheng, "Design of Deep Learning Mixed Language Short Text Sentiment Classification System Based on CNN Algorithm," 2022 IEEE 2nd International Conference on Mobile Networks and Wireless Communications (ICMNWC), Tumkur, Karnataka, India, 2022, pp. 1-5, doi: 10.1109/ICMNWC56175.2022.10031786.

[7] L. J. Shyan, T. H. Lim and D. N. Pg Hj Muhammad, ”Real time road traffic sign detection and recognition systems using Convolution Neural Network on a GPU platform,” 2022 International Conference on Digital Transformation and Intelligence (ICDI), Kuching, Sarawak, Malaysia, 2022, pp. 236-240, doi: 10.1109/ICDI57181.2022.10007277

[8] United Nations Economic Commission for Europe, ”Convention on Road Signs and Signals (1968), Consolidated Version,” United Nations, Geneva, Switzerland, 2006. [Online]. Available: https://unece.org