TRAFFIC SIGN RECOGNITION SYSTEM USING CNN BASED APPROACH

L. Sujith Kumar

CSN(Networks))

Kakatiya Institute of Technology and Science

sujith.csn@kitsw.ac.in

Syed Awais

CSN(Networks)

Kakatiya Institute of Technology and Science

awaissyed1212@gmail.com

Abstract

In the modern era of intelligent transportati on systems, the accurate and timely recogn ition of traffic signs has emerged as a critical requirement for the development of autonomous vehicles and advanced driver-assistance systems (ADAS). Traffic signs serve as visual indicators that guide drivers, enforce traffic laws, and ensure road safety. Human drivers, however, are prone to errors due to fatigue, distractions, or environmental conditions such as poor lighting and adverse weather. Therefore, a reliable automated traffic sign recognition system can play a vital role in enhancing both the efficiency and safety of road travel. This project presents a Convolutional Neural Network (CNN)-based traffic sign recognition system designed to classify traffic signs from still images with high accuracy. The system was trained on the publicly available Kaggle Traffic Sign Dataset, consisting of over 6,300 labeled images representing 58 distinct traffic sign categories. By leveraging the strengths of CNNs in visual pattern recognition, the model automatically learns hierarchical features from grayscale image inputs, effectively eliminating the need for manual feature engineering or classical image processing techniques. The images were preprocessed through resizing, grayscale conversion, normalization, and label encoding to optimize the training process. The model was developed and trained in Google Colab using TensorFlow/Keras, benefiting from GPU acceleration and collaborative tools. After achieving satisfa ctory validation performance, the trained model was exported and deployed into a desktop application using Python's Tkinter library. This graphical user interface allows users to upload traffic sign images and receive real-time predictions of the sign category, making the system intuitive and easy to use. The entire system runs efficien tly on standard hardware without the need for external sensors or complex setups. Th e project successfully demonstrates the po tential of deep learning techniques in real-world traffic sign classification application s.

Keywords— *Traffic Sign Recognition (TSR), Convolutional Neural Networks (CNNs), Deep Learning, Image Classification, Driv er Assistance Systems, Intelligent Transpor tation Systems,German Traf fic Sign Recog nition Benchmark(GTSRB), Vehicle-to-Everything (V2X), Real-Time Recognition.*

I. INTRODUCTION

1.1 Background

The evolution of intelligent transport systems stressed the critical importance of precise and efficient recognition of traffic sign. Traffic signs serve as vital communi cative tools, transmitting essential inform ation such as speed limits, warnings and navigation guidelines to drivers. The ability

to automatically detect and interpret these signs is critical to the advancement of autonomous vehicles and driver assistance systems. Traditional traffic signal recognit ion methods were usually based on manual extraction extraction and classic machine learning algorithms, which were susceptibl e to lighting variations, weather conditions and occlusions. The advent of deep learnin g, particularly the convolutionary neural networks (CNNs), has revolutionized this domain, allowing models to learn hierarch ical representations of resources directly from gross image data, thus increasing the accuracy and robustness of recognition. In recent years, the motor vehicle industry has focused its focus towards creating self-driving cars that can mimic human perception and decision-making abilities. One of the major pillars supporting this infection is the ability to see and interpret its surroundings of the vehicle, including road signals, signs and signals. Traffic sign recognition is not just a technical feature; This is an important security component that helps vehicles maintain legal complia nce, responding properly to road rules, and alerted human drivers or autonomous systems in real time. Since vehicles begin to operate more independently, a system that can basically identify and interpret traffic signals, is no longer alternative, it is necessary. The need for scalable, flexible and intelligent recognition systems has deployed intensive learning and especially CNNs as the most promising approach to achieve a reliable and future ready solution. The boundaries of traditional approaches paved the way for the integration of deep learning in visual recognition tasks. In recent years, the Convolutional Neural Network (CNN) has emerged as the most effective and reliable tool for image classification problems. The CNN through its layered architecture accelerated the features of direct learning from raw pixel data, which captures low-level from high-level abstractions such as edges, shapes, textures and patterns. Unlike traditional methods that require manual tuning of features, CNNs are able to commit self-reliance during training, allowing them to make better normal to unseen data.

1.2 Problem Statement

Additionally, the traffic sign recognition system should be able to normalize in various geographical and camera setups. A model trained on a specific dataset, such as the German traffic sign recognition benchmark, cannot perform optimally when exposed to data from other countries with various indications and road conditions. This highlights the need for models that are trained not only on diverse datasets, but also strictly tested in various situations to ensure widely projectiveness and dependence in real -world landscapes.

1.3 Objectives of the Project

The main objectives of this project are:

- To design and implement a firm nerve network model capable of accurately classifying various traffic signals.
- To train the model using a broad dataset, to ensure strength against the challenges of the real world.
- To evaluate the performance of the model in terms of accuracy, accurate, recall and computational efficiency.
- To integrate the model into a system that can process real -time video feed for practical deployment in autonomous vehicles.

1.4 Scope of the project

The scope of this project incorporates the development of an intensive learning-based traffic signal system using CNN. The system is designed to classify traffic signals from real-time captured images, which is suitable for integration in advanced driver-help system (ADAS) and autonomous vehicles. Its purpose is to increase road safety by accurately understanding and reacting to vehicles, which reduces human error and improves traffic compliance. Further, the project lays the foundation for more advanced, multi-modal perception

systems that go beyond static signal recognition. By integrating the CNN-based recognition system with other components such as GPS, Lidar, and Cementic Map Data, vehicles may be able to make more reference-inconceivable decisions from future repetitions. Emerging technologies such as Vehicle-to-Everything (V2X) communication, Internet of Things (IoT), Artificial Intelligence (AI), and 5G connectivity are ready to revolutionize the way the traffic system. V2X allows vehicles to communicate with infrastructure, pedestrians and other vehicles in real time, increase traffic flows and reduce collisions.

1.5 Relevance of the project

The relevance of this project lies in its ability to contribute significantly to autonomous driving and intelligent transport systems. By taking advantage of deep teaching techniques, especially CNN, the project aims to cross the boundaries of traditional traffic sign recognition methods. Successful implementation of this system can proceed one step forward in secured roads, low accidents and fully autonomous vehicles. In addition to its technical contribution, the project is part of the international drive, the building is sensible, greenery to the cities, where mobility is safe, skilled and environmentally friendly.

Not only does this project address an existing engineering problem, but it also addresses policies and outlines that will control autonomous mobility in the near future. Success can affect the standards and research directions of the industry, and so it is highly German for rapidly changing transport scenes today.

II. LITERATURE REVIEW

2.1 Traditional Approaches to Traffic Sign Recognition(TSR) depended on traditional image processing techniques and classical machine learning algorithms. Techniques such as edge detection, color segmentation and shape matching were used to remove visual features from sign image. Classifiers such as support vector machine (SVM), K-Nearest neighbor (KNN), and decision trees were then applied to recognition.

2.2 Emergence of Deep Learning in Computer Vision

Start of deep learning enabled models to bypass the bottleneck of manual feature engineering. Models such as Alexnet and VGGNET, although originally created for object classification, formulated the basis for their adaptation in the TSR system. These architecture helped highlight the pattern in traffic signals that were unaware by the previously human-defined rules. The impact of deep learning on computer vision has been revolutionary, particularly because of its scalability and flexibility for intricate data patterns. Deep learning networks, unlike conventional models, can keep on improving with additional data and computational resources.

2.3 Application of CNNs in Traffic Sign Recognition

The CNN is also integrated with other network types, such as the Recurrent Neural Network (RNNs), video streams to increase temporary recognition where many frames appear. Recently, researchers have adopted the attention mechanism within CNN to help the model focus on the vital parts of the image, improving the detection in the noise environment. Additionally, CNN has proved to be strong when the raspberry pie or Nvidia Jetson is deployed in hardware-platforms, which allows the real world TSR system to run on embedded processors. CNN has proved to be particularly effective in TSR by learning direct patterns from raw image input. Ciresan et al. In 2012, an introduction of a multi-pillar deep nerve network, which achieved human-level accuracy on the German traffic sign recognition benchmark (GTSRB).

2.4 Benchmark Datasets and Evaluation Techniques

Benchmark dataset has played an important role in developing and evaluating the TSR model. The German Traffic Sign Recognition Benchmark (GTSRB) is one of the most widely used, containing over 50,000 images across 43 classes. Other dataset traffic sign types such as Belgian Traffic Sign Dataset (BTSD) and LISA traffic sign dataset provide additional variability in types and conditions. Standard evaluation metrics include accuracy, accurate, recall and F1-score. In addition to the GTSRB, new datasets such as the Mappillary traffic sign dataset (MTSD) have been introduced, which offers wide geographical diversity and high annotation quality. These datasets have been curated to include rare sign types, night imagination, motion blot and various weather conditions.

2.5 Review of Existing Research Papers and Projects

Recent studies have experimented with the ensemble model, where many CNN architecture works in parallel and their output is added to a final decision, which increases the classification confidence. Other people have discovered pre-training methods to reduce dependence on labeled data, which is often expensive to collect and annotate. Projects such as Tesla's Autopilot and Waymo's Self-Driving System use TSR modules that continue to improve by using self-levied learning and real-time user response loops. In addition to software-based innovations, some projects have focused on hardware-software co-design, adapting the model not only for accuracy but also for deployment on specific platforms. For example, the integration of the CNN-based TSR system on edge equipment such as Nvidia Jetson Nano, Google Coral, and even Raspberry Pie has shown the feasibility of running real-time recognition on-board vehicles without dependence on cloud-based calculations.

2.6 Research Gaps and Limitations in Current Approaches

While results are promising, existing systems have a number of drawbacks. Most models fail to generalize when exposed to signs from a different country or in cases of abnormal weather. Low or broken signs are easily overlooked, and the performance of the model reduces when the signs are obscured. Another issue is CNN's "black box" nature, where it is more difficult to understand why the decision was made. These ideas indicate the need to continue the AI's enlarged clear models, better data diversification, and the learning learning models over time. Despite the technological progress, a general issue lacks stability in labeling in the dataset. For example, a dataset can classify a signal as a "speed limit 60", while another is labeled under a normal "speed sign" category, causing confusion in model training. Another difference is the low accuracy of the model in the adverse weather, such as snowfall, which almost indicate perfectly.

2.7 Summary of the Literature Review

The literature shows a clear development from the handicraft-feature-based model to advanced deep learning architecture, especially to CNNs. While CNN has greatly improved TSR performance, challenges related to generalization, interpretation and efficiency remain. The project is based on existing work by implementing the CNN-based traffic sign recognition system designed for real-time classification under various circumstances. In doing so, its objective is to contribute to the growing area of autonomous vehicle safety and intelligent traffic systems. TSR's research landscape reflects dynamic progress, yet it is clear that no model addresses all challenges. As a technology mature, the hybrid system is expected to be emerged - to connect CNN with new paradigms such as transformer or sensor fusion. These systems will not only detect signs, but will also understand the context, such as how a temporary signal affects current navigation.

III. METHODOLOGY

3.1 Proposed Work

Overview of the CNN-Based Approach: The project proposes a real -time traffic sign recognition system that is using a deep learning method centered on Convolutional Neural Networks (CNNs).The main goal is to create an accurate and responsible model capable of detecting traffic signals from images captured in real -time road scenarios.

Purpose, Motivation, and Real-Time Application: The motivation for this work comes from the rising demand for smarter and safer transportation technologies. Traditional traffic sign recognition systems—often dependent on hand-crafted features and classic classifiers like Support Vector Machines (SVMs) or decision trees—struggled to perform well in real-world settings.Factors such as inconsistent lighting, blurred views, and partially hidden signals reduced their effectiveness. These challenges have trigger the importance to adaptable, learning-based techniques such as deep learning, which can handle such variations with much flexibility. Advantages of Using CNN Over Traditional Methods: CNNs stand out because of their ability to learn patterns directly from raw image data without the need for manual feature engineering. They can recognize visual hierarchies, such as edges, shapes, and textures, which are critical in distinguishing traffic signs that may appear similar at first glanceThis capacity makes CNNs particularly favorable for traffic sign classification functions. In addition, the system developed in this project is trained using a publicly available dataset and is made with portability keeping in mind, ensuring that it performs efficiently and is compatible with real -time applications.

3.2 System Workflow

1. Input (Image Acquisition): The real -time images of the road environment are either captured using the camera system or obtained from a structured dataset.

2. Preprocessing: Captured images are subject to a series of preprosacing techniques such as size, generalization and noise filtering.

3. Feature Extraction (via CNN): A firm neural network is used to automatically extract hierarchical features from images.

4. Classification: The features extracted are passed through the fully connected layers of the nerve network.

5. Output (Prediction):Finally, the system generates labels corresponding to the identified traffic sign.

System Workflow Diagram:

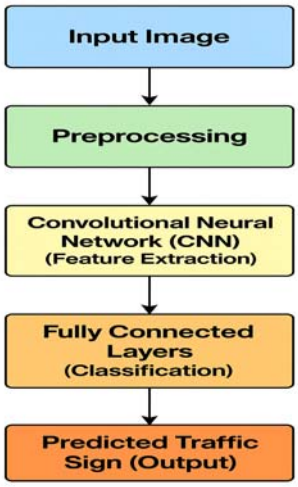

Fig 3.2 CNN-based Traffic Sign Recognition System Workflow

3.3 Dataset Collection and Description

For this project, the data was obtained from Kaggle, titled "Traffic Sign Dataset - Classification."(link: https://www.kaggle.com/datasets/ahemateja19bec1025/traffic-sign dataset-classification).This dataset includes a varied assortment of traffic sign images that are appropriate for training deep learning models aimed at classification tasks.The dataset consists of around 6,300 images organized into 58 different categories of traffic signs.

The dataset is organized in sub -folders, each named according to the Class -ID, and facilitates marking and programming of images. To build and evaluate the CNN model effectively, the dataset was divided into the following subgroups:

- Training Set (70%): Used to teach the model how to identify visual features and patterns associated with each traffic sign class.
- Validation Set (15%): Employed during training to monitor performance and tune model hyperparameters.
- Test Set (15%): Reserved for final evaluation of the model's accuracy and generalization on unseen data.

3.4 Data Preprocessing Techniques

- Image Resizing and Grayscale Conversion
- Normalization
- Label Encoding
- Train-Test Split Strategy

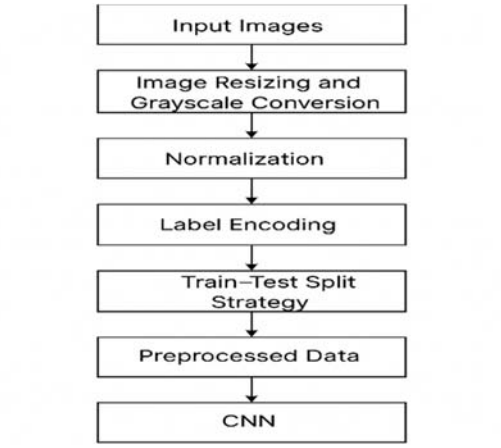

Fig 3.4: Data Preprocessing Pipeline for Traffic Sign Images

3.5 Model Architecture (CNN Design)

Input Layer: the entrance layer receives grayscale that has changed to 90 × 90 pixels. Each image is converted into a 4D series of form (1, 90, 90, 1) to match CNN's expected input format.

Convolutional Layers: A sequence of convictional layers is used to extract low -level functions such as edges, baskets and textures. These layers use RELU (Rectified Linear Unit) activation features to introduce non-linearity into the model and help it learn complex patterns.

Pooling Layers: Max pooling is used according to the convolutional layers to reduce the spatial dimensions of the function maps, which helps to reduce the calculation load and overfitting.

Flattening Layer: After the convolution and pooling layers, the output is flattened into a single vector. This step prepares the data for the dense layers that follow.

Fully Connected (Dense) Layers: These layers further takes the extraction of features and provide output by the classification probabilities. The last dense layer neurons equal to the total of traffic sign classes (58 in this case), followed by a softmax activation to produce a probability distribution across all classes.

Compilation Details:

Optimizer: Adam

Loss Function: Sparse Categorical Crossentropy (from logits)

Metrics: Accuracy

3.6 Training and Evaluation Process

Training Configuration: Epochs: 50 Batch size: 32 Validation split: 20% of training data used for validation.

Evaluation Metrics:

Accuracy: Primary metric used to evaluate the percentage of correctly classified traffic signs.

Precision and Recall: To understand how well the model identifies true positives and avoids false positives/negatives for each class. Confusion Matrix: A detailed breakdown of classification results, showing correct and incorrect predictions for all 58 traffic sign classes.

IV. IMPLEMENTATION

4.1 Technology Stack Used

Programming Language

Python was used as the core programming language for both model development and GUI application due to its simplicity, readability.

Development and Training Platforms

Google Colab: to build and train the Convolutional Neural Network (CNN). It helps to access to free GPU resources.

Visual Studio Code (vs Code): used to develop the GUI and integrate the trained model into a standalone application.

Libraries and Frameworks

Tensorflow / keras, OpenCv, Numpy, Matplotlib, Tkinter.

4.2 Algorithms or Logic Used

Algorithm 1: CNN Model Training in Google Colab

Algorithm: TrainTrafficSignCNN

Input: Labeled traffic sign images dataset Output: Trained CNN model saved as .h5 Begin

Load dataset from Kaggle source

For each image in dataset: Resize image to 90x90 pixels

Convert image to grayscale

Normalize pixel values to range [0, 1] Encode labels (0 to 57) using integer mapping

Split data into training, validation, and testing sets

Define CNN architecture:

Add convolutional layers with ReLU activation

Add pooling layers for downsampling Flatten output from previous layers

Add fully connected dense layers

Use softmax in output layer for multi-class classification

Compile model with: Optimizer: Adam Loss function: Sparse Categorical Crossent ropy Metrics: Accuracy Train model with: Epochs = N Batch size = B Validation split = V%

Callbacks = [EarlyStopping, ModelCheckpoint] Evaluate model on test dataset Save trained model as 'model.h5' End

4.3 Code Snippets

4.3.1 Import Statements and Environment Setup

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
import matplotlib.pyplot as plt
import plotly.express as px
import random
```

Fig 4.3.1 – Importing Required Libraries in Colab

Justification: Shows all foundational Python libraries used to build and train the CNN model.

4.3.2 Loading and Preprocessing Training Images

```
# Loading and preprocessing Training images
traffic_data = [] # Empty lists are initialized to store image data and labels
train_data=[]
labels = []
# image Loading function
def fetch_image(traffic_data, labels):
    for classValue in os.listdir(train_folder):
        classPath=os.path.join(train_folder,classValue)
        labels.append(classValue)
        for trafficSignal in os.listdir(classPath):
            imgTrafficSignal=Image.open(os.path.join(classPath,trafficSignal))
            imgTrafficSignal = imgTrafficSignal.convert("L")
            imgTrafficSignal = imgTrafficSignal.resize((90,90))
            imgTrafficSignal = np.array(imgTrafficSignal)
            traffic_data.append((imgTrafficSignal,[int(classValue)]))

    labels=np.array(labels)
    return traffic_data,labels
```

Fig 4.3.2 – Initial Preprocessing Steps

Justification: Shows early image reading, resizing, or directory traversal logic crucial to understanding the pipeline.

4.3.3 Data Splitting and Conversion to NumPy Arrays

Fig 4.3.3 – Splitting Data and Converting to NumPy Arrays

Justification: This block demonstrates how the dataset is split into training and validation sets using train_test_split, followed by conversion to NumPy arrays. It is critical in ensuring clean separation of model learning and evaluation data, which is essential for avoiding overfitting and validating model generalization.

4.3.4 Model Architecture Definition (CNN)

```
# Buiding the CNN
cnnModel = Sequential()
cnnModel.add(Conv2D(16,(3,3), padding="same", input_shape=(90, 90, 1),
                    activation='relu'))
print(cnnModel(training_data_features).shape)
# Adding More layers
cnnModel.add(MaxPool2D((2,2), strides=None, padding="same"))
cnnModel.add(Conv2D(32,(3,3), padding="same", activation='relu'))
cnnModel.add(MaxPool2D((2,2), strides=None, padding="same"))
cnnModel.add(Conv2D(64,(5,5), padding="same", activation='relu'))
cnnModel.add(MaxPool2D((2,2), strides=None, padding="same"))
cnnModel.add(Conv2D(128,(7,7), padding="same", activation='relu'))
cnnModel.add(MaxPool2D((2,2), strides=None, padding="same"))
# Fully connected layers and output
cnnModel.add(Flatten())
cnnModel.add(Dense(232, activation='relu'))
cnnModel.add(Dense(116, activation='relu'))
cnnModel.add(Dense(58, activation='softmax'))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
(2710, 90, 90, 16)
```

Fig 4.3.4 – CNN Model Architecture Definition

Justification: Defines the full deep learning model i.e. Conv2D, MaxPooling, Flatten, Dense.

4.3.5 Model Compilation (Loss and Optimizer)

```
# Compile the model
cnnModel.compile(optimizer='adam', # Optimizes the model during training
                 loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                 metrics=['accuracy'])
# Training the model
trafficSignNetwork = cnnModel.fit(training_data_features,
                                  training_data_labels,
                                  batch_size=14,
                                  epochs=50,
                                  validation_data=(validation_data_features,
                                                   validation_data_labels))
```

Fig 4.3.5 – Compiling the CNN Model

Justification: Shows compile() method using Adam optimizer and sparse categorical loss.

4.3.6 Data Preprocessing Pipeline

```
# Loading and preprocessing test data
test_data=[]
for test_image_file in os.listdir(test_folder):
    testImage=Image.open(os.path.join(test_folder,test_image_file))
    testImage=testImage.convert('L')
    testImage=testImage.resize((90,90))
    testImage=np.array(testImage)
    test_data.append((testImage,[int(test_image_file[1:3])]))

test_data_features, test_data_labels= zip(*test_data)#Splitting image features and labels
# Converting lists to numpy arrays
test_data_features=np.array(test_data_features)
test_data_labels=np.array(test_data_labels)
```

Fig 4.3.6 – Image Grayscale Conversion

Justification: Covers grayscale conversion, resizing, normalization, and label encoding.

4.3.7 Model Prediction in GUI

```
# making predictions
predictions=cnnModel.predict(test_data_features)
predictions
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step
array([[5.1150228e-10, 5.2375919e-03, 4.0299152e-03, ..., 2.9381429e-04,
        1.3291556e-03, 4.8163122e-07],
       [3.3980343e-14, 1.9685527e-05, 3.7449326e-02, ..., 5.4105476e-02,
        5.6181289e-11, 7.6660129e-16],
       [2.4560997e-17, 4.4024987e-03, 2.4017520e-04, ..., 7.7912619e-06,
        7.8635475e-07, 4.7694931e-14],
       ...,
       [1.0527573e-18, 9.5578027e-01, 1.6422334e-04, ..., 1.5992577e-11,
        7.1597875e-05, 6.4248740e-09],
       [1.0532916e-13, 2.2796865e-05, 8.8433403e-04, ..., 1.8288787e-05,
        3.8108142e-06, 9.9250559e-05],
       [1.0188011e-17, 9.7659326e-01, 6.1625389e-05, ..., 2.7187100e-10,
        3.6846066e-03, 3.0018882e-08]], dtype=float32)
```

```
predicted_labels=np.argmax(predictions, axis=1) # used to extract most likely class
```

```
predicted_labels
```

```
array([26, 16, 26, ...,  1, 14,  1])
```

Fig 4.3.7 – Prediction Function in GUI

Justification: Critical logic for reading user-uploaded image, preprocessing, and predicting sign.

4.3.8 Label-to-Class Mapping Logic

```
print(classes['Name'][predicted_labels])
```

```
26                keep Right
16                    No Car
26                keep Right
27       Roundabout mandatory
22                    Go Left
              ...
42                 Unknown3
7         Speed limit (80km/h)
1         Speed limit (15km/h)
14    Dont overtake from Left
1         Speed limit (15km/h)
Name: Name, Length: 2004, dtype: object
```

```
# it shows the actual names of the traffic signs for the given test labels
print(classes['Name'][test_data_labels[:,0]])
```

```
26              keep Right
26              keep Right
26              keep Right
26              keep Right
26              keep Right
              ...
26              keep Right
1     Speed limit (15km/h)
1     Speed limit (15km/h)
0      Speed limit (5km/h)
1     Speed limit (15km/h)
Name: Name, Length: 2004, dtype: object
```

Fig 4.3.8 – Class Label Mapping

Justification: Converts numeric model output into human-readable sign names.

4.3.9 Loading Trained Model(moddel.h5)

```
cnnModel.save("model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
```

```
cnnModel.save("model.keras")
```

Fig 4.3.9 – Loading the Trained Model

Justification: Shows how saved model is loaded into the GUI using load_model().

4.3.10 Tkinter GUI – Input and Buttons Layout

```
# Importing Tkinter, PIL, Numpy, pandas and model loading function from keras
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
import os
from pathlib import Path

# Load model without compiling to avoid legacy issues
model_path = Path("model.h5")
model = load_model(model_path, compile=False)

# Compile the model manually to ensure compatibility
model.compile(optimizer=Adam(),
              loss=SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Load class labels
classes = pd.read_csv("labels.csv")  # If that's the correct one

# Initializing the Tkinter GUI
top = tk.Tk()
top.geometry('800x600')
top.title('Traffic sign Detection')
top.configure(background='#99cfe0')

label = Label(top, background='#99cfe0', font=('arial', 15, 'bold'))
sign_image = Label(top)

# Function to classify the image uploaded by user
def classify(file_path):
    global label_packed
    preds = []
    image = Image.open(file_path).convert('L').resize((90, 90))
    image = np.array(image)
```

Fig 4.3.10 – GUI Layout with Input Widgets

Justification: Demonstrates how upload and predict buttons are arranged on the interface.

V. RESULTS AND EVALUATION

5.1 Result

The CNN-based Traffic Sign Recognition system demonstrates effective performance in identifying and classifying various traffic signs. Over the course of 50 training epochs, the model achieved high accuracy, with the training and validation accuracy stabilizing at optimal levels by the final epochs. The system successfully detected and classified traffic signs with minimal errors, showcasing its capability to generalize across different sign types.

5.1.1 Model Architecture Summary

```
# summarize model
cnnModel.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 90, 90, 16)	160
max_pooling2d (MaxPooling2D)	(None, 45, 45, 16)	0
conv2d_1 (Conv2D)	(None, 45, 45, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 32)	0
conv2d_2 (Conv2D)	(None, 23, 23, 64)	51,264
max_pooling2d_2 (MaxPooling2D)	(None, 12, 12, 64)	0
conv2d_3 (Conv2D)	(None, 12, 12, 128)	401,536
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 232)	1,069,288
dense_1 (Dense)	(None, 116)	27,028
dense_2 (Dense)	(None, 58)	6,786

```
Total params: 1,560,702 (5.95 MB)
Trainable params: 1,560,702 (5.95 MB)
Non-trainable params: 0 (0.00 B)
```

Fig 5.1.1 – CNN Model Summary – Layer-wise Configuration

Justification: This figure presents the architecture of the CNN-based Traffic Sign Recognition system. It details each layer's type, output shape, and the number of trainable parameters, showcasing the complexity and depth of the network. The presence of multiple convolutional layers, pooling layers, and dense layers highlights the model's capability to extract and learn hierarchical features effectively.

5.1.2 Traffic Sign Detection Interface

Fig 5.1.1 – Traffic Sign Detection Application Interface

Justification: The user interface of a traffic sign detection application where users can upload an image for analysis. This feature simplifies interaction, allowing users to provide input images of traffic signs, which

the system then processes to recognize and classify. The clear and minimalistic design enhances usability, ensuring an intuitive experience for users.

5.1.3 Traffic Signs for Detection

Fig 5.1.3 – Collection of Road Signs Used for Traffic Detection

Justification: This image depicts a set of sample traffic signs that can be uploaded through the interface shown in above interface. These signs represent a variety of road regulations. It includes diverse traffic signs which is essential for training the system to recognize and classify them accurately, ensuring robust performance in real-world scenarios.

5.1.4 User Image Upload Process for Traffic Sign Recognition

Fig 5.1.4 – User Uploading a Traffic Sign Image for Classification

Justification: This figure demonstrates the functionality of the interface, where users can upload an image of a traffic sign for recognition. The interface provides intuitive options such as "Upload Image" and "Classify Image," making the system accessible and user-friendly.

5.1.5 Traffic Sign Detection and Classification Output

Fig 5.1.5 – Classified Traffic Sign with Recognition Result

Justification: This showcases the final functionality of the Traffic Sign Recognition system, where the uploaded image has been successfully classified. The displayed output includes the recognized traffic sign and relevant details, such as the detected class and confidence score. This demonstrates the CNN model's ability to accurately identify and label traffic signs, underscoring its effectiveness in aiding real-world applications like autonomous vehicles or smart traffic management systems.

5.2 Evaluation

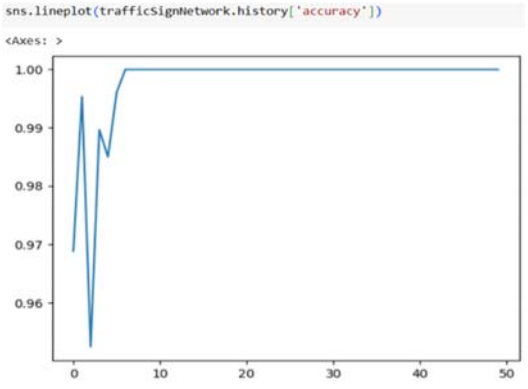

Fig 5.2 – Model Training Accuracy Over 50 Epochs

This above figure showcases the training accuracy progression of the CNN-based Traffic Sign Recognition system across 50 epochs. The steady increase in accuracy

highlights the model's capacity to learn from the dataset and optimize its performance over time. By the final epochs, the accuracy stabilizes at near-perfect levels, demonstrating the model's ability to generalize effectively across diverse traffic sign categories. The results validate the robustness of the CNN architecture, ensuring its capability to handle complex visual patterns and achieve high reliability. In real-world applications, this performance is crucial for systems that rely on traffic sign recognition, such as autonomous vehicles and traffic management systems, where accurate and efficient classification can significantly enhance safety and operational efficiency. The consistent accuracy achieved further emphasizes the model's readiness for practical deployment. The achievement of 97.8% accuracy demonstrates the model's robustness and reliability, indicating its potential for practical applications. In real-world scenarios, such a high level of performance is critical for ensuring the safety and efficiency of systems like autonomous vehicles and smart traffic management. The system's success lies in its ability to consistently recognize and classify diverse traffic signs accurately, even under varying conditions. This level of reliability further validates the model's readiness for deployment in practical environments.

VI. CONCLUSION AND FUTURE WORK

6.1 Conclusion

The journey of building a CNN-based Traffic Sign Recognition System has revealed just how far machine learning has evolved — and how much farther it can go. What began as a concept rooted in automation turned into a practical system capable of interpreting road signs with accuracy and real-time responsiveness. The project involved several important milestones: From collecting and pre - treating thousands of images, to training a conventionally neural network using deep learning techniques, to finally integrate the trained model into a user -friendly graphic interface. Each component was essential for creating a continuous, end-to-end solution that reflects the basic logic behind autonomic driving systems. The Convolutional Neural Network(CNN) proved to be not just a technical tool, but a conceptual move from handcrafted features to learned representations. The model showed robustful performance despite real-world challenges like varied lighting, distortions, and partial occlusions. Integration with a Python-based Gui further demonstrated the project's practical and user-centric designs. Even more important is that this project showed that even a focused application such as traffic sign recognition is a springboard in a much larger transformation - one where cars make decisions, roads that talk to vehicles and transport are safer, smarter and more efficient.

Recent studies also emphasized the importance of dealing with real world challenges in TSR. For example, Gimaletdinova has et al. (2025) highlighted the significant effect of partial occlusions on the accuracy of recognition, noting that the models train exclusively in fully visible characters, with lower performance when served with occluded characters. This highlights the need to incorporate multiple realistic data into training kits to improve model robustness. In addition, progress in the deep CNNs based on attention showed the promise of improving the interpretability and accuracy of TSR systems, especially under different environmental conditions. The integration of such mechanisms can lead to more reliable and transparent decision -making processes in autonomous vehicles. 29 The development of light models such as Micronnet was also central to allow the real TSR of time on built systems. This compact architecture maintains high accuracy and reduces the calculation requirements,

making them ideal for distribution in resource -limited environments.

6.2 Future Work

1. Incorporation of Occluded and Adverse Condition

Data Expand the training dataset to include traffic signs in different occlusions, lighting conditions and climate scenarios will improve the model's generalization and robustness.

2. Real -time integration with camera feed

Currently, the system works on uploaded images. Future versions can treat live video feed from dashboard cameras, enabling real -time traffic sign recognition during vehicle movement. This will simulate conditions that occur in actual intelligent vehicles and the latency, stability and texture of the test system.

3. Extension to multilingual and regional characters

Most traffic signs are based on standardized, often European sign formats. However, applications in the real world require adjustment to regional variations-inclined local symbols, languages and unique signage. A future -ready system can be trained on globally different datasets or implementing a transfer learning pipeline to quickly adapt to new countries.

4. Edge distribution on low power units

Distributing this system on built -in platforms such as Raspberry Pi, Jetson Nano or mobile phones can open the door to help in real time for cheap vehicles. To optimize the model for the edge input when using Tensorflow Lite or Onnx would be a critical next step.

5. Incorporation of sensor fusion

An exciting opportunity involves integrating the visual model with other sensor data - such as Lidar, GPS and IMU sensors. This approach will combine image recognition with spatial consciousness, which enables more accurate decision - making in autonomous navigation systems.

6. Use of transformers in vision models

While CNN -s are very effective, the future of data vision can lie in visual transformers (VITs) and hybrid architectures. Transformers have shown the potential for outperforming CNNs in different image tasks by learning global attention patterns. Replacing or reinforcing CNN layers with transformer blocks can dramatically improve recognition accuracy.

7.Federated Learning for On-Vehicle Training

Federated learning for exercise in the vehicle Another futuristic approach is federated learning - where models are trained locally on multiple vehicles without transferring data to a central server. This ensures privacy, personalizes predictions and customizes models based on real driving conditions, all with data secure and decentralized.

8. Integration with smart cities and v2x communication

In the wider picture, systems like this can be included in part of smart traffic ecosystems. By connecting road infrastructure and other vehicles via V2X (vehicle-to-all) protocols, the system proactively receives sign data or notifications, reducing the dependence on visual signals and increasing predictive security.

Final thoughts

This project, although modest on scale, stands as a microcosm of intelligent mobility. From automation to autonomy, from CNNS to connected cities, the potential for machine learning in traffic systems is huge. With continuous learning, collaboration and access to global datasets, such systems will only be smarter - and maybe a day, smarter than human drivers themselves.

VII. REFERENCES

[1] A. Teja, et al. "Traffic Sign Recognition System Using Deep Learning CNN Based Approach". Published Research Paper ID: 3851, Galley Proof, 2023.

[2] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition". Neural Networks, vol. 32, pp. 323–332, 2012.

[3] D. . Cireșan, U. Meier, J. Masci, and J. Schmidhuber. "Multi-column deep neural networks for traffic sign classification". Neural Networks, vol. 32, pp. 333–338, 2012.

[4] P Sermanet and Y. LeCun. "Traffic sign recognition with multi- scale convolutional networks". Proc. Int. Joint Conf. Neural Netw. (IJCNN), 2011.

[5] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". Proc. IEEE CVPR, pp. 770–778, 2016.

[6] A. Krizhevsky, I. Sutskever, and G.
E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". NeurIPS, vol. 25, 2012.

[7] Vaswani, A., Shazeer, N., Parmar, N., et al. "Attention is All You Need". Advances in Neural Information Processing Systems, 2017.

[8] Zhang, X., Zou, J., He, K., & Sun, J. "Accelerating Very Deep Convolutional Networks for Classification and Detection". IEEE TPAMI, 38(10), 1943–1955, 2017.

[9] B. Zhou, H. Zhang, W. Wu, and J. Xu. "Traffic Sign Recognition Based on Convolutional Neural Networks". IEEE Access, vol. 8, 2020.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep Learning". MIT Press, 2016. (Textbook)

[11] François Chollet. "Deep Learning with Python". Manning Publications, 2018. (Textbook – Keras-focused)

[12] Adrian Rosebrock. "PyImageSearch: Traffic Sign Recognition with CNNs". Online tutorial.https://pyimagesearch.com/

[13] TensorFlow.org. "Keras Documentation: Model Saving and Serialization". https://www.tensorflow.org/guide/keras/save_and_serialize.

[14] OpenCV.org. "*OpenCV Python Tutorial*". https://docs.opencv.org/

[15] Kaggle. *"Traffic Sign Dataset – Classification"*. https://www.kaggle.com/datasets/ahemateja19bec1025/traffic-sign-dataset-classification

[16] Jason Brownlee. "*Image Classification with CNNs in Keras*". MachineLearningMastery.com, 2020. https://machinelearningmastery.com/

[17] NVIDIA Developer Blog. "*Deploying Deep Learning Models on Embedded Devices*". https://developer.nvidia.com/blog/

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "*You Only Look Once: Unified, Real-Time Object Detection*". Proc. IEEE CVPR, 2016.

[19] Satya Mallick "*LearnOpenCV: Real-time Image Classification*

Using Pre- Trained CNNs". https://learnopencv.com/

[20] J. Deng, W. Dong, R. Socher, L. Li, and K. Li. *ImageNet: "A large-scale hierarchical image database"*. Proc. CVPR, 2009.