# Traffic Sign Recognition Using Convolutional Neural Networks (CNN)

Dominik Barukčić*, Nika Božić*, Borna Josipović*, Andrija Merlin*, Vedran Moškov*
*Faculty of Electrical Engineering and Computing*
*University of Zagreb*
Zagreb, Croatia
{dominik.barukcic, nika.bozic, borna.josipovic, andrija.merlin, vedran.moskov}@fer.hr

***Abstract*—Traffic sign recognition plays a vital role in autonomous driving and traffic monitoring systems. This paper explores the use of Convolutional Neural Networks (CNNs) for the classification of traffic signs using the German Traffic Sign Recognition Benchmark (GTSRB) dataset. A custom-designed neural network was developed, trained, and evaluated to test its effectiveness. Furthermore, the performance of our model was compared with state-of-the-art networks, including ResNet-18, EfficientNet, MobileNetV2 and DenseNet121 to highlight its strengths and limitations. Our results demonstrate [summarize findings briefly, e.g., high accuracy, specific insights]. The study emphasizes the potential of CNNs for robust traffic sign recognition while providing insights into optimizing neural network architectures for this task.**

## I. Introduction

Traffic sign recognition is a crucial component of advanced driver-assistance systems (ADAS) and autonomous vehicles. Accurate recognition and classification of traffic signs ensure safe navigation, compliance with traffic regulations, and prevention of accidents. With the increasing reliance on intelligent systems in transportation, developing robust and efficient methods for traffic sign recognition has become a critical research area.

The German Traffic Sign Recognition Benchmark (GTSRB) [1] dataset provides a comprehensive collection of traffic sign images, encompassing diverse classes and challenging scenarios such as varying illumination, occlusion, and distortions. Leveraging this dataset, researchers have explored machine learning and deep learning techniques to improve classification accuracy and system reliability.

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision, offering state-of-the-art performance in image recognition tasks. Their ability to automatically learn hierarchical features from raw images makes them particularly suited for traffic sign recognition. In this work, we present a comparative study involving a custom-designed CNN and several well-established deep learning architectures on the GTSRB dataset.

The objectives of this study are as follows:

1) To design and implement a lightweight CNN tailored for traffic sign recognition.
2) To evaluate the performance of the proposed model on the GTSRB dataset.
3) To compare the results of the custom CNN with renowned architectures, such as ResNet and EfficientNet, in terms of accuracy and computational efficiency.
4) To analyze the advantages and limitations of different models for this task.

The rest of this paper is organized as follows: Section II gives an overview of the existing approaches to the selected problem. Section III describes the GTSRB dataset and preprocessing steps. Section IV outlines the architecture and design of the proposed CNN. Section V presents the experimental setup, results, and comparison with other models. Section VI discusses the findings and their implications. Finally, Section VII concludes the paper and outlines directions for future research.

## II. Overview of Existing Approaches to Solving the Selected Problem

The problem of traffic sign recognition has garnered considerable attention in the field of computer vision, primarily due to its critical role in autonomous driving systems and intelligent transportation. This section provides an overview of existing approaches, encompassing both traditional methods and contemporary techniques based on deep learning.

### A. Traditional Methods

Traditional methods for traffic sign recognition involved hand-crafted feature extraction followed by classification using conventional machine learning algorithms. These methods typically consisted of the following stages:

- **Feature Extraction:** Techniques like Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), and Speeded-Up Robust Features (SURF) were commonly utilized to extract geometric and texture-related features from traffic sign images.
- **Classification:** Algorithms such as Support Vector Machines (SVM), Random Forests, and K-Nearest Neighbors (KNN) were used to classify traffic signs based on the extracted features.

While these methods performed well on simple and constrained datasets, they struggled to handle real-world challenges such as occlusions, variations in lighting, and diverse sign appearances.

### B. Deep Learning-Based Methods

In recent years, deep learning has emerged as a dominant approach for traffic sign recognition. Convolutional Neural Networks (CNNs), in particular, have revolutionized the field by enabling automatic feature extraction and robust classification. Key advancements include:

- **Convolutional Neural Networks:** Architectures such as AlexNet, VGGNet, and ResNet have demonstrated high accuracy in image recognition tasks, including traffic sign recognition.
- **Transfer Learning:** Pretrained models like EfficientNet and MobileNetV2 have been fine-tuned on traffic sign datasets, leveraging their learned features to achieve state-of-the-art results with minimal additional training.
- **Data Augmentation and Regularization:** Techniques such as rotation, scaling, and flipping have been applied to augment the dataset, improving model generalization and robustness.

Deep learning approaches have shown remarkable performance in terms of accuracy and reliability, even in challenging scenarios with noisy or partially occluded data.

### C. Comparison of Approaches

Traditional methods rely heavily on feature engineering, requiring significant domain expertise and often resulting in suboptimal performance on complex datasets. In contrast, deep learning-based methods offer a more automated and scalable solution, achieving superior accuracy by learning hierarchical representations directly from raw input data.

## III. German Traffic Sign Recognition Benchmark (GTSRB) Dataset

The German Traffic Sign Recognition Benchmark (GTSRB) dataset is a widely recognized benchmark for multi-class traffic sign classification. It was originally introduced as part of a challenge at the International Joint Conference on Neural Networks (IJCNN) in 2011. Designed to promote research in traffic sign recognition and computer vision, the dataset provides a comprehensive collection of traffic sign images, reflecting real-world conditions such as varying lighting, rotations, and partial occlusions.

The GTSRB dataset is publicly available and was obtained for this study from Kaggle at this link. The dataset has several noteworthy characteristics:

1) **Multi-Class Classification**: The dataset contains images spanning 43 classes.
2) **Large Dataset Size**: It comprises over 50,000 labeled images, providing sufficient data for training and testing.
3) **Realistic Variations**: The images include various challenges commonly encountered in real-world applications, such as perspective distortions, blurring, varying resolutions, and changes in weather or lighting.
4) **Lifelike Data Distribution**: The dataset includes a realistic distribution of traffic signs, simulating real-world driving conditions.


Fig. 1. Examples of images of various classes from the dataset.

### Dataset Structure

The dataset is manually divided into a training set and a test set, the test set being 20% of the original dataset. Each image is accompanied by a label corresponding to the traffic sign category it represents. The training set contains a rich diversity of images for each class, which will contribute to the robustness of the trained model. The test set is utilized for performance evaluation, ensuring that our model is tested on unseen data.

### Dataset Preprocessing

To prepare the German Traffic Sign Recognition Benchmark (GTSRB) dataset for use in our models, we performed several preprocessing steps to ensure the images were appropriately formatted and standardized. The steps are described below:

**1. Loading and Organizing the Data:**

- The GTSRB dataset is organized into folders, with each folder corresponding to a specific traffic sign category (0–42). The images are stored within these category folders.
- The training data was loaded by iterating through these category folders and reading all image files.

**2. Resizing and Normalization:**

- Each image was resized to $64 \times 64$ pixels to ensure uniformity in input size. This resolution was selected to balance computational efficiency and the retention of essential image details.
- The pixel values of the images were normalized to standardize the data. Each image was converted to RGB format, followed by scaling each pixel to the range [0, 1].

**3. Data Structure:**

- The preprocessed images were stored in a NumPy array of shape $(\text{num\_images}, 64, 64, 3)$, representing the size of

the dataset, the dimensions of each image ($64 \times 64$), and the three color channels (red, green, and blue).
- Corresponding labels, representing the traffic sign category for each image, were stored in another NumPy array.

## IV. Custom CNN Architecture

For this study, a custom Convolutional Neural Network (CNN) was designed and implemented to classify traffic signs in the GTSRB dataset. Below, we detail the layers of the custom CNN.

### *Architecture Overview*

The TraffiKING model is designed with a modular structure, consisting of convolutional layers for feature extraction, batch normalization for stability, and fully connected layers for classification. The architecture can be summarized as follows:

1) **Input Layer:** The input to the network is a $64 \times 64$ pixel RGB image (3 channels).
2) **Feature Extraction Layers:** The model extracts features through the following layers:
   - **BatchNorm2d + ReLU Activation:** The input image is normalized and passed through a ReLU activation function to improve convergence during training.
   - **Conv1:** A convolutional layer with 32 filters of size $3 \times 3$, stride 1, and padding 1. This layer captures low-level features such as edges and textures.
   - **BatchNorm2d + ReLU Activation:** Batch normalization is applied to the output of Conv1 to stabilize learning, followed by a ReLU activation function for non-linearity.
   - **Conv2:** A second convolutional layer with 64 filters of size $3 \times 3$, stride 1, and padding 1. This layer extracts higher-level features from the input.
   - **Max Pooling:** A $2 \times 2$ max pooling operation reduces the spatial dimensions by half, summarizing feature maps and lowering computational complexity.
3) **Flattening Layer:** After the max pooling operation, the feature maps are flattened into a single-dimensional vector. For a $64 \times 64$ input image, this vector has dimensions of $64 \times 32 \times 32$ (64 filters of $32 \times 32$ each).
4) **Fully Connected Layer:**
   - **FC1:** The flattened vector is directly connected to a layer with 43 neurons (one for each class). This serves as the output layer, where the activations represent the probabilities of the image belonging to each traffic sign class.
5) **Output Layer:** The output is a vector of size 43, with each element representing the confidence score for a particular class. The class with the highest score is selected as the predicted class.

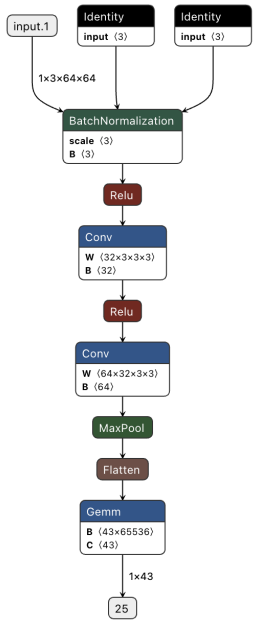


Fig. 2. Custom TraffiKING network architecture.

## V. Experimental Process

### *Methodology*

The experiments and analyses conducted in this study were carried out using the Python programming language in a Jupyter Notebook environment. For this study, the following libraries were used:

- **NumPy** [2]: Used for efficient numerical computations and handling multidimensional arrays, used for dataset storage and handling.
- **OpenCV**: Utilized for image processing tasks such as reading, resizing, and color conversion of images from the dataset.
- **os**: Used to navigate and manage file system paths for dataset organization and image loading.
- **Torchvision [3]**: Part of the PyTorch library, it provided utilities for handling image datasets, including preprocessing and transformation functions.

### *Training Methodology*

Below, the training methodology, optimization techniques, and key parameters are described in detail:

1) **Data Preparation:** The dataset was split into training, validation, and testing sets. The training set was used to update model weights, the validation set to monitor performance during training, and the testing set to evaluate the final model. Data augmentation techniques, including resizing and normalization, were applied to improve generalization.
2) **Training Configuration:** The training process was configured with the following settings:

- *Loss Function:* Cross-Entropy Loss (`nn.CrossEntropyLoss`) was used, as it is well-suited for multi-class classification tasks.
- *Optimizer:* Adam Optimizer (`optim.Adam`) was used to update model weights, with an initial learning rate of 0.001.
- *Learning Rate Scheduler:* A Step Learning Rate Scheduler (`lr_scheduler.StepLR`) was implemented to reduce the learning rate by a factor of 0.1 every 4 epochs, helping the model converge more effectively.

3) **Training Process:** The training loop involved the following steps for each batch of images:
   - The model and data were transferred to the GPU.
   - Forward propagation was performed to compute predictions.
   - The loss was calculated using the Cross-Entropy Loss function.
   - Backpropagation was executed to compute gradients, followed by an optimization step to update weights.
   - The learning rate scheduler was applied after each batch to adjust the learning rate dynamically.

   During each epoch, the model's performance was assessed on both the training and validation sets. The accuracy and loss were logged for analysis. The best-performing model (based on validation accuracy) was saved for later evaluation.

*Testing and Evaluation*

After training, the custom CNN was evaluated on the testing dataset to measure its final performance. The evaluation process involved the following steps:

### A. Testing Configuration

The model was switched to evaluation mode to disable gradient computations and dropout layers (if any). This ensured consistent predictions during testing. The model's predictions were compared with the true labels, and the overall accuracy was calculated as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}} \times 100 \quad (1)$$

The test accuracy was logged and compared with the validation accuracy to assess the generalization capability of the model. For comparative analysis, a pretrained version of the custom CNN model (saved during training) was loaded and tested on the same testing dataset. This allowed for consistent benchmarking.

To assess the performance of our custom CNN model, we conducted a comparative analysis against several well-established models widely used in image classification tasks: **ResNet-18** [4], **DenseNet121** [5], **EfficientNet-B0** [6], and **MobileNetV2** [7] (a compact and efficient model). The evaluation was performed on the validation set, using both pretrained versions of the models and those initialized with random weights. This approach ensured a comprehensive understanding of the performance differences across models under varying conditions.

## VI. Results and Comparative Analysis

### A. Accuracy measures

The recorded validation accuracies of the pretrained models are as follows:

- **TraffiKING** (custom neural network): 97.28%
- **ResNet-18** : 99.90%
- **EfficientNet-B0**: 99.96%
- **DenseNet121**: 99.96%
- **MobileNetV2**: 99.94%

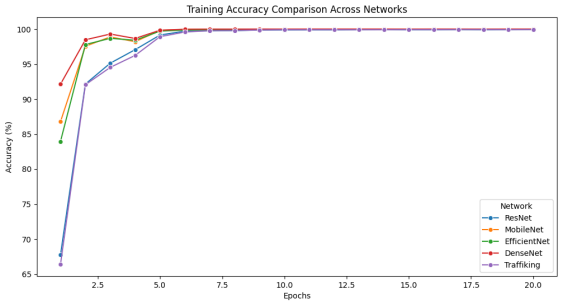


Fig. 3. Training accuracies of the pretrained models.

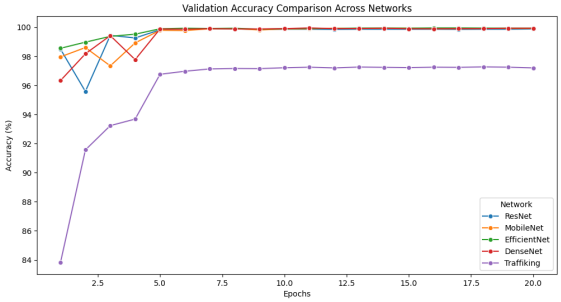


Fig. 4. Validation accuracies of the pretrained models.

**Key findings:**

- The **TraffiKING** custom CNN model demonstrated reasonable accuracy, though the pretrained models generally outperformed it in terms of classification accuracy.
- **EfficientNet-B0** and **DenseNet121** provided the best accuracy.
- **ResNet-18** also performed well, benefiting from its residual architecture.
- **MobileNetV2**, though designed for efficiency, provided competitive accuracy with a significantly smaller model size.

It is important to highlight the trade-off between the size of the model and the accuracy of the model. Our custom made

model achieves a reasonably high accuracy on the validation set while maintaining a small factor, much smaller than the state of the art models it was compared to. The table below highlights the discrepancy quite well:

| Model | Best Validation Accuracy | State Dict Size | Parameters |
|---|---|---|---|
| **TraffiKING** | 97.28% | 10.8MB | 2,841,649 |
| **ResNet-18** | 99.90% | 44.7 MB | 11,689,512 |
| **MobileNetV2** | 99.94% | 13.6MB | 3.4M |
| **EfficientNet-B0** | 99.96% | 20.5MB | 5.3M |
| **DenseNet121** | 99.96% | 31MB | 7,978,856 |

TABLE I
COMPARISON OF DIFFERENT MODELS IN TERMS OF VALIDATION ACCURACY, STATE DICTIONARY SIZE AND NUMBER OF PARAMETERS

As mentioned, our custom model was also compared to models with randomly initialized weights. When evaluated with randomly initialized weights, the models achieved the following best validation accuracies:

- **TraffiKING** (custom neural network): 97.28%
- **ResNet-18**: 99.76%
- **EfficientNet-B0**: 99.63%
- **DenseNet121**: 99.80%
- **MobileNetV2**: 99.43%

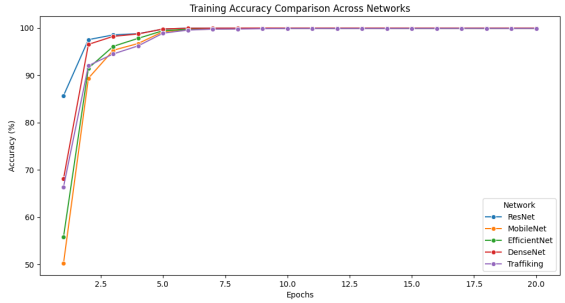


Fig. 5. Training accuracies of models with randomly initialized weights.

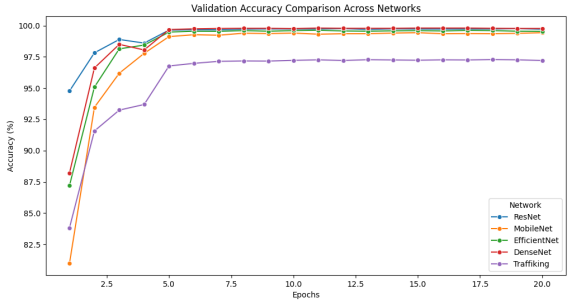


Fig. 6. Validation accuracies of models with randomly initialized weights.

**Key findings:**

- The **TraffiKING** demonstrated consistent performance across both scenarios, achieving a validation accuracy of 97.28
- Among the randomly initialized models, DenseNet121 outperformed the others with an accuracy of 99.80
- Pretrained models consistently outperformed their randomly initialized counterparts, highlighting the advantages of transfer learning on image classification tasks.

### *B. Domain change analysis*

A *domain change* refers to evaluating a model's performance on data that originates from a different distribution [8] than the training data. This scenario is common in real-world applications where training and testing datasets might differ in conditions such as lighting, weather, image quality, or geographic region. The objective is to measure how well models generalize to unseen, real-world conditions.

**Results:** The reported test accuracies highlight the performance of each model on the new domain:

| Model | Test Accuracy (%) |
|---|---|
| **TraffiKING** | 64.32 |
| **ResNet18** | 90.23 |
| **EfficientNet-B0** | 83.64 |
| **MobileNetV2** | 84.51 |
| **DenseNet121** | 93.25 |

TABLE II
TEST ACCURACIES ON THE NEW DOMAIN DATASET FOR DIFFERENT MODELS.

**Key findings:**

- **TraffiKING Model:** While achieving a test accuracy of 64.32%, the custom **TraffiKING** model demonstrates a solid performance given its relatively simpler architecture compared to the larger, more complex pretrained models.
- **ResNet18:** With a test accuracy of 90.23%, **ResNet18** adapts well to the new domain, leveraging its residual architecture to handle domain shifts effectively. However, its higher computational demands may not always align with resource-limited deployments.
- **EfficientNet-B0 and MobileNetV2:** These models achieve competitive test accuracies of 83.64% and 84.51%, respectively. Both are designed to balance efficiency and accuracy, making them strong contenders in scenarios where both performance and computational efficiency are priorities.
- **DenseNet121:** Delivering the highest accuracy of 93.25%, **DenseNet121** shows excellent domain adaptability and feature extraction capabilities.

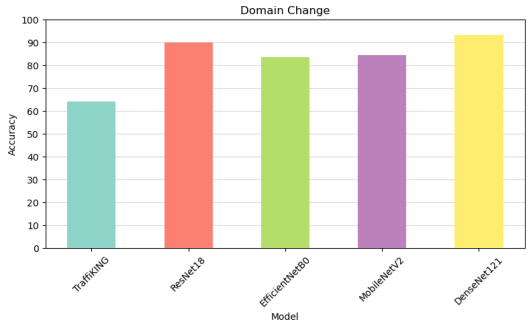


Fig. 7. Model accuracies after domaing change.

## VII. Conclusion

This work presented an exploration of traffic sign recognition using convolutional neural networks, focusing on the performance of a custom-designed model, TraffiKING, and its comparison with well-established pretrained models such as ResNet-18, EfficientNet-B0, DenseNet, and MobileNetV2.

The study demonstrated that while the custom TraffiKING model performed reasonably well, achieving high accuracy in its primary domain, pretrained models exhibited superior generalization capabilities, particularly when applied to a new domain. These results emphasize the importance of leveraging transfer learning and advanced architectures to tackle complex real-world challenges effectively.

Key contributions of this work include:

- Designing and evaluating a lightweight custom neural network tailored for traffic sign recognition.
- Demonstrating the impact of domain shifts on model performance and highlighting the robustness of pretrained models.
- Providing a detailed analysis of traditional and modern approaches to traffic sign recognition, fostering a better understanding of the field's evolution.

Future work could explore hybrid approaches, combining the efficiency of custom models with the robust feature extraction capabilities of pretrained architectures. Additionally, extending the evaluation to include real-time inference scenarios and diverse datasets would provide further insights into the practical applications of traffic sign recognition systems.

In summary, this research contributes to the ongoing development of intelligent transportation systems by showcasing both the challenges and opportunities inherent in applying machine learning to traffic sign recognition.

## References


[1] https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign (accessed 16th January)

[2] https://numpy.org/doc/2.1/index.html (accessed 16th January)

[3] https://pytorch.org/vision/stable/index.html (accessed 16th January)

[4] 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 (pp. 770-778).

[5] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).

[6] Tan, M., Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning (pp. 6105-6114). PMLR.

[7] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

[8] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., ... Lempitsky, V. (2016). Domain-adversarial training of neural networks. Journal of machine learning research, 17(59), 1-35.