# **Project Default Option #1: Road Condition Detection**

**Team Members:**

Tommy Dao - 013958178

Phuc Le - 013873028

Phuoc Le - 013868400

**Task distribution:** For this project Tommy Dao will be responsible for testing of training parameters and in charge of running the models training. Phuc Le will be responsible for data augmentation and combining and splitting of the datasets used for training and evaluation of the models. Phuoc Le will be responsible for evaluation of the models and creating the frontend for the project.

# **Introduction**

Maintaining road infrastructure is a critical aspect in ensuring public safety, minimizing vehicle damage, and optimizing transportation efficiency. However, traditional methods of inspecting road surfaces such as manual surveys or civilian reports are time-consuming and costly. These methods may also lack consistency and fail to provide real-time insights, especially across large urban and rural roads.

The objective of this project is to use a state-of-the-art object detection model to detect road surface damage, primarily cracks and potholes. Our goal is to train a baseline model capable of detecting cracks and potholes, as well as other common road objects such as cars and pedestrians, and then explore some architectural modifications that could further improve the model’s performance. We aim to deliver a model that is able to detect different road damage that may interfere with traffic conditions.

Our final product will be an object detection model accompanied by a UI app to let users upload videos for detection. The app will use the model to detect road damages and output a video back to the user to view. Our model should also utilize an optimized runtime like Onnx or Openvino to speed up inference time.

# 

# **Model**

To pick the best base model, three different models were tested for their performance and they are: Ultralytic’s yolov5n and yolov8n, and detectron2 Faster R-CNN. The model’s mAP and inference speed will be used as the comparison metrics and the COCO2017 val dataset will be used as the evaluation dataset.

| **Model** | **mAP@0.5** | **mAP@0.5:0.95** | **Evaluation Time (s)** |
| --- | --- | --- | --- |
| YOLOv5n | 0.492 | 0.341 | 193.44 |
| YOLOv8n | 0.521 | 0.372 | 177.69 |
| Detectron2 Faster R-CNN | 0.541 | 0.367 | 550.66 |

From the results, Detectron2 Faster R-CNN achieved the highest mAP@0.5 of 0.541, which indicates the best performance in terms of detection accuracy at a 0.5 IoU threshold. However, it had the longest evaluation time of 550.66 seconds, which is more than three times longer when compared to yolov8n. Yolov8n offered a strong balance between performance and speed. It outperformed yolov5n in both mAP metrics and had the fastest evaluation time among the three models. Yolov8n has a mAP@0.5:0.95 of 0.372, which indicates that it is more consistent when detecting objects across varying IoU thresholds. Considering both accuracy and inference efficiency, Yolov8n was selected as the base model for further development and fine-tuning in this project.

# **Dataset**

The combination of COCO2017 and the RDD2022 dataset will be used to train the road damage detection model. The two dataset contain different data and will train the model on different classes. COCO2017 will contain data for cars, bicycles, trucks, and people, while the RDD2022 dataset contains data on road cracks (longitudinal, alligator, transverse, and other) and potholes. After pulling the targeted dataset from the combined dataset, 25% of the combined dataset will be used for training. Also, since the RDD2022 dataset is smaller than COCO2017 dataset, data from the RDD2022 will be oversampled three times to make up for the class imbalance and data augmentation will be used to minimize the overfitting of the repeated data.

COCO: <https://cocodataset.org/>

RDD2022: <https://github.com/sekilab/RoadDamageDetector?tab=readme-ov-file>

# 

# **Methodology**

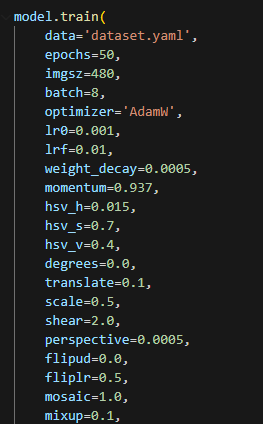
For purposes of training and modifying the model’s architecture, the Ultralytics library will be used for its built-in feature set for preprocessing, training, post processing, and evaluation of YOLO models. Its strengths and the ease of modifying yolov8n architecture will speed up the training and evaluation process. Starting from yolov8n, hyperparameters were changed to test effectiveness on performance. After testing hyperparameters, the yolov8n architecture was changed using a yaml file in Ultralytics to test custom architecture.

## **Baseline Model (YOLOv8)**

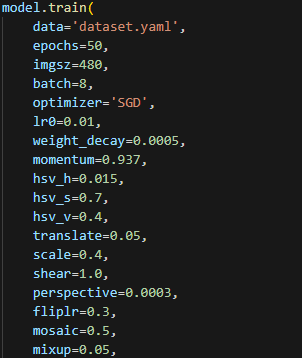
Using yolov8n from Ultralytics

## Configured Model Version 1

* Hsv (color variation)
* Scale, translate, shear(size and angles)
* Mosaic (helps small objects)
* Mixup (class boundaries and generalization)
* Fliplr (horizontal symmetry)



## Configured Model Version 2

****

## 

## Custom Model V1

Added a Conv and c2f layer before the detection head - Better refinement of features

[-1, 1, Conv, [768, 3, 1]]

[-1, 2, C2f, [768]]

## Custom Model V2

depth\_multiple

* Changed from 0.33 → 0.50
* Adds more layers → better pattern learning (especially for small cracks/potholes)

width\_multiple

* Changed from 0.50 → 0.75
* Increases channel size → more features retained per layer = better representational power

nc

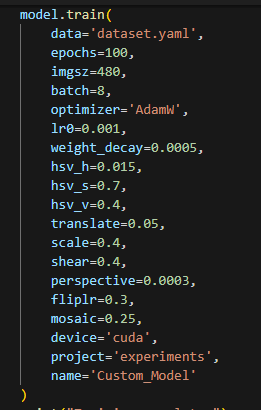
* Set explicitly to 9
* Matches dataset: 4 COCO classes + 5 RDD classes

backbone

* Changed the final Conv layer before SPPF from 512 to 768
* Slightly deepens the backbone → improves context for hard-to-detect damage classes like cracks

head

* Slightly widened intermediate Conv and C2f blocks (e.g., from 512 → 768)
* Added two Conv layers in the head before upsample layers
* Keeps head balanced with wider backbone → avoids bottlenecks in feature fusion



# 

# **Results and Evaluation**

The performance of the models was evaluated using two key metrics: mAP@0.5 and mAP@0.5:0.95. Below is a summary of the results:

| **Model** | **mAP@0.5** | **mAP@0.5:0.95** | **Eval Time (s)** |
| --- | --- | --- | --- |
| Baseline Model | 0.4235 | 0.2323 | 13 |
| Configured Model V1 | 0.4066 | 0.2260 | 13 |
| Configured Model V2 | 0.3885 | 0.2168 | 13 |
| Custom Model V1 | 0.4830 | 0.2583 | 50 |
| Custom Model V2 | 0.4631 | 0.2421 | 29 |

The baseline model was trained using the default yolov8n configuration from the Ultralytics library, with minimal adjustments. It served as the control setup to compare against modified versions. The results indicate solid performance, particularly considering yolov8n’s lightweight architecture. It effectively recognized larger and more distinct objects like cars and pedestrians but showed limitations on subtle features like cracks and potholes.

Configured Model V1 experimented with data augmentation combinations, which includes: Color augmentation (HSV), Geometric transformations (scaling, translation, shear), Mosaic and Mixup (aimed at helping with small object detection), and Horizontal flipping. The model underperformed compared to the baseline model, which is likely due to overly aggressive augmentations like mosaic, shear, and translation that might have distorted fine-grained features necessary for accurately detecting small, low-contrast classes like road cracks and potholes.

Configured Model V2 slightly reduced the augmentation intensity in hopes of improving stability while retaining generalization power. However, the model underperformed compared to the baseline, and did worse than Configured Model V1.

The custom YOLO Model V1 used the same data augmentation from Configured Model V1 but with some changes to the model’s architecture. A Conv layer and a c2f layer was added right before the detection head to improve refinement of features. This architectural change resulted in the best performing model with a mAP@0.5 of 0.483 and mAP@0.5:0.95 of 0.2583. This change did come with a major increase in inference time which resulted in the model being less practical to use.

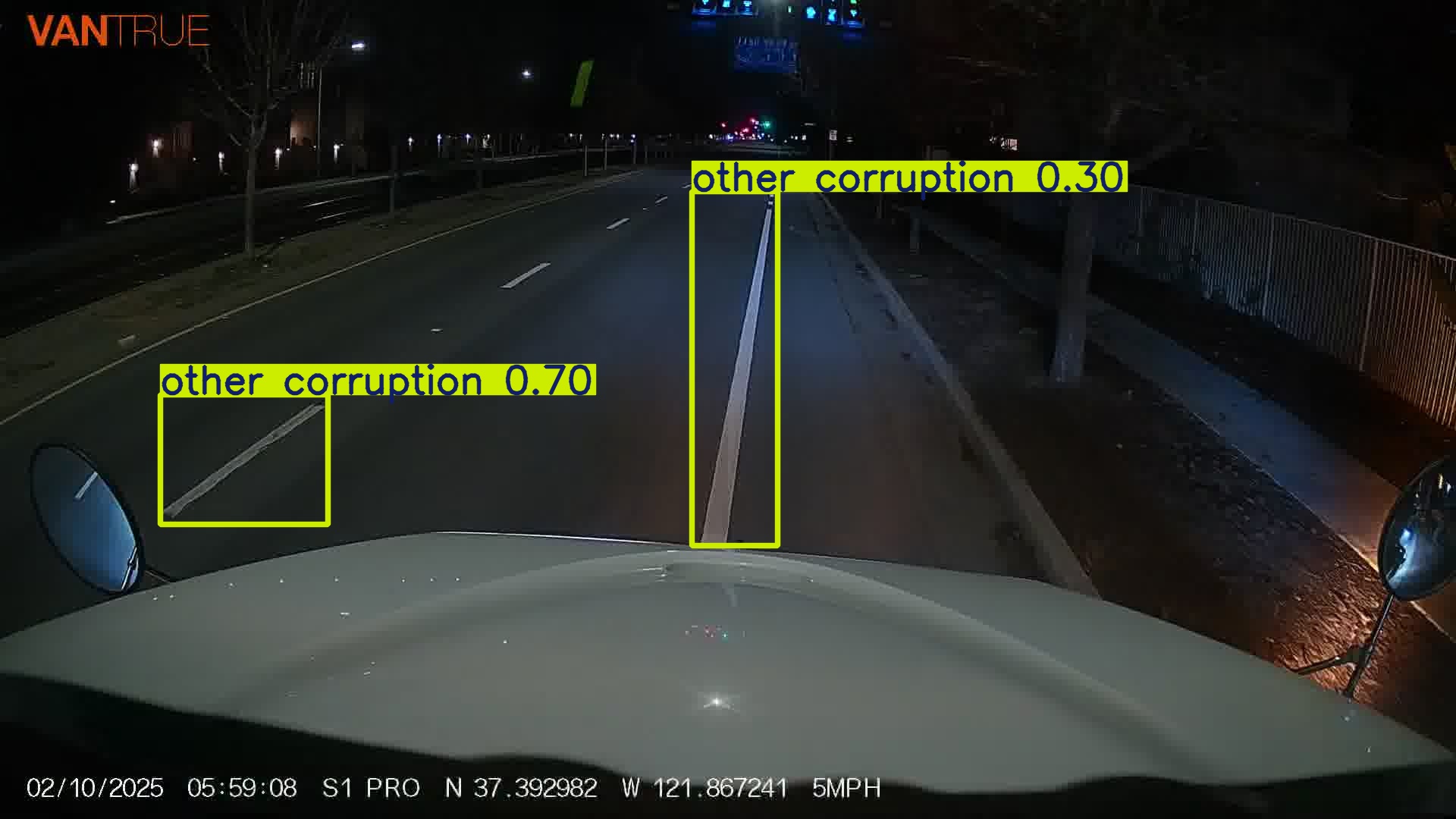
Custom Model V2 used the same data augmentation from Configured Model V1 but with different changes to the model's architecture when compared to custom YOLO Model V1. The architecture change includes the increase in the model's depth and width multiple, increasing the backbone’s final Conv layer before SPPF, and the addition of 2 Conv layers at the start of the head. The result is a model with a bit worse mAP than that of custom YOLO Model V1 but with less of a hit to the inference time. Considering that the custom Model V2 still has better mAP than the base yolov8n model, this model would be more practical than custom Model V1, and will be used for our application.

**Examples of good detections made with Custom Model V2:**



In conditions with good lighting and no visual obstructions like bad weather, the Custom Model V2 is able to detect different kinds of road damages like cracks and potholes. The bounding boxes and class labels are closely aligned with the actual locations and types of defects, confirming that the model has correctly learned road damage features during training. However, when viewing the inferred images, it is clear that the model was not able to identify all of the damages present in the image. This failure in detecting all visual road damages highlights the model’s shortcoming when it comes to sensitivity, which suggests a problem in the model’s training data. This could mean that the model was not trained on enough data or may not have seen enough examples of certain types of damages or damages under certain conditions.

**Examples of bad detections made with Custom Model V2:**



The bad inference results shows that the model is able to detect road cracks and potholes in certain conditions, but fail at others. The model has a high rate of false positives that will cause the model to detect road markings as cracks or other corruptions. This false positive issue seems to be prevalent when the model inferences images in low-light or dark conditions. Another problem relating to low-light conditions is that the model would often not detect anything at all. This problem with the model struggling to detect road damages in low-light conditions falls within expectation since the RDD2022 dataset mostly contains data in daylight and contains little to no data at night.

# 

# **Optimized Runtime**

To improve inference performance even further, an optimized runtime will be used to the detection model in the application. Below are the results comparing the two tested runtimes: Onnx and OpenVino.

| **Model** | **mAP@0.5** | **mAP@0.5:0.95** | **Eval Time (s)** |
| --- | --- | --- | --- |
| Custom Model V2 Base | 0.4631 | 0.2421 | 29 |
| Custom Model V2 Onnx | 0.4508 | 0.2283 | 35 |
| Custom Model V2 Openvino | 0.4508 | 0.2283 | 15 |

From this runtime inference comparison, the fastest optimized runtime is OpenVino, which resulted in an eval time twice as fast as the base pytorch model. The Onnx model was found to be slower than the original model. The openvino runtime will be used in our final road damage detection application.

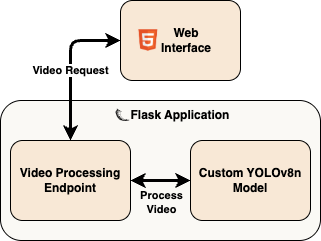
# **Frontend Application**

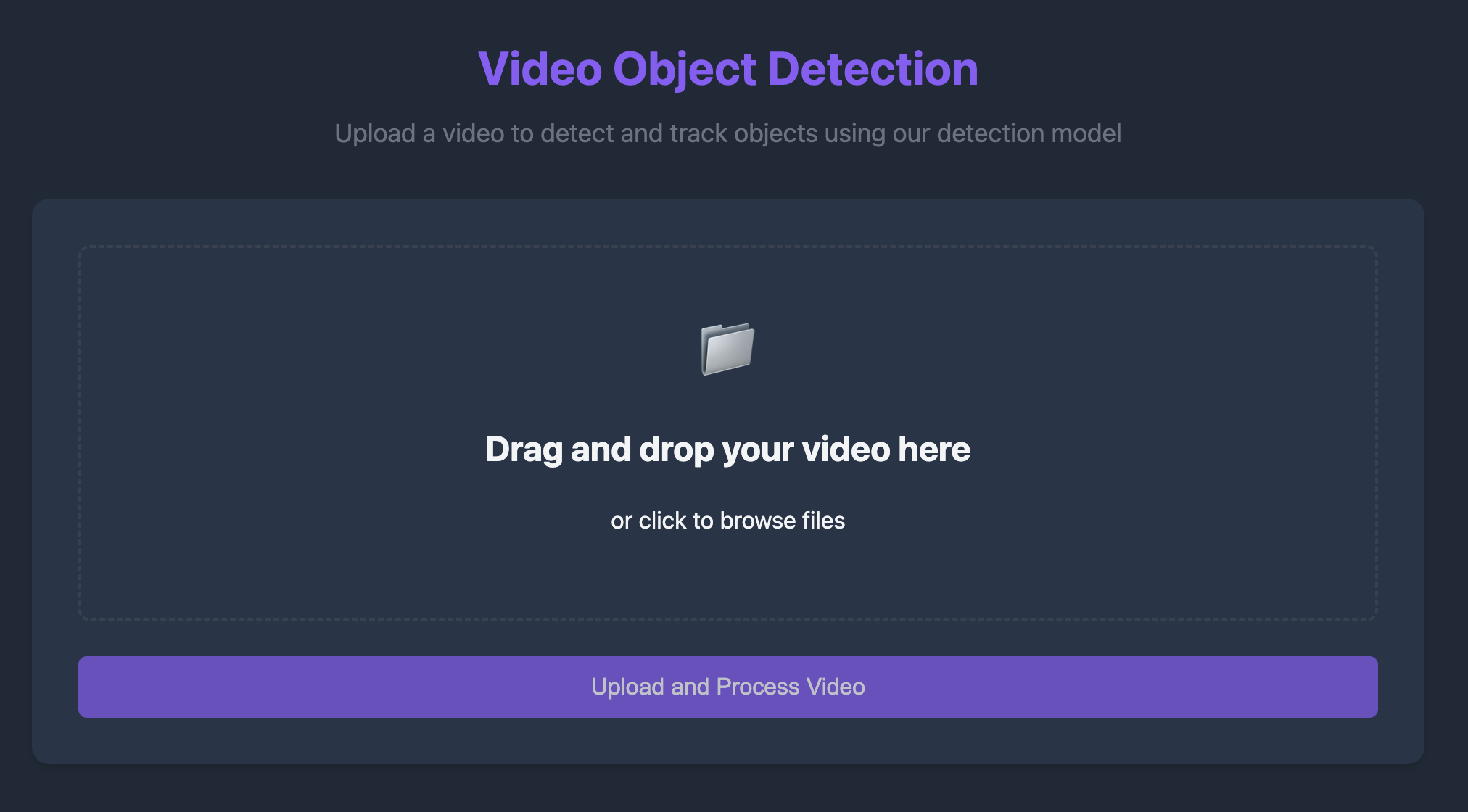
To complement the custom object detection model, a user-friendly frontend application was developed using Python Flask. The goal of this application is to make the road damage detection system accessible to non-technical users by allowing them to upload and analyze road footage through a simple web interface.

Key features of the frontend application are:

* **Video Upload Interface:** Users can upload a video file of road footage directly through the UI.
* **Backend Processing:** Once the video is submitted, the Flask server processes it using the custom YOLOv8-based model. The model is used on each frame to detect road damages such as cracks and potholes, as well as other road entities like vehicles and pedestrians.
* **Annotated Output:** After processing, the application generates a new video with bounding boxes and labels overlayed on detected objects.
* **Result Display:** The processed video is then automatically displayed back to the user within the same interface.

This setup ensures a smooth end-to-end experience for users from uploading the video to viewing detection results.





# 

# **Discussion and Future Works**

The experimentation and results in this project offer several insights into both the potential and limitations of current object detection techniques for road condition assessment. The performance decline observed in the Configured Model Versions 1 and 2 compared to the baseline model, indicates that augmentation strategies need to be carefully tailored to the nature of the data. Overly aggressive augmentations like mosaic, shear, and heavy translation can distort subtle visual elements such as fine cracks or minor potholes, leading to worse performance. In contrast, the Custom YOLO models, which incorporated architectural enhancements like increased depth and width and added conv layers, performed better. This supports the hypothesis that deeper and wider architectures are more capable of capturing fine-grained patterns needed for nuanced damage detection.

Despite these improvements, several limitations persist. The model struggles with scenarios involving poor lighting, such as nighttime footage, and is less effective in detecting cracks or potholes obscured by shadows, mud, or water. Blurry frames or motion artifacts in videos also negatively impact detection reliability. The custom model is also slower than the default model in inferencing tasks.

For future work, we propose two main directions. First is to enhance robustness by training the model with additional data collected under diverse environmental conditions such as nighttime, rain, snow, and glare, in order to improve their ability to generalize in different environmental conditions. Second, we recommend expanding the class labels to include more nuanced categories, such as water-filled potholes, mud or dirt piles, or shadow to improve the model's performance. Additionally, more architecture changes could also be explored to further improve the model’s mAP score and inference time.

# **Conclusion**

This project demonstrates the feasibility of using object detection models, like yolov8, for road damage detection. The final custom YOLO model provided the best balance of speed and accuracy for detecting cracks, potholes, and common road objects such as cars and pedestrians.

# 

# **References**

1. COCO2017 Dataset

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2015). *Microsoft COCO: Common objects in context*. arXiv.<https://arxiv.org/abs/1405.0312>

2. RDD2022 Dataset

Arya, D., Maeda, H., Ghosh, S., Toshniwal, D., & Sekimoto, Y. (2022, September 18). *RDD2022: A multi-national image dataset for automatic road damage detection*. arXiv. <https://doi.org/10.48550/arXiv.2209.08538>

3. Ultralytics YOLO

Jocher, G., Qiu, J., & Chaurasia, A. (2023). Ultralytics YOLO (Version 8.0.0) [Computer software]. Ultralytics. <https://github.com/ultralytics/ultralytics>