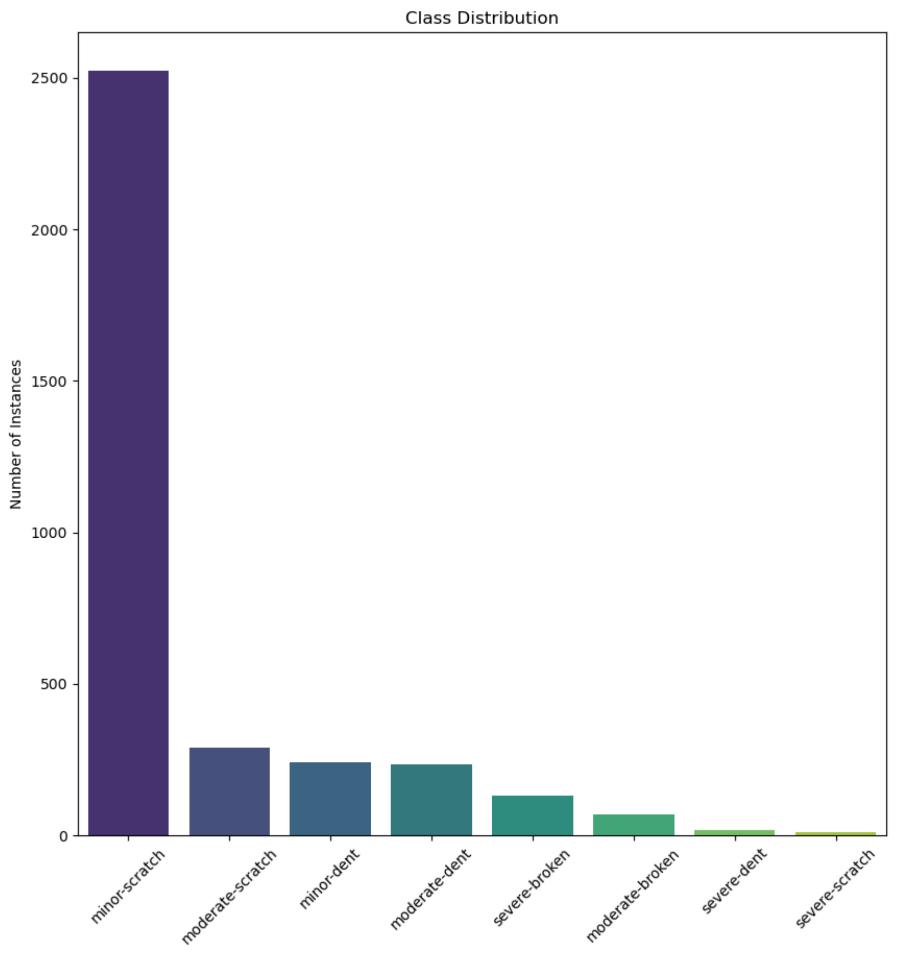
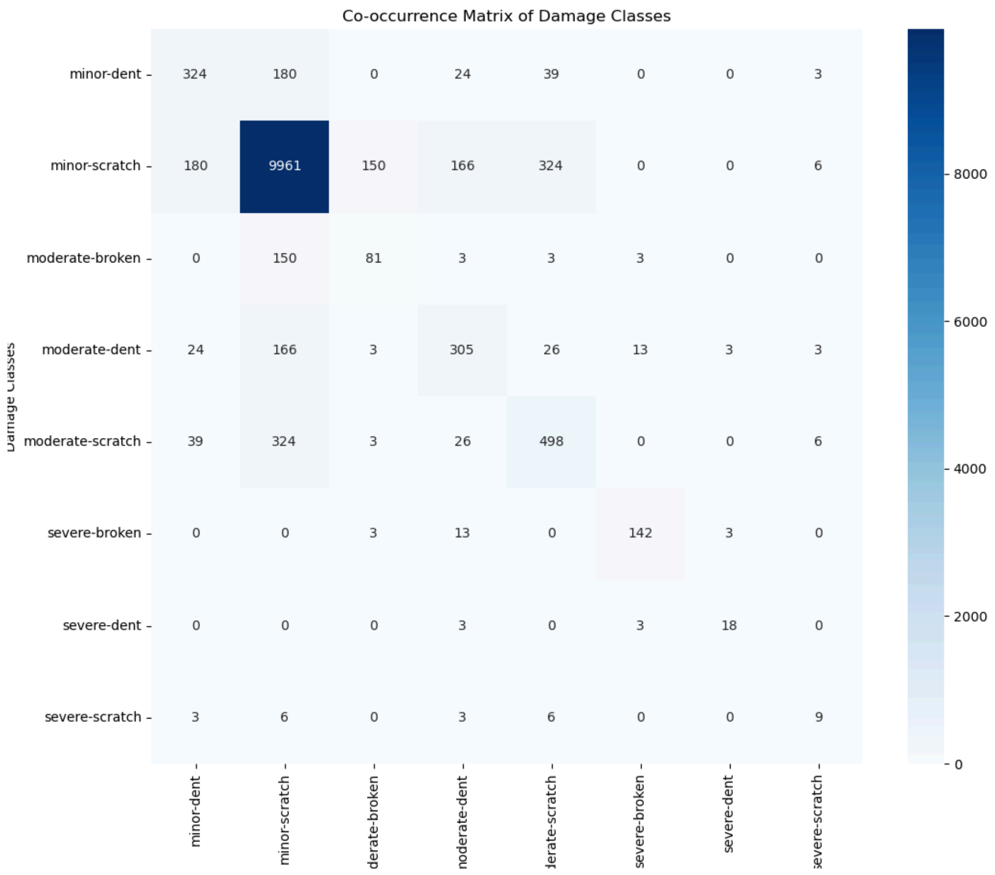
Lensor Damage Detection Challenge

1. Data Analysis

Number of Training Images : 1600
Number of Classes : 9
Number of Annotations: 3512
Number of Images with zero annotations: 21

Distribution of all classes

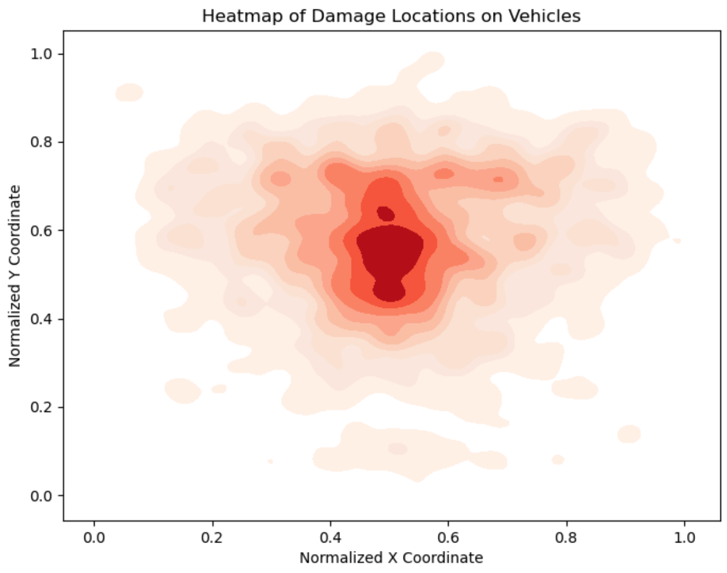




	minor-dent	minor-scratch	derate-broken	noderate-dent	derate-scratch	severe-broken	severe-dent	evere-scratch
minor-dent	324	180	0	24	39	0	0	3
minor-scratch	180	9961	150	166	324	0	0	6
moderate-broken	0	150	81	3	3	3	0	0
moderate-dent	24	166	3	305	26	13	3	3
moderate-scratch	39	324	3	26	498	0	0	6
severe-broken	0	0	3	13	0	142	3	0
severe-dent	0	0	0	3	0	3	18	0
severe-scratch	3	6	0	3	6	0	0	9

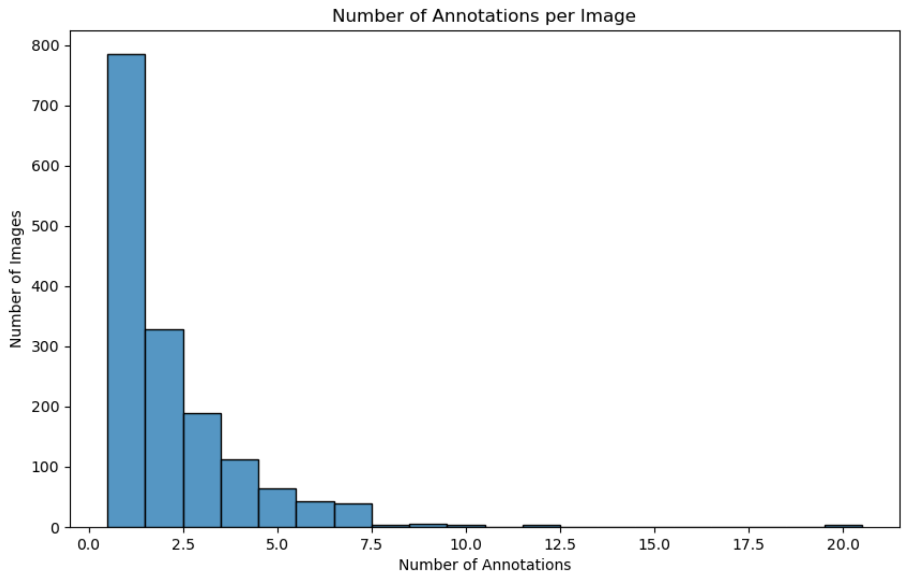
- The class distribution of the given dataset shows a highly imbalanced dataset. This may lead to bias in the model predictions
    - Steps to avoid this is to collect more data that contains under represented classes. If this is not possible, then upsampling of those class data will help.
    - Can implement clustering of minority class data to improve the variability of the oversampled data.
    - Weighted loss function.
- The co-occurrence matrix shows that the dataset may lead to co-occurrence bias [1].

Localization of Damages:



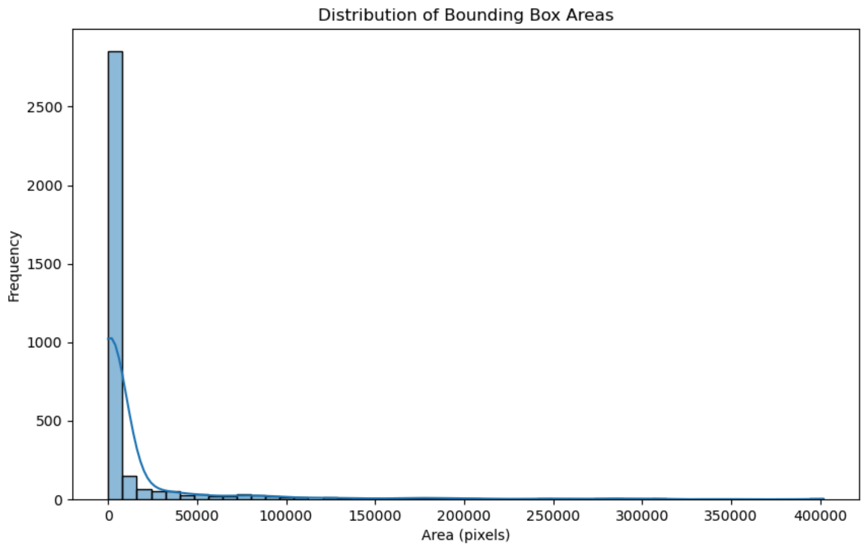
- Concentration of damage location around a specific region might affect the model performance, overfitting the prediction to these areas.
    - Can be solved by data augmentation like, cropping, flipping or shifting to ensure the location varies during training.
    - Attention mechanism.

Annotation Density



- Images with high density of annotation might lead to overlapping of bounding boxes. In the given dataset however, most of the images have less than 4 annotations.

Bounding Box Area



- Majority of bounding boxes have small areas focussing on small damages. (which is also evident from the uneven class distribution)
- This leads to a long tailed problem, where larger damages become difficult to predict.

Model Choice

- Given the time-frame, I decided to explore only three options. The options being YOLO, Faster RCNN and SSD.
- Tradeoffs considered for the three options are

Criteria	**Faster RCNN**	**YOLO**	**SSD**
Speed	Slow	Fast	Moderate
Accuracy	High	Low	Almost similar

			to YOLO
Fine details	Good at capturing smaller details	Weak	Moderate

- I decided to go with Faster RCNN.
- One of the main reasons is the use case and requirements. Since, YOLO and SSD are more suited for real time applications, due to its faster inference time. I figured, speed is not the main priority here as the application is not in real time (to my knowledge).
- Accuracy is the priority as it is important to capture the damages with high precision. Since, out of the three options, Faster RCNN had better accuracy than YOLO, I decided to go with it [2].
- Yolo detects the objects in a single pass, due to which small details have a higher chance of being missed. It is important to capture even fine details like minor scratches and dents on vehicles. Faster RCNN is the better choice as it provides better localization.
- Also from the data analyses done earlier, most of the bounding boxes have smaller areas. This suggests that the model should be able to detect small objects with better accuracy. Another reason for choosing Faster RCNN.

Design Choices

Modular Code Structure:

- The entire code base is organized into separate modules into data, models, training_pipeline etc. This is done mainly to maintain and manage the code better. Also it is easier for multiple developers to collaborate on the project.
- All the configuration parameters like hyperparameters, dataset paths and model related settings are defined in a central file. This permits flexibility and can be updated easily when required.

Data Pre-processing:

- After analysing the data, some of the pre-processing methods have been suggested in the analyses part. However, not all has been implemented. This is because the focus was more towards developing an end-to-end pipeline rather than a best solution.

Model Abstraction:

- When in future the need for using a different model occurs, it is easier to swap among the available models. Also, it is easier to define new models and can easily be instantiated based on a model name string (e.g., faster_rcnn).
- This is mainly done to enable the scalability of the pipeline.

Loss Functions:

- Faster RCNN has its own loss function at the two stages of the network.
- At RPN stage - the objectness loss obtained using Binary cross entropy loss, measures how well it distinguishes between foreground and background.
- At RPN stage - the bounding box regression loss obtained using smooth L1 loss measures the localization of objects.
- At the head part: the classifier loss measures how well the detected regions are classified into the correct categories.
- Again at the head part, the bounding box coordinates are refined using the obtained loss.
- The total loss is the sum of all the 4 losses, used to optimize the model.

Unit Test:

- I added a couple of unit tests to test the functionality of the dataset class. One is to test the dataset initialization and the other is to test the get_item function.

Evaluation Metrics:

- For benchmarking the model, standard COCO evaluation metrics are used, which can provide a meaningful assessment of the performance.
- Precision, Recall and mAP, calculated using IoU provides the model's ability to identify the areas of interest (vehicle damages) within the input images.
- Using such metrics can ease the comparison with other models.

Logging and Visualization:

- Used tensorboard to log and visualize the training and validation process.
- The real-time visualization using tensorboard assists the developer in identifying issues like overfitting or underfitting by monitoring the training curves.

Command Line Interface (CLI):

- Created a single entry point for the training pipeline that performs training, evaluation and inference at the endpoint.
- By using argparse, developers can specify the required action.

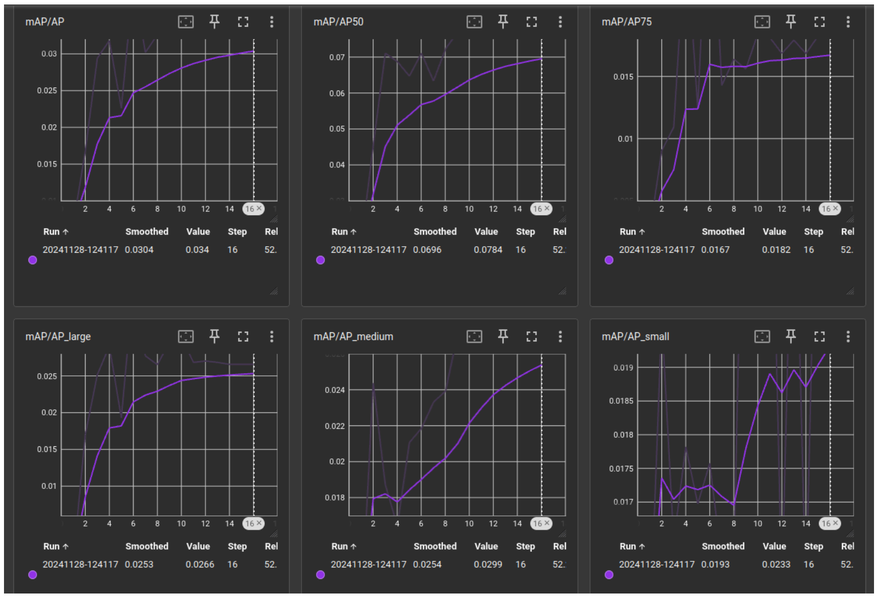
- This facilitates the integration of the project into a larger pipeline or scheduling the jobs via scripts when using HPC to run the training.

Packaging and Deploying the Model:

- GitHub: Version control and continuous deployment.
- Docker: Containerization of the model and dependencies
- Google Cloud Platform (GCP): Hosting the trained model using Google Cloud Run.
- FastAPI: Building the API endpoint.
- Postman: Testing the API by sending the input image.

This solution ensures that the changes are automatically updated when the code is pushed to Github.

Analyses of the model training:



mAP/AP

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.0304	0.034	16	52.

mAP/AP50

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.0696	0.0784	16	52.

mAP/AP75

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.0167	0.0182	16	52.

mAP/AP_large

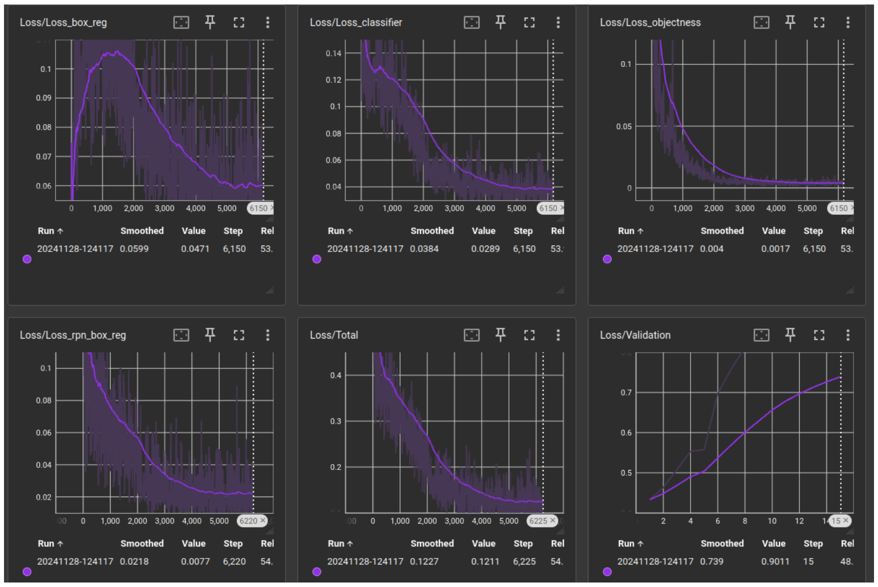
Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.0253	0.0266	16	52.

mAP/AP_medium

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.0254	0.0299	16	52.

mAP/AP_small

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.0193	0.0233	16	52.



Loss/Loss_box_reg

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.0599	0.0471	6,150	53.

Loss/Loss_classifier

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.0384	0.0289	6,150	53.

Loss/Loss_objectness

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.004	0.0017	6,150	53.

Loss/Loss_rpn_box_reg

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.0218	0.0077	6,220	54.

Loss/Total

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.1227	0.1211	6,225	54.

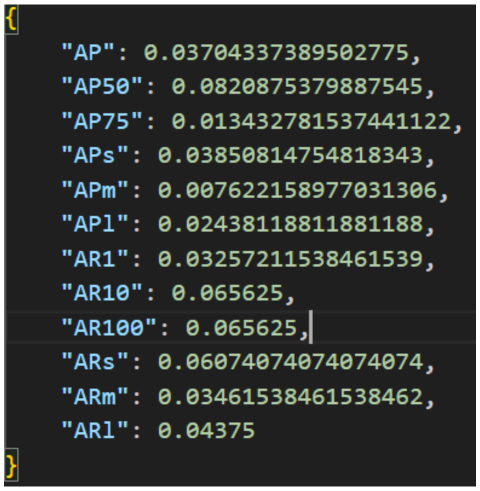
Loss/Validation

Run ↑	Smoothed	Value	Step	Rel
20241128-124117	0.739	0.9011	15	48.

- From the mAP graph, it can be seen that overall mAP is improving. But the value is still low and there is room for improvement. This low value can also be due to the lack of pre-processing done on data prior to the training. And also, the model performs better when the IoU threshold is 0.5 compared to 0.75. This shows that the model struggles to achieve higher localization precision.
- From the Loss trend, the total loss shows a downward curve, which shows that the model is learning. However, the validation loss curve shows that the model might be overfitting. This suggests that the model finds it difficult to generalize.
- The training has been carried out only for around 30 epochs.
- **More analyses and the insights gained, can be discussed further during the meeting.**

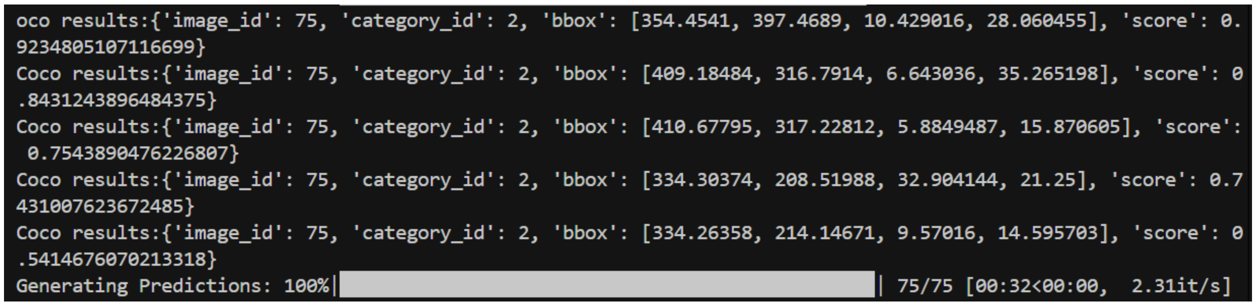
Analyses of the Inference results:

- The inference has been performed on the test set provided. The model has been evaluated using the same metrics and some sample visualization has been provided. The pipeline is designed in a way that sample visualization and evaluation results are saved inorder to revisit it in the future.

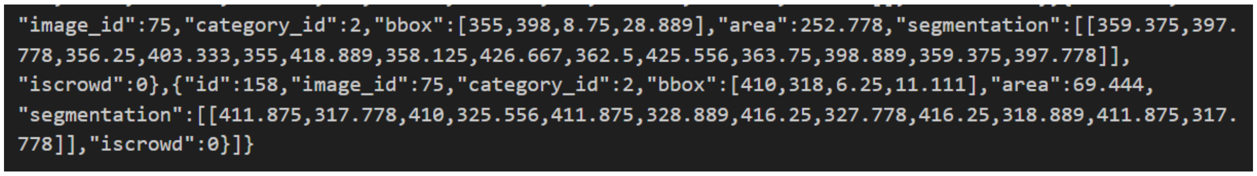


```
{
    "AP": 0.03704337389502775,
    "AP50": 0.0820875379887545,
    "AP75": 0.013432781537441122,
    "APs": 0.03850814754818343,
    "APm": 0.007622158977031306,
    "APl": 0.0243811881188,
    "AR1": 0.03257211538461539,
    "AR10": 0.065625,
    "AR100": 0.065625,
    "ARs": 0.06074074074074074,
    "ARm": 0.03461538461538462,
    "ARl": 0.04375
}
```

- The precision and recall score suggest that the model can be improved much better.
- I believe overfitting is one of the main reasons for the low precision and recall values. This can also be solved by early stopping criteria.



```
oco results:{'image_id': 75, 'category_id': 2, 'bbox': [354.4541, 397.4689, 10.429016, 28.060455], 'score': 0.
9234805107116699}
Coco results:{'image_id': 75, 'category_id': 2, 'bbox': [409.18484, 316.7914, 6.643036, 35.265198], 'score': 0
.8431243896484375}
Coco results:{'image_id': 75, 'category_id': 2, 'bbox': [410.67795, 317.22812, 5.8849487, 15.870605], 'score':
 0.7543890476226807}
Coco results:{'image_id': 75, 'category_id': 2, 'bbox': [334.30374, 208.51988, 32.904144, 21.25], 'score': 0.7
431007623672485}
Coco results:{'image_id': 75, 'category_id': 2, 'bbox': [334.26358, 214.14671, 9.57016, 14.595703], 'score': 0
.5414676070213318}
Generating Predictions: 100%|██████████████████████████████| 75/75 [00:32<00:00,  2.31it/s]
```



```
"image_id":75,"category_id":2,"bbox":[355,398,8.75,28.889],"area":252.778,"segmentation":[[359.375,397.
778,356.25,403.333,355,418.889,358.125,426.667,362.5,425.556,363.75,398.889,359.375,397.778]],
"iscrowd":0},{"id":158,"image_id":75,"category_id":2,"bbox":[410,318,6.25,11.111],"area":69.444,
"segmentation":[[411.875,317.778,410,325.556,411.875,328.889,416.25,327.778,416.25,318.889,411.875,317.
778]],"iscrowd":0}]}
```

- For example, in the above predictions, there are 5 instances of damage detected of the same category. But, in the ground truth there are only 2 instances. Out of the 5 predictions, 2 of them closely match with the 2 ground truths. This shows that the model is biased towards the majority class.
- There are several methods that can be tried to improve the model, such as good data pre-processing techniques that have been suggested previously, experimenting with hyperparameters, tweaking the model architecture etc.

Overall the provided solution gives an end-to-end solution to train, test and deploy a neural network for vehicle damage detection. The solution however does not focus on developing a best model, but on a good pipeline for object detection.

Looking forward to discussing the solution with the team!

References.

[1] . Witzgall, H., & Shen, W. (2022, March). Reducing co-occurrence bias to improve classifier explainability and zero-shot detection. In *2022 IEEE Aerospace Conference (AERO)* (pp. 1-8). IEEE.

[2] . Mittal, U., Chawla, P., & Tiwari, R. (2023). EnsembleNet: A hybrid approach for vehicle detection and estimation of traffic density based on faster R-CNN and YOLO models. *Neural Computing and Applications*, *35*(6), 4755-4774.