**Instance Segmentation using YOLO V8**

Instance segmentation involves not only detecting objects in an image, but also segmenting them into individual instances.

To perform instance segmentation using YOLO, we can follow these general steps:

Train a YOLO model on a dataset that includes annotations for both object detection and segmentation. The annotations should include bounding boxes for the objects as well as segmentation masks for each instance.

Modify the YOLO architecture to output both the bounding boxes and segmentation masks for each detected object. This can be done by adding additional output channels to the network.

During inference, run the YOLO model on an input image and use the output segmentation masks to segment each object into its individual instance. This can be done by applying a segmentation algorithm, such as watershed or mean shift, to the segmentation masks.

Finally, post-process the segmented instances to refine the results and remove any false positives or duplicates.

albumentations is an image augmentation library in Python that allows users to easily apply a variety of transformations to their image datasets. The transformations you mentioned are:

Blur(p=0.01, blur_limit=(3, 7)): This transformation applies a random blurring effect to an image. The p parameter controls the probability of the transformation being applied to an individual image, and the blur_limit parameter controls the range of possible blur radii.

MedianBlur(p=0.01, blur_limit=(3, 7)): This transformation applies a median blur effect to an image. The p and blur_limit parameters function in the same way as for Blur.

ToGray(p=0.01): This transformation converts an image to grayscale. The p parameter controls the probability of the transformation being applied to an individual image.

CLAHE(p=0.01, clip_limit=(1, 4.0), tile_grid_size=(8, 8)): This transformation applies Contrast Limited Adaptive Histogram Equalization (CLAHE) to an image. CLAHE is a technique for enhancing the contrast of an image while avoiding over-amplifying noise or other artifacts. The p parameter controls the probability of the transformation being applied to an individual image, and the clip_limit and tile_grid_size parameters control the specific details of the CLAHE transformation.

These transformations can be used to augment image datasets for tasks such as object detection, image classification, and segmentation. By applying random transformations to images, the resulting dataset can be made more diverse and robust, which can improve the performance of machine learning models.

**The YOLOv8s-seg model is a computer vision model with 195 layers, 11,780,374 parameters, 0 gradients, and 42.4 GFLOPs. The model was evaluated on a dataset containing 73 images, with a total of 102 instances. The overall mAP50 was 0.647, and the mAP50-95 was 0.357.**

**The model was evaluated on two specific classes: "Cracks-and-spalling" and "object". For the "Cracks-and-spalling" class, the model achieved a mAP50 of 0.777 and a mAP50-95 of 0.442, with precision and recall values of 0.698 and 0.788, respectively. For the "object" class, the model achieved a mAP50 of 0.517 and a mAP50-95 of 0.273, with precision and recall values of 0.633 and 0.466, respectively.**

**Overall, the YOLOv8s-seg model performed reasonably well on the evaluation dataset, achieving a moderate mAP50 score and showing promise for detecting instances of the "Cracks-and-spalling" class in particular. However, further evaluation and testing would be necessary to determine the model's performance on other datasets and in other use cases.**

Here is a breakdown of some of the key metrics being tracked:

**Epoch: The current epoch number.**

**GPU_mem: The GPU memory usage during training.**

**box_loss: The loss value for the bounding box regression task.**

**seg_loss: The loss value for the segmentation task.**

**cls_loss: The loss value for the classification task.**

**dfl_loss: The loss value for the deformable convolutional layers (DFL) task.**

**Instances: The number of instances of objects being detected in the training images.**

**Size: The input size of the images during training.**

**After each epoch of training, the model's performance is evaluated on a validation dataset. The evaluation results are shown in the "Class" section, where "all" indicates the overall performance of the model. The following metrics are being tracked for each class:**

**Box(P): Precision of the bounding box predictions.**

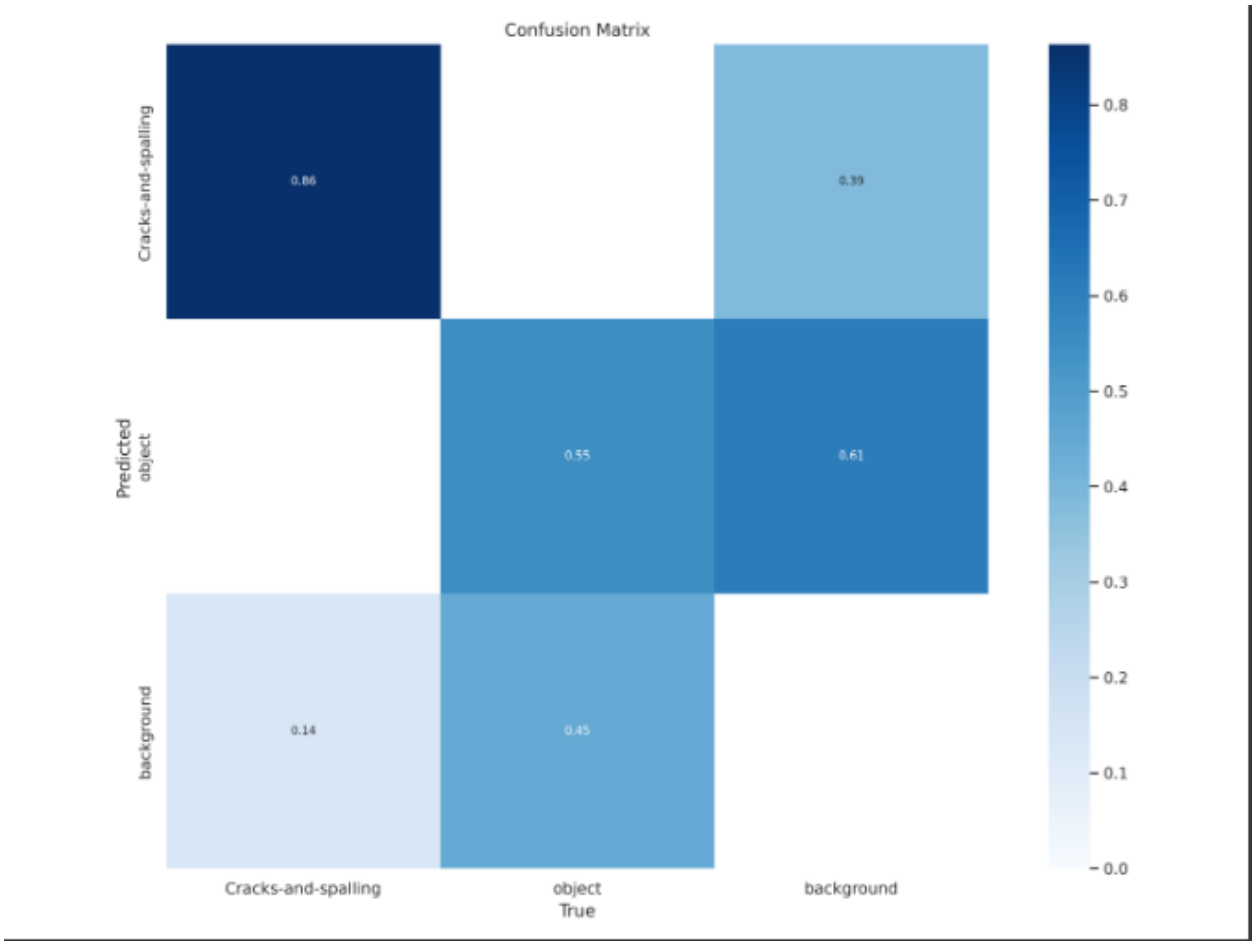
**Box(R): Recall of the bounding box predictions.**

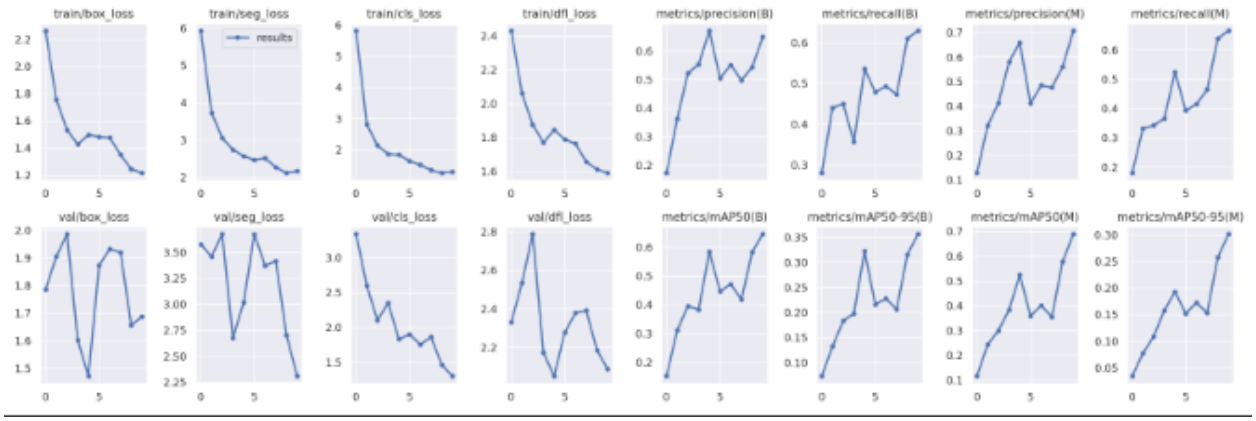
**mAP50: Mean Average Precision (mAP) at 50% intersection over union (IoU) threshold.**

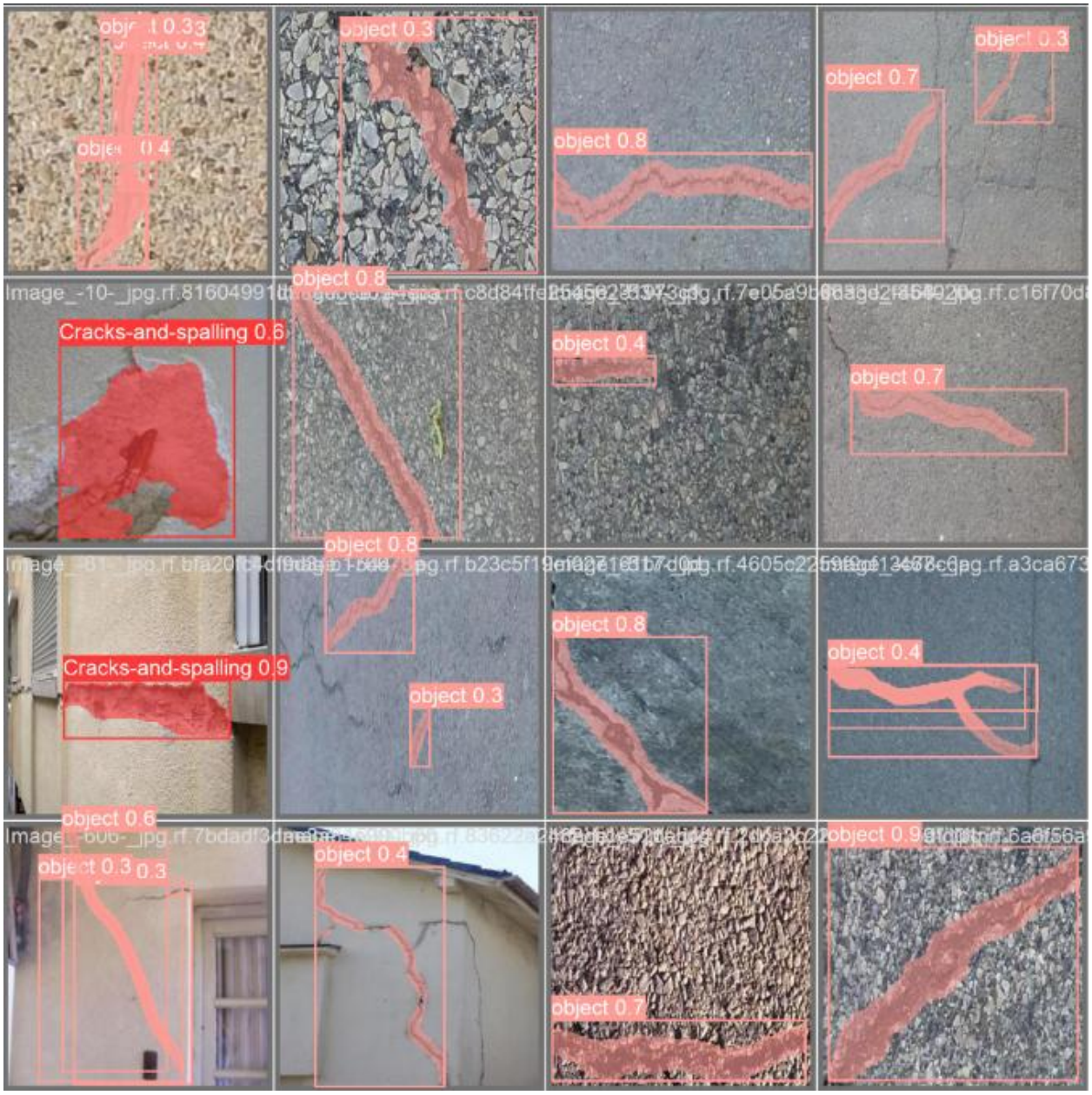
**mAP50-95: mAP averaged over IoU thresholds from 50% to 95%.**

These metrics are used to assess the performance of the object detection model during training and to identify areas that need improvement. The ultimate goal is to maximize the mAP metric, which is a measure of how well the model can detect objects in an image.

Output:




object 0.3
object 0.8
object 0.7
object 0.3
Cracks-and-spalling 0.6
object 0.8
object 0.4
object 0.7
Cracks-and-spalling 0.9
object 0.8
object 0.3
object 0.8
object 0.4
object 0.6
object 0.3
object 0.4
object 0.7