TRAFFIC SIGN RECOGNITION SYSTEM USING DEEP LEARNING

[1]Sri. S. RAMA GOPALA REDDY, Associate Professor

[2]B. Tharun Kumar [2]P. Vamsi [2]N. Aditya Varma [2]I. Akshay Kumar

[1] Associate Professor of SRKR Engineering College, Dept of IT, Bhimavaram-534204, India

[2] Students of SRKR Engineering College, Dept of IT, Bhimavaram-534204, Andhra Pradesh, India

ABSTRACT

The requirement for traffic sign recognition has grown in importance along with the demand for driver assistance systems and autonomous cars. This is essential for maintaining the effectiveness and safety of driving assistance systems and self-driving automobiles, greatly enhancing traffic monitoring and safety protocols. This paper presents a technique that uses convolutional neural networks (CNNs) to recognise traffic signs with high accuracy in a variety of settings. Preprocessing traffic sign data in an efficient manner is essential to guarantee accurate detection in autonomous driving systems. Creating a deep neural network model especially for traffic sign classification is the suggested method. The ability to read traffic signs is made possible by this model, which is essential for all driver assistance programmers and autonomous cars. The process described includes crucial phases.

Keywords: Deep learning, Keras , TensorFlow , OpenCV ,Convolution neural network

1.INTRODUCTION

Traffic sign identification is a focus, in computer vision research and a vital element of Advanced Driver Assistance Systems (ADAS). It comprises two areas; detecting and recognizing traffic signs, where the accuracy of detection directly influences the identification results. Traffic signs play a role in providing information on road safety current traffic conditions, traffic rules and potential dangers assisting drivers in navigating roads safely. In todays technology driven world with a growing reliance on AI leading companies such, as Google, Tesla, Uber, Ford, Audi, Toyota and Mercedes Benz are dedicated to enhancing vehicle automation technology for autonomous or self driving vehicles.

There are worries, about the safety of vehicles and the potential for accidents. We must recognize the work of researchers in improving road safety and accuracy through algorithms. When driving, encountering traffic signals such as traffic lights, speed limits, directional arrows and warning signs is an occurrence. Following these signals correctly is essential, for navigation. Self driving cars must. React to these signals to operate effectively.

Categorizing traffic signs requires identifying the category to which each sign belongs. In our Deep Learning project we employ a Convolutional Neural Network (CNN) using the Karas library to develop a model of classifying different kinds of traffic signs. The dataset comprises, over 30,000 images showing types of traffic signs, like speed limits, pedestrian crossings and traffic signals.

The dataset contains, around 43 classification categories with varying numbers of images in each category. Some categories have images compared to others. Despite its size of 314.36 MB the dataset allows for downloading and storage. CNNs, which

are neural networks designed for processing pixel inputs are highly effective in tasks related to image recognition and processing. By utilizing the processing mechanisms of the brain CNNs can efficiently handle complete images making it easier to address challenges associated with processing images piece, by piece.

Pythons Keras is a user deep learning library that simplifies the implementation of networks. While it may not be the option developers prefer Keras for its ease of use and seamless compatibility, with TensorFlow. Acting as the high level interface for TensorFlow the combination of TensorFlow Core API and Keras offers developers a platform with intuitive tools, for deep learning tasks. This integration enables prototyping and deployment of machine learning solutions speeding up the process of tackling challenges.

2. LITERATURE REVIEW

Zaibi et al. [1]: They propose a lightweight traffic sign classification model based on enhancing the LeNet-5 architecture, highlighting the benefits of batch normalization, dropout layers, and grayscale input images. Their enhanced model achieves high accuracy on the German Traffic Sign Recognition Benchmark dataset while being more compact and efficient, with suggestions for exploring lightweight architectures and real-world deployment challenges.

Bharath Kumar et al. [2]: They present a CNN-based approach for traffic sign detection, emphasizing the effectiveness of data augmentation techniques and hierarchical feature learning. Achieving high accuracy on the German Traffic Sign Recognition Benchmark dataset, they suggest further exploration in advanced architectures and real-world challenges like illumination variations and occlusions.

Ellahyani et al. [3]: Their survey comprehensively reviews traffic sign detection techniques, categorizing them based on color information, shape analysis, and machine learning models including CNNs. They highlight the dominance of CNNs in achieving high accuracy and discuss challenges like dataset collection and real-world deployment, with suggestions for future exploration in lightweight model compression and domain adaptation.

Sakthivel et al. [4]: They develop a CNN model for traffic sign recognition, emphasizing the significance of data augmentation and proposing avenues for further exploration in advanced architectures and real-world challenges. Their model achieves high accuracy on the German Traffic Sign Recognition Benchmark dataset, paving the way for future research in optimizing models for embedded deployment and integrating with other perception modules.

Rustad et al. [5]: Proposing a CNN model for Indonesian traffic sign classification, they achieve promising accuracy and suggest further exploration in advanced architectures and data augmentation techniques. Their study provides insights for improving classification accuracy and robustness in real-world scenarios.

3. EXISTING SYSTEMS

Traditional approaches for traffic sign detection and recognition relied heavily on hand-crafted features and classical machine learning algorithms like support vector machines (SVMs), random forests, etc. These methods involved preprocessing steps like color thresholding, edge detection, and shape analysis to identify candidate traffic sign regions, followed by extracting features like histograms of oriented gradients (HOG), color histograms, and geometric descriptors for classification. While these techniques showed reasonable performance, they had several limitations:

1)Dataset Limitations: Hand-crafted features often require extensive dataset curation and labeling, which is time consuming and labour intensive. The availability of large, diverse, and accurately labeled datasets is a bottleneck.

2)Environmental Variations: Traditional methods struggle to handle significant variations in lighting conditions, weather, and scene complexity, leading to performance degradation in real-world scenarios.

3)Scalability: As the number of traffic sign classes increases or new sign designs are introduced, existing systems may require extensive feature engineering and retraining, making them less scalable and adaptable.

4) Real Time Performance: Many traditional approaches are computationally intensive, making it difficult to achieve real-time performance, especially on resource-constrained embedded devices in vehicles.

5) Lack of Generalization: Hand-crafted features and models trained on specific datasets may not generalize well to unseen traffic sign variations or different geographic regions, limiting their practical applicability.

4. PROPOSED SYSTEM

Our main goal is to create a CNN model that outperforms the accuracy of models and can be used for all types of traffic signs. The process we followed includes importing and preparing the dataset optimizing data, for better model performance designing the model using CNN structure and testing it with sets of training and testing data.During development we kept the design simple by using a layers max pooling layers and batch normalization layers to ensure accuracy. We stopped adding layers when we noticed overfitting and achieved loss during training.

Our strategy involves training a CNN model from scratch on the GTSRB dataset with 39,209 training images. This approach aims to enhance accuracy in recognizing traffic signs across categories while maintaining flexibility, in adapting to types of signs.

4.1 PROBLEM STATEMENT

In the context of increasing demand for autonomous vehicles and driver assistance systems, the critical need for accurate and reliable traffic sign recognition systems has emerged. The goal of this project is to develop and implement a robust traffic sign recognition solution using convolutional neural networks (CNNs) to enable autonomous vehicles and driver assistance systems to accurately identify and interpret traffic signs in real-time scenarios. The project aims to address challenges such as effective preprocessing of traffic sign data, optimizing CNN model architecture for classification accuracy, and ensuring the system's robustness and generalization across diverse traffic sign types and environmental conditions.

5. SYSTEM ARCHITECTURE

There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. When these layers are stacked, a CNN architecture will be formed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined below.

1. Convolutional Layer

The convolutional layer extracts features by applying a filter to the input image, producing a feature map that highlights edges and corners. This layer maintains spatial relationships between pixels and is fundamental in recognizing intricate patterns within images.

2. Pooling Layer

The Pooling Layer in a CNN follows the Convolutional Layer, aiming to decrease the size of the feature map, thus reducing computational costs. It operates independently on each feature map, employing methods like Max Pooling, Average Pooling, or Sum Pooling to summarize features. Acting as a bridge between convolution and fully connected layers, it enables independent feature recognition, leading to reduced computational load.

3. Fully Connected Layer

The Fully Connected (FC) layer connects neurons between different layers in a CNN, typically positioned before the output layer. Input from previous layers is flattened and passed to the FC layer for classification. Multiple FC layers allow for complex mathematical operations and enhance classification accuracy. These layers minimize human supervision in the CNN architecture.

4. Dropout Layer

To address overfitting in training datasets, dropout layers are employed in neural networks. Dropout randomly removes a portion of neurons during training, effectively reducing the model's size and complexity. For instance, setting a dropout of 0.3 means 30% of nodes are randomly dropped out from the neural network. This technique prevents overfitting by making the network more robust and simpler, ultimately improving model performance.

5. Activation Functions

Activation functions play a crucial role in CNN models, as they introduce non-linearity and determine which information should be transmitted forward in the network. Commonly used activation functions include ReLU, Softmax, tanH, and Sigmoid, each serving specific purposes. For binary classification CNN models, Sigmoid and Softmax functions are preferred, while Softmax is generally used for multi-class classification. Activation functions decide whether a neuron should be activated based on the input, thus influencing the network's predictive capabilities through mathematical operations.

6. SYSTEM IMPLEMENTATION

6.1 DATA COLLECTION

Each of the 43 folders in our dataset folder represents a different class. We iterate over all of the classes using the OS module, appending images and their labels to the data and labels list. To open picture content into an array, the PIL library is utilized. Finally, we organized all of the images and labels into lists (data and labels).To feed the model, we must turn the list into NumPy arrays. The train_test_split() method in the sklearn package is used to split training and testing data. To transform the labels in y_train and y_test into one-hot encoding, we utilize the to_categorical method from the Keras.utils package.

6.2 DATA PREPROCESSING

1.Image Resizing: Resizing original images, typically with dimensions of 450x600 pixels, to a smaller size, such as 30x30 pixels, is a crucial preprocessing step. This resizing reduces the computational burden, particularly for machine learning models with limited resources, while preserving essential details. By resizing images, we can train models more efficiently, leading to improved performance and faster computation.

2.Normalization: Normalization is another essential preprocessing step that involves scaling raw image pixel values to a standardized range. Typically, raw pixel values range from 0 to 255. Normalization entails dividing each pixel value by 255, thereby scaling all values to a range between 0 and 1. This ensures consistency in the data format across the dataset, facilitating better model learning and faster training times. In our model, we employ the standard scaler technique for normalization to enhance model performance.

3.Grayscale Conversion: Grayscale conversion is a preprocessing technique wherein colored images are transformed into grayscale representations. This process involves computing the weighted average of the RGB channels for each pixel, resulting in a single-channel image where pixel values represent gray intensity ranging from 0 to 255. Grayscale conversion simplifies images to focus on luminance information, which can aid in image processing tasks.

6.3 MODEL TRAINING

For convolution neural network model training the steps required are **Initialization:**

Initialize the CNN's weights and biases randomly.

Forward Pass: Input data is fed forward through the network, passing through convolutional and pooling layers to extract features and reduce dimensionality.

Prediction and Loss Calculation: Predictions are made based on the output layer's activation. Loss is calculated by comparing these predictions with the ground truth labels using a predefined loss function.

Backward Pass (Backpropagation): Errors are propagated backward through the network, computing gradients of the loss function with respect to the model parameters.

Parameter Updates (Optimization): Model parameters are updated using an optimization algorithm (e.g., stochastic gradient descent) to minimize the loss function. Regularization techniques may be applied to prevent overfitting.

Iteration and Epochs: Repeat steps 0 for multiple batches of training data, completing one epoch. Continue for the specified number of epochs, updating the entire dataset.

Validation and Evaluation: Periodically evaluate the model's performance on a separate validation dataset to monitor generalization. Metrics like accuracy and loss are computed.

Completion and Model Deployment: Training concludes after the specified epochs. The final trained model is evaluated on a test dataset. Once satisfactory, it can be deployed for inference on new data**.**

7.RESULTS AND DISCUSSIONS

7.1 Model Loss & Accuracy

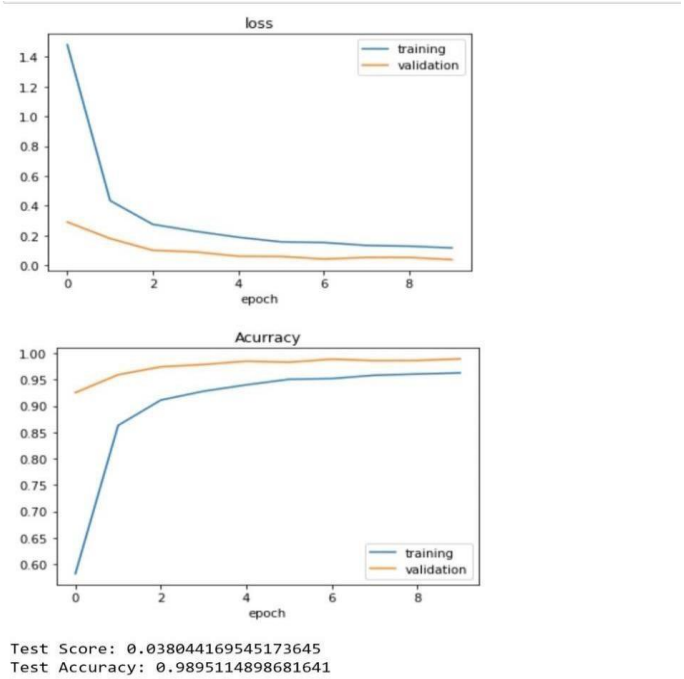

Fig-5: Model Accuracy

This graph visualizes the model's learning process over training epochs. The blue curve (training loss) shows how well the model performs on the training data as it learns. Ideally, this decreases as the model improves. The orange curve (validation loss) tracks performance on unseen validation data. It's crucial for preventing overfitting, where the model memorizes training data but struggles with new examples. In a successful training run, both curves should generally go down, indicating the model is learning effectively from the training data while still generalizing well to unseen data.

7.2 Sample Results

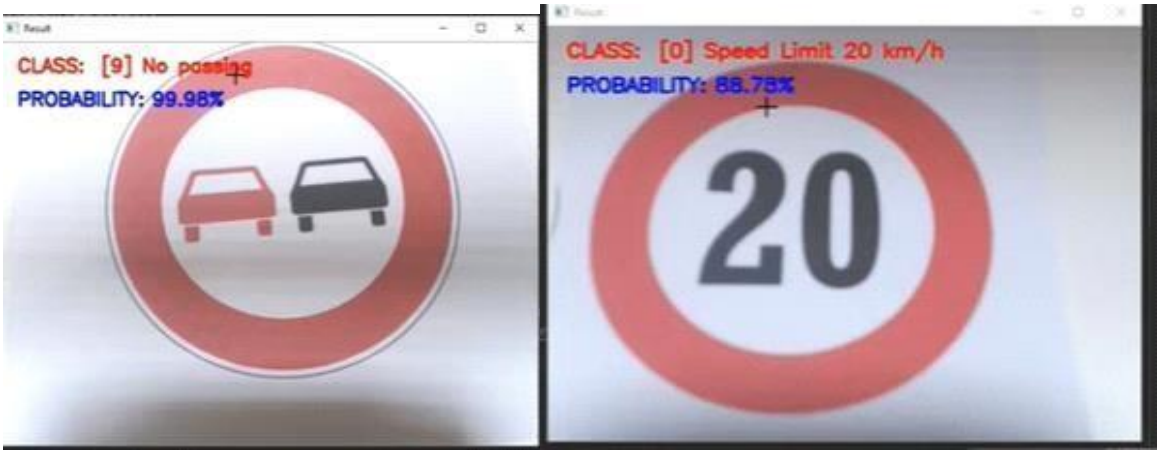

fig.sample outputs

8. CONCLUSION

The paper proposes a method for traffic sign recognition leveraging deep learning techniques, specifically a convolutional neural network (CNN) implemented using KerasThe paper proposes a method for traffic sign recognition leveraging deep learning techniques, specifically a convolutional neural network (CNN) implemented using Keras. The primary objective is to detect various types of traffic signs. Through image preprocessing and utilizing a dataset sourced from Kaggle, the method achieves effective detection, recognition, and classification of traffic signs. These results enable the identification of traffic signs, providing assistance to users in both manual and automatic driving modes. In manual mode, the results are displayed on the dashboard screen, while in automatic mode, the system aids safe driving by recognizing traffic signs for the car. Test results indicate that the method achieves a very high level of accuracy.

9. BIBLIOGRAPHY

1. Zaibi A., Ladgham A., & Sakly A. (2021). A Lightweight Model for Traffic Sign Classification Based on Enhanced LeNet-5 Network. Journal of Sensors, 2021.

2. G. Bharath Kumar, N. Anupama Rani, “TRAFFIC SIGN DETECTION USING CONVOLUTION NEURAL NETWORK A NOVEL DEEP LEARNING APPROACH”, International Journal of Creative Research Thoughts (IJCRT), ISSN:2320-2882, Vol.8, Issue 5, May 2020.

3. A.Ellahyani, I. E. Jaafari, and S. Charfi, “Traffic Sign Detection for Intelligent Transportation Systems: A Survey,” E3S Web of Conferences, vol. 229, p. 01006, 2021.

4. Sakthivel, K., Raghul, B., & Raghul, E. (2022). Traffic sign recognition system using CNN and Keras.

5. Pradana, A.I.; Rustad, S.; Shidik, G.F.; Santoso, H.A. Indonesian Traffic Signs Recognition Using Convolutional Neural Network. In Proceedings of the 2022 International Seminar on Application for Technology of Information and Communication (iSemantic), Semarang, Indonesia, 17–18 September 2022; pp. 426–430.