Fine-Tuning YOLOv8 for Small Object Detection in Traffic Monitoring

Gaurav Patil
012629189
CSCI611 - Spring 2025

March 19, 2025

GitHub Repository for Submission

The complete project, including source code, trained models, and examples, is available on GitHub:

GitHub Repository - Assignment 3

1 Introduction

Object detection is a fundamental task in computer vision, playing a crucial role in areas such as autonomous driving, traffic surveillance, and smart city development. While state-of-the-art models like YOLOv8 excel at detecting large objects such as vehicles and pedestrians, they often struggle with **small and distant objects** like traffic lights, stop signs, and speed limit signs.

This project investigates the effectiveness of fine-tuning YOLOv8 for small object detection, specifically in the context of traffic monitoring. The objective is to enhance the model's performance in detecting traffic signs while analyzing the trade-offs between **specialization and generalization** when training on a **new, limited dataset**.

2 Dataset and Preprocessing

2.1 Dataset Source

For this experiment, I used the **Self-Driving Car dataset** from Roboflow [1], which contains a variety of **traffic signs, signals, and road mark-**

ers. The dataset was downloaded and extracted using the following setup in Google Colab:

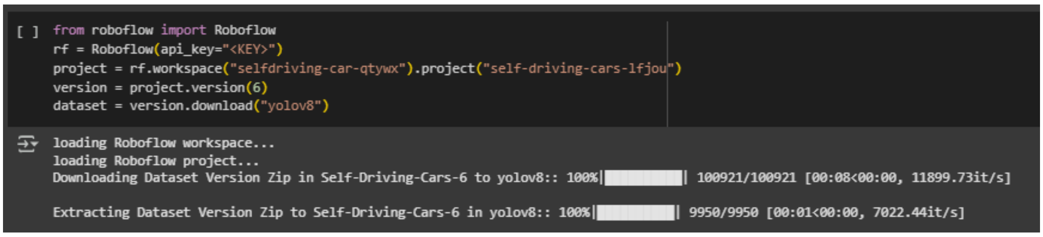

Figure 1: Downloading and extracting the dataset in Google Colab.

2.2 Data Preprocessing

The dataset was carefully prepared to ensure compatibility with YOLOv8's training pipeline. The key preprocessing steps included:

- **Image resizing:** Adjusted to 1280x1280 pixels for improved detection of small objects.
- **Data augmentation:** Applied techniques such as flipping, scaling, rotation, and blurring to improve generalization.
- **Annotation verification:** Checked and converted bounding box labels to YOLO format.
- **Class mapping:** Ensured that class names matched expected model inputs.

3 Training Configuration and Constraints

3.1 Training Setup

The model was trained on **Google Colab** using a Tesla T4 GPU. The following image shows the actual training setup, including hyperparameters, validation results, and final accuracy:

```python
from ultralytics import YOLO
import os
# Load pre-trained YOLOv8 model
model = YOLO("yolov8n.pt")  # You can also use yolov8s.pt, yolov8m.pt, etc. for different sizes
# Train the model
model.train(
    data=data_yaml,           # Path to the dataset YAML file
    epochs=20,                # Number of training epochs
    imgsz=1280,                # Image size (increase if small objects are hard to detect)
    batch=16,                 # Batch size (adjust based on your GPU memory)
    workers=4,                 # Number of workers for data loading
    lr0=0.01,                  # Initial learning rate
    weight_decay=0.0005,       # Regularization to prevent overfitting
    optimizer="SGD",           # SGD is better for object detection
    momentum=0.937            # Helps stabilize SGD updates
)
# Evaluate the model performance
metrics = model.val()
print(metrics)
# Export the trained model
model.export(format="onnx")  # Export to ONNX format (or use 'pt', 'tflite', etc.)
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      20/20      10.1G     0.8082     0.5661     0.9496         14       1280: 100%|██████████| 221/221 [03:36<00:00,  1.02it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 26/26 [01:17<00:00,  2.99s/it]


20 epochs completed in 1.443 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.4MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.4MB

Validating runs/detect/train/weights/best.pt...
Ultralytics 8.3.92 🚀 Python-3.11.11 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
Model summary (fused): 72 layers, 3,008,573 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 26/26 [01:13<00:00,  2.84s/it]
                   all        801        944      0.932      0.899      0.959      0.804
           Green Light         87        122      0.838      0.836      0.872      0.501
             Red Light         74        108      0.799      0.778       0.81      0.493
       Speed Limit 100         52         52      0.948      0.942      0.979      0.883
       Speed Limit 110         17         17      0.774          1      0.987      0.858
       Speed Limit 120         60         60      0.983      0.917      0.991      0.884
        Speed Limit 20         56         56      0.981       0.91       0.98        0.8
        Speed Limit 30         71         74      0.935      0.959      0.987      0.895
        Speed Limit 40         53         55      0.954      0.927      0.974      0.868
        Speed Limit 50         68         71        0.9      0.845      0.957      0.821
        Speed Limit 60         76         76       0.99      0.921      0.959      0.855
        Speed Limit 70         78         78      0.997      0.962       0.99       0.88
        Speed Limit 80         56         56      0.962      0.916      0.988      0.861
        Speed Limit 90         38         38          1      0.691      0.959      0.785
                  Stop         81         81      0.979      0.988      0.992       0.87
```

Figure 2: Training YOLOv8 model in Google Colab.

3.2 Training Constraints and Accuracy Limitations

This project was conducted on **Google Colab**, which provides limited access to GPU resources. The constraints included:

- **Limited Training Epochs:** Due to hardware restrictions, the model was only trained for 20 epochs. A longer training duration could have significantly improved accuracy.

- **GPU Memory Limitations:** Batch sizes had to be reduced to prevent out-of-memory errors.

- **Restricted Training Duration:** Colab imposes time limits on GPU usage, restricting extensive hyperparameter tuning.

- **Limited Dataset Size:** Since the dataset used for fine-tuning contained only traffic signs and not other objects such as vehicles, the model lost its ability to detect previously recognized objects.

These constraints resulted in a model that, while improved for small object detection, had limited generalization capability. Training on a **larger dataset** with **higher computational resources** would have likely improved accuracy and robustness.

4 Challenges and Observations

4.1 Loss of General Object Detection

A key issue observed after fine-tuning was that the model **stopped detecting vehicles and other previously recognized objects**. This occurred due to:

- **Catastrophic Forgetting:** The new dataset contained only traffic signs and signals, causing YOLOv8 to gradually "forget" how to detect vehicles.
- **Class Overwriting:** YOLO relies on dataset annotations to determine what objects exist. Since vehicles were absent in the fine-tuning dataset, the model stopped recognizing them.
- **Limited Class Set:** Without training on both old and new classes, YOLOv8 restructured its detection head to focus only on newly learned objects.

4.2 Stop Sign Detection - Pre-trained vs Fine-tuned Model

The stop sign detection results before and after fine-tuning are shown below:

Figure 3: Stop sign detection using the pre-trained YOLOv8 model.

Figure 4: Stop sign detection using the fine-tuned YOLOv8 model.

The fine-tuned model demonstrated improved confidence in detecting traffic signs, but it no longer recognized other objects such as vehicles.

4.3 Processed Model and Video Links

The trained models and processed videos can be accessed here:

Google Drive Link for Models and Processed Videos

5 Evaluation and Results

5.1 Training Loss and Metrics

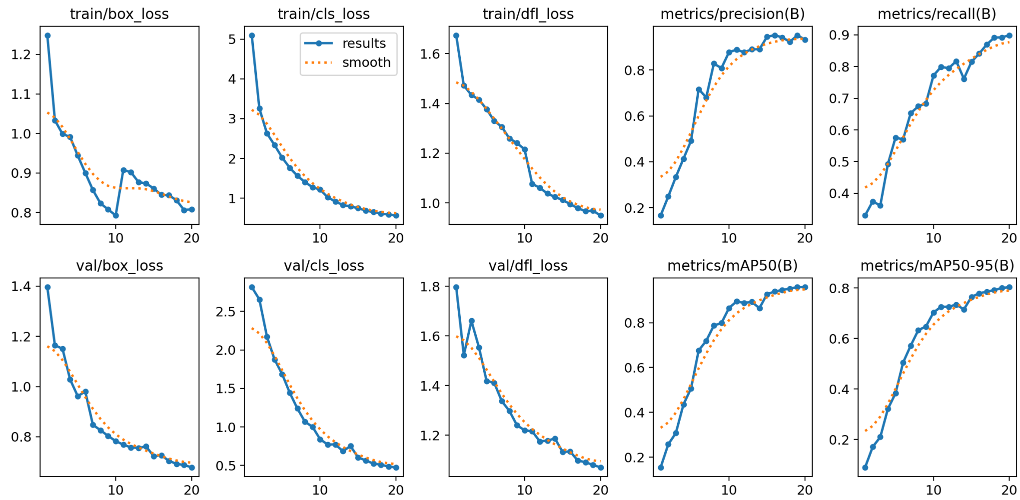

Figure 5: Training Loss and Performance Metrics

5.2 Confusion Matrices

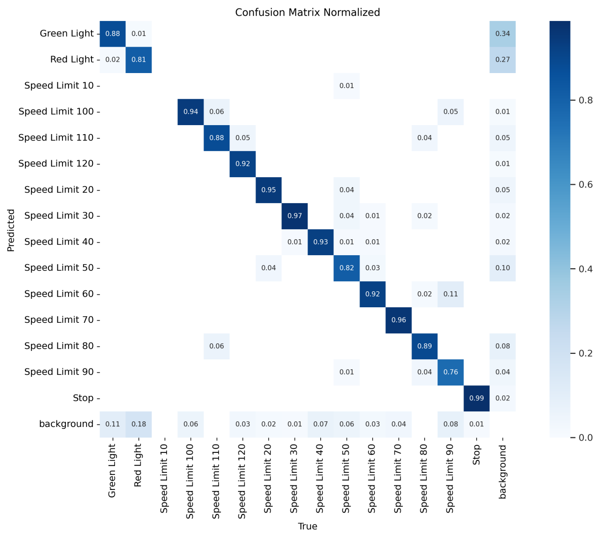

Figure 6: Normalized Confusion Matrix

5.3 Precision-Recall and Confidence Curves

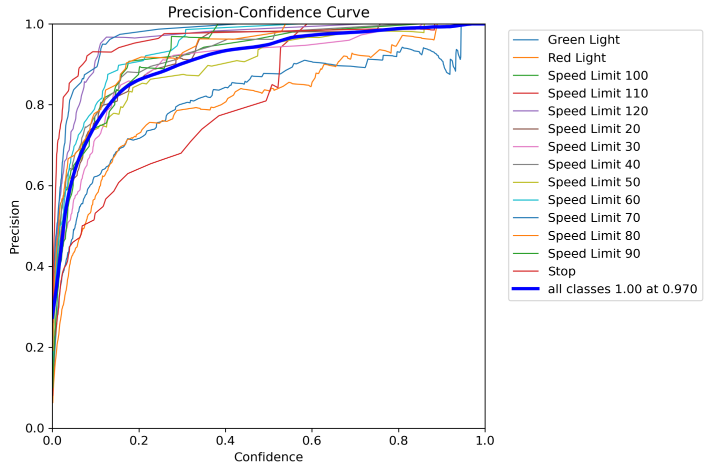

Figure 7: Precision-Confidence Curve

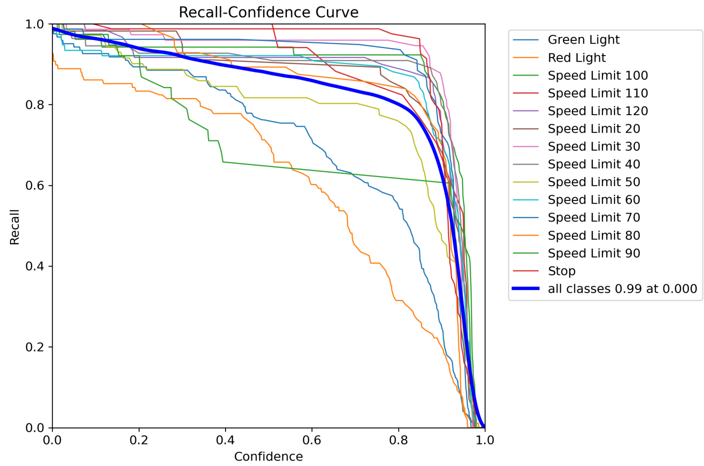

Figure 8: Recall-Confidence Curve

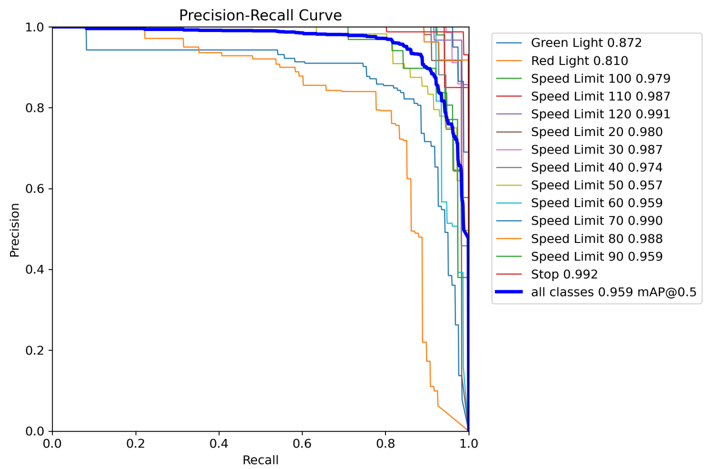

Figure 9: Precision-Recall Curve

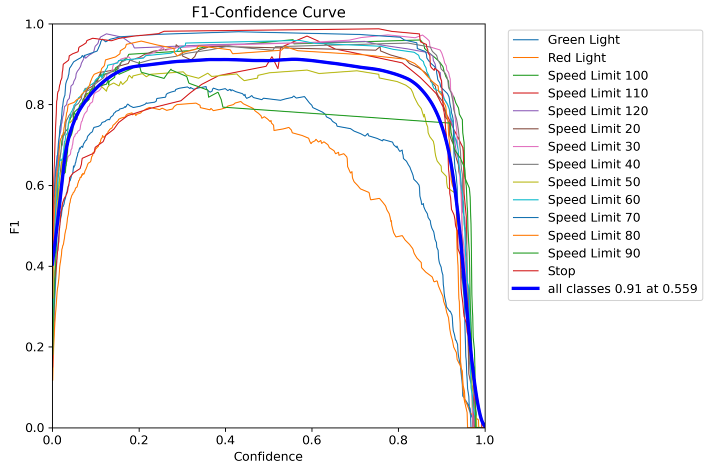

Figure 10: F1-Confidence Curve

6 Conclusion and Future Work

This project highlights the benefits and trade-offs of fine-tuning YOLOv8 for small object detection in traffic environments. While detection accuracy for **traffic lights and signs** improved, the model lost its ability to detect larger objects such as vehicles.

Key Takeaways:

- **Fine-tuning improved small object detection** but caused the model to forget previously learned classes.
- **Higher resolution training** (1280x1280) enhanced small object visibility.
- **Limited epochs and dataset size** restricted accuracy improvements and generalization.

Future Work:

- **Integrate a Multi-Task Dataset:** Training on both small and large objects will help maintain general detection.
- **Increase Training Duration:** Running for 50+ epochs with better hardware will likely improve accuracy.
- **Use Transfer Learning:** Freezing early layers and using knowledge distillation can mitigate catastrophic forgetting.

- **Optimize Video Processing:** Explore more efficient encoding techniques to reduce output file size.

References

[1] Roboflow Self-Driving Car Dataset. Available: https://universe.roboflow.com/selfdriving-car-qtywx/self-driving-cars-lfjou/dataset/6