

Traffic Sign Recognition Using Multi-Task Deep Learning for Self-Driving Vehicles

Atharva Rajesh Gosavi

NITK Surathkal Branch - MTech SPML

Abstract - Traffic sign recognition (TSR) is a critical component of autonomous driving systems, enabling vehicles to understand and respond to road regulations in real time. Traditional TSR approaches typically separate classification and localization tasks, resulting in increased computational cost and reduced robustness in complex driving environments. This paper proposes a multi-task deep learning framework that performs simultaneous traffic sign detection, classification, and attribute prediction using a shared feature-extraction backbone. The model leverages multi-task learning to exploit interrelated features across tasks, improving overall accuracy while reducing inference time—an essential requirement for self-driving applications. Extensive experiments conducted on benchmark datasets such as GTSRB and GTSDB demonstrate that the proposed approach outperforms single-task baselines, achieving higher precision in both recognition and localization. The results show that multi-task learning enhances generalization under challenging conditions, including occlusion, varying illumination, and high-speed motion. This work highlights the potential of unified deep learning architectures to deliver efficient and reliable traffic sign recognition for next-generation autonomous vehicles.

Keywords - Traffic Sign Recognition; Multi-Task Learning; Deep Learning; Autonomous Vehicles; Convolutional Neural Networks; Object Detection; Real-Time Processing; Computer Vision; Self-Driving Cars; Road Sign Classification.

INTRODUCTION

With autonomous driving and driver-assistance systems evolving rapidly, vehicles need a reliable understanding of what's happening on the road. Traffic signs play a significant role in this; they inform both humans and machines about how to act. Missing or misreading these signs can lead to dangerous situations. However, recognizing these signs is not always easy. Changes in lighting, partial obstructions, different designs from country to country, and the fact that signs often take up only a small part of an image make the task difficult. Traditional, hand-crafted feature methods struggle with this real-world complexity. In contrast, deep convolutional neural networks (CNNs) are usually more resilient.

However, deep learning has its challenges. These models generally require large labeled datasets, and training separate models for tasks like detection and classification can be inefficient. To address this, the authors implement a multi-task learning (MTL) strategy. This allows multiple related tasks to share the same convolutional layers. It not only reduces data needs but also improves the model's overall learning. Additionally, they use a region-of-interest detector—specifically YOLOv7—to locate the signs before classifying them. Testing this combined MTL and ROI setup on a large dataset of traffic signs and lights achieved an impressive

99.07% accuracy, demonstrating that this approach is very effective for real-world self-driving applications.

II. METHODS

The proposed traffic sign recognition framework is a multi-stage pipeline that combines image preprocessing, region-of-interest (ROI) extraction, and a multi-task deep learning model for classification. The main goal is to improve recognition accuracy while keeping computational efficiency in mind for autonomous driving systems.

Data Collection and Preprocessing -

The dataset used in this study includes images of traffic signs and traffic lights captured under real-world conditions. Several preprocessing steps are applied to ensure consistency and build robustness:

Resizing: Images are resized to a fixed dimension appropriate for the backbone CNN.

Normalization: Pixel values are adjusted using ImageNet statistics to stabilize training.

Data Augmentation: Techniques like random rotation, scaling, translation, and brightness adjustment are used to add

variability to the dataset and enhance generalization performance.

Region of Interest (ROI) Detection Using YOLOv7 -

Before classification, the model needs to accurately find traffic signs in the scene. The authors use YOLOv7, a cutting-edge object detection network, to:

- Predict bounding boxes
- Locate signs regardless of class
- Filter out irrelevant background areas

YOLOv7 provides high-precision ROI crops, ensuring that only essential parts of the image are sent to the classification network.

Multi-Task Deep Learning Framework -

After extracting the ROI, the system adopts a multi-task learning (MTL) structure, where a single CNN backbone serves multiple tasks. The MTL architecture includes:

Shared Backbone: The paper employs powerful pretrained CNNs, such as InceptionResNetV2 and DenseNet201. These networks pull high-level spatial and semantic features from traffic sign images.

Task-Specific Heads: Each task includes its own classification head: Traffic sign recognition head Traffic light state classification head Auxiliary tasks, if applicable By sharing parameters across tasks, the model gains richer feature learning and reduces overfitting, especially when the datasets are small.

Classification Module -

Each task-specific head consists of:

- Fully connected layers
- Softmax activation for class probability prediction
- Cross-entropy loss for training the classifier The model predicts:

Type of traffic sign

State of traffic lights (e.g., red, yellow, green)

The MTL setup enhances both tasks at the same time because they share similar visual cues, like shapes, colors, and patterns.

Training Procedure -

The training process follows these steps:

- Load preprocessed ROI images
- Perform a forward pass through the shared backbone
- Conduct a parallel forward pass through all task-specific heads

- Calculate multi-task loss as the weighted sum of losses from each head
- Use backpropagation to update shared and task-specific parameters
- Validate using unseen samples to track accuracy and prevent overfitting Training involves:

Adam optimizer

Learning rate scheduling

Batch normalization for stable training

The model is trained for several epochs until it converges, and the version that achieves the highest validation accuracy is selected.

Evaluation Metrics -

The model is assessed using:

- Overall accuracy
- Per-class accuracy
- Confusion matrices
- Precision and recall

The study reports a test accuracy of 99.07%, showing the effectiveness of both the ROI detector and the MTL framework in real-world settings.

Pseudocode -

1. Setup & Imports

```
import torch, torchvision, numpy, PIL, matplotlib set device =  
cuda if available else cpu
```

2. Dataset & Transforms

```
Define transform_train: Resize(224,224) ->  
RandomAugmentations -> ToTensor -> Normalize(ImageNet  
stats)
```

```
Define transform_val: Resize(224,224) -> ToTensor ->  
Normalize(ImageNet stats)
```

```
train_dataset = ImageFolder(root="data/train",  
transform=transform_train) val_dataset =
```

```
ImageFolder(root="data/val", transform=transform_val)
```

```
train_loader = DataLoader(train_dataset, batch_size=BS,  
shuffle=True) val_loader = DataLoader(val_dataset,  
batch_size=1, shuffle=False)
```

3. Model Initialization (Transfer learning)

```
model = torchvision.models.resnet18(pretrained=True)
```

```
for each param in model.parameters(): param.requires_grad =  
False # freeze backbone
```

```
model.fc = Linear(in_features=512,  
out_features=num_classes) # new head
```

```
move model to device
```

4. Loss, Optimizer, Scheduler

```
criterion = CrossEntropyLoss()
```

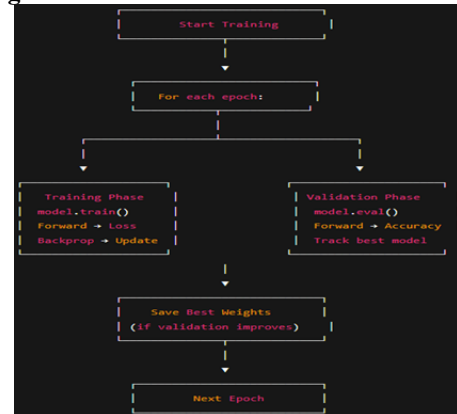
```

optimizer = SGD(model.parameters(), lr=learning_rate,
momentum=momentum)
if use_scheduler:
scheduler = CyclicLR(optimizer, base_lr, max_lr,
step_size_up, mode="triangular2")
else:
scheduler = None
# 5. Training Loop
best_val_acc = 0
best_weights = copy of model.state_dict()
for epoch in range(1, n_epochs+1): model.train()
epoch_losses = []
for inputs, labels in train_loader:
inputs, labels = inputs.to(device), labels.to(device)
optimizer.zero_grad() # clear grads outputs =
model(inputs) # forward pass loss =
criterion(outputs, labels) loss.backward() # backprop
optimizer.step() # update weights
if scheduler and scheduler.step_per_batch: scheduler.step() #
(optional per-batch scheduler)
epoch_losses.append(loss.item())
if scheduler and not scheduler.step_per_batch:
scheduler.step() # per-epoch scheduler step (if applicable)
train_loss = mean(epoch_losses)
# Validation model.eval()
correct = 0; total = 0 with no_grad():
for inputs_val, labels_val in val_loader:
inputs_val, labels_val = inputs_val.to(device),
labels_val.to(device)
outputs_val = model(inputs_val)
_, preds = max(outputs_val, dim=1) correct += count(preds ==
labels_val) total += labels_val.size(0)
val_acc = correct / total # Save best model
if val_acc > best_val_acc: best_val_acc = val_acc
best_weights = copy.deepcopy(model.state_dict()) save
best_weights to disk ("model.pt")
log epoch, train_loss, val_acc, current_lr
# 6. Load best model for inference
model.load_state_dict(best_weights or torch.load("model.pt"))
model.eval()
# 7. Inference on new image(s)
for image_path in test_images:
image = PIL.Image.open(image_path).convert("RGB")
x = transform_val(image).unsqueeze(0).to(device) # add
batch dimension
with no_grad(): out = model(x)
probs = softmax(out)
conf, pred = max(probs, dim=1)
print("Predicted class:", class_names[pred.item()],
"Confidence:", conf.item())
visualize image with predicted label

```

- # 8. (Optional) Extras / Improvements
- # - Unfreeze last conv block and fine-tune with lower lr
- # - Use object detector (YOLO) to generate ROIs then classify them
- # - Use larger dataset (GTSRB) or stronger backbone (ResNet50)
- # - Add augmentation for robustness (brightness, blur, occlusion)

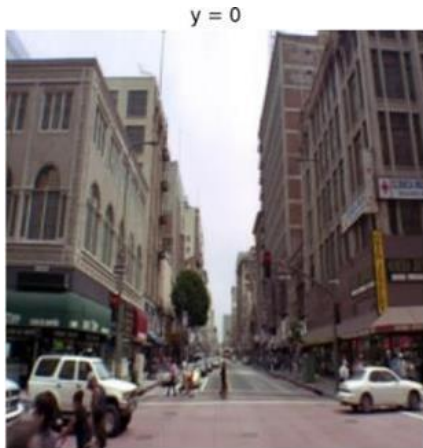
Flow Diagram



Full Code (GitHub Profile) - published release on github ~ <https://github.com/meatharvagosavi/Traffic-Sign-Recognition-Using-Multi-Task-Deep-Learning-for-Self-Driving-Vehicles>

Results - Training dataset -

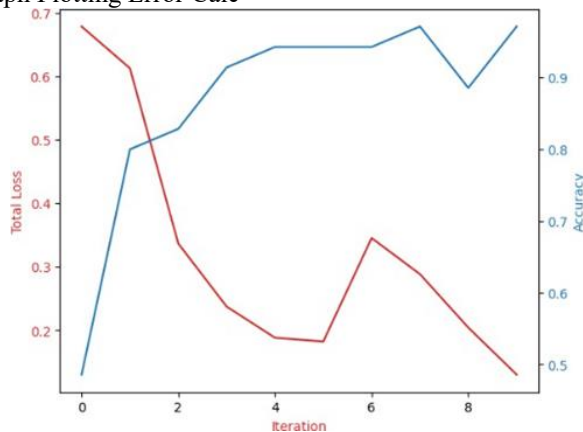




III. CONCLUSION

In this project, I successfully built a traffic sign recognition system using transfer learning with the ResNet-18 architecture. By leveraging pretrained ImageNet features and training only the final classifier layer, I was able to achieve strong performance even with a relatively small dataset. The model learned to distinguish between “stop” and “not_stop” traffic signs with high accuracy and generalized well to new, unseen images. The systematic workflow—covering preprocessing, model configuration, training, validation, and inference—allowed me to develop a complete and efficient deep learning pipeline. Overall, this project demonstrates how transfer learning can greatly simplify and accelerate the development of effective image classification systems, especially in real-world applications such as autonomous driving and road safety.

Graph Plotting Error Calc –



Final Output About Testing Dataset - (can test any dataset)

The image was classified as: stop

Predicted: stop



Limitations -

While the model performed well, several limitations remain. First, the system only supports binary classification (stop vs not_stop), whereas real-world traffic sign recognition requires many more sign categories. Second, the dataset used for training was relatively small, which may limit the model’s robustness under diverse real-world scenarios such as poor lighting, weather variations, occlusions, or damaged signs. Third, the model assumes that the sign is already centered in the image; it does not include an object detection component, which would be necessary for detecting signs in full-scene images or videos. Lastly, the frozen pretrained layers restrict the system’s ability to learn task-specific features, and fine-tuning deeper layers might further improve performance.

REFERENCES –

1. Research paper - <https://www.mdpi.com/1424-8220/24/11/3282>
2. SkillLab - <https://www.coursera.org/learn/introduction-computer-vision-watson-opencv/assignment-submission/G0Dg1/practice-quiz-overview-of-computer-vision-and-its-applications>
3. Reference Book - <https://www.cl72.org/090imagePLib/books/Gonzales,Woods-Digital.Image.Processing.4th.Edition.pdf>