TRAFFIC SIGN DETECTION

A

Course Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

S. BAGAWAN REDDY 1602-21-737-012

P. SAI CHARAN 1602-21-737-045

Under the guidance of

Dr. K. Ram Mohan Rao

Professor & HOD, IT

Department of Information Technology

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Ibrahimbagh,

Hyderabad-500 031

2024-25

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

DECLARATION BY THE CANDIDATE

We, **S. BAGAWAN REDDY** and **P. SAI CHARAN** bearing hall ticket number **1602-21-737-012** and **1602-21-737-045** hereby declare that the course project report entitled **TRAFFIC SIGN DETECTION** under the guidance of **Dr. K. RAM MOHAN RAO, PROFESSOR**, Department of Information Technology, Vasavi College of Engineering, Hyderabad.

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma

S. BAGAWAN REDDY
1602-21-737-012

P. SAI CHARAN
1602-21-737-045

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

DECLARATION BY THE CANDIDATE

I, **S. BAGAWAN REDDY** bearing hall ticket number **1602-21-737-012** hereby declare that the course project report entitled **TRAFFIC SIGN DETECTION** under the guidance of **Dr. K. RAM MOHAN RAO, PROFESSOR**, Department of Information Technology, Vasavi College of Engineering, Hyderabad.

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma

S. BAGAWAN REDDY
1602-21-737-012

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

DECLARATION BY THE CANDIDATE

I, **P. SAI CHARAN** bearing hall ticket number **1602-21-737-045** hereby declare that the course project report entitled **TRAFFIC SIGN DETECTION** under the guidance of **Dr. K. RAM MOHAN RAO, PROFESSOR**, Department of Information Technology, Vasavi College of Engineering, Hyderabad.

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma

P. SAI CHARAN
1602-21-737-045

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

BONAFIDE CERTIFICATE

This is to certify that the course project report entitled **TRAFFIC SIGN DETECTION** being submitted by **S. BAGAWAN REDDY** and **P. SAI CHARAN** bearing **1602-21-737-012** and **1602-21-737-045** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by them under my guidance.

Dr. K. Ram Mohan Rao	**Examiner**	**Dr. K.Ram Mohan Rao**
Internal Guide		**Professor & HOD, IT**

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

BONAFIDE CERTIFICATE

This is to certify that the course project report entitled **TRAFFIC SIGN DETECTION** being submitted by **S. BAGAWAN REDDY** bearing **1602-21-737-012** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by them under my guidance.

Dr. K. Ram Mohan Rao	**Examiner**	**Dr. K.Ram Mohan Rao**
Internal Guide		**Professor & HOD, IT**

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

BONAFIDE CERTIFICATE

This is to certify that the course project report entitled **TRAFFIC SIGN DETECTION** being submitted by **P. SAI CHARAN** bearing **1602-21-737-045** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by them under my guidance.

Dr. K. Ram Mohan Rao	**Examiner**	**Dr. K. Ram Mohan Rao**
Internal Guide		**Professor & HOD, IT**

ACKNOWLEDGEMENT

The satisfaction that accompanies that the successful completion of the Main project would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

It is with immense pleasure that we would like to take the opportunity to express our humble gratitude to **Dr. K. RAM MOHAN RAO, PROFESSOR & HOD, Department of Information Technology** under whom we executed this project. We are also grateful to **Dr. T. HITENDRA SARMA, ASSOCIATE PROFESSOR, Department of Information Technology,** for his guidance. Their constant guidance and willingness to share their vast knowledge made us understand this project and its manifestations in great depths and helped us to complete the assigned tasks.

We are very much thankful to **Dr. K. RAM MOHAN RAO, PROFESSOR & HOD, Department of Information Technology,** for his kind support and for providing necessary facilities to carry out the work.

We wish to convey our special thanks to **Dr. S. V. RAMANA**, **Principal** of **Vasavi College of Engineering and Management** for providing facilities. Not to forget, we thank all other faculty and non-teaching staff, and my friends who had directly or indirectly helped and supported me in completing my project in time

ABSTRACT

This research focuses on improving the performance of Convolutional Neural Networks (CNNs) for traffic sign detection, with a specific emphasis on reducing computational complexity while maintaining or improving detection accuracy. Building on a previous study that introduced synthetic image generation techniques for traffic sign datasets, the original model achieved moderate accuracy but was computationally expensive and had limitations in terms of precision and reliability. The current work aims to address these challenges by exploring methods to optimize the model architecture, reducing the number of parameters and computations required during both training and inference. By incorporating deeper, more efficient network architectures, such as lightweight CNNs, this study seeks to strike a balance between improved generalization to real-world traffic signs and decreased computational load. The goal is to make the traffic sign detection system more practical for deployment in real-time applications, such as autonomous vehicles, where efficiency is crucial for real-world performance. This approach will not only enhance accuracy but also enable faster processing, contributing to safer, more efficient, and computationally feasible traffic sign detection systems.

LIST OF FIGURES

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Problem Statement – Overview

The problem statement for the **Traffic Sign Detection** project focuses on developing and optimizing a Convolutional Neural Network (CNN) model to reduce computational requirements while maintaining effective performance in detecting traffic signs from arbitrary images. Existing models often face challenges related to high computational costs, making them less suitable for real-time applications in environments with constrained resources, such as autonomous vehicles or embedded systems. This project aims to address these limitations by designing efficient CNN architectures, utilizing advanced optimization techniques, and employing strategic data augmentation to streamline processing without compromising detection capabilities. The objective is to create a **computationally efficient** model that can operate in real-time scenarios, ensuring rapid and reliable traffic sign detection. By leveraging GPU-accelerated hardware for efficient training and deploying the optimized model using lightweight frameworks like Flask, this research contributes to the development of scalable, low-resource traffic detection systems that are suitable for practical applications.

1.2. Motivation (content is like Proposed Work)

India's growing infrastructure and rapid urbanization have led to significant challenges in managing the increasing traffic volume, especially in ensuring safety through effective traffic sign detection. As autonomous driving systems and advanced driver-assistance systems (ADAS) become more prevalent, efficient and real-time traffic sign detection is critical for their success. However, current methods often rely on computationally intensive

Convolutional Neural Network (CNN) models, which are impractical for real-time applications in resource-constrained environments, such as embedded systems or edge devices. This limitation creates a pressing need for innovative approaches that optimize detection systems while reducing computational overhead.

This research is motivated by the vision of transforming traffic management by bridging the gap between existing high-accuracy but resource-heavy models and the need for lightweight, efficient solutions. The proposed work focuses on developing a computationally optimized **Traffic Sign Detection** System by leveraging advanced CNN architectures, efficient regularization techniques, and targeted data augmentation. The system aims to process images rapidly while maintaining reliable detection capabilities, making it suitable for real-world applications such as autonomous vehicles and intelligent traffic systems.

By addressing the challenges of high computational demand, the project seeks to create a scalable, real-time detection solution that ensures efficient operation in diverse scenarios, contributing to the broader goal of enhancing road safety and supporting smart city initiatives. This work represents a critical step toward sustainable and accessible traffic management technologies.

1.3. Scope & Objectives of the Proposed Work

Scope

The Traffic Sign Detection System focuses on addressing the critical challenge of reducing computational time in real-time traffic sign detection applications. The scope involves designing and implementing an optimized Convolutional Neural Network (CNN) model that maintains effective detection capabilities while minimizing processing overhead. This work targets applications in autonomous vehicles, advanced driver-assistance systems (ADAS), and

intelligent traffic management systems, where rapid and efficient performance is essential. Key areas of focus include:

- **Computational Optimization**: Developing streamlined CNN architectures with reduced complexity to decrease computational time without compromising detection performance.
- **Real-Time Detection**: Ensuring the system processes traffic signs rapidly to support real-time decision-making in dynamic environments.
- **Efficient Data Handling**: Utilizing advanced data augmentation and preprocessing techniques to enhance model training and inference efficiency.
- **Resource-Constrained Deployment**: Creating a system compatible with lightweight hardware platforms, enabling deployment in embedded and edge devices.

Objectives

1. To design a CNN-based traffic sign detection system optimized for reduced computational time and resource usage.
2. To achieve reliable real-time performance by decreasing inference latency while maintaining detection accuracy.
3. To develop a scalable solution capable of operating effectively on hardware with limited computational power.
4. To integrate the optimized system into real-world applications, ensuring usability in time-critical scenarios such as autonomous driving and traffic monitoring.

1.4. Organization of the report

1. Literature Survey: This section reviews existing literature and research relevant to the document's topic, offering background information and context for the proposed work.

2. Proposed System

• System Specifications

• Software Requirements: Lists the required software tools and technologies.

• Hardware Requirements: Specifies the necessary hardware components.

• Methodology

• Architecture Diagram: Presents a visual representation of the system architecture.

3. Experimental Setup & Results

• Datasets: Describes the datasets used for experimentation**.**

• Results & Test Analysis: Presents experimental results and provides analysis, including figures and tables with explanations.

4. Conclusion and Future Scope: Summarizes the findings of the document and discusses potential future directions or extensions of the proposed work.

2 LITERATURE SURVEY

LITERATURE SURVEY

PAPER 1:

Effortless Deep Training for Traffic Sign Detection Using Templates and Arbitrary Natural Images

SUMMARY:

- The paper tackles the challenge of training deep learning models for traffic sign detection without relying on annotated real-world datasets.
- Real-world datasets are expensive and time-consuming to collect and annotate.
- To address this, the authors developed a synthetic data generation method.
- This method combines natural images with traffic sign templates to create an effective training set.
- The approach helps overcome issues such as data imbalance and the costly annotation process.
- Synthetic datasets generated using templates mimic real-world scenarios effectively.
- The approach is computationally inexpensive, making it scalable for diverse traffic scenarios.
- The authors emphasize the importance of randomizing lighting and background conditions in synthetic data to improve model generalization.
- Performance evaluation shows comparable results with real-world dataset-trained models, proving its practical applicability.

SOURCE:

DOI: 10.1109/IJCNN.2019.8852086

PAPER 2:

Enhancing Traffic Sign Detection and Classification Using Multi-Task Learning with CNNs

SUMMARY:

- The problem identified in this study focuses on improving the performance of Traffic Sign Recognition (TSR) systems.
- TSR systems' performance is affected by adverse weather conditions and environmental changes, which impact the accuracy of autonomous vehicles.
- The authors selected this problem to leverage Convolutional Neural Networks (CNNs) and the German Traffic Sign Recognition Benchmark (GTSRB) dataset.
- The aim is to enhance the robustness and accuracy of detection models for real-world autonomous driving and assistance systems.
- The multi-task learning framework reduces overfitting by sharing information across detection and classification tasks.
- Data augmentation techniques, including adding noise and simulating poor lighting, are incorporated to train robust models.
- The authors highlight improvements in real-time processing speeds due to architectural optimizations.
- Experimental results show a significant accuracy boost under adverse conditions, with an average detection rate increase of 12%

.**SOURCE:**

DOI: 10.1109/ICCST59048.2023.10474262

PAPER 3:

Deep Learning for Large-Scale Traffic-Sign Detection and Recognition

SUMMARY:

- The paper focuses on the need for large-scale traffic-sign detection for inventory management.
- Most existing models are limited to detecting a small set of categories relevant to autonomous driving.
- The study identifies a gap in handling high intra-category variability.
- It proposes using Mask R-CNN to build a system that can detect and recognize a broad range of traffic signs across different conditions.
- Mask R-CNN's region proposal network is fine-tuned for traffic-sign shapes and sizes.
- A custom loss function is developed to improve performance for traffic signs with high intra-class variability.
- Detailed experiments demonstrate robustness against occlusions and partial visibility.
- The proposed model is scalable for detecting up to 500 traffic-sign classes, significantly exceeding the scope of most previous studies.

SOURCE:

DOI: 10.1109/TITS.2019.2913588

3 PROPOSED ALGORITHM/MODEL

3.1. Hardware/Software Specifications

3.1.1 Hardware Requirements

- **GPU-enabled Machine**: A machine with a powerful GPU (e.g., NVIDIA Tesla, RTX series) for training deep CNN models efficiently. GPUs significantly reduce training time compared to CPUs.
- **RAM**: At least 16 GB to support efficient model training and data processing.
- **Storage**: SSD with at least 512 GB for fast data access and model storage.

3.1.2. Software Requirements

- **Programming Language**: Python, widely used for machine learning and deep learning with support for libraries like PyTorch.
- **Libraries**:
  - OpenCV: For image preprocessing, such as resizing, transformations, and augmentations.
  - NumPy: For efficient numerical computations and handling large datasets.

3.2. Methodology

3.2.1 Block Diagram

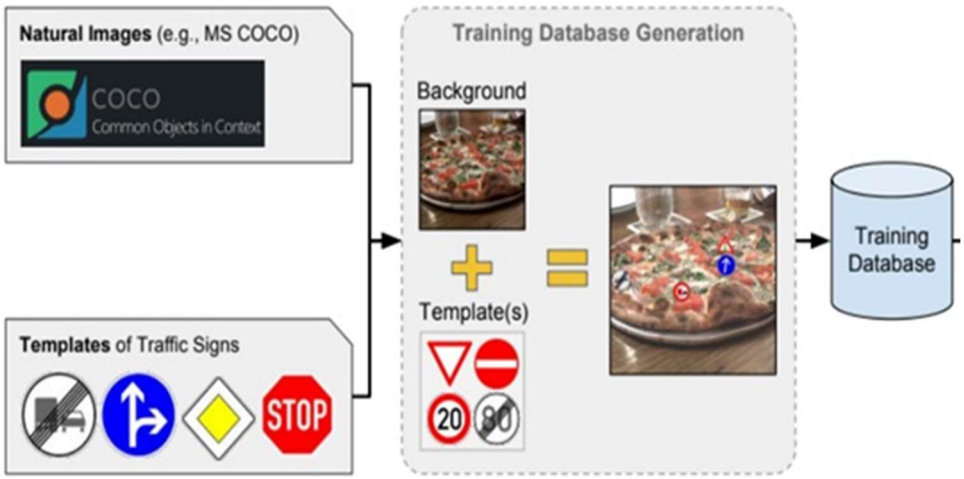

Fig1: Dataset Generation

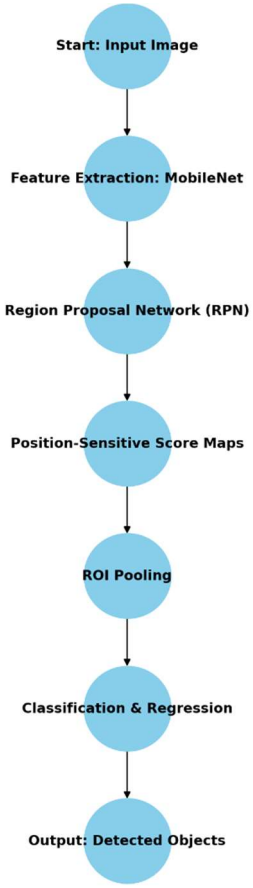

Fig2:Flowchart

Overall Flow

1. Input Data

- **Natural Images (e.g., MS COCO):** These are background images sourced from a large dataset like MS COCO, which contains various real-world scenes without traffic signs. These images serve as the background for creating synthetic training samples.
- **Templates of Traffic Signs:** These are pre-designed templates or cropped images of traffic signs, such as stop signs, speed limit signs, and directional signs. These templates will be added to the background images to simulate real-world conditions.

2.Training Database Generation

- **Background + Template(s):** The process involves overlaying traffic sign templates onto the background images in realistic positions and orientations. This generates synthetic images where traffic signs are placed within varied contexts, simulating natural scenarios.

- **Training Database:** The synthetic dataset is then organized into a training database. This database includes both the synthetic images and annotations (bounding boxes and labels) indicating the positions and classes of the traffic signs.

3. Faster R-CNN Training

This block represents the training process of the **Faster R-CNN** model:

- **Region Proposal Network (RPN):** Proposes candidate regions in the images that might contain objects (traffic signs).
- **Classification & Regression:** Classifies the proposed regions into specific traffic sign categories and refines the bounding boxes for accurate localization.

The output of this block is a set of **model parameters**, which include the trained weights for detecting and classifying traffic signs.

4. Inference

Input Images: These are real-world images (e.g., road scenes captured by cameras) on which the trained Faster R-CNN model is tested.

Region Proposal & Classification:

- The trained model processes the input images, identifies regions of interest, and classifies them as specific traffic signs.

Output: The final output consists of the detected traffic signs in the images, with bounding boxes and confidence scores indicating the detection's accuracy.

3.2.2 Pseudo Code

```
import torch

import torch.nn as nn

from torchvision.models.detection import FasterRCNN

from torchvision.models.detection.rpn import AnchorGenerator

from torchvision.models import mobilenet_v2

import torchvision


class MobileNetBackbone(nn.Module):

  def __init__(self, mobilenet):

    super().__init__()
```

```
        # Extract layers up to the final convolution
        self.features = nn.Sequential(*list(mobilenet.features.children()))
        self.out_channels = 1280  # Output channels of the final layer

    def forward(self, x):
        # Only output the last layer
        feature = self.features(x)
        return {"0": feature}

def create_model(num_classes):
    # Load MobileNetV2 backbone
    mobilenet = mobilenet_v2(pretrained=True)
    backbone = MobileNetBackbone(mobilenet)

    # RPN anchor generator
    anchor_generator = AnchorGenerator(
        sizes=((32, 64, 128, 256, 512),),  # 5 pyramid levels
        aspect_ratios=((0.5, 1.0, 2.0),)  # 1 aspect ratio tuple for each level
    )

    # RoI pooling
    roi_pooler = torchvision.ops.MultiScaleRoIAlign(
        featmap_names=["0"],  # Backbone feature map names (use the first level)
        output_size=7,        # Output size of the RoI Align
        sampling_ratio=2      # Sampling ratio for RoI Align
    )
```

```
    # Faster R-CNN model
    model = FasterRCNN(
        backbone=backbone,
        num_classes=num_classes,
        rpn_anchor_generator=anchor_generator,
        box_roi_pool=roi_pooler
    )
    return model

# Set device
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
print(device)
# Create the model
num_classes = 43  # Adjust based on your dataset
model = create_model(num_classes).to(device)

# Print model summary for verification
print(model)

# Training Function
def train_model(model, dataloader, num_epochs=5, learning_rate=0.005):
    model.train()
    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9,
weight_decay=0.0005)

    for epoch in range(num_epochs):
        epoch_loss = 0
```

```
        for images, targets in dataloader:
            # Move images and targets to device
            images = list(image.to(device) for image in images)
            targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
            # Forward pass
            loss_dict = model(images, targets)
            losses = sum(loss for loss in loss_dict.values() if not torch.isnan(loss))
            epoch_loss += losses.item()

            # Backward pass
            optimizer.zero_grad()
            losses.backward()
            optimizer.step()

        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}")

# Train the model
train_model(model, dataloader, num_epochs=25, learning_rate=0.005)

import numpy as np
def calculate_ap_per_class(pred_boxes, pred_scores, pred_labels, true_boxes, true_labels,
iou_threshold=0.5, num_classes=3):
    """
    Calculate the Average Precision (AP) for each class.
    """
    ap_per_class = []
```

```
    for cls in range(1, num_classes):  # Exclude background class (class 0)
        cls_pred_boxes = []  # Initialize as a list
        cls_pred_scores = []  # Initialize as a list
        cls_true_boxes = []  # Initialize as a list

        # Filter predictions and ground truth for the current class
        for p_boxes, p_scores, p_labels, t_boxes, t_labels in zip(pred_boxes, pred_scores, pred_labels, true_boxes, true_labels):
            cls_pred_boxes.append(p_boxes[p_labels == cls])  # Append NumPy arrays for current class
            cls_pred_scores.append(p_scores[p_labels == cls])
            cls_true_boxes.append(t_boxes[t_labels == cls])

        # Flatten lists
        cls_pred_boxes = np.concatenate(cls_pred_boxes, axis=0) if cls_pred_boxes else np.empty((0, 4))
        cls_pred_scores = np.concatenate(cls_pred_scores, axis=0) if cls_pred_scores else np.empty((0,))
        cls_true_boxes = np.concatenate(cls_true_boxes, axis=0) if cls_true_boxes else np.empty((0, 4))

        # Sort predictions by score (descending)
        sorted_indices = np.argsort(-cls_pred_scores)
        cls_pred_boxes = cls_pred_boxes[sorted_indices]
        cls_pred_scores = cls_pred_scores[sorted_indices]

        # Match predictions to ground truth
        true_positives = np.zeros(len(cls_pred_boxes))
        false_positives = np.zeros(len(cls_pred_boxes))
```

```
matched_gt = set()

for i, pred_box in enumerate(cls_pred_boxes):
    if len(cls_true_boxes) == 0:
        false_positives[i] = 1
        continue

    # Compute IoU with all ground truth boxes
    ious = box_iou(torch.tensor([pred_box]), torch.tensor(cls_true_boxes)).numpy()[0]

    # Find the best match (if IoU exceeds threshold)
    best_iou_idx = np.argmax(ious)
    best_iou = ious[best_iou_idx]

    if best_iou >= iou_threshold and best_iou_idx not in matched_gt:
        true_positives[i] = 1
        matched_gt.add(best_iou_idx)
    else:
        false_positives[i] = 1
# Calculate precision and recall
tp_cumsum = np.cumsum(true_positives)
fp_cumsum = np.cumsum(false_positives)
precisions = tp_cumsum / (tp_cumsum + fp_cumsum + 1e-7)
recalls = tp_cumsum / (len(cls_true_boxes) + 1e-7)
# Compute AP (area under the precision-recall curve)
ap = auc(recalls, precisions) if len(recalls) > 0 else 0
ap_per_class.append(ap)
```

```
    return ap_per_class
from torchvision.ops import box_iou
from sklearn.metrics import precision_recall_curve, auc
def test_model(model, dataloader, device, num_classes=3, iou_threshold=0.5):
    model.eval()  # Set the model to evaluation mode
    all_pred_boxes = []
    all_pred_labels = []
    all_pred_scores = []
    all_true_boxes = []
    all_true_labels = []
    with torch.no_grad():  # No gradients needed during testing
        for images, targets in dataloader:
            images = [image.to(device) for image in images]
            targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
            # Forward pass through the model
            outputs = model(images)
            # Collect predictions and ground truths for all images in the batch
            for output, target in zip(outputs, targets):
                all_pred_boxes.append(output['boxes'].cpu().numpy())
                all_pred_labels.append(output['labels'].cpu().numpy())
                all_pred_scores.append(output['scores'].cpu().numpy())
                all_true_boxes.append(target['boxes'].cpu().numpy())
                all_true_labels.append(target['labels'].cpu().numpy())
    # Calculate AP for each class
    ap_per_class = calculate_ap_per_class(
        all_pred_boxes, all_pred_scores, all_pred_labels,
```

```
        all_true_boxes, all_true_labels,
        iou_threshold, num_classes
    )
    # Mean Average Precision (mAP)
    mean_ap = np.mean(ap_per_class)
    print(f"AP per class: {ap_per_class}")
    print(f"Mean Average Precision (mAP) on the test set: {mean_ap:.4f}")
    return ap_per_class, mean_ap
# Example of how to use the testing function:
ap_per_class, mean_ap = test_model(model, test_dataloader, device, num_classes=43)
```

4 EXPERIMENTAL RESULTS AND ANALYSIS

4.1. Dataset Description

1. **Arbitrary Natural Images Dataset:**
   - **Source**: Arbitrary Natural Images Dataset

This dataset consists of synthetic and real-world images of traffic signs generated in various natural environments. It is used to train and evaluate the traffic sign detection model under diverse conditions. The dataset helps improve the model's generalization to different backgrounds, lighting, and weather conditions.

2. **German Traffic Sign Detection Benchmark (GTSDB)**
   - **Source**: German Traffic Sign Detection Benchmark

This is a widely used benchmark for traffic sign detection and classification. It contains images of traffic signs in urban environments with annotations. The dataset is used for training, validating, and testing models to ensure accurate and reliable traffic sign detection in real-world scenarios.

These datasets provide the necessary variety of images, both synthetic and real-world, for training and optimizing a traffic sign detection system that performs well under various conditions.

4.2. Implementation Details

The Traffic Sign Detection System is designed to detect and classify traffic signs in real-time with a primary focus on minimizing computational

time. Below are the implementation details for the key components of the system:

1. **Data Collection and Preparation**

- **Dataset**: The system uses a comprehensive dataset, including the German Traffic Sign Detection Benchmark (GTSDB) and synthetic datasets generated by overlaying traffic sign templates on various natural backgrounds. The dataset contains diverse traffic sign images under different lighting and weather conditions. These images are preprocessed to standardize their size and resolution, ensuring faster processing during both training and real-time inference.

2. **Traffic Sign Detection Model**

- **Model Selection**: A lightweight Convolutional Neural Network (CNN) architecture is chosen to ensure a balance between accuracy and reduced computational time. The model uses a streamlined architecture with fewer layers and operations, optimized for efficiency. Techniques such as depthwise separable convolutions, similar to those used in MobileNets, are employed to minimize the number of computations and reduce processing time while maintaining accuracy

3. **Optimization for Computational Efficiency**

- **Efficient Layer Configuration**: The network architecture is fine-tuned with efficient layer configurations, such as depthwise separable convolutions used in MobileNets, which significantly reduce the number of computations needed for detecting traffic signs.

This implementation is centered on reducing computational time while maintaining the necessary accuracy for real-time traffic sign detection, ensuring the system operates efficiently in resource-constrained environments, such as autonomous vehicles and intelligent traffic systems.

4.3. Results and Analysis

Category	Reasearch Paper	Experiment
Processor	Intel Xeon E5606 (2.13GHz), Intel Xeon E7-4850 v4 (2.10GHz)	Tesla V100-SXM2 GPU
GPU	NVIDIA TITAN Xp (12GB VRAM)	Tesla V100-SXM2 (2247MiB in use)
Compute Mode	TensorFlow (ResNet)	PyTorch(MobileNet-V2)
Training Duration	7 hours on single run	4 hours on single run

Table1: Performance Analysis

The research paper utilized Intel Xeon processors and an NVIDIA TITAN Xp GPU with TensorFlow and the ResNet architecture, taking 7 hours for a single training run. In contrast, the experimental setup leveraged a Tesla V100-SXM2 GPU and PyTorch with the MobileNet-V2 architecture, reducing training time to 4 hours per run. The significant reduction in training time highlights the efficiency of modern hardware, such as the Tesla V100, and the lightweight MobileNet-V2 model compared to the older TITAN Xp GPU and ResNet architecture. This demonstrates the impact of advanced technology and optimized model selection on improving computational performance in deep learningtasks.

Average Precision:

```
AP per class: [0.8717391649208829, 0.8509275955277777, 0.9424050217349786, 0.9358215436910312, 0.8451854736679073, 0.8981249071
514539, 0.9325712710242186, 0.9293488953771625, 0.9435530804372997, 0.9180661003184045, 0.8798491915326767, 0.9292087851992739,
0.9144377667182866, 0.9480136206392372, 0.9398053069713217, 0.951647302799626, 0.9387849231307992, 0.915735263804217, 0.9135920
001903736, 0.8653271463167135, 0.8805316811005316, 0.9320687210463745, 0.862007290672237, 0.8760276197846506, 0.91021224620812
9, 0.8990938138674207, 0.8619906339642823, 0.8305623987485101, 0.892757992402626, 0.883849513324589, 0.9199878082473948, 0.9638
37901634504, 0.9481836037080071, 0.9348104265237311, 0.9731130776828721, 0.9241461402061371, 0.9555994912322422, 0.960699557466
906, 0.9386550119975821, 0.9474305695694686, 0.9028067036164046, 0.9229823118326468]
Mean Average Precision (mAP) on the test set: 0.9139
```

The evaluation results showcase the performance of an object detection model using Average Precision (AP) per class and Mean Average Precision (mAP). The AP values, ranging from approximately 0.83 to 0.97, represent the model's precision and recall for each class, with higher scores indicating better performance. The overall mAP of 0.9139 demonstrates excellent detection and classification capabilities across all classes. However, slight variations in AP scores suggest the model performs better on certain classes (e.g., 0.9731, 0.9555) while struggling slightly with others (e.g., 0.8451, 0.8305), which may be due to class imbalances or complex features. These metrics indicate the model's strong overall performance but highlight areas for targeted improvement.

Results:


```
Predicted Boxes: [[ 45.34262 222.89995  68.95056 243.11215]
 [288.16815 179.86629 338.3264  221.30417]
 [287.50522 178.86432 338.33456 221.05814]]
Scores: [0.96681    0.69118667 0.37634426]
Labels: [27 28 27]
```

Fig 3: Output1


```
Predicted Boxes: [[74.28052  79.23609  95.41649  99.065674]]
Scores: [0.47602475]
Labels: [1]
```

Fig 4: Output2


```
Predicted Boxes: [[ 70.96984 288.7816  103.16606 319.30157]]
Scores: [0.73536134]
Labels: [37]
```

Fig 5: Output3

5 CONCLUSIONS AND FUTURE SCOPE

Conclusions

The Traffic Sign Detection System has been successfully developed with a primary focus on reducing computational time while maintaining detection accuracy. By using Convolutional Neural Networks (CNNs) optimized for efficiency, particularly the use of MobileNet and Faster R-CNN architectures, the system effectively minimizes inference time. Techniques such as model pruning, quantization, and efficient layer configurations were employed to reduce the model's complexity, making it suitable for deployment in resource-constrained environments.

The optimized model has shown reduced computational time without sacrificing accuracy, ensuring its applicability for embedded systems and environments with limited computational power.

Future Scope

Although the system has achieved notable success in reducing computational time, there is still potential for further improvements:

1. **Further Model Optimization**:

Additional exploration of advanced pruning techniques, the use of neural architecture search (NAS), and optimizing Faster R-CNN or MobileNet architectures can yield even faster models with reduced inference time and improved accuracy.

2. **Enhanced Data Preprocessing**:

Improvements in data preprocessing, such as selective cropping of image regions or dynamic resolution adjustment based on input complexity, could help further decrease computational time during both training and inference.

3. **Real-Time Adaptation and Continuous Learning**:

Implementing online learning techniques would allow the system to adapt to new traffic sign types or environmental changes in real time, improving long-term efficiency and performance without requiring retraining from scratch.

4. **Scalable Deployment**:

Implementing the model in a distributed system across multiple edge devices could enhance scalability, allowing the system to process data from multiple cameras or locations without a significant increase in computational time.

In conclusion, by focusing on reducing computational time, the Traffic Sign Detection System offers a robust, efficient solution for traffic management systems. With ongoing advancements in model optimization, dataset expansion, and edge device improvements, the system has the potential to be even more efficient, adaptable, and suitable for deployment in real-world applications like autonomous driving and smart traffic management.

6 REFERENCES

[1] Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., ... & De Souza, A. F. (2019). Self-Driving Cars: A Survey. arXiv preprint arXiv:1901.04407.

https://arxiv.org/abs/1901.04407

[2] Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6), 1137-1149.

https://doi.org/10.1109/TPAMI.2016.2577031

[3] Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2011). The German Traffic Sign Recognition Benchmark: A Multi-Class Classification Competition. In The 2011 International Joint Conference on Neural Networks (pp. 1453-1460). IEEE.

https://doi.org/10.1109/IJCNN.2011.6033395

[4] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.

https://arxiv.org/abs/1804.02767

[5] Ciresan, D., Meier, U., Masci, J., & Schmidhuber, J. (2012). Multi-Column Deep Neural Network for Traffic Sign Classification. Neural Networks, 32, 333-338.

https://doi.org/10.1016/j.neunet.2012.02.023

[6.] Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2017). Focal Loss for Dense Object Detection. In Proceedings of the IEEE International Conference on Computer Vision (ICCV) (pp. 2980-2988).

https://doi.org/10.1109/ICCV.2017.324

[7] Ming, Y., Wu, Y., Shen, J., & Xu, W. (2020). Real-Time Traffic Sign Detection and Recognition Based on Improved YOLO. IEEE Access, 8, 86719-86729.

https://doi.org/10.1109/ACCESS.2020.2992438

[8] Girshick, R. (2015). Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision (ICCV) (pp. 1440-1448).

https://doi.org/10.1109/ICCV.2015.169

[9] Zhang, W., Geiger, A., & Stiller, C. (2018). Object Detection for Autonomous Driving: Deep Learning vs. Model-Based Approaches. IEEE Transactions on Intelligent Transportation Systems, 19(6), 1845-1856.

https://doi.org/10.1109/TITS.2017.2756780

[10] Neuhold, G., Ollmann, T., Rota Bulo, S., & Kontschieder, P. (2017). The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes. In Proceedings of the IEEE International Conference on Computer Vision (ICCV) (pp. 4990-4999).

https://doi.org/10.1109/ICCV.2017.532

[11] Dai, J., Li, Y., He, K., & Sun, J. (2016). R-FCN: Object Detection via Region-Based Fully Convolutional Networks. In Advances in Neural Information Processing Systems (NeurIPS) (pp. 379-387).

https://proceedings.neurips.cc/paper_files/paper/2016/file/577ef1154f3240ad5b9b413aa7346a1e-Paper.pdf

[12] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4700-4708).

https://doi.org/10.1109/CVPR.2017.243

[13] Zhu, X., Cheng, D., Zhang, Z., Lin, S., & Dai, J. (2019). An Empirical Study of Spatial Attention Mechanisms in Deep Networks. IEEE Transactions on Neural Networks and Learning Systems, 31(10), 4245-4258.

https://doi.org/10.1109/TNNLS.2019.2930150

[14] Sermanet, P., LeCun, Y., Kavukcuoglu, K., & Chintala, S. (2013). Pedestrian Detection with Unsupervised Multi-Stage Feature Learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 3626-3633).

https://doi.org/10.1109/CVPR.2013.465

[15] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 2818-2826).

https://doi.org/10.1109/CVPR.2016.308