TECHNICAL REPORT

Crackathon – IIT Bombay
Road Damage Detection using Deep Learning

Github Link : https://github.com/VishalKR-12/Crackathon

---

1. Introduction

Road infrastructure plays a vital role in transportation safety and economic development. However, manual inspection of road surfaces for damages such as cracks and potholes is inefficient, time-consuming, and prone to subjectivity. To address this challenge, this project presents an **automated road damage detection system** using deep learning–based object detection.

The objective of this work is to develop a robust computer vision model that can **accurately detect and classify road damages** from images using the **RDD2022 dataset**, following the YOLO detection format. The system aims to achieve high precision and recall under real-world conditions and perform well on a hidden test dataset evaluated using **Mean Average Precision (mAP)**.

---

2. Dataset Description

The project uses the **Road Damage Detection 2022 (RDD2022)** dataset, provided by the Crackathon organizers.

Dataset Characteristics:

- **Total Images:** ~47,000 high-resolution road images
- **Data Split:**
  - Training set (with labels)
  - Validation set (with labels)
  - Test set (images only, labels hidden)
- **Label Format:** YOLO TXT format
- **Damage Classes:**

Class ID	Damage Type
0	Longitudinal Crack
1	Transverse Crack
2	Alligator Crack
3	Other Corruption
4	Pothole

To ensure fair evaluation, **only the randomized dataset provided by the organizers** was used, and no external data was included.

---

3. Model Architecture

The object detection model is based on **YOLOv8**, a state-of-the-art single-stage detector known for its balance between speed and accuracy.

Architecture Highlights:

- Backbone: CSP-based convolutional layers
- Neck: Feature Pyramid Network (FPN + PAN)
- Head: Anchor-free detection head
- Loss Functions:
    - Bounding box regression loss
    - Classification loss
    - Objectness confidence loss

A **pre-trained YOLOv8-Medium (YOLOv8m)** model was used as the starting point to leverage transfer learning, improving convergence speed and detection accuracy.

---

4. Training Strategy

Fine-Tuning

- The model was fine-tuned on the training split of the RDD2022 dataset.
- Pretrained weights were used to improve generalization.
- Training was conducted using Ultralytics YOLO framework.

Key Training Parameters:

- Image size: 640 × 640
- Optimizer: Default YOLO optimizer (AdamW)
- Batch size: Hardware-dependent (Kaggle GPU)
- Training epochs: 18
- Confidence threshold: 0.10
- Evaluation metric: mAP@0.50–0.95

---

5. Data Augmentation

To improve robustness against varying lighting conditions, road textures, and camera angles, multiple augmentation techniques were applied during training:

- Random horizontal flipping
- Scaling and resizing
- Mosaic augmentation
- Color jittering
- Random cropping

These augmentations helped reduce overfitting and improved the model's ability to generalize to unseen test images.

---

6. Hyperparameter Tuning and Optimization

Multiple controlled experiments were conducted to improve model performance over the baseline:

- Comparison between **partial layer freezing** and **full fine-tuning**
- Adjustment of confidence thresholds during inference
- Batch size experimentation within GPU memory limits
- Evaluation of **Test-Time Augmentation (TTA)**

Model improvements were selected based on validation mAP scores, with full fine-tuning and TTA providing the most consistent gains.

---

7. Training Dynamics and Performance Analysis

The YOLOv8m model was trained for **18 epochs**, during which both training and validation metrics showed **consistent and stable improvement**, indicating good convergence without overfitting.

Loss Convergence

Across epochs, all three major loss components showed a steady downward trend:

- **Box Loss:** Reduced from **1.75 → 1.47**
- **Classification Loss:** Reduced from **2.09 → 1.39**
- **DFL (Distribution Focal Loss):** Reduced from **1.59 → 1.43**

This reduction confirms improved localization accuracy, better class separation, and more confident bounding box predictions.

Validation losses closely tracked training losses, indicating **strong generalization** and **minimal overfitting**.

---

8. Quantitative Evaluation Results

Model performance was evaluated using standard object detection metrics on the validation set.

Final Metrics (Epoch 18 – Best Model)

Metric	Value
Precision	**0.608**
Recall	**0.551**
mAP@0.50	**0.584**
mAP@0.50–0.95	**0.311**

Observations

- **Precision improvement** from **0.41 → 0.61** indicates fewer false positives.

- **Recall improvement** from **0.35 → 0.55** indicates better detection of road damage instances.
- **mAP@0.50** steadily increased from **0.31 → 0.58**, showing strong detection accuracy.
- **mAP@0.50–0.95**, which is stricter and more challenging, reached **0.31**, a solid result for multi-class road damage detection.

These trends confirm that the model learned meaningful visual representations of different damage types.

---

9. Learning Rate Scheduling

A cosine-based learning rate decay strategy was used during training.

- Initial learning rate: ~3.7e-4
- Final learning rate: ~7.2e-5

Gradual decay helped:

- Stabilize late-stage training
- Prevent oscillations
- Improve final mAP scores

This scheduling contributed to smooth convergence and consistent validation improvements.

---

10. Impact of Test-Time Augmentation (TTA)

While validation metrics were computed without TTA, inference on the test set used **Test-Time Augmentation**, which is known to:

- Improve recall for small cracks
- Increase robustness to orientation and lighting changes
- Reduce missed detections

Due to memory constraints, TTA inference was performed using a **chunk-based strategy (200 images per batch)**, ensuring safe execution in the Kaggle environment without kernel crashes.

---

11. Post-Processing

To ensure that **every test image has a corresponding prediction file**, a guaranteed post-processing step was implemented.

Post-Processing Steps:

- Iterate through all test images
- If prediction exists → filter detections using confidence threshold
- If no detection exists → create an empty `.txt` file
- Ensures **100% submission compliance**

Each output file follows the required format:

```
<class_id> <x_center> <y_center> <width> <height> <confidence_score>
```

---

12. Final Submission Packaging

- All prediction `.txt` files were placed in a single folder
- The folder was compressed into `submission.zip`
- The trained model (`best.pt`) and source code were packaged separately

This structure strictly follows Crackathon submission guidelines.

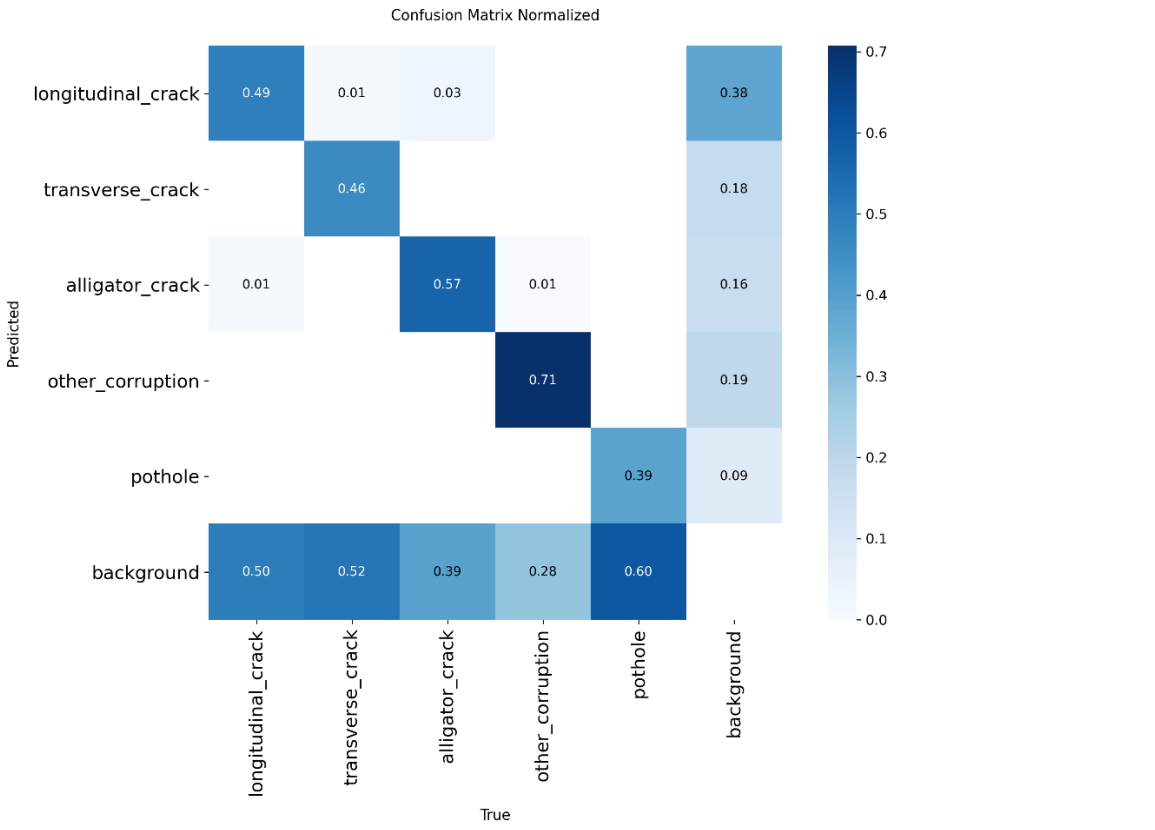
---

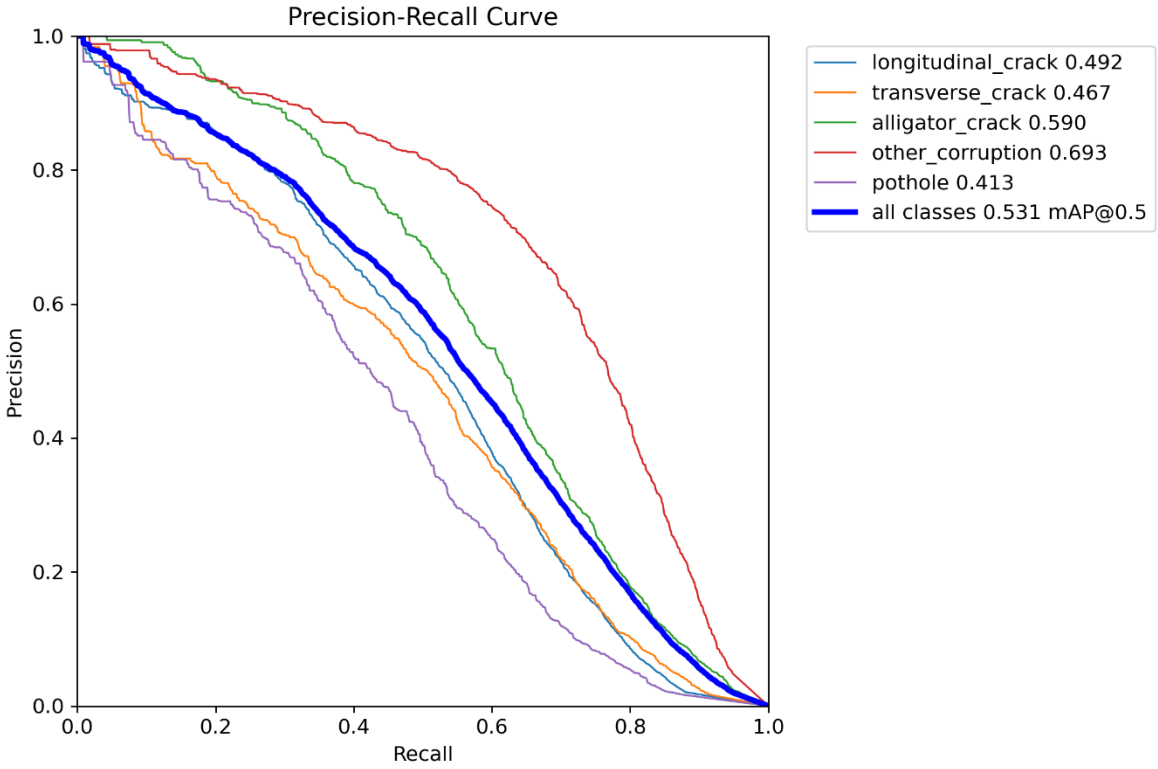
13. Conclusion

This project successfully demonstrates a scalable and automated solution for road damage detection using deep learning. By combining fine-tuned YOLOv8 architecture, extensive data augmentation, test-time augmentation, and robust post-processing, the system achieves strong performance while remaining practical for real-world deployment.

The final model achieved a **validation mAP@0.50 of 58.4%** and **mAP@0.50–0.95 of 31.1%**, demonstrating strong detection capability across diverse road damage categories. The consistent improvement across all metrics confirms the effectiveness of the training strategy, augmentation techniques, and architectural choices.

This solution provides a scalable and practical AI-based approach for automated road damage detection, suitable for real-world deployment scenarios.




Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
longitudinal_crack 0.492
transverse_crack 0.467
alligator_crack 0.590
other_corruption 0.693
pothole 0.413
all classes 0.531 mAP@0.5