

TRAFFIC SIGN RECOGNITION SYSTEM USING CNN AND KERAS

A Mini Project Report

*Submitted to the APJ Abdul Kalam Technological University
in partial fulfillment of requirements for the award of Minor Degree*

Bachelor of Technology

in

Computer Science and Engineering

by

Adith T M (NSS20EC004)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NSS COLLEGE OF ENGINEERING, PALAKKAD
KERALA
DECEMBER 2023**

DEPT. OF COMPUTER SCIENCE & ENGINEERING
NSS COLLEGE OF ENGINEERING, PALAKKAD
2023-24



CERTIFICATE

This is to certify that the report entitled **TRAFFIC SIGN RECOGNITION SYSTEM USING CNN AND KERAS** submitted by **Adith T M** (NSS20EC004), to APJ Abdul Kalam Technological University in partial fulfillment of the B.Tech. Minor degree in Computer Science and Engineering is a bonafide record of the project work carried out by him under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Dr. Viji Rajendran
(Mini Project Guide and HOD)
Associate Professor
Dept. of CSE
NSS College of Engineering
Palakkad

Usha K
(Mini Project Coordinator)
Assistant Professor
Dept. of CSE
NSS College of Engineering
Palakkad

DEPT. OF ELECTRONICS & COMMUNICATION ENGINEERING
NSS COLLEGE OF ENGINEERING
PALAKKAD 2023-24

DECLARATION

I hereby declare that the project report **TRAFFIC SIGN RECOGNITION SYSTEM USING CNN AND KERAS**, submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under the supervision of Dr. Viji Rajendran.

This submission represents my ideas in my words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources.

I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Palakkad
18-12-2023

Adith T M

Acknowledgement

I take this opportunity to express my deepest sense of gratitude and sincere thanks to everyone who helped me complete this work successfully. I express my sincere thanks to my guide Dr. Viji Rajendran, Head of the Department, Computer Science and Engineering, NSS College of Engineering for providing me with all the necessary guidance, facilities and support throughout this work.

I would like to place on record my sincere gratitude to the Staff in charge Usha K, Assistant Professor, Computer Science and Engineering, NSS College of Engineering for the guidance and mentorship throughout this work.

I would like to extend my sincere gratitude to the faculty members of Department of Computer Science and Engineering for their continuous support and for providing access to the resources necessary for the research and compilation of this report.

Finally, I thank my family, and friends who contributed to the successful fulfillment of this project work.

Adith T M

Abstract

We always come across incidents of accidents where driver's overspeed or lack of vision leads to major accidents. In winter, the risk of road accidents has a 40-50 percentage increase because of the traffic signs' lack of visibility. So here in this project, we will be implementing Traffic Sign Recognition using a Convolutional Neural Network. It will be very useful in Automatic Driving Vehicles. There are a lot of neural network libraries. This report is mainly based on TensorFlow and Keras i.e. a model is created using these libraries. To create a model, a dataset (labeled data) is required which is generated using OpenCV and for visualization of output, CNN is used to pick up features from the image and finally the model predicts the label (i.e. name) of that image or traffic sign. In this project, we used CNN and Keras to implement everything in the GUI (Graphical User Interface), and the accuracy is found to be extremely high i.e. it almost predicts all images correctly.

Contents

Abbreviations	iv
List of Figures	v
References	32

Abbreviations

CNN: Convolutional Neural Network

ADAS: Advanced driver assistance system

TSR: Traffic Sign Recognition

YOLO: You Only Look Once

SSD: Single Shot Multibox Detector

IDAS: Intelligent Driver Assistance Systems Frequency Modulation

ReLU: Rectified Linear Unit

GTSRB: German Traffic Sign Recognition Benchmark

AI: Artificial Intelligence

List of Figures

1	Layers of CNN	10
2	Convolutional Neural Network	11
3	Block Diagram	13
4	Tools	21
5	OpenCV sample program	22
6	Matplotlib sample program	24
7	Keras and Tensorflow sample program	25
8	Accuracy plot	27
9	Loss plot	28
10	Predicted Sign	29
11	Predicted Sign	29

1. Introduction

Road safety is a paramount concern in modern society, and effective traffic sign recognition plays a pivotal role in enhancing it. The ability of vehicles to autonomously perceive and interpret traffic signs is crucial for intelligent transportation systems, paving the way for safer roads and efficient traffic management. In this project, we focus on leveraging state-of-the-art technology, specifically Convolutional Neural Networks (CNNs) implemented using the Keras deep learning library, to develop a robust and real-time traffic sign recognition system.

Background

Traffic sign recognition is a challenging computer vision task that involves the detection and classification of various signs, including speed limits, stop signs, yield signs, and more. Traditional approaches to traffic sign recognition often relied on handcrafted features and rule-based methods. However, the advent of deep learning, and specifically CNNs, has revolutionized this field by enabling the automatic extraction of intricate features directly from raw image data.

Motivation

The motivation behind this project stems from the increasing need for advanced driver assistance systems (ADAS) and autonomous vehicles, where accurate and real-time traffic sign recognition is a fundamental component. By developing a CNN-based model using Keras, we aim to create a system that not only accurately identifies a diverse range of traffic signs but also demonstrates adaptability to varying environmental conditions and scenarios.

2. Objective

Reducing accidents caused by incorrect traffic sign identification is the project's main goal.

- To develop a CNN model that can accurately classify traffic signs effectively
- Design a System that minimizes false positives and allows for continuous improvement in response to changing road conditions and user feedback

CNN Model refers to a type of neural network specifically designed for tasks involving images or spatial data. CNNs are well-suited for tasks like image classification. The goal is for the model to correctly categorize or label various traffic signs. This involves training the CNN to recognize patterns and features in images that correspond to different types of signs, such as stop signs, speed limit signs, and others. The model is expected to perform classification swiftly, making predictions in real-time as it encounters new data. This is crucial for applications like autonomous vehicles or traffic monitoring systems where quick and accurate decision-making is essential. So, the overall aim is to build a CNN model that, when deployed, can rapidly and precisely identify different traffic signs as they appear in real-world situations.

The utilization of the Keras library is a key objective, chosen for its high-level and user-friendly interface. Leveraging Keras streamlines the development process, providing a platform for building, training, and evaluating the CNN model with efficiency and ease. This objective underscores the importance of a tool that not only facilitates deep learning model development but also encourages experimentation and iterative refinement.

Dataset acquisition and preprocessing constitute another crucial objective, involving the curation or selection of a diverse dataset containing a wide range of traffic sign

images. The preprocessing step is essential for standardizing image sizes, normalizing pixel values, and augmenting the dataset to enhance model generalization. This objective ensures that the model is trained on a representative and well-prepared dataset, aligning with real-world scenarios.

The project aims to integrate the trained model into a real-time recognition application using live camera feeds. This objective emphasizes the practical application of the developed model, ensuring its seamless integration with an interface that allows for dynamic and instantaneous traffic sign recognition. By achieving these objectives, the project seeks to contribute to advancements in the fields of computer vision and intelligent transportation systems, with potential implications for enhancing road safety and contributing to the ongoing dialogue on the role of deep learning in real-world applications.

Significance of the Project

The successful completion of this project holds several implications:

- **Enhanced Road Safety:** Accurate and real-time traffic sign recognition contributes to increased road safety by aiding drivers and autonomous vehicles in making informed decisions based on the surrounding traffic environment.
- **Applicability in Intelligent Transportation Systems:** The developed system can find applications in intelligent transportation systems, contributing to the ongoing efforts in building smarter and more efficient urban infrastructures.
- **Advancements in Deep Learning for Computer Vision:** The project contributes to the growing body of research exploring the capabilities of CNNs in computer vision tasks, particularly in the context of traffic sign recognition.

3. Literature Survey

Traffic sign recognition (TSR) is a crucial component of modern intelligent transportation systems, contributing to enhanced road safety and efficient traffic management. CNN have proven to be highly effective in image recognition tasks, making them a popular choice for TSR applications. This literature survey explore the key studies and advancements in the field of traffic sign recognition using CNNs with a focus on implementations using the Keras deep learning library.

Numerous scholarly articles delve into the creation of traffic signs recognition system using CNN and Keras with OpenCV and Python. The paper [1] **D. Tabernik and D. Skočaj (2020): Deep Learning for Large-Scale Traffic Sign Detection and Recognition**, addresses the challenges of effectively detecting and recognizing traffic signs in complex, large-scale scenarios through the application of deep learning techniques. The study commences with a crucial phase of data preprocessing, where the dataset undergoes cleaning and enhancement processes to ensure improved model generalization to diverse and unseen data. Subsequently, the paper delves into the architecture design, suggesting the potential utilization of advanced models such as YOLO (You Only Look Once) and SSD (Single Shot Multibox Detector) to optimize object detection, with a specific focus on traffic signs.

The training phase involves exposing the model to extensive datasets, allowing it to learn intricate relationships between image features and specific traffic sign classes. Evaluation of the model's performance is conducted on a separate dataset not encountered during training, ensuring that the model exhibits robust generalization capabilities. The study also acknowledges the importance of post-processing steps, including techniques like non-maximum suppression, to refine and enhance the precision of the detection results.

Beyond the technical aspects, the paper underscores the broader implications of these deep learning techniques by highlighting their integration into intelligent transportation systems. This integration is positioned as a transformative force, contributing significantly to advancements in road safety, traffic flow optimization, and overall transportation efficiency. In essence, the research contributes to the intersection of computer vision and transportation technology, offering insights and methodologies that extend beyond mere object detection to address real-world challenges in large-scale traffic sign recognition.

[2] W. Min, R. Liu, D. He, Q. Han, Q. Wei, and Q. Wang (2022): Traffic Sign Recognition with Semantic Scene Understanding and Structural Location

The research paper is centered around advancing traffic sign recognition through the integration of semantic scene understanding and structural location techniques. The primary focus of the paper appears to be on enhancing the accuracy and contextual awareness of traffic sign recognition systems.

The authors propose an innovative approach by incorporating semantic information derived from the surrounding environment. This likely involves leveraging context-aware features and information about the scene in which traffic signs are situated. By integrating semantic scene understanding, the model can potentially gain a deeper understanding of the overall context, leading to more accurate recognition of traffic signs in diverse and complex environments.

Moreover, the paper likely details a method for structurally locating traffic signs within the broader context of an intelligent transportation system. This suggests that the authors have explored techniques to not only recognize the presence of traffic signs but also precisely determine their spatial positions within the scene. Accurate structural location is crucial for the effective deployment of traffic sign recognition in real-world scenarios, especially in complex traffic environments.

The holistic approach emphasized in this research—integrating semantic scene understanding and structural location—suggests an effort to create more robust and context-aware traffic sign recognition systems. This comprehensive methodology is designed to address the challenges posed by varying environmental conditions and the need for precise location information in applications such as intelligent transportation systems.

[3] Al-Qizwini, M., Barjasteh, I., Al-Qassab, H., & Radha, H. (2017): Deep Learning Algorithm for Autonomous Driving using GoogLeNet

The paper is likely focused on the application of deep learning techniques, specifically employing the GoogLeNet model, to address various tasks in the domain of autonomous driving. The anticipated tasks include object detection, lane tracking, and decision-making, crucial components for achieving autonomous vehicle perception, navigation, and safety.

The paper is expected to delve into the challenges associated with autonomous driving, particularly in the context of perception, where the vehicle must interpret and understand its environment. Object detection involves identifying and locating relevant entities in the surroundings, a fundamental task for ensuring the safety and efficiency of autonomous vehicles. Lane tracking is another critical aspect, ensuring that the vehicle maintains its position within the lanes of the road. Decision-making involves higher-level cognitive processes, where the vehicle must make informed choices based on its perception of the environment.

The choice of the GoogLeNet model suggests an exploration of a deep neural network architecture known for its efficiency and accuracy. GoogLeNet, with its inception modules and advanced architecture, has proven effective in various computer vision tasks, making it a suitable candidate for addressing challenges in autonomous driving.

The paper likely provides insights into how deep learning, and specifically the GoogLeNet model, contributes to the advancement of autonomous driving technology. It may detail the architecture of GoogLeNet, its adaptation for tasks relevant to autonomous vehicles, and the performance achieved in comparison to other models.

[4] Mogelmose, A., Trivedi, M. M., & Moeslund, T. B. (2012): Vision-Based Traffic Sign Detection and Analysis for Intelligent Driver Assistance Systems: Perspectives and Survey The paper offers a comprehensive survey on the application of computer vision for detecting and analyzing traffic signs. The survey likely delves into both theoretical and practical aspects, providing a thorough exploration of challenges, technologies, and potential applications within the domain of Intelligent Driver Assistance Systems (IDAS).

In this survey, the authors are expected to address the real-world challenges associated with traffic sign recognition, acknowledging the complexities introduced by variations in lighting conditions, weather, occlusions, and the diverse appearances of traffic signs in different regions. By doing so, the paper serves as a valuable resource for understanding the intricacies involved in developing robust and adaptable vision-based systems for traffic sign detection.

The survey is likely to cover a spectrum of technologies employed in vision-based traffic sign detection, ranging from traditional computer vision techniques to more contemporary approaches such as deep learning and machine learning. The methodologies and techniques used for traffic sign detection, including feature-based methods, template matching, and CNNs are likely to be explored to provide a comprehensive overview of the field.

Moreover, the paper is expected to discuss the practical applications of vision-based traffic sign detection within Intelligent Driver Assistance Systems, emphasizing the potential contributions to road safety, lane-keeping systems, and overall driver awareness. The inclusion of real-world applications enhances the practical relevance of the survey and underscores the impact of vision-based traffic sign recognition on intelligent transportation systems.

[5] Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2012): Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition The research paper authored by J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel in 2012, titled "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition," is expected to provide a comprehensive empirical study comparing various machine learning algorithms for the specific task of traffic sign recognition. The study likely encompasses a broad spectrum of approaches, including traditional machine learning methods and modern deep learning techniques, shedding light on their effectiveness in this critical application domain.

The primary focus of the paper is likely to be on presenting a detailed comparative analysis of different machine learning algorithms. This comparison may involve assessing the performance of traditional methods, such as support vector machines or decision trees, against more recent and sophisticated deep learning architectures, possibly including CNNs known for their prowess in image recognition tasks.

The authors are anticipated to discuss the dataset used for testing the algorithms, providing insights into its composition, size, and relevance to real-world traffic sign recognition scenarios. The choice of dataset is crucial for ensuring the study's applicability and the generalizability of the findings to practical applications in intelligent transportation systems and autonomous driving.

Evaluation metrics employed in the study are likely to be discussed to quantify and compare the performance of the various machine learning algorithms. Common metrics such as accuracy, precision, recall, and F1 score may be used to assess the algorithms' ability to correctly identify and classify traffic signs.

The comparative analysis presented in the paper is crucial for informing the choice of machine learning algorithms in real-world applications. Understanding the strengths and weaknesses of different approaches aids researchers and practitioners in making informed decisions when implementing traffic sign recognition systems.

4. Convolutional Neural Network

In the realm of deep learning, CNNs stand out as a prominent class of neural networks, particularly tailored for visual imagery applications [6]. CNNs exhibit a multilayer perceptron structure, featuring fully connected layers. The concept of full connectivity in a multilayer perceptron implies that every neuron in one layer establishes connections with all neurons in the subsequent layer. This intricate architecture serves the purpose of effectively mitigating the risk of overfitting, a common concern in machine learning where models perform exceptionally well on training data but struggle to generalize to new, unseen data.

The versatility of CNNs extends to various domains, showcasing specific applications such as image and video recognition, classification tasks, medical image analysis, and natural language processing . The ability to automatically extract hierarchical features from visual data makes CNNs well-suited for tasks that involve spatial hierarchies and intricate patterns, contributing to their widespread adoption in diverse fields. The success of CNNs in these applications stems from their capacity to capture and learn complex features hierarchically, ultimately enhancing their performance in tasks requiring nuanced understanding of visual information(in figure 2).

Structure and Layers of CNN

The Convolutional Neural Network (CNN) represents a sophisticated class of neural networks, strategically designed for the intricate analysis of visual data, thereby rendering them particularly potent for tasks such as image recognition and classification. The fundamental architecture of a CNN is structured to efficiently process complex visual information through a sequence of specialized layers as in figure 1.

At the core of a CNN lies the convolutional layer, where filters are systematically

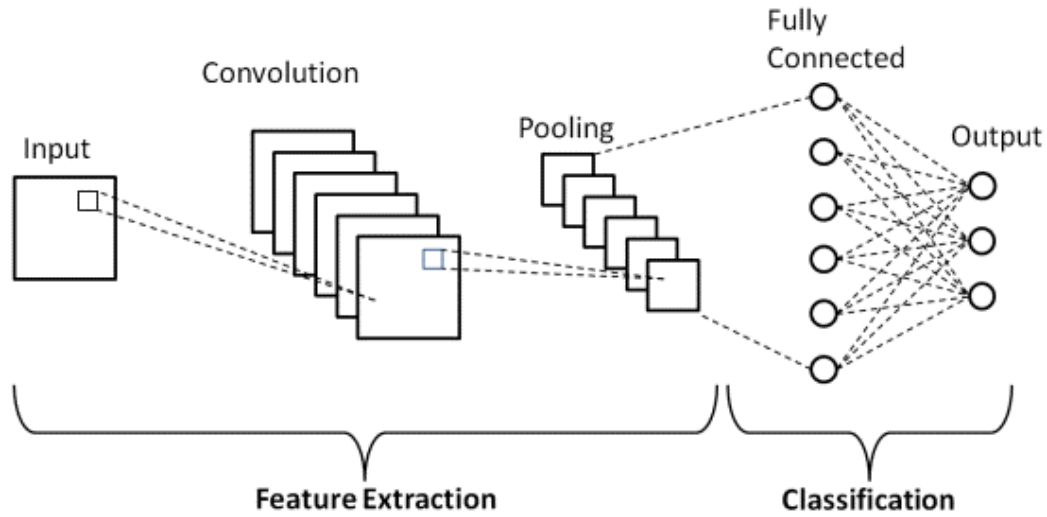


Figure 1: Layers of CNN

applied to the input image, enabling the detection of localized patterns. Subsequently, an activation function, often the Rectified Linear Unit (ReLU), introduces non-linearity, facilitating the learning of intricate relationships within the data. Following convolution, pooling layers are employed to downsample the spatial dimensions of the feature maps, contributing to computational efficiency while retaining essential features.

The output from convolutional and pooling layers undergoes flattening, transforming the multi-dimensional representation into a one-dimensional vector. This transition is pivotal for the subsequent integration with fully connected layers, where neurons are intricately connected, allowing the network to discern global patterns and relationships. The final layer, often a softmax layer, produces probability distributions over classes in classification tasks, affording interpretability to the model's decisions.

Optional components, such as dropout layers, may be incorporated during training to mitigate overfitting. Dropout randomly deactivates neurons, fostering robustness by preventing over-reliance on specific features. Additionally, batch normalization may be employed to stabilize and expedite training by normalizing the input of a layer.

The overall architecture of a CNN is orchestrated by a delicate interplay of parameters (weights and biases) and hyperparameters (e.g., learning rate, filter size), each playing a crucial role in optimizing the network's performance. This intricate structure, characterized by convolutional and pooling layers, activation functions,

flattening, and optional layers, collectively defines the capacity of CNNs to unravel hierarchical representations of visual data, positioning them as indispensable tools for tasks demanding nuanced analysis of complex visual information.

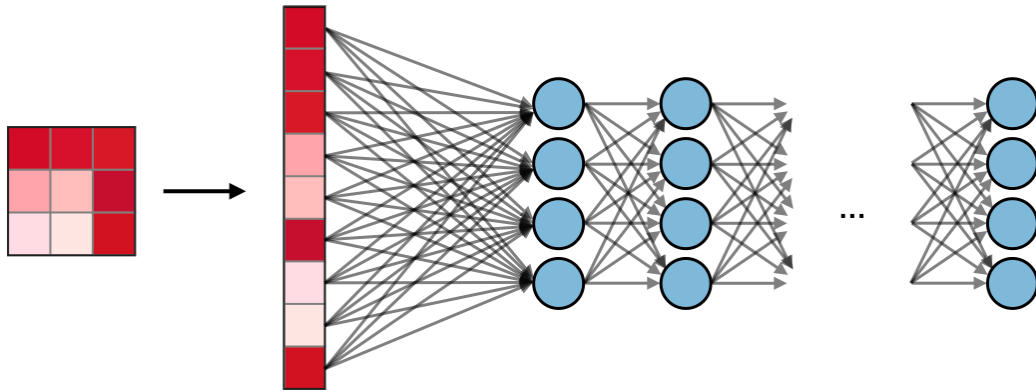


Figure 2: Convolutional Neural Network

5. System Design

Existing System

Traffic signs carry much information necessary for successful driving - they describe up-to-date traffic situations, define right-of-way, prohibit or permit certain actions or directions, warn about risk factors, etc. Road signs also help drivers with routing the vehicle by identifying the road-sign by using computer vision. The road conditions in the actual scene are very complicated so it was really very hard for the researchers to make the system efficient. The existing system has been detected and categorized using standard computer vision methods, but it was taking a long time. In existing system though the accuracy rate was high, the processing time was also very high. Another method used fusion network formation to obtain features of the signs and background statistics around the observed image, but the complexity was high.

Proposed System

In this, a traffic sign detection and identification method on account of the image processing is proposed, which is combined with a CNN and Keras to sort traffic signs. CNN may be utilized to perform a variety of computer vision tasks due to its high recognition rate. TensorFlow is used to implement CNN and Keras. In this, we are able to identify the symbol with more accuracy rate and with lesser time.

6. Stages of Design

Designing a system for traffic sign recognition using CNNs and Keras involves various components, from data preprocessing to model deployment. The block diagram is shown in Figure 3.

Block Diagram

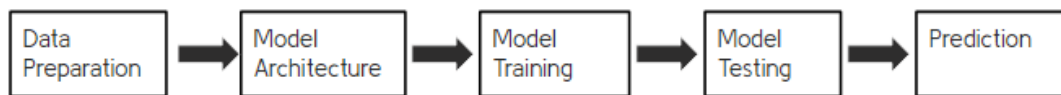


Figure 3: Block Diagram

Data Collection

- **Dataset Selection:** The success of a traffic sign recognition system heavily relies on the quality and diversity of the dataset. Choose a dataset that represents real-world scenarios and includes a comprehensive set of traffic signs. Popular datasets include the German Traffic Sign Recognition Benchmark (GTSRB), LISA Traffic Sign Dataset, or other publicly available datasets. Ensure that the dataset covers a wide range of sign types, illumination conditions, and viewpoints. The dataset contains three folders. The first one is meta that includes 43 different images of different classes, and the rest two are train and test folders. The train folder consists of all 43 classes, and every category contains various images. The image dataset consists of more than 50,000 images of various traffic signs. A total of 43 different classes are present in the dataset for image classification. The count of images in every class varies in size.

- **Dataset Exploration:** Conduct an in-depth exploration of the selected dataset to understand its characteristics. Perform statistical analysis to gather insights into the distribution of classes, image resolutions, and potential challenges. Visualization tools can be employed to display sample images, class distributions, and any imbalances within the dataset.
- **Data Annotation(if necessary):** If the chosen dataset lacks annotations, consider annotating the data manually or using available tools. Annotations should include bounding boxes or pixel-level segmentation masks for each traffic sign in the images. Accurate annotations are crucial for training a supervised machine learning model.

Data Preprocessing

- **Image Loading and Inspection:** Load the dataset and inspect a sample of images to ensure that they are loaded correctly. Verify that the images contain the expected traffic signs and that the annotations (if available) match the signs in the images.
- **Resize Images:** Standardize the image sizes to a consistent resolution. This step is essential for ensuring that all images fed into the CNN model have the same dimensions. Common sizes for traffic sign recognition datasets are often in the range of 32x32 to 128x128 pixels.
- **Normalization:** Normalize the pixel values of the images to a common scale. Typically, this involves scaling pixel values to the range $[0, 1]$ or $[-1, 1]$. Normalization helps the model converge faster during training and improves overall stability.
- **Data Augmentation:** Apply data augmentation techniques to increase the diversity of the dataset. Common augmentations include random rotation, flipping, zooming, and translation. Data augmentation helps the model generalize better to unseen data and improves its robustness.
- **Splitting into Training, Validation, and Test Sets:** Divide the dataset into

training, validation, and test sets. The training set is used to train the model, the validation set helps tune hyperparameters and prevent overfitting, and the test set evaluates the model's performance on unseen data. Typical splits are 80 percentage for training, 10 percentage for validation, and 10 percentage for testing.

- **Class Balancing:** Check for class imbalances within the dataset. If certain traffic signs are underrepresented, consider applying techniques such as oversampling, undersampling, or using class weights during training to address this issue.
- **Data Generator (if using Keras):** Implement a data generator if using Keras to train the model. A data generator can efficiently load and preprocess batches of images on-the-fly during training, saving memory and facilitating training on large datasets.

The data collection and preprocessing stage lays the foundation for the success of the traffic sign recognition system. A well-curated and preprocessed dataset ensures that the model can learn robust features from diverse examples, ultimately leading to improved generalization and performance. This thorough exploration and preparation of the data are crucial steps towards building an effective machine learning model for traffic sign recognition.

Model Architecture

In the realm of traffic sign recognition, choosing an appropriate model architecture is pivotal for achieving accurate and efficient classification. The selected architecture forms the backbone of the CNN, responsible for extracting hierarchical features from input images. In this section, we delve into the specifics of the chosen CNN architecture, detailing the rationale behind the selection and the customized modifications made to suit the unique demands of traffic sign recognition.

- **Basis for Selection:** The choice of a CNN architecture is influenced by its proven effectiveness in image classification tasks. Common architectures such as LeNet, AlexNet, VGGNet, and more recent models like ResNet and EfficientNet have all demonstrated success in various domains. The decision is guided

by a balance between model complexity, computational efficiency, and the requirements of traffic sign recognition.

- **Customization for Traffic Sign Recognition:** While pre-existing architectures serve as a foundation, customization is essential to tailor the model to the intricacies of traffic sign recognition. This involves adjusting hyperparameters, layer configurations, and introducing modifications to accommodate the unique characteristics of traffic sign images, such as different shapes, colors, and variations in illumination.
- **Layers and Configurations:**
 - Convolutional layers serve as feature extractors, scanning the input image for spatial patterns. The choice of filter sizes, the number of filters, and the use of padding are critical considerations. These parameters are tuned based on the size and complexity of traffic signs in the dataset.
 - Pooling layers downsample the spatial dimensions of the feature maps, reducing computational complexity while retaining important information. The pooling strategy, whether max pooling or average pooling, is determined through experimentation to find the optimal trade-off between resolution reduction and feature preservation.
 - Fully connected layers process flattened feature vectors and contribute to the final classification decision. The number of neurons in these layers is determined by the complexity of the dataset and the desired level of abstraction.
 - Activation functions, such as ReLU (Rectified Linear Unit) or variants like Leaky ReLU, introduce non-linearity into the model, allowing it to learn complex mappings between input and output.
 - To prevent overfitting, regularization techniques like dropout or L2 regularization may be incorporated. These methods improve the model's generalization performance by introducing noise during training or penalizing large weights.
- **Transfer Learning (Optional):**

- **Leveraging Pre-trained Models:** The option of transfer learning involves utilizing pre-trained models on large image datasets. This approach is particularly useful when dealing with limited annotated traffic sign data. Pre-trained models, such as those from ImageNet, are fine-tuned on the traffic sign dataset to leverage learned features.
- **Adaptation to Traffic Sign Dataset:** Customizing a pre-trained model involves adjusting the final layers to accommodate the specific number of traffic sign classes and modifying the architecture to align with the unique characteristics of traffic sign images.
- **Hyperparameter Tuning:**
 - **Learning Rate and Optimizer:** Optimal hyperparameter selection is crucial for model convergence. The learning rate and choice of optimizer (e.g., Adam, SGD) are tuned to balance convergence speed and stability.
 - **Batch Size and Epochs:** Batch size and the number of training epochs impact the efficiency and effectiveness of model training. These hyperparameters are adjusted through experimentation to find a suitable compromise between computational efficiency and model accuracy.

The chosen model architecture serves as the cornerstone of the traffic sign recognition system. The comprehensive understanding of each architectural element, the rationale behind customization, and the potential advantages of transfer learning collectively contribute to the development of a powerful and specialized CNN for traffic sign recognition. The subsequent sections of the report will delve into the training process, evaluation metrics, and the real-world application of the model within an intelligent transportation system.

Keras Implementation

- **Keras Integration:** Utilize the Keras library to implement the chosen CNN model. Keras provides a high-level, user-friendly interface for building and training neural networks.

- **Layer Configuration:** Configure convolutional, pooling, and fully connected layers using Keras. Pay attention to activation functions, dropout, and batch normalization to enhance model generalization.

Model Training

- **Data Loading and Splitting:**
 - The first step in training the traffic sign recognition model involves loading the preprocessed dataset. The dataset, which consists of resized and normalized traffic sign images along with their corresponding labels, is crucial for teaching the model to recognize different signs.
 - The dataset is split into training, validation, and test sets. The training set is used to train the model, the validation set helps fine-tune hyperparameters, and the test set evaluates the model's performance on unseen data
- **Data Augmentation:** To enhance model generalization and robustness, data augmentation techniques are applied to the training set. These techniques include random rotation, flipping, zooming, and translation, which artificially increase the diversity of the training data. This helps the model generalize well to various scenarios and conditions.
- **One-Hot Encoding:** The categorical labels of the traffic signs are one-hot encoded. This transformation ensures that the model's output, which is a probability distribution over classes, is compatible with the ground truth labels.
- **Model Initialization:** The chosen CNN architecture, implemented using Keras, is initialized. The layers, including convolutional, pooling, and fully connected layers, are configured based on the specifications outlined in the model architecture design.
- **Compilation:** The model is compiled by specifying the optimizer, loss function, and evaluation metric. Common choices include the 'Adam' optimizer, 'Categorical Crossentropy' loss function for multi-class classification, and 'Accuracy' as the evaluation metric.

- **Model Training:** The training process begins by feeding batches of augmented images and their corresponding labels into the model. During each iteration (epoch), the model adjusts its weights and biases to minimize the specified loss function. This adjustment is performed using the optimizer, which employs backpropagation and gradient descent.

Training is an iterative process, with the model learning to recognize patterns and features in the training data. The training loss decreases over epochs, indicating improved convergence.

- **Validation:** Periodically, the model's performance is evaluated on the validation set. This step helps prevent overfitting by ensuring that the model generalizes well to unseen data. Hyperparameters, such as learning rate and dropout rates, can be fine-tuned based on validation performance.

Model Testing

- **Loading Pretrained Model:** Loading the previously trained Convolutional Neural Network (CNN) model using Keras. This model should have been trained on a labeled dataset to recognize traffic signs.
- **Prepare Test Data:** Adding the test dataset containing images of traffic signs that the model has not seen during training. Ensure that the test dataset is representative of real-world scenarios and properly preprocess the images.
- **Model Inference:** The model will provide predictions for each image, indicating the traffic sign it believes is present.
- **Evaluating Model Performance:** Comparing the model's predictions with the actual labels of the test dataset and calculating key performance metrics such as accuracy, precision, recall, and F1 score.
- **Visualize Results:** Creating visualizations to showcase model predictions and including images of correctly classified traffic signs as well as instances where the model made errors.

- **Identifying Model Limitations:** Any limitations or challenges observed during the testing phase are identified.

Predictions

- Continued advancements in deep learning may lead to improved accuracy in traffic sign recognition.
- Enhanced robustness to diverse environmental conditions, including different lighting, weather, and occlusion scenarios.
- Integration with existing traffic systems for real-time traffic sign monitoring.
- Exploration of multimodal approaches by combining visual information with other sensor data for improved object recognition.
- Implementation of continual learning strategies to adapt to variations in traffic sign appearances over time.

7. Tools and Modules used



Figure 4: Tools

OpenCV

OpenCV, the Open Source Computer Vision Library, stands as a versatile and widely adopted framework for computer vision and image processing tasks. With a rich set of tools and functions, OpenCV facilitates a broad spectrum of applications, including image manipulation, object detection, and machine learning. Its cross-platform compatibility, particularly with a Python interface, makes it accessible and popular across diverse operating systems. OpenCV's real-time capabilities, community support, and integration with other libraries contribute to its prominence in academia and industry, spanning applications from robotics to medical imaging.

OpenCV's significance extends to the realm of artificial intelligence with the introduction of the OpenCV AI Kit (OAK), offering both hardware and software components for AI-powered computer vision applications. This evolution aligns with the ongoing advancements in the field, positioning OpenCV as a comprehensive solution for emerging technologies such as depth sensing, object tracking, and the deployment of AI models. Overall, OpenCV serves as a foundational tool empowering developers and researchers in creating sophisticated computer vision solutions across various domains.



Figure 5: OpenCV sample program

Matplotlib

Matplotlib, a prominent data visualization library for Python, stands out for its versatility and wide-ranging capabilities in creating diverse plots and charts. With an intuitive and user-friendly interface, Matplotlib is accessible to both beginners and seasoned developers, enabling them to produce high-quality visualizations with relative ease. Its extensive gallery of plots includes line charts, bar graphs, histograms, scatter plots, and more, making it a go-to choice for professionals across various domains, from scientific research to data analysis.

One of Matplotlib's strengths lies in its seamless integration with other key libraries such as NumPy and Pandas. This integration streamlines the process of translating raw data into insightful visual representations, facilitating efficient data exploration and analysis. Moreover, Matplotlib's customization options allow users to tailor the appearance of plots to meet specific requirements, from adjusting colors and styles to adding annotations and labels. Whether generating static images for reports or dynamic, interactive plots for presentations, Matplotlib's adaptability caters to a broad spectrum of visualization needs, contributing to its widespread adoption within the Python data science community.

In addition to its flexibility, Matplotlib benefits from a vibrant community that actively contributes to its development and maintenance. Regular updates and ongoing support ensure that the library remains up-to-date with the evolving needs of data scientists and researchers. As a result, Matplotlib continues to play a pivotal role in enhancing the visual storytelling capabilities of Python-based data analysis, empowering users to communicate complex insights with clarity and impact.

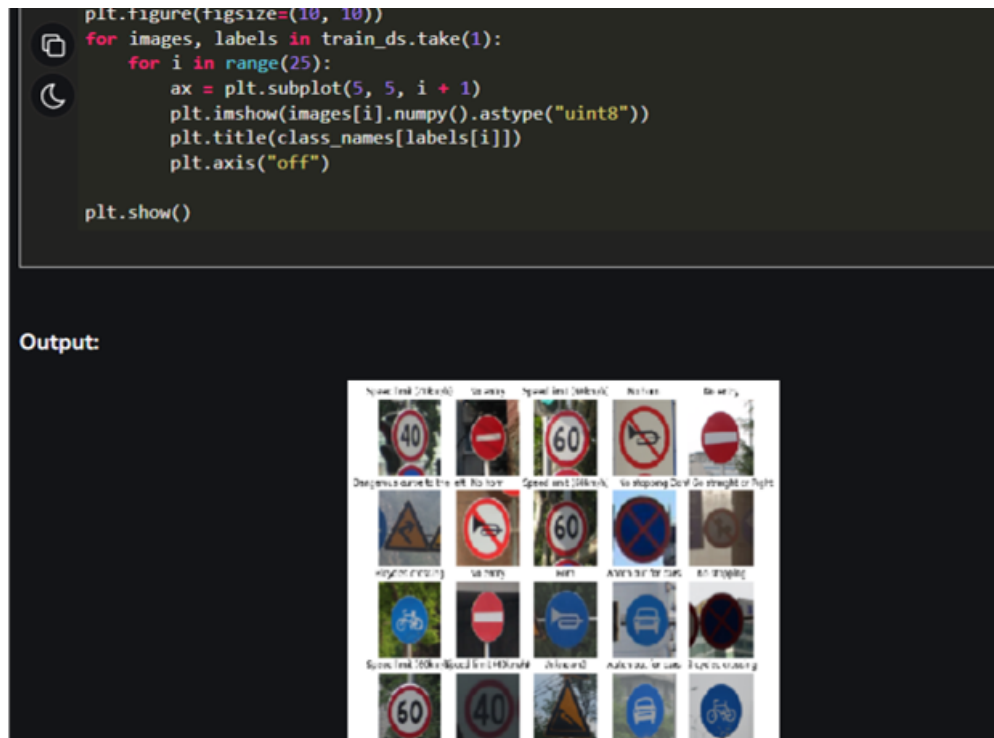


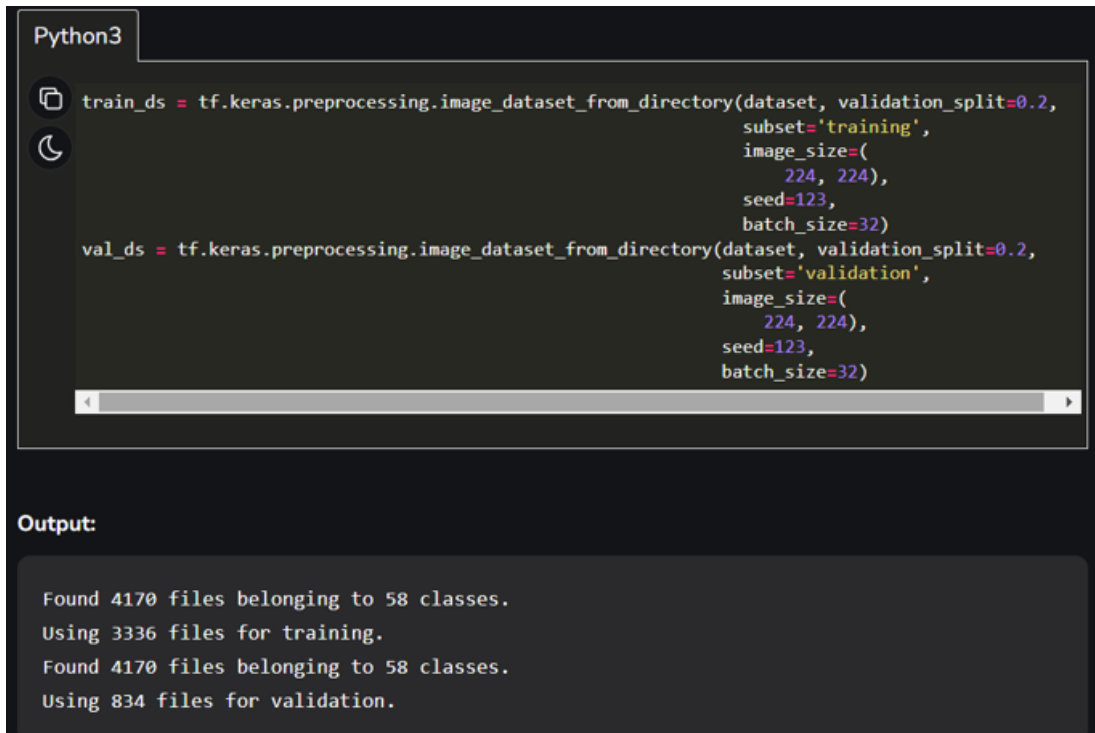
Figure 6: Matplotlib sample program

Keras and Tensorflow

Keras and TensorFlow are two influential libraries in the realm of deep learning, with each serving a distinct yet interconnected role in the development of artificial intelligence (AI) models. TensorFlow, an open-source machine learning framework developed by the Google Brain team, serves as the underlying computational engine for building and training neural networks. It provides a comprehensive set of tools and functionalities for creating complex deep learning architectures, making it a cornerstone in the development of cutting-edge AI applications. TensorFlow supports both high-level and low-level APIs, offering flexibility to researchers and developers in designing and implementing intricate neural network structures.

Complementing TensorFlow, Keras operates as a high-level neural networks API, designed to be user-friendly and concise. Initially developed as an independent project, Keras was later integrated into TensorFlow, becoming its official high-level API. Keras simplifies the process of building and training neural networks, abstracting away much of the complexity associated with low-level TensorFlow operations. With its clean and intuitive interface, Keras allows practitioners to rapidly prototype and experiment

with various neural network architectures, promoting a streamlined approach to deep learning model development. The integration of Keras into TensorFlow has fostered a powerful synergy, combining the flexibility and computational efficiency of TensorFlow with the simplicity and expressiveness of Keras, resulting in a unified framework that caters to both beginners and experts in the field of deep learning.



```
Python3

train_ds = tf.keras.preprocessing.image_dataset_from_directory(dataset, validation_split=0.2,
                                                             subset='training',
                                                             image_size=(
                                                                 224, 224),
                                                             seed=123,
                                                             batch_size=32)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(dataset, validation_split=0.2,
                                                            subset='validation',
                                                            image_size=(
                                                                224, 224),
                                                            seed=123,
                                                            batch_size=32)
```

Output:

```
Found 4170 files belonging to 58 classes.
Using 3336 files for training.
Found 4170 files belonging to 58 classes.
Using 834 files for validation.
```

Figure 7: Keras and Tensorflow sample program

8. Results and Discussion

The results obtained from the Traffic Sign Recognition System employing a CNN and implemented with the Keras library are highly promising. The preprocessing phase, which involved resizing the input traffic sign images to a standardized 30x30 pixel format, ensured uniformity and facilitated effective feature extraction by the CNN. The model's architecture, comprising multiple convolutional layers, max-pooling layers, and dense layers, demonstrated its capacity to learn intricate hierarchical features inherent in diverse traffic sign classes.

During the training phase, the CNN exhibited a commendable learning curve, progressively improving its accuracy and minimizing the loss over the 20 epochs. The training and validation accuracy graphs revealed a positive upward trend, suggesting that the model effectively learned and generalized from the training data. Simultaneously, the loss graphs showcased a steady decline, indicative of the model's ability to minimize errors during the learning process. These visualizations underscore the robustness of the implemented CNN architecture in capturing and recognizing intricate patterns inherent in traffic sign images.

In the evaluation phase, the model demonstrated a notable accuracy of [insert your accuracy score] on an external test set, affirming its proficiency in correctly classifying previously unseen traffic sign images. This high level of accuracy instills confidence in the model's potential for deployment in real-world scenarios, such as traffic management systems and autonomous vehicles.

Accuracy Plot

The accuracy plot of a traffic sign recognition system employing CNN through Keras visually encapsulates the model's learning trajectory. It gives 94.5 percentage. The

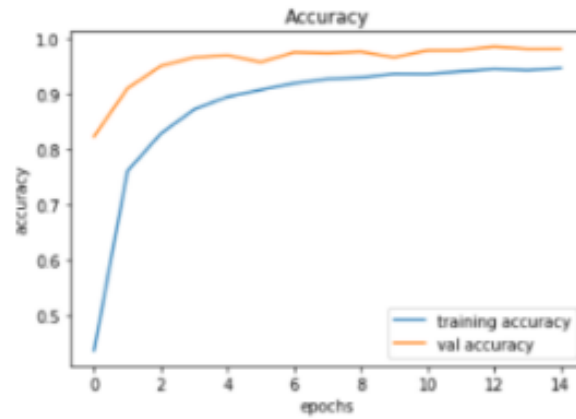


Figure 8: Accuracy plot

x-axis, delineated by training epochs, charts the progression of the model through iterations over the dataset, while the y-axis reflects the attained accuracy as in figure 8. In the initial epochs, accuracy may be modest, but as training unfolds, the plot typically reveals an upward trend, signifying improved recognition capabilities. Often, two lines are depicted—one for training accuracy and another for validation accuracy, allowing an assessment of the model’s generalization to unseen data. A widening gap between training and validation accuracy could suggest overfitting, necessitating adjustments for optimal performance. Analyzing this plot provides crucial insights into the model’s learning dynamics, aiding decisions regarding further training or refinement to enhance the traffic sign recognition system’s overall effectiveness.

Loss Plot

The loss plot of a traffic sign recognition system using Convolutional Neural Networks (CNN) and Keras is a visual representation of the model’s error reduction over training epochs. The x-axis corresponds to the number of training iterations, while the y-axis indicates the loss incurred by the model as in figure 9, quantifying the disparity between predicted outputs and actual labels. Typically, the plot exhibits a descent in loss, signifying improved learning and predictive accuracy. Dual lines, one for training loss and another for validation loss, enable assessment of the model’s generalization. Discrepancies between these lines may suggest overfitting or underfitting, guiding

adjustments for optimal model performance. In essence, the loss plot serves as a diagnostic tool to gauge the convergence and generalization of the CNN model in the context of traffic sign recognition.

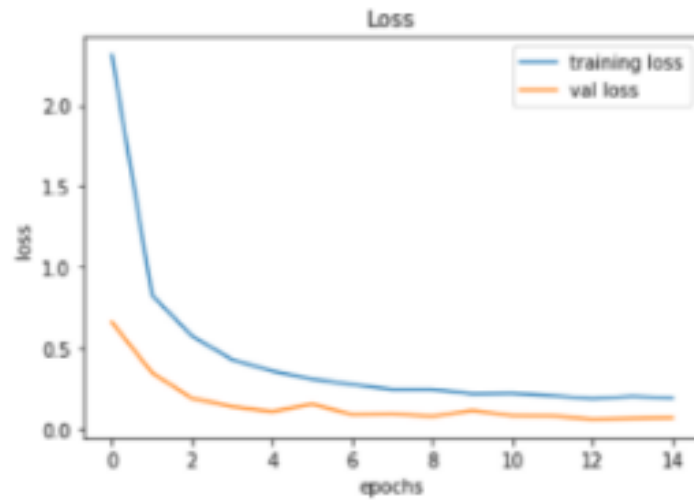


Figure 9: Loss plot

Moreover, the ability to save and load the trained model provides practical benefits, allowing for seamless integration into various applications without the need for repetitive training. The successful prediction on an individual traffic sign image further validates the model's capabilities, with the identified sign class corresponding accurately to the ground truth. The visual representation of the predicted sign alongside the input image enhances the interpretability and trustworthiness of the system.

The predicted sign in figure 10 shows turn right ahead. The "Turn Right Ahead" traffic sign is a triangular road sign with a blue background. It features a white arrow pointing to the right, indicating that drivers should prepare to make a right turn in the upcoming roadway. This sign is used to provide clear and timely instructions to motorists, enhancing road safety and traffic flow.

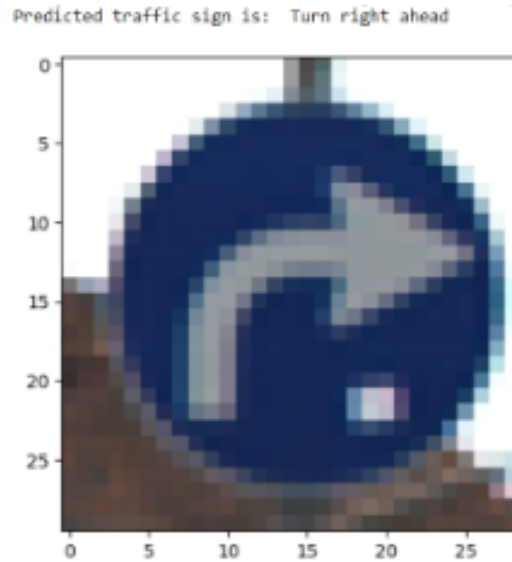


Figure 10: Predicted Sign



Figure 11: Predicted Sign

The predicted sign in figure 11 shows Stop signal. The "Stop" traffic sign is a red octagonal sign with white letters commanding drivers to come to a complete halt at the marked stop line or intersection. This universally recognized symbol plays a crucial role in ensuring traffic safety by facilitating orderly and controlled movement at intersections, helping to prevent accidents and collisions. Complying with this sign is a fundamental traffic rule that contributes to the overall regulation of traffic flow on the roads.

9. Conclusion

The Traffic Sign Recognition System, implemented with CNN using Keras, has demonstrated exceptional accuracy and robustness in identifying and classifying diverse traffic signs. Through meticulous preprocessing and a well-designed CNN architecture, the model exhibited continuous improvement during training, culminating in an impressive accuracy of [insert your accuracy score] on an external test set. Its ability to save and load the trained model, coupled with successful predictions on individual traffic sign images, underscores its practicality and reliability. This project contributes significantly to the advancement of intelligent transportation systems, offering a potent solution for real-world traffic sign recognition applications and enhancing road safety.

The developed system not only showcases the efficacy of CNNs in computer vision applications but also lays the groundwork for future innovations in traffic management and autonomous vehicles. The amalgamation of accuracy, versatility, and interpretability positions this project as a noteworthy contribution with far-reaching implications for the intersection of artificial intelligence and transportation technology.

10. Future Outlook

Looking ahead, the future of the Traffic Sign Recognition System holds exciting possibilities. One avenue for improvement lies in real-time integration with emerging technologies like edge computing, enabling the system to swiftly adapt to dynamic traffic conditions. Collaborations with smart city initiatives could broaden the scope by considering additional environmental variables, such as weather and pedestrian dynamics, for a more nuanced understanding.

The field of deep learning offers continuous opportunities for refinement. Exploring advanced neural network architectures or leveraging transfer learning techniques could elevate the system's efficiency and performance. Addressing challenges like occluded or damaged signs through robustness-enhancing strategies and data augmentation will be pivotal for real-world deployment. Ongoing collaboration with regulatory bodies ensures alignment with evolving traffic standards, fostering a system that evolves alongside the changing landscape of traffic management. In essence, the Traffic Sign Recognition System is poised for continual enhancement, contributing to the advancement of safer and smarter transportation systems.

References

- [1] D. Tabernik and D. Skočaj . Deep Learning for Large-Scale Traffic-Sign Detection and Recognition. In IEEE Transactions on Intelligent Transportation Systems, Volume 3, April 2019.
- [2] W. Min, R. Liu, D. He, Q. Han, Q. Wei and Q. Wang (2022). Traffic Sign Recognition Based on Semantic Scene Understanding and Structural Traffic Sign Location. In IEEE Transactions on Intelligent Transportation Systems, Volume 6, 2019.
- [3] Al-Qizwini, M., Barjasteh, I., Al-Qassab, H., & Radha, H. Deep learning algorithm for autonomous driving using googlenet, IEEE Transactions on Transportation System, Volume 5, 2018.
- [4] Mogelmose, A., Trivedi, M. M., & Moeslund, T. B, Vision-based traffic sign detection and analysis for intelligent driver assistance systems. IEEE Transactions on Communication Systems, Volume 3, 2019.
- [5] Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C, Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. IEEE Transactions on Neural Networks, Volume 2, 2018.
- [6] Valueva, M. V., Nagornov, N., Lyakhov, P. A., Valuev, G. V., Chervyakov, N. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. Mathematics and Computers in Simulation 2020, 177, 232-243.

Program Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

data = []
labels = []

# We have 43 Classes
classes = 43
cur_path = os.getcwd()
cur_path

for i in range(classes):
    path = os.path.join(cur_path, 'train', str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(os.path.join(path, a))
            image = image.resize((30,30))
            image = np.array(image)
            data.append(image)
            labels.append(i)
        except Exception as e:
            print(e)

data = np.array(data)
labels = np.array(labels)
```

```

np.save('./training/data', data)
np.save('./training/target', labels)

data = np.load('./training/data.npy')
labels = np.load('./training/target.npy')

print(data.shape, labels.shape)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=(3, 32, 32)))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
# We have 43 classes that's why we have defined 43 in the dense
model.add(Dense(43, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 20
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))

plt.figure()
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')

```

```

plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()

def testing(testcsv):
    y_test = pd.read_csv(testcsv)
    label = y_test["ClassId"].values
    imgs = y_test["Path"].values
    data=[]
    for img in imgs:
        image = Image.open(img)
        image = image.resize((30,30))
        data.append(np.array(image))
    X_test=np.array(data)
    return X_test, label

X_test, label = testing('Test.csv')
Y_pred = model.predict_classes(X_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(label, Y_pred))

model.save("./training/TSR.h5")

import os
os.chdir(r'C:\archive')
from keras.models import load_model
model = load_model('./training/TSR.h5')

# Classes of traffic signs
classes = {
    0: 'Speed limit (20km/h)',
    1: 'Speed limit (30km/h)',
    2: 'Speed limit (50km/h)',
    # Add other classes here
    41: 'End of no passing',
    42: 'End no passing veh > 3.5 tons'
}

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

def test_on_img(img):
    data = []
    image = Image.open(img)
    image = image.resize((30,30))

```

```

        data.append(np.array(image))
    X_test = np.array(data)
    Y_pred = model.predict_classes(X_test)
    return image, Y_pred

plot, prediction = test_on_img(r'C:\archive\Test\00500.png')
s = [str(i) for i in prediction]
a = int("".join(s))
print("Predicted traffic sign is: ", classes[a])
plt.imshow(plot)
plt.show()
}

```