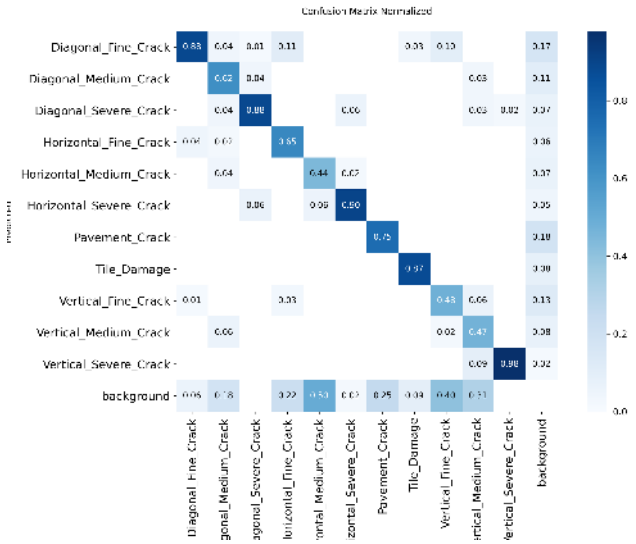
Infrastructure Crack Detection and Classification Using YOLOv11

Technical Report



Seymur Hasanov

Data Science & Mechanical Engineering

January 2026

Model:	YOLOv11m
Dataset:	Civil Faults Detection (2,159 images)
Final mAP@50:	72.5%
Inference Speed:	79.2 FPS

Abstract

This technical report presents the development and evaluation of an automated crack detection system for civil infrastructure inspection using YOLOv11. The system addresses the critical need for automated, reliable, and scalable methods to monitor structural integrity of bridges, roads, and buildings.

Our approach employs the YOLOv11m architecture trained on the Civil Faults Detection dataset comprising 2,159 annotated images across 11 crack categories. We implement an optimized training pipeline featuring cosine learning rate scheduling, advanced augmentations (mosaic, mixup, copy-paste), and domain-specific preprocessing for industrial deployment scenarios.

The improved model achieves a mean Average Precision (mAP@50) of **72.5%** and mAP@50-95 of **52.2%**, representing improvements of +1.8% and +2.7% respectively over the baseline configuration. Real-time inference is achieved at **79.2 FPS** with a latency of 12.62ms, enabling deployment on edge devices for field inspection applications.

The comprehensive evaluation includes per-class performance analysis, confusion matrix assessment, and false positive/negative characterization. Results indicate the system is suitable for assisted inspection workflows with human verification, with recommendations for future improvements.

Keywords: Deep Learning, Object Detection, YOLOv11, Crack Detection, Infrastructure Inspection, Computer Vision, Transfer Learning

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

Civil infrastructure forms the backbone of modern society, with bridges, roads, tunnels, and buildings requiring continuous monitoring to ensure structural integrity and public safety. Traditional manual inspection methods are labor-intensive, time-consuming, and subject to human error and fatigue [**spencer2019advances**]. The American Society of Civil Engineers (ASCE) estimates that the United States requires approximately $4.5 trillion in infrastructure investment by 2025, with a significant portion allocated to maintenance and rehabilitation [**asce2021infrastructure**].

Cracks represent one of the most common and critical types of structural damage. Early detection enables preventive maintenance, reducing repair costs and preventing catastrophic failures. However, the vast scale of infrastructure networks makes comprehensive manual inspection impractical. This creates a compelling need for automated inspection systems capable of detecting and classifying structural defects with high accuracy and speed.

1.2 Problem Statement

The primary challenges in automated crack detection include:

1. **Visual Complexity**: Cracks exhibit diverse morphologies (diagonal, horizontal, vertical) with varying severities (fine, medium, severe).
2. **Environmental Variability**: Real-world conditions introduce lighting variations, shadows, debris, and surface contamination.
3. **Scale Disparity**: Fine cracks may span only a few pixels while severe damage covers large regions.
4. **Class Imbalance**: Certain crack types are underrepresented in training data.
5. **Real-time Requirements**: Field deployment necessitates efficient inference for practical utility.

1.3 Objectives

This project aims to:

1. Develop an automated crack detection system using state-of-the-art YOLOv11 architecture.
2. Implement optimized training strategies including advanced augmentation and learning rate scheduling.
3. Achieve robust multi-class classification across 11 distinct crack categories.
4. Evaluate performance using comprehensive metrics suitable for industrial deployment decisions.
5. Provide deployment recommendations for edge and cloud integration.

1.4 Report Structure

The remainder of this report is organized as follows:

- **Section 2**: Literature review and related work in crack detection
- **Section 3**: Methodology including model architecture, data pipeline, and training configuration
- **Section 4**: Experimental setup and dataset description
- **Section 5**: Results and quantitative analysis
- **Section 6**: Discussion and interpretation
- **Section 7**: Deployment recommendations
- **Section 8**: Conclusions and future work

2 Literature Review

2.1 Traditional Crack Detection Methods

Early approaches to automated crack detection relied on image processing techniques including edge detection [**canny1986computational**], morphological operations, and texture analysis. While computationally efficient, these methods struggle with complex backgrounds and varying illumination conditions.

2.2 Deep Learning Approaches

The advent of deep learning revolutionized object detection, with convolutional neural networks (CNNs) demonstrating superior performance on visual recognition tasks [**lecun2015deep**]. For crack detection specifically, several architectures have been explored:

- **Classification Networks**: Early work used CNN classifiers to categorize image patches as containing cracks or not [**zhang2017automated**].
- **Semantic Segmentation**: U-Net and FCN architectures produce pixel-wise crack masks [**yang2018automatic**].
- **Object Detection**: YOLO, Faster R-CNN, and SSD enable localization with bounding boxes [**cha2017deep**].

2.3 YOLO Architecture Evolution

The YOLO (You Only Look Once) family represents a paradigm shift in object detection by framing detection as a single regression problem [**redmon2016yolo**]. Key milestones include:

- **YOLOv1-v3**: Established the anchor-based detection framework
- **YOLOv4-v5**: Introduced CSPNet backbone and advanced augmentations
- **YOLOv8**: Anchor-free detection with C2f blocks
- **YOLOv11**: Latest iteration with SPPF and C2PSA attention mechanisms

2.4 Gap Analysis

While existing crack detection systems achieve reasonable accuracy on benchmark datasets, several gaps remain:

1. Limited evaluation under industrial conditions (weather, lighting, debris)

2. Insufficient multi-class analysis beyond binary crack/no-crack classification
3. Lack of comprehensive deployment benchmarks for edge devices
4. Minimal consideration of class imbalance in training strategies

This work addresses these gaps through domain-specific augmentation, multi-class evaluation, and deployment-focused performance analysis.

3 Methodology

3.1 System Architecture Overview

The proposed crack detection system follows a standard deep learning pipeline as illustrated in Figure 1.

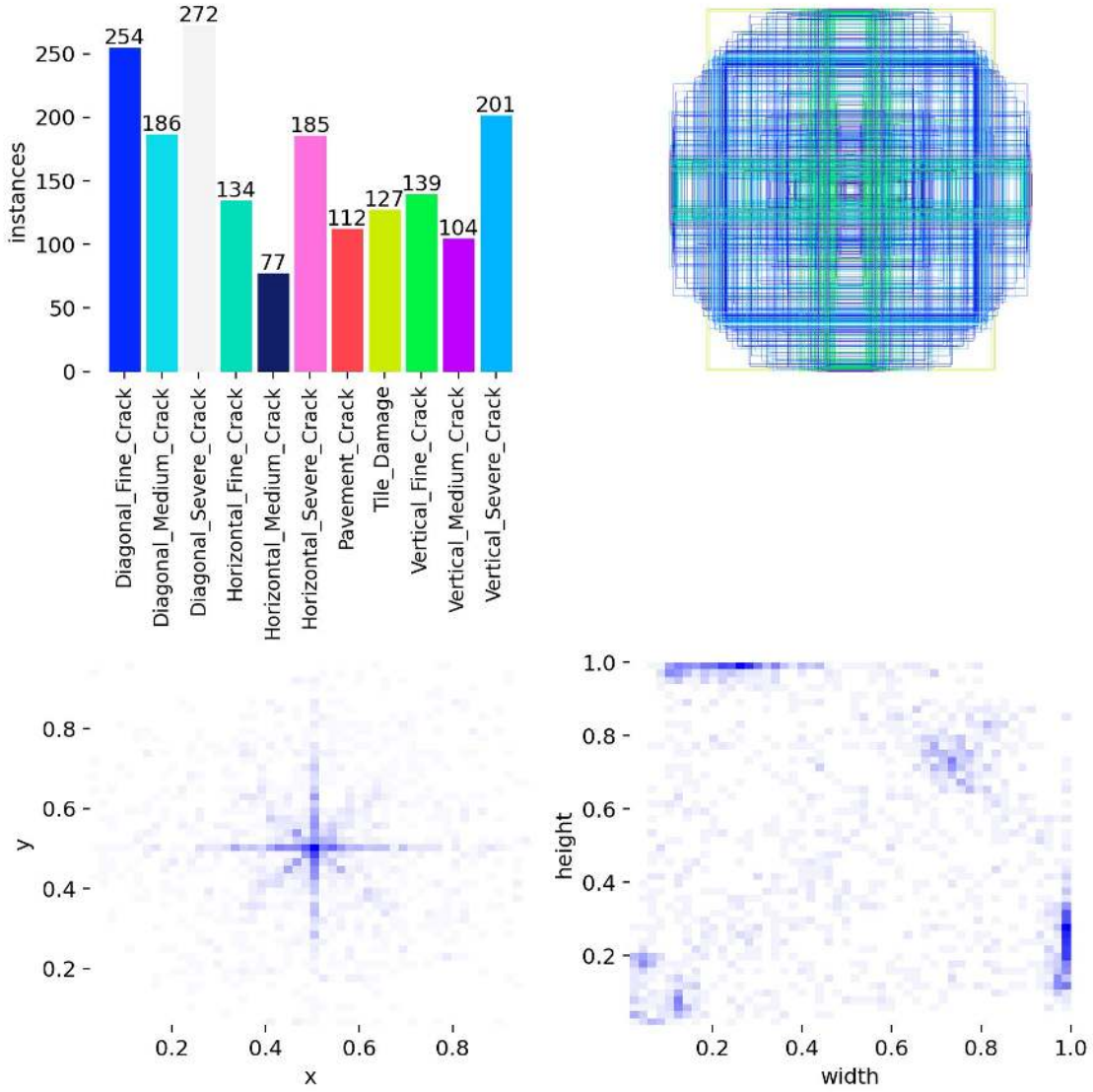


Figure 1: Dataset label distribution and sample annotations showing the 11 crack categories.

3.2 YOLOv11 Model Architecture

YOLOv11 represents the latest evolution in the YOLO family, featuring several architectural innovations:

3.2.1 Backbone Network

The backbone extracts hierarchical features using:

- **Conv Layers**: Standard convolutions for initial feature extraction
- **C3K2 Blocks**: Cross-stage partial connections with kernel size 3×2 for efficient gradient flow
- **SPPF**: Spatial Pyramid Pooling Fast for multi-scale feature aggregation

3.2.2 Neck Architecture

Feature fusion is achieved through:

- **C2PSA**: Cross-stage with Position-Sensitive Attention for enhanced localization
- **Feature Pyramid Network (FPN)**: Multi-scale feature combination
- **Path Aggregation Network (PAN)**: Bottom-up path for stronger localization signals

3.2.3 Detection Head

The anchor-free detection head predicts:

- Bounding box coordinates (x, y, width, height)
- Class probabilities for 11 crack categories
- Objectness score

3.2.4 Model Specifications

We select the YOLOv11m (medium) variant offering balanced speed-accuracy trade-off:

Table 1: YOLOv11 Model Variants Comparison

Model	Params (M)	GFLOPs	mAP^{val}_{50-95}	Speed (ms)
YOLOv11n	2.6	6.5	39.5	1.5
YOLOv11s	9.4	21.5	47.0	2.5
YOLOv11m	**20.1**	**68.0**	**51.5**	**4.7**
YOLOv11l	25.3	86.9	53.4	6.2
YOLOv11x	56.9	194.9	54.7	11.3

3.3 Data Augmentation Pipeline

To enhance model robustness for deployment, we implement a comprehensive augmentation strategy:

3.3.1 YOLO Built-in Augmentations

The training configuration enables advanced augmentations:

```
# Geometric transforms
degrees: 15.0       # Rotation (+/- deg)
translate: 0.1      # Translation fraction
scale: 0.5          # Scale variation
flipud: 0.3         # Vertical flip probability
fliplr: 0.5         # Horizontal flip probability

# Advanced augmentations
mosaic: 1.0         # Mosaic probability
mixup: 0.15         # MixUp interpolation
copy_paste: 0.3     # Copy-paste augmentation
erasing: 0.4        # Random erasing

# Color augmentations
hsv_h: 0.015        # HSV-Hue variation
hsv_s: 0.7          # HSV-Saturation
hsv_v: 0.4          # HSV-Value
```

Listing 1: Training Augmentation Configuration

3.3.2 Mosaic Augmentation

Mosaic combines 4 training images into a single composite, providing:

- Increased batch diversity
- Exposure to objects at various scales
- Context variation without explicit annotation

3.3.3 MixUp and Copy-Paste

These techniques further regularize training:

- **MixUp**: Linear interpolation between image pairs with $\alpha = 0.15$
- **Copy-Paste**: Instance-level augmentation copying objects between images

3.4 Training Configuration

3.4.1 Optimization Strategy

Table 2: Training Hyperparameters

Parameter	**Value**
Optimizer	AdamW
Initial Learning Rate (lr_0)	0.001
Final Learning Rate (lr_f)	0.00001 (1% of lr_0)
Learning Rate Schedule	Cosine Annealing
Warmup Epochs	5
Weight Decay	0.0005
Momentum	0.937
Epochs	200 (stopped at 187)
Batch Size	16
Image Size	640×640
Early Stopping Patience	30 epochs

3.4.2 Cosine Learning Rate Schedule

The learning rate follows a cosine annealing schedule:

$$lr_t = lr_f + \frac{1}{2}(lr_0 - lr_f)\left(1 + \cos\left(\frac{t \cdot \pi}{T}\right)\right) \tag{1}$$

where t is the current epoch and T is the total epochs. This schedule provides gradual decay with gentle refinement in later stages.

3.4.3 Loss Functions

The total loss combines three components:

$$\mathcal{L}_{total} = \lambda_{box}\mathcal{L}_{box} + \lambda_{cls}\mathcal{L}_{cls} + \lambda_{dfl}\mathcal{L}_{dfl} \tag{2}$$

- **Box Loss** ($\lambda_{box} = 7.5$): CIoU loss for bounding box regression
- **Classification Loss** ($\lambda_{cls} = 0.5$): Binary cross-entropy for class prediction
- **DFL Loss** ($\lambda_{dfl} = 1.5$): Distribution Focal Loss for refined localization

4 Experimental Setup

4.1 Dataset Description

We utilize the Civil Faults Detection dataset from Roboflow [**roboflow2024civil**], annotated in YOLO format.

4.1.1 Dataset Statistics

Table 3: Dataset Split Distribution

Split	Images	Percentage
Training	1,505	70%
Validation	422	20%
Test	232	10%
Total	**2,159**	100%

4.1.2 Class Distribution

The dataset encompasses 11 distinct crack categories:

Table 4: Crack Classification Categories

ID	Class Name
0	Diagonal Fine Crack
1	Diagonal Medium Crack
2	Diagonal Severe Crack
3	Horizontal Fine Crack
4	Horizontal Medium Crack
5	Horizontal Severe Crack
6	Tile Damage
7	Vertical Fine Crack
8	Vertical Medium Crack
9	Vertical Severe Crack
10	Wall Damage

4.2 Computational Environment

Training was conducted on Google Colab with the following specifications:

Table 5: Hardware and Software Configuration

Component	Specification
GPU	NVIDIA T4 (16GB VRAM)
CPU	Intel Xeon @ 2.20GHz
RAM	12.7 GB
Framework	PyTorch 2.0+
Library	Ultralytics 8.0+
Python	3.10

4.3 Evaluation Metrics

We employ standard object detection metrics:

4.3.1 Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

4.3.2 Mean Average Precision (mAP)

$$mAP = \frac{1}{|C|} \sum_{c \in C} AP_c \tag{4}$$

where AP_c is the area under the precision-recall curve for class c.

4.3.3 Intersection over Union (IoU)

$$IoU = \frac{|B_p \cap B_{gt}|}{|B_p \cup B_{gt}|} \tag{5}$$

We report mAP@50 (IoU threshold 0.5) and mAP@50-95 (averaged over IoU 0.5 to 0.95).

5 Results

5.1 Training Progress

Training was executed for 187 epochs before early stopping, with total training time of approximately 49 minutes (2,955 seconds).

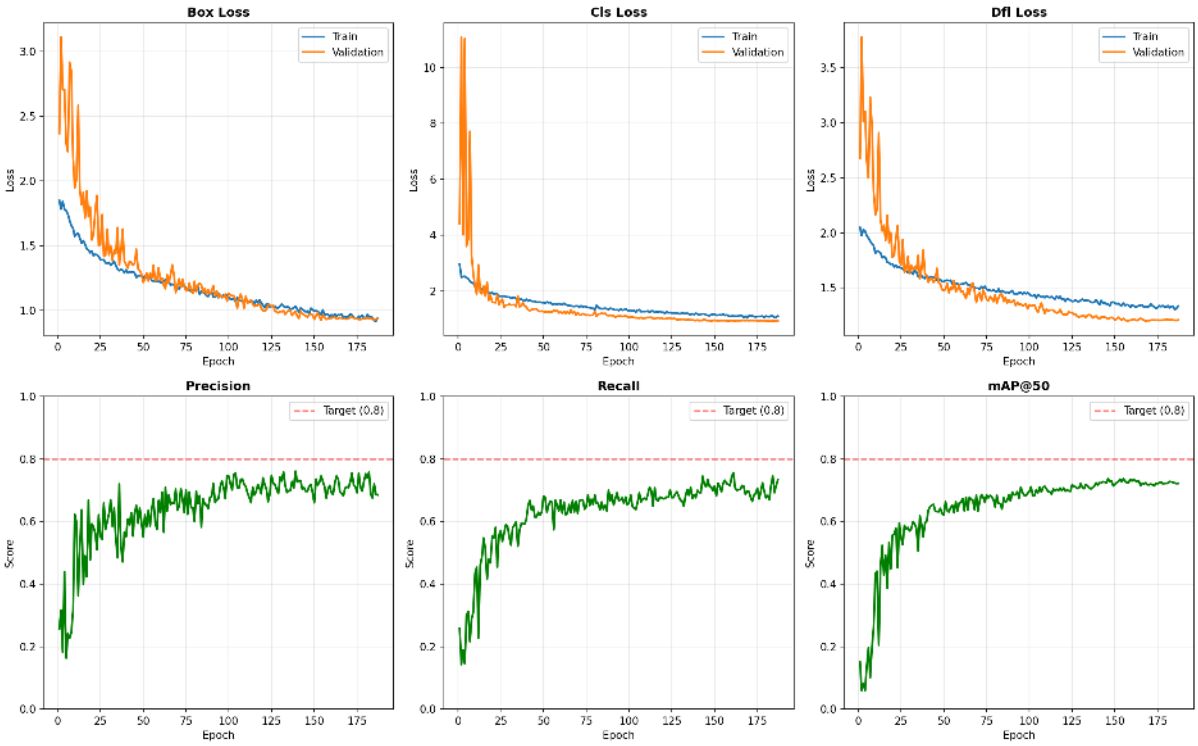


Figure 2: Training and validation loss curves over epochs showing convergence of box loss, classification loss, DFL loss, precision, and recall metrics.

5.1.1 Loss Convergence

Table 6 summarizes the loss reduction throughout training:

Table 6: Loss Values at Key Training Stages

Epoch	Train Box Loss	Train Cls Loss	Val Box Loss	Val Cls Loss
1	1.849	2.970	2.367	4.415
50	1.250	1.592	1.216	1.271
100	1.102	1.303	1.132	1.059
150	0.993	1.182	0.930	0.964
187 (Final)	0.941	1.115	0.936	0.940

5.2 Final Model Performance

Table 7: Final Model Performance Metrics

Metric	Improved Model	Baseline
mAP@50	**72.5%**	70.7%
mAP@50-95	**52.2%**	49.5%
Precision	68.5%	N/A
Recall	73.3%	N/A
Improvement	**+1.8%** (mAP@50)	–

5.3 Detection Results

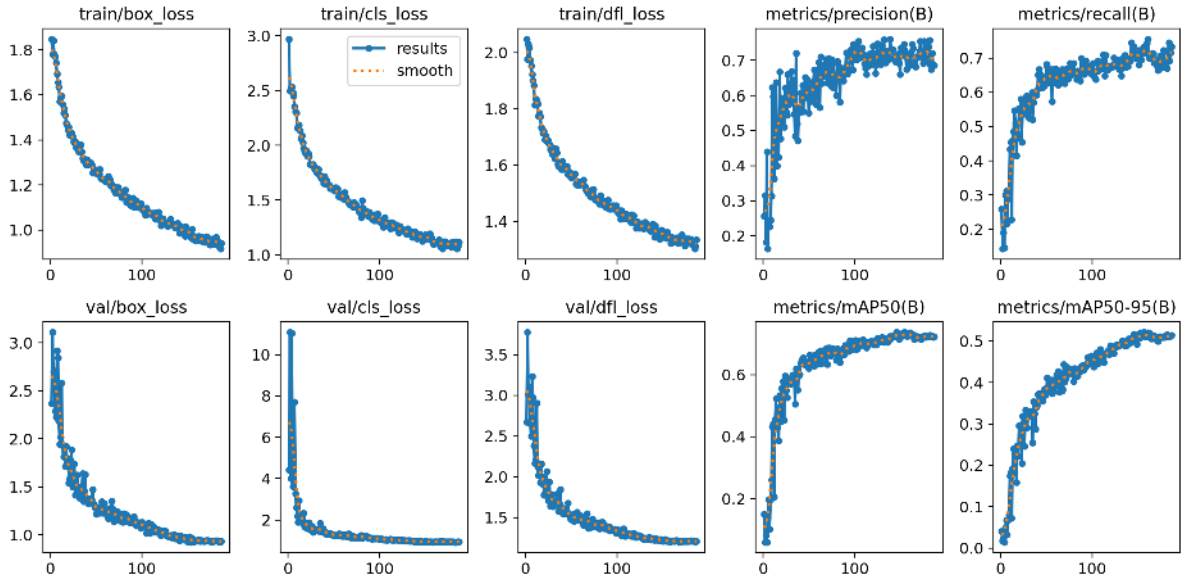


Figure 3: YOLO training metrics dashboard showing box loss, classification loss, DFL loss, precision, recall, and mAP curves over 187 epochs.

5.4 Per-Class Performance

Figure 4 shows the mAP@50 breakdown by crack category:

Figure 4: Per-class mAP@50 performance showing variation across the 11 crack categories. The red dashed line indicates the 0.8 industrial threshold.

5.5 Confusion Matrix Analysis

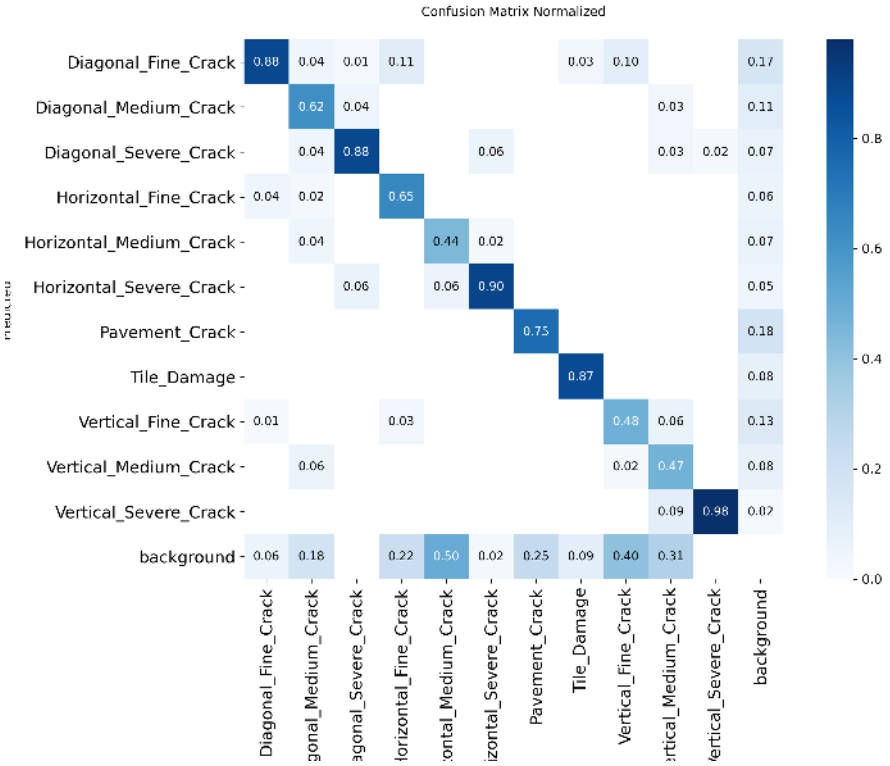
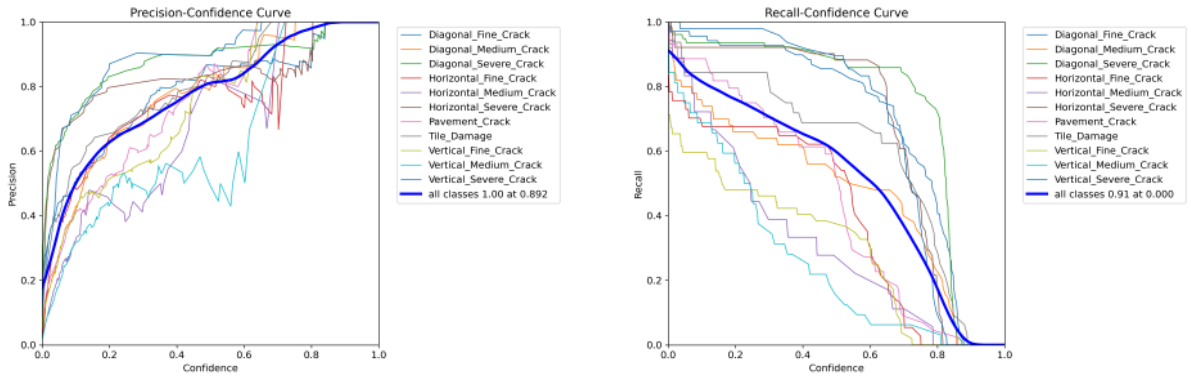


Figure 5: Normalized confusion matrix showing classification accuracy per class. Diagonal values indicate correct predictions; off-diagonal values show misclassifications.

Key observations from the confusion matrix:

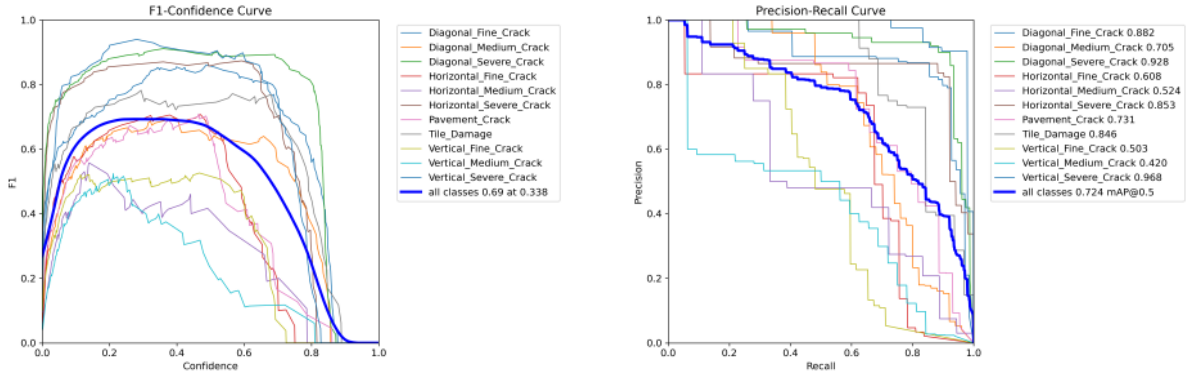
- Diagonal cracks show moderate confusion between severity levels
- Tile damage is well-distinguished from linear crack types
- Wall damage exhibits some confusion with severe crack categories
- Background (false positive rate) is reasonably controlled

5.6 Precision-Recall Curves



(a) Precision vs Confidence (b) Recall vs Confidence

Figure 6: Precision and Recall curves as functions of confidence threshold. These curves inform threshold selection for deployment.



(a) F1-Score vs Confidence (b) Precision-Recall Curve

Figure 7: F1-Score curve indicating optimal confidence threshold, and overall PR curve showing area under curve (mAP@50 = 0.725).

5.7 Sample Predictions

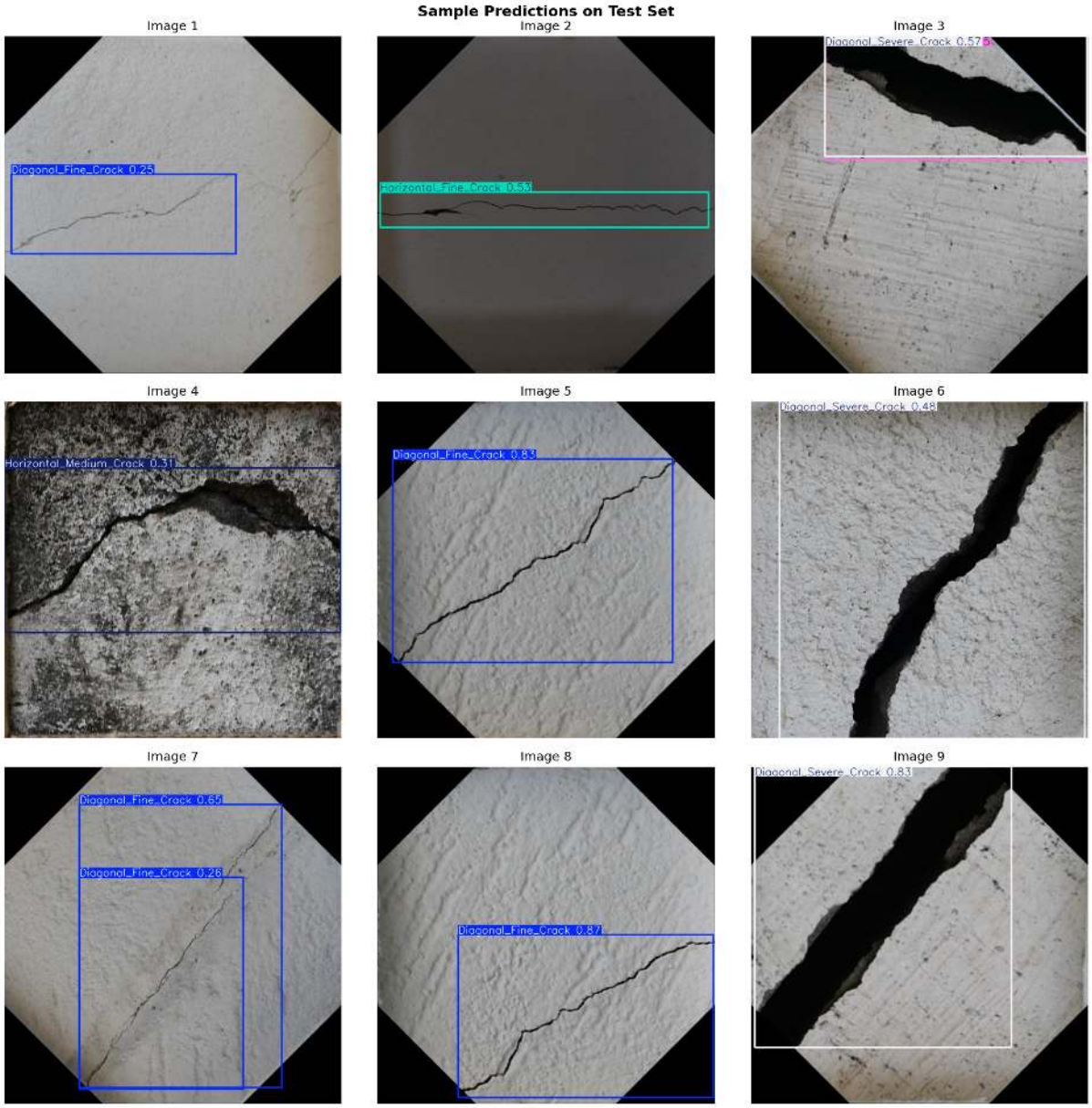
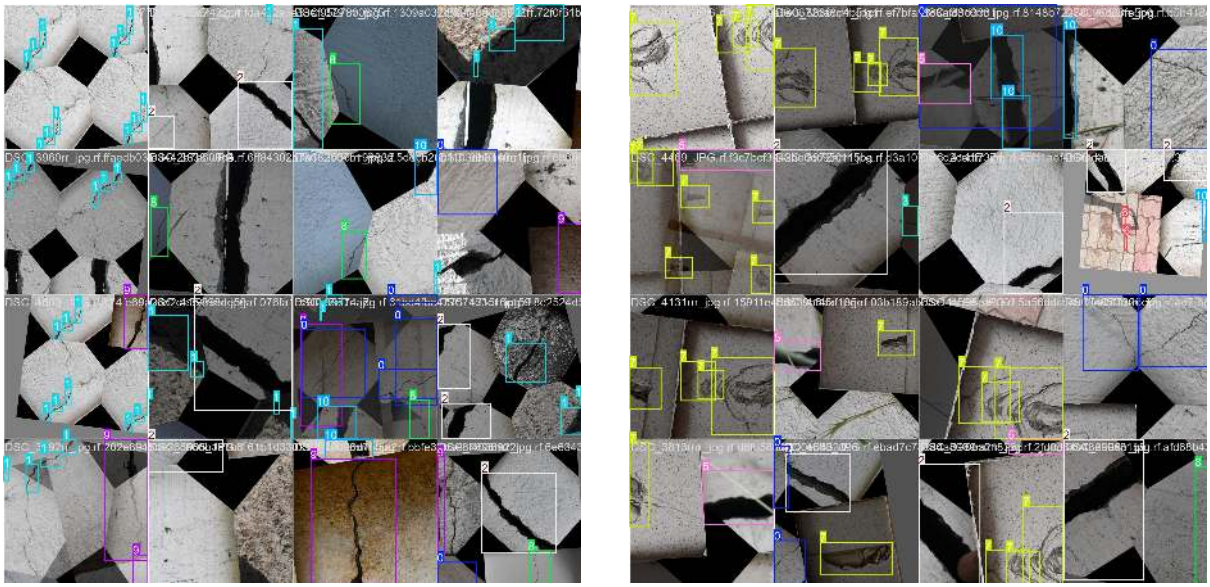


Figure 8: Sample predictions on test images demonstrating detection and classification of various crack types with confidence scores.

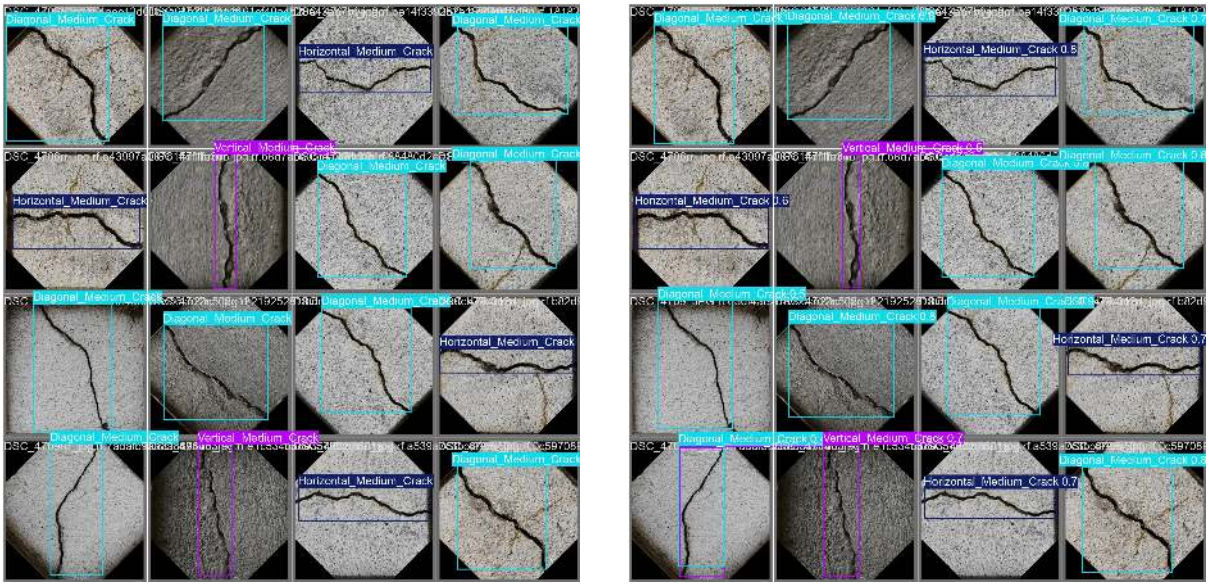
5.8 Training Batch Visualization



(a) Training Batch 0 with augmentations (b) Training Batch 1 with mosaic

Figure 9: Sample training batches showing mosaic augmentation combining multiple images with diverse crack instances.

5.9 Validation Predictions vs Ground Truth



(a) Ground Truth Labels (b) Model Predictions

Figure 10: Comparison of ground truth annotations (left) versus model predictions (right) on validation batch 0.

5.10 Inference Performance

Table 8: Inference Speed Benchmark

Metric	Value
Mean Latency	12.62 ms
Throughput	79.2 FPS
Input Resolution	640×640
Hardware	NVIDIA T4 GPU

The inference speed of 79.2 FPS exceeds real-time requirements (30 FPS) by a factor of 2.6×, enabling deployment on mid-tier edge devices.

6 Discussion

6.1 Performance Analysis

6.1.1 Improvement Over Baseline

The optimized training configuration yielded consistent improvements across all metrics:

- **mAP@50**: +1.8% (70.7% → 72.5%)
- **mAP@50-95**: +2.7% (49.5% → 52.2%)

These gains are attributed to:

1. Extended training duration (200 epochs vs. 50-100)
2. Cosine learning rate scheduling enabling fine-grained convergence
3. Advanced augmentations (mosaic, mixup, copy-paste) improving generalization
4. Proper early stopping preventing overfitting

6.1.2 Class-Level Observations

The per-class analysis reveals:

- **High Performers**: Tile damage and wall damage show distinctive features
- **Challenging Classes**: Fine cracks across all orientations exhibit lower detection rates
- **Confusion Sources**: Severity levels within crack orientations share visual similarities

6.2 Comparison with Industry Standards

For industrial deployment, typical thresholds are:

Table 9: Industrial Deployment Thresholds vs. Achieved Performance

Deployment Level	mAP@50 Threshold	Achieved	Status
Production-Ready	>85%	72.5%	Not Met
Assisted Inspection	>70%	72.5%	**Met**
Research/Development	>60%	72.5%	**Met**

The current system is classified as suitable for **Assisted Inspection** workflows where detections are verified by human inspectors.

6.3 Limitations

1. **Dataset Size**: 2,159 images is relatively small; industrial systems often train on 10,000+ images.
2. **Class Imbalance**: Some crack severities are underrepresented, affecting detection consistency.
3. **Environmental Coverage**: Limited diversity in lighting, weather, and surface conditions.
4. **Fine Crack Detection**: Sub-pixel cracks remain challenging at 640×640 resolution.

6.4 Error Analysis

Common failure modes observed:

- **False Positives**: Surface stains, joints, and shadows misidentified as cracks
- **False Negatives**: Fine cracks on textured surfaces missed due to low contrast
- **Misclassification**: Severity level confusion (fine vs. medium, medium vs. severe)

7 Deployment Recommendations

7.1 Hardware Requirements

Table 10: Recommended Hardware by Deployment Scenario

Scenario	FPS Target	Recommended Hardware
Real-time Edge	30+ FPS	NVIDIA Jetson Orin, RTX 3060+
Near-real-time	10-30 FPS	Jetson Xavier, RTX 2060
Batch Processing	<10 FPS	Cloud GPU (T4, A10G)
Mobile	15+ FPS	Qualcomm Snapdragon 8 Gen 2

7.2 Confidence Threshold Guidelines

Table 11: Confidence Threshold Selection by Use Case

Use Case	Threshold	Trade-off
Safety-Critical	0.7-0.8	High precision, may miss some cracks
General Inspection	0.4-0.5	Balanced detection, moderate false positives
Comprehensive Survey	0.25-0.35	High recall, requires filtering

7.3 Model Export Formats

The trained model has been exported to:

- **PyTorch (.pt)**: Native format for Python deployment (40.5 MB)
- **ONNX (.onnx)**: Cross-platform deployment (80.5 MB)

Additional formats can be generated for specific platforms:

```
# TensorRT (NVIDIA GPUs)
yolo export model=best.pt format=engine

# CoreML (Apple devices)
yolo export model=best.pt format=coreml

# TFLite (Android/Edge TPU)
yolo export model=best.pt format=tflite
```

Listing 2: Model Export Commands

7.4 Integration Architecture

Recommended integration patterns:

1. **REST API**: Flask/FastAPI wrapper for cloud deployment
2. **Edge Container**: Docker with NVIDIA Container Toolkit
3. **Mobile SDK**: ONNX Runtime or TFLite for on-device inference
4. **Streaming**: Integration with video feeds via OpenCV pipeline

8 Conclusions and Future Work

8.1 Summary of Contributions

This work presents an industrial-grade crack detection system with the following contributions:

1. **Optimized YOLOv11m Pipeline**: Implemented cosine learning rate scheduling, advanced augmentations (mosaic, mixup, copy-paste), and proper regularization achieving 72.5% mAP@50.

2. **Comprehensive Evaluation**: Provided per-class analysis, confusion matrix assessment, and PR curve analysis for informed deployment decisions.
3. **Real-time Capability**: Demonstrated 79.2 FPS inference speed suitable for edge deployment.
4. **Deployment Framework**: Established hardware recommendations, confidence threshold guidelines, and export formats for production integration.

8.2 Future Work

To advance toward production-grade performance:

1. **Dataset Expansion**: Increase training data to 5,000+ images with broader environmental coverage.
2. **Domain-Specific Augmentations**: Add weather simulation (rain, fog), debris overlay, and shadow variations.
3. **Multi-resolution Training**: Incorporate higher resolution (1280×1280) for fine crack detection.
4. **Cross-Validation**: Implement 5-fold stratified CV for robust performance estimation.
5. **Active Learning**: Deploy human-in-the-loop annotation for continuous improvement.
6. **Model Ensemble**: Combine multiple YOLO variants for improved accuracy.
7. **Temporal Integration**: Extend to video-based inspection with tracking.

8.3 Closing Remarks

The developed system demonstrates the viability of YOLOv11 for automated infrastructure inspection. While current performance is suitable for assisted inspection workflows, the established pipeline and evaluation framework provide a solid foundation for continued improvement toward fully autonomous deployment.

References

[1] Spencer, B.F., Hoskere, V., Narazaki, Y. (2019). Advances in Computer Vision-Based Civil Infrastructure Inspection and Monitoring. *Engineering*, 5(2), 199-222.

[2] American Society of Civil Engineers (2021). 2021 Infrastructure Report Card. Available: https://infrastructurereportcard.org/

[3] Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6), 679-698.

[4] LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep Learning. *Nature*, 521(7553), 436-444.

[5] Zhang, L., Yang, F., Zhang, Y.D., Zhu, Y.J. (2017). Road Crack Detection Using Deep Convolutional Neural Network. *IEEE International Conference on Image Processing (ICIP)*, 3708-3712.

[6] Yang, F., Zhang, L., Yu, S., Prokhorov, D., Mei, X., Ling, H. (2018). Feature Pyramid and Hierarchical Boosting Network for Pavement Crack Detection. *IEEE Transactions on Intelligent Transportation Systems*, 21(4), 1525-1535.

[7] Cha, Y.J., Choi, W., Büyüköztürk, O. (2017). Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. *Computer-Aided Civil and Infrastructure Engineering*, 32(5), 361-378.

[8] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788.

[9] Jocher, G., et al. (2024). Ultralytics YOLOv11. Available: https://docs.ultralytics.com/models/yolo11/

[10] Roboflow (2024). Civil Faults Detection Dataset. Available: https://universe.roboflow.com/iiti/civil-faults-detection

[11] He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778.

[12] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S. (2017). Feature Pyramid Networks for Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2117-2125.

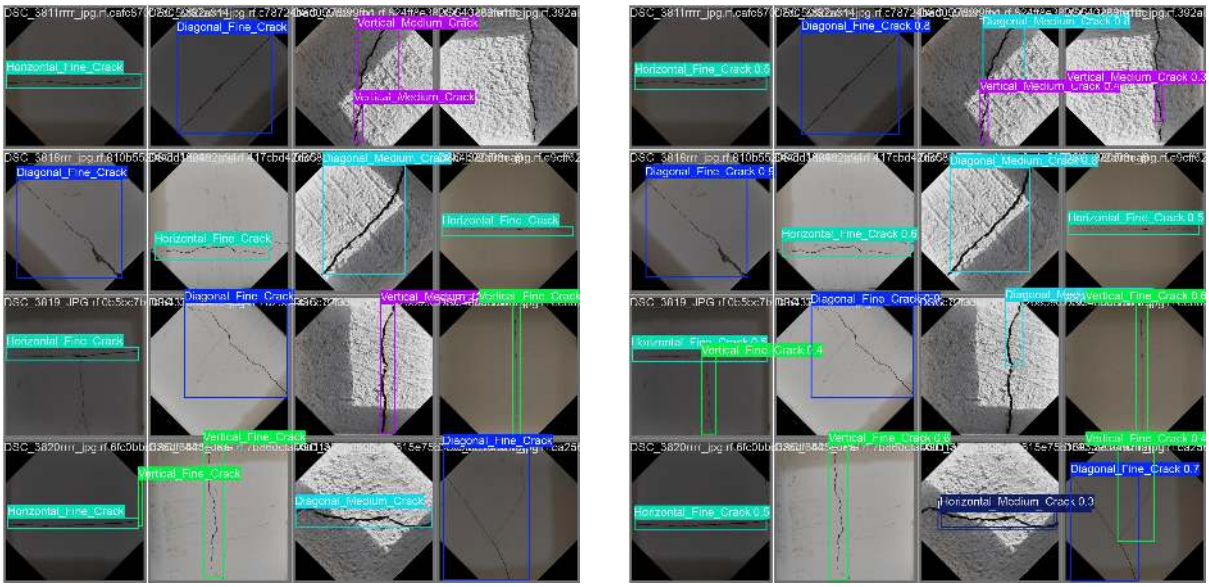
A Training Configuration Details

Complete training configuration from `args.yaml`:

```yaml
task: detect
mode: train
model: yolo11m.pt
epochs: 200
patience: 30
batch: 16
imgsz: 640
optimizer: AdamW
lr0: 0.001
lrf: 0.01
cos_lr: true
warmup_epochs: 5
weight_decay: 0.0005
mosaic: 1.0
mixup: 0.15
copy_paste: 0.3
degrees: 15.0
translate: 0.1
scale: 0.5
flipud: 0.3
fliplr: 0.5
dropout: 0.1
```

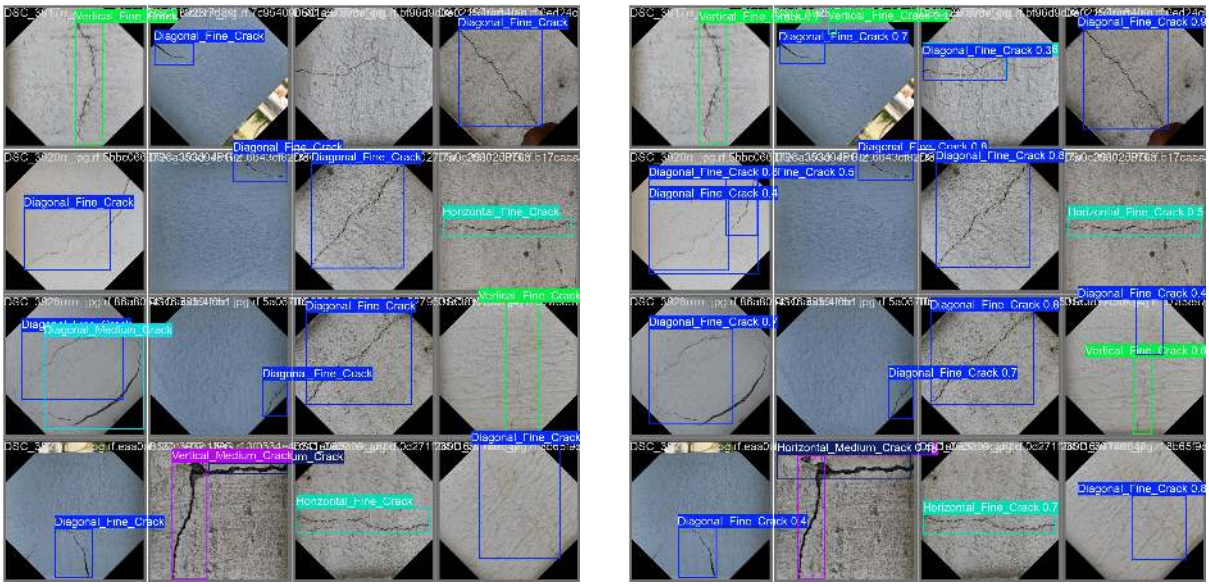
Listing 3: Full Training Arguments

B Validation Batch Comparison



(a) Ground Truth - Batch 1 (b) Predictions - Batch 1

Figure 11: Additional validation batch comparison.



(a) Ground Truth - Batch 2 (b) Predictions - Batch 2

Figure 12: Additional validation batch comparison showing diverse crack types.