***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. 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. 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. 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. 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]

C. The Mathematics in Convolution

Technically, a convolution involves two main players: the **Input** and the **Kernel**. [6]

D. A. The Input

This is usually an image represented as a grid of numbers (pixels).

[6]

- Example: A 6×6 grid representing a small grayscale image.

E. B. The Kernel (or Filter)

This is a much smaller grid of numbers, typically 3×3 or 5×5, that contains learnable **weights**. [6]

- The numbers in the kernel determine what feature it searches for. For example, a kernel with positive numbers on the left and negative numbers on the right acts as a **vertical edge detector**. [6]

F. C. The Operation (The Sliding Window)

1) **Placement:** The kernel is placed over the top-left corner of the input image.
2) **Multiplication:** The value of each image pixel is multiplied by the corresponding value in the kernel.

3) **Summation:** All these products are added together to get a **single number**.
4) **Movement:** The kernel "slides" (moves) one step to the right and repeats the process.

The result of this sliding operation is a new grid called a **Feature Map** (or Activation Map).

$$\text{Output Pixel} = \sum(\text{Input Pixel} \times \text{Kernel Weight})$$

The fundamental operation of a Convolutional Neural Network (CNN) is the discrete convolution (mathematically implemented as cross-correlation in most frameworks), which enables the extraction of spatial hierarchies from input data [6]. Given an input tensor $\mathbf{X}$ and a learnable kernel $\mathbf{W}$ of size $H \times W$, the value of the output feature map $\mathbf{Y}$ at spatial coordinates (i, j) is computed as:

$$Y_{i,j} = \sigma\left(\sum_{m=0}^{H-1}\sum_{n=0}^{W-1} X_{i+m,j+n} \cdot W_{m,n} + b\right) \tag{1}$$

where b denotes the bias term and σ represents a non-linear activation function. The most prevalent activation is the Rectified Linear Unit (ReLU), defined as $\sigma(x) = \max(0, x)$, which introduces the sparsity necessary for deep representation learning [6]. To manage computational complexity and introduce translation invariance, pooling layers are interleaved between convolutional layers. For a stride s, the Max-Pooling operation is defined as:

$$Y_{i,j} = \max_{m\in[0,H-1],n\in[0,W-1]} \{X_{i\cdot s+m,j\cdot s+n}\} \tag{2}$$

The spatial dimensions of the resulting feature map are governed by the input size N, kernel size K, padding P, and stride S, following the relation $O = \lfloor \frac{N-K+2P}{S} \rfloor + 1$ [6].

II. 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]

III. 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.

IV. 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.

V. 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 transformation pipeline used is shown below:

Listing 1. Image transformation pipeline used during training

```
transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5],
                         std=[0.5, 0.5, 0.5])
])
```

The dataset is split into:

- 80% training
- 20% validation

ensuring a fair measure of generalization performance.

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.

The model is defined as:

Listing 2. Convolutional neural network used in this project

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128*4*4, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )
```

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.

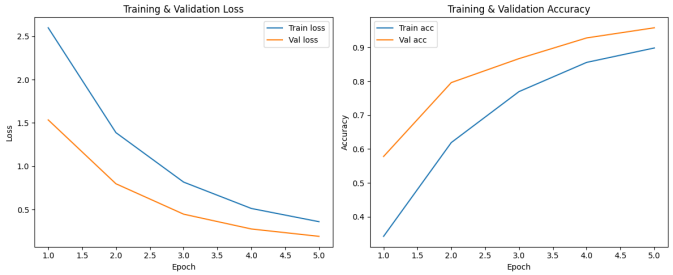

Fig. 1. Training and validation loss/accuracy over epochs.

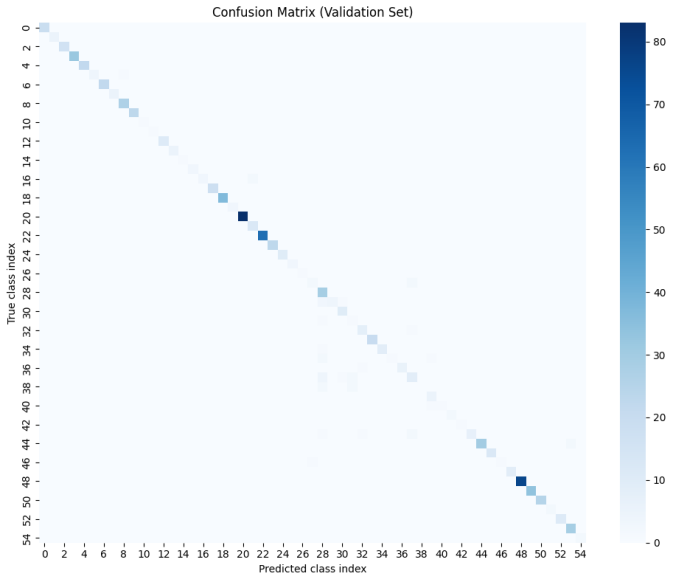

Fig. 2. Confusion matrix for the validation set

VI. 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.

Listing 3. Inference time measurement for a single image

```
import time
import torch

model.eval()

sample_image, sample_label = val_dataset[0]
sample_image = sample_image.unsqueeze(0).to(device

with torch.no_grad():
    _ = model(sample_image)

with torch.no_grad():
    start_time = time.time()
    _ = model(sample_image)
    end_time = time.time()

inference_time_ms = (end_time - start_time) * 1000
print(f"Inference_time_for_a_single_image:
{inference_time_ms:.2f}_ms")
```

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 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]

VII. 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]

VIII. 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.

IX. Conclusion

This project successfully implemented a compact CNN capable of accurately classifying traffic sign images. The model reaches high accuracy, exhibits stable learning behavior, and performs well across all evaluated metrics. This work demonstrates that deep learning, even at modest scale, is an effective tool for real time traffic sign understanding an essential component of autonomous driving and intelligent transportation systems.Despite their effectiveness, CNNs have several limitations: Data Dependence: CNNs require sufficiently diverse labeled data to generalize well. Performance can degrade when trained on small or biased datasets. [6] Sensitivity to Unseen Conditions: Extreme lighting changes, occlusions, or unusual viewpoints can reduce accuracy. [6] Computational Cost: Although efficient, CNNs still require more computation than classical methods, especially for real-time embedded deployment. [6] Limited Global Context Awareness: CNNs focus on local patterns and may struggle with tasks requiring long-range dependencies without architectural modifications. [6] These limitations motivate future extensions such as data augmentation, larger datasets, or hybrid architectures.

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.