Final Project

Crack Detection and Localization in Civil Infrastructure using YOLOv11

(Detecting various cracks for structural health monitoring)

Hasanov, Seymur

CSCI E-89 Deep Learning, Fall 2024

Harvard University Extension School

Prof. Zoran B. Djordjević

Introduction

Problem Statement

- Develop an automated solution for **crack detection and classification** in civil infrastructure using deep learning to enhance safety, efficiency, and maintenance processes.

- **Challenges in Manual Crack Detection** - Time-consuming and labor-intensive processes.

- Objective
  - Detect and classify cracks into **11 categories** (e.g., fine, medium, severe cracks).

Original image source: https://times-age.co.nz/infrastructure/riddiford-bridge-restrictions-remain/

Benefits of Automated Crack Detection

- **Efficiency**: Reduces inspection time and effort, enabling large-scale assessments quickly.
- **Accuracy:** Provides consistent and precise detection of subtle cracks, minimizing human errors.
- **Cost Savings**
- **Scalability**
- **Safety**

Video reference: https://youtu.be/4OBpmIh3PzE?si=29jRRmE4m7_GPpo4

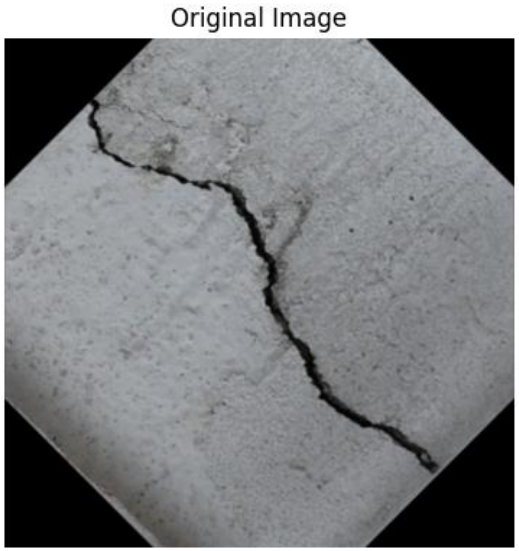
Data Description

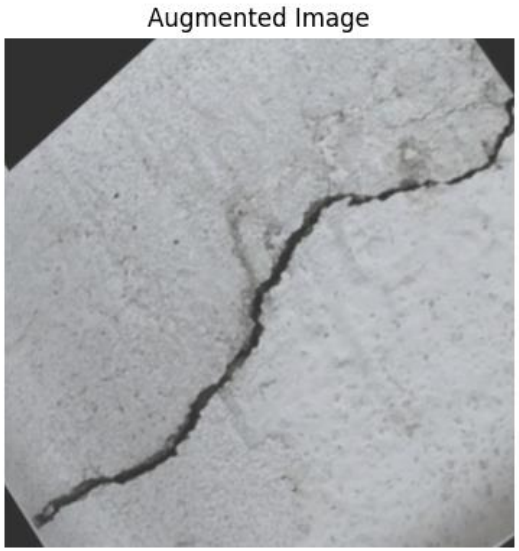
- **Dataset Source**: Roboflow Civil Faults Detection Dataset.
- **Total Images**: 2,159 annotated images.
- **Dataset Splits**:
  - **Train**: 1,505 images (70%)
  - **Validation**: 422 images (20%)
  - **Test**: 232 images (11%)

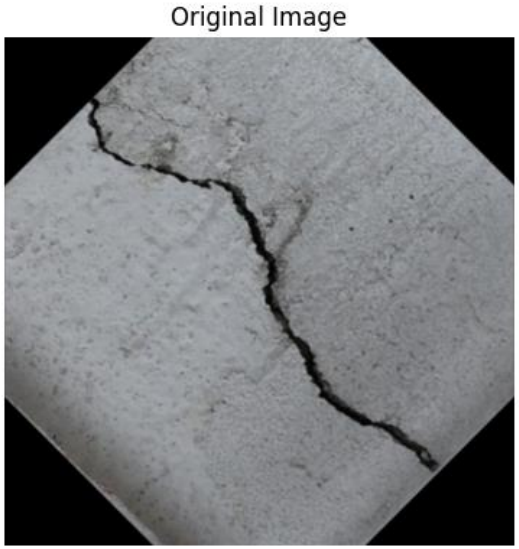
- **Classes**: 11 crack types, including: Diagonal Fine, Medium, Severe Cracks Horizontal Fine, Medium, Severe Cracks Vertical Fine, Medium, Severe Cracks Tile Damage, Pavement Crack

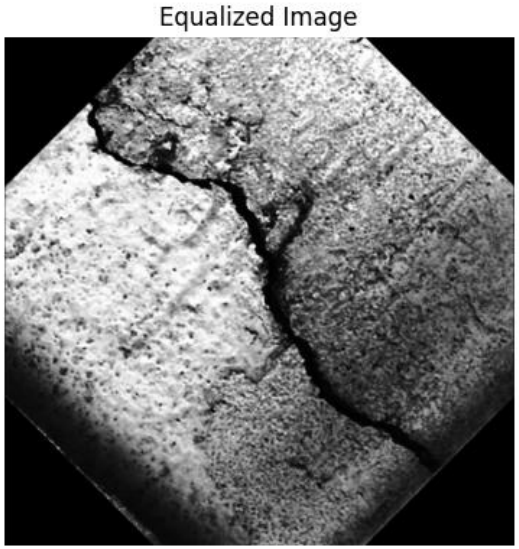
Preprocessing - augmentations

- **Preprocessing**:
  - Grayscale conversion,
  - histogram equalization,
  - and augmentations applied to enhance image quality.



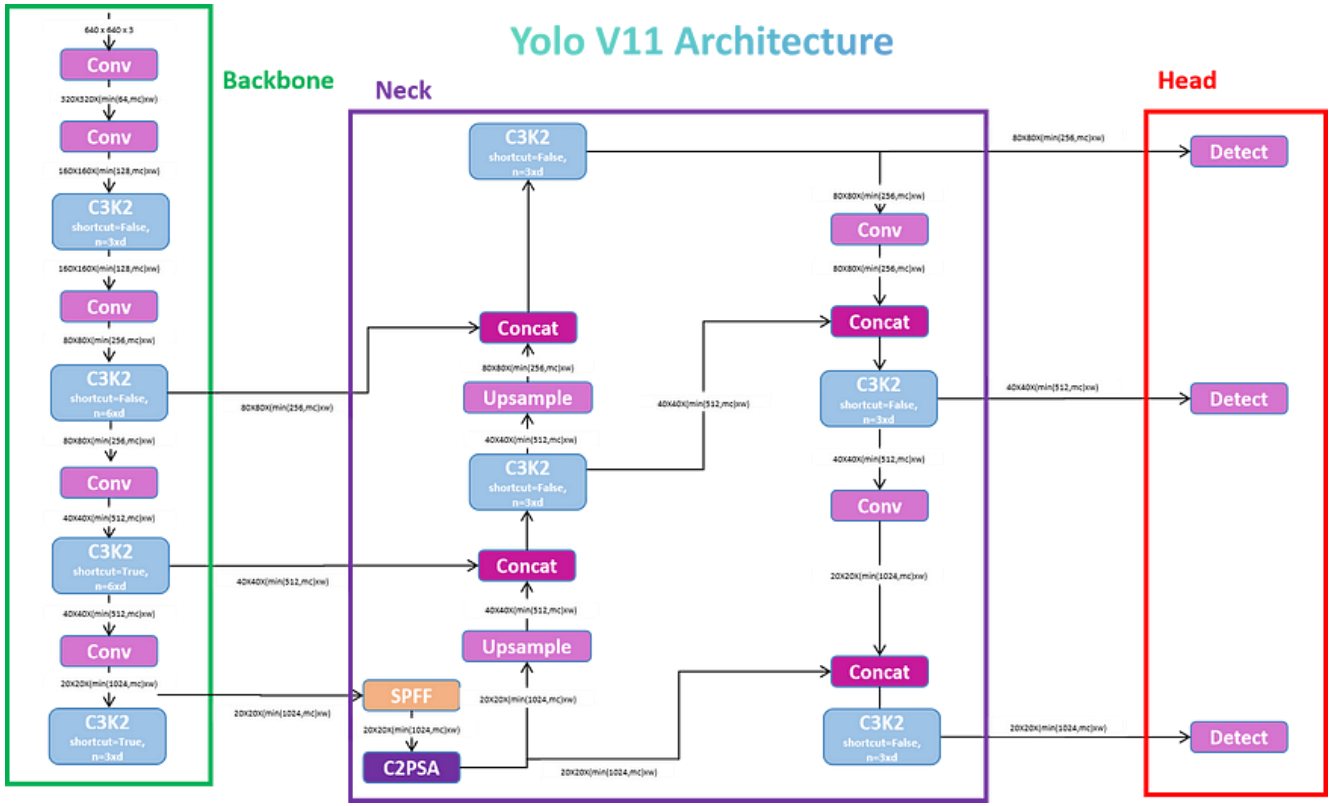






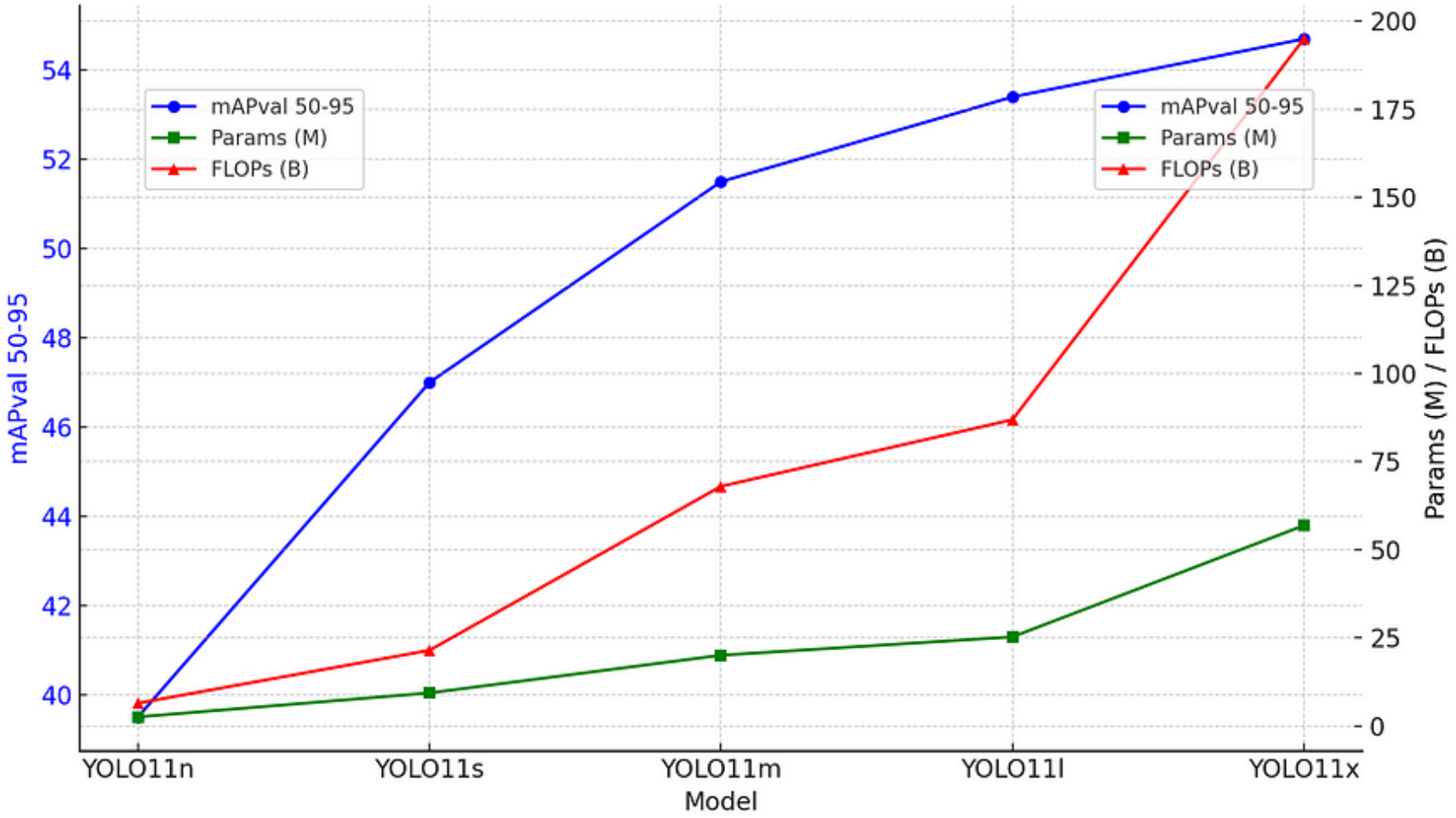
Technology

- YOLO (You Only Look Once), a popular object detection and image segmentation model, was developed by Joseph Redmon and Ali Farhadi at the University of Washington. Launched in 2015, YOLO quickly gained popularity for its high speed and accuracy.



Original YOLO source: https://arxiv.org/abs/1506.02640
Original image source: YOLOv11 Architecture Explained: Next-Level Object Detection with Enhanced Speed and Accuracy | by S Nikhileswara Rao | Oct, 2024 | Medium

YOLOv11 Model Comparison



Model Training

```
[ ] from ultralytics import YOLO
    model = YOLO("yolo11n.yaml").load("yolo11n.pt")  # build from YAML and transfer weights
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n.pt to 'yolo11n.pt'...
100%|██████████| 5.35M/5.35M [00:00<00:00, 65.3MB/s]Transferred 499/499 items from pretrained weights
```

```
[ ] model.info()
```

```
YOLO11n summary: 319 layers, 2,624,080 parameters, 2,624,064 gradients, 6.6 GFLOPs
(319, 2624080, 2624064, 6.614336)
```

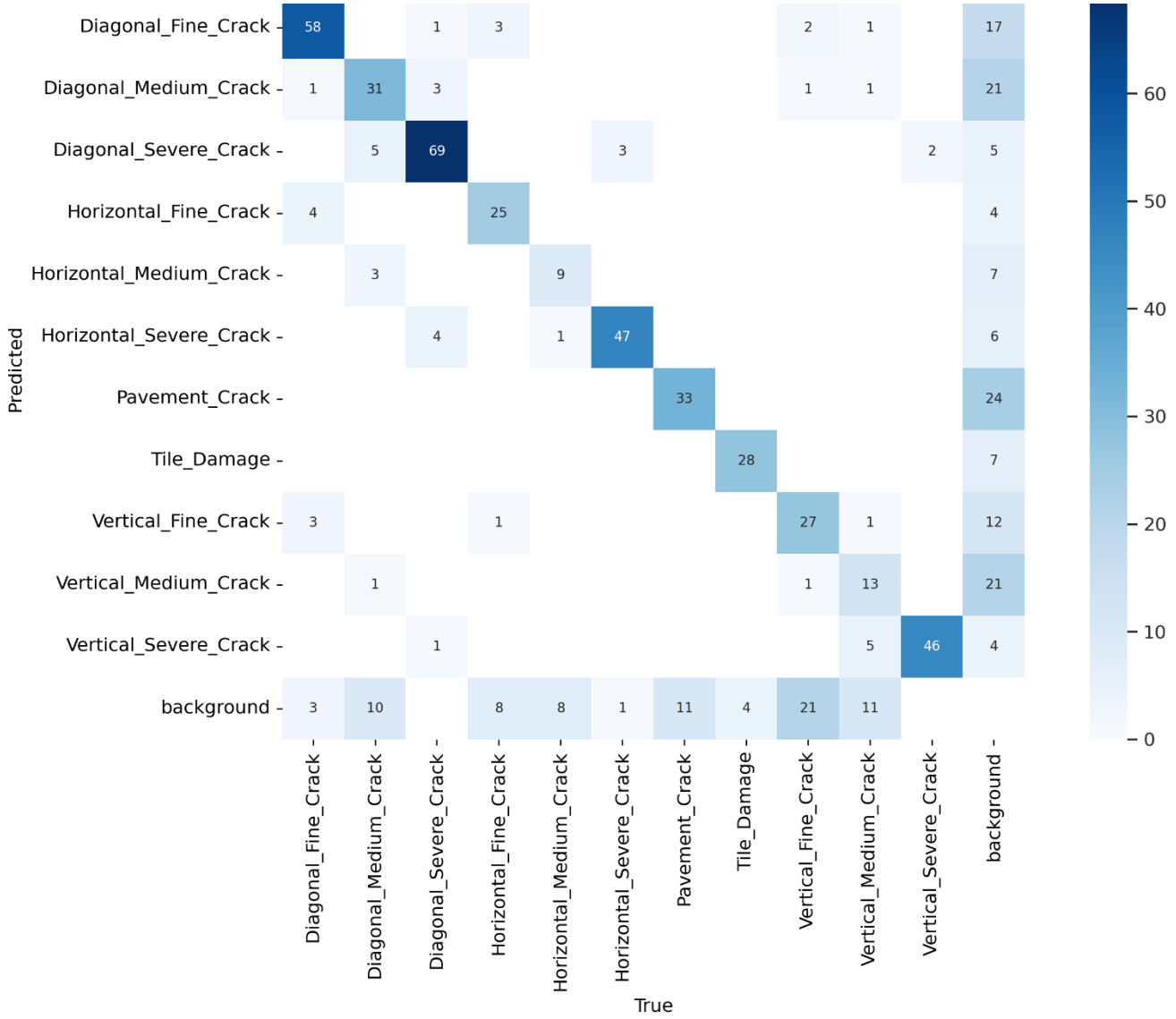
```
[ ] # Train the model
    results = model.train(
        data="/content/Civil-Faults-Detection--1/data.yaml",
        epochs=50,
        imgsz=640,
        plots=True,
        patience=10
    )
```

- **Final mAP50: 70.7%**
- **Final mAP50-95: 49.5%**
- **Precision: 75.0%**
- **Recall: 69.8%**

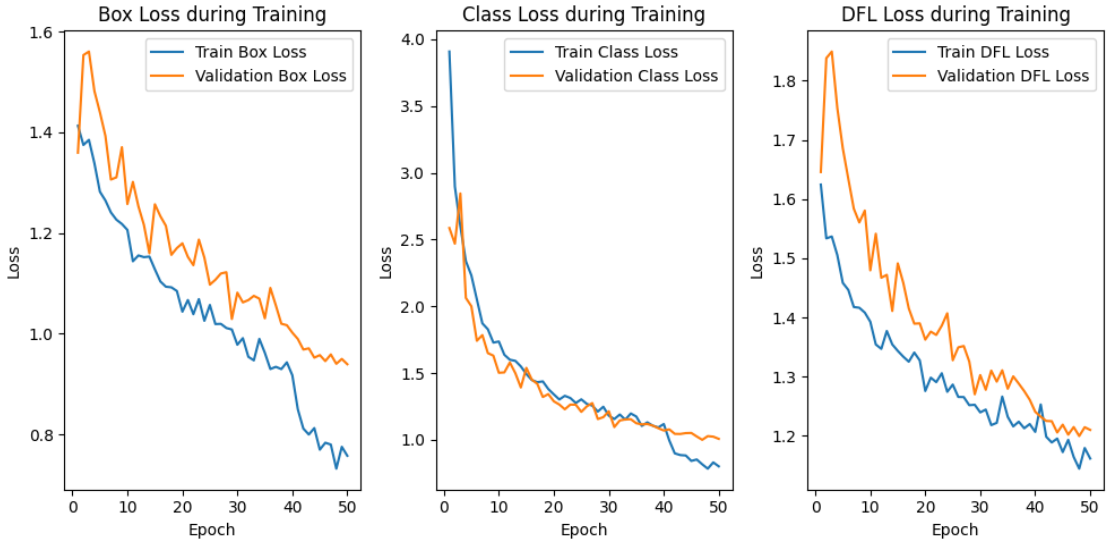
Class-wise Performance

Class	Precision	Recall	mAP50	mAP50-95
Diagonal_Fine_Crack	78.5%	85.5%	80.4%	61.5%
Diagonal_Medium_Crack	67.3%	66.0%	63.1%	51.8%
Diagonal_Severe_Crack	91.1%	89.7%	92.7%	84.5%
Horizontal_Fine_Crack	76.8%	62.6%	62.6%	38.0%
Horizontal_Medium_Crack	58.3%	38.9%	47.1%	28.4%
Horizontal_Severe_Crack	81.9%	88.9%	82.1%	60.7%
Pavement_Crack	65.3%	65.9%	71.7%	36.8%
Tile_Damage	89.9%	83.5%	88.5%	52.9%
Vertical_Fine_Crack	62.8%	48.6%	51.6%	31.5%
Vertical_Medium_Crack	64.5%	40.6%	42.5%	23.5%
Vertical_Severe_Crack	89.1%	97.9%	95.1%	74.6%

Confusion matrix

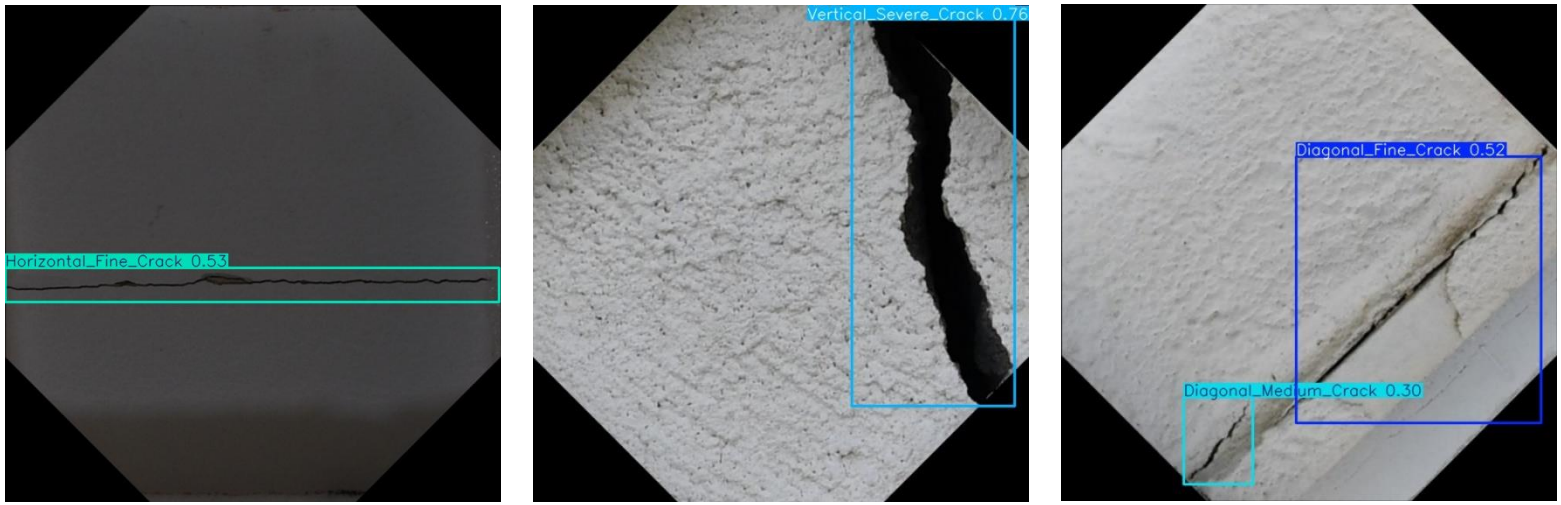


Metrics, Validation and Testing



- Precision stabilizes around **72-75%**, while recall reaches approximately **71%**, indicating balanced performance in detecting and classifying cracks.

Predicting on test images



Hyperparameter Adjustments

- **Learning Rate**: Reduced to **0.0005** to stabilize training and avoid overshooting during optimization.
- **Optimizer**: Changed to **SGD** for improved convergence and better weight decay handling on complex datasets.
- **Weight Decay**: Set to **0.0005** to penalize large weights, reducing overfitting and enhancing generalization on unseen data.

```
[ ]  from ultralytics import YOLO

     model = YOLO('/content/yolo11n.pt')

     # Train the model
     model.train(
         data='/content/Civil-Faults-Detection--1/data.yaml',
         epochs=100,
         imgsz=640,
         lr0=0.0005,
         optimizer='SGD',
         weight_decay=0.0005
         )
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     98/100      2.61G      0.769      1.825       1.15          1        640: 100%|██████████| 95/95 [00:10<00:00,  8.91it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 14/14 [00:01<00:00,  7.02it/s

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      2.62G     0.7544      1.781      1.151          1        640: 100%|██████████| 95/95 [00:10<00:00,  8.98it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 14/14 [00:01<00:00,  7.05it/s

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      2.61G     0.7455       1.78       1.14          1        640: 100%|██████████| 95/95 [00:10<00:00,  8.78it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 14/14 [00:02<00:00,  6.92it/s
```

Adjusted model performance

- **Box Loss**:
  - Training loss indicates potential underfitting or learning rate limitations.
- **Class Loss**:
  - suggests challenges in generalizing the classification.
- **DFL Loss**:
  - Both training and validation losses show consistent downward trends, though a slight gap remains.

YOLOv11X Model

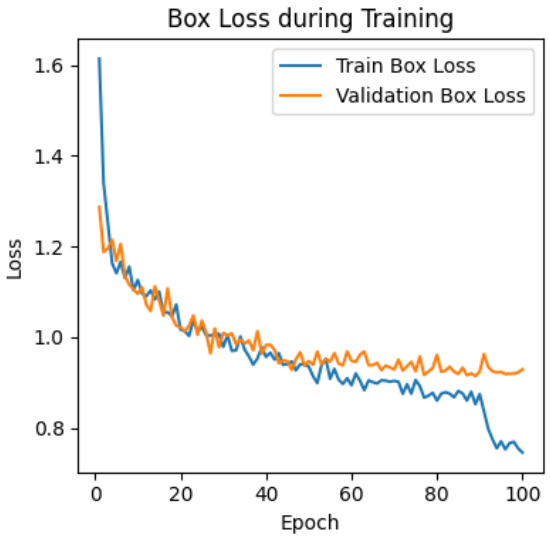
- The **YOLOv11x** model, a larger and more powerful variant, is trained with the same dataset and default (optimized) configuration:

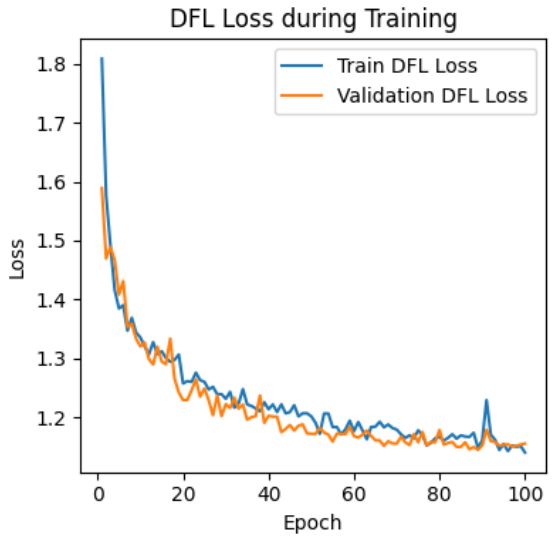
```
[ ]  from ultralytics import YOLO

     model = YOLO('yolo11x.pt')

     # Train the model
     model.train(
         data='/content/Civil-Faults-Detection--1/data.yaml',
         epochs=100,
         imgsz=640,
     )
```

- **Box Loss**:
  - **Train Loss**: Decreased from **1.6** to **0.75**.
  - **Validation Loss**: Stabilized around **0.9** after epoch 40.
- **Class Loss**:
  - **Train Loss**: Reduced from **4.0** to **1.2**.
  - **Validation Loss**: Stabilized around **1.0**.
- **DFL Loss**:
  - **Train Loss**: Dropped from **1.8** to **1.15**.
  - **Validation Loss**: Closely follows training loss, ending near **1.2**.





Conclusion

- ✓ **YOLOv11n**: Lightweight, fast, with decent mAP scores but higher validation losses.
- ✓ **YOLOv11x**: Improved accuracy, lower losses, and better generalization at higher computational cost.

- **Hyperparameter Adjustments**:
- Lowered **learning rate**, increased **batch size**, and tested **AdamW** and **SGD** optimizers for stability and convergence.

- **Loss Trends**: Steady reductions in training and validation losses.
- **Precision & Recall**: Significant improvements, with YOLOv11x achieving the best performance.
- **Class-specific Performance**: High accuracy for **Severe Cracks**, challenges remain for **Horizontal_Medium_Crack**.

- ➢ **YOLOv11x**: Recommended for accuracy-demanding tasks.
- ➢ **YOLOv11n**: Ideal for real-time applications with resource constraints.

Future work can focus on class balancing, advanced augmentations, and hyperparameter tuning.

References

- Roboflow. "Train YOLOv11 Object Detection on Custom Dataset". Roboflow Colab
- Ultralytics Documentation. "YOLOv11: Supported Tasks and Modes." Ultralytics YOLOv11 Docs
- Rao, Nikhil. "YOLOv11 Explained: Next-Level Object Detection with Enhanced Speed and Accuracy." Medium, 2024 .Medium Article
- Roboflow Dataset: "Civil Fault Detection. "Roboflow Dataset
- Ultralytics GitHub Repository. "Ultralytics YOLOv11 - Open-Source Object Detection Models." Ultralytics GitHub
- Google Drive. "YOLOv11 Architecture Diagram."Google Drive Link
- PyPI. "Ultralytics Release Notes." PyPI Ultralytics
- TensorFlow Hub: "Object Detection Models and Performance Metrics." TensorFlow Hub

YouTube URLs

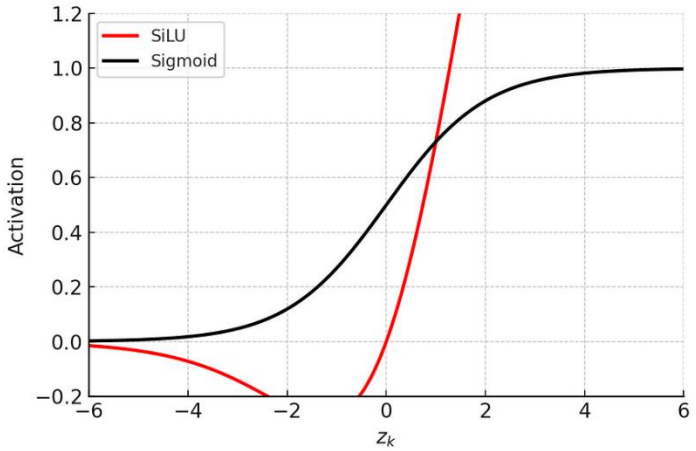
- 3 minute (short): https://youtu.be/_oMe7VelVI8
- 15 minutes (long): https://youtu.be/TGYzXQFG7sI

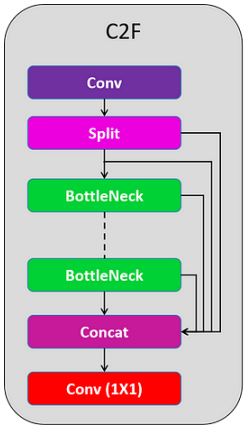
APPENDIX

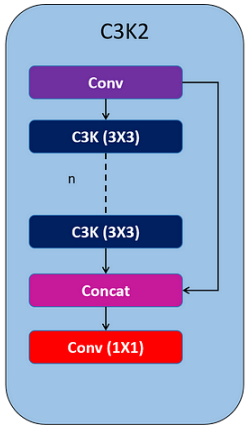
- Backup slides

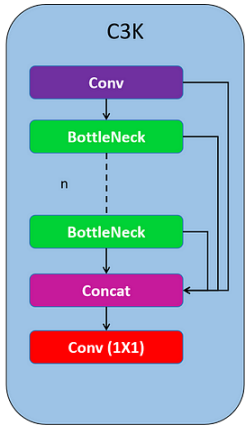
Detailed Explanation of the YOLOv11 Architecture

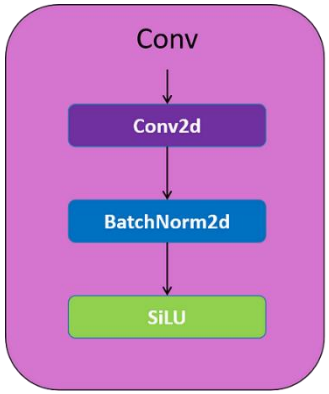
- The main architectural innovations in YOLOv11 revolve around the **C3K2 block**, the **SPFF module**, and the **C2PSA block**, all of which enhance its ability to process spatial information while maintaining high-speed inference.
- **1. Backbone**
  - Convolutional Block
  - Bottle Neck
  - C2F (YOLOv8)
  - C3K2

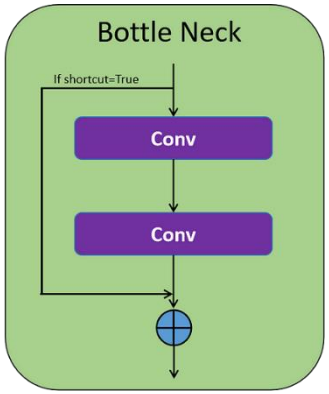












Detailed Explanation of the YOLOv11 Architecture

- **2. Neck: Spatial Pyramid Pooling Fast (SPFF) and Upsampling**
- **3. Attention Mechanisms: C2PSA Block**
- **4. Head: Detection and Multi-Scale Predictions**