Data Analysis

Data Quality Issues Identified

1. Inconsistent annotations:
    a. Some images had incorrectly labeled bounding boxes, and some where the labels were missing for some bowls
    b. Although, not a big concern here, but in some cases the bounding boxes were not fitting the bowl tightly
    c. Inconsistent image sizes in the dataset
2. Small dataset size limiting model generalization
3. Class imbalance could be a problem here, but that's something which can be mitigated to some extent using appropriate loss functions like Focal loss, etc.

Data Preprocessing and Improvement

I ended up doing a few things to improve the overall dataset quality and have a good dataset to start with before training any models (also to make it easier to train models). I ended up using Robotflow for manual corrections, and also used their data augmentation tool to increase the size of the dataset. This ideally would be implemented in code using Albumentations or similar library, but in this case having visibility into the augmented dataset was helpful.

- Label correction: Manually corrected mislabeled annotations using RoboFlow interface and made the bounding boxes better fit the bowls. Also, took the liberty of creating two classes, empty bowls and filled bowls.
- Through RoboFlow export, I converted the dataset into COCO format to make it a lot easier to import and read.
- Data augmentation: Applied transformations including:
    - Random flips and crops
    - Color jitter
    - Resize, Rotation, Scaling

Recommendations for Larger Dataset Collection

1. Through a systematic data collection pipeline with the robots, more data can be collected to get a better dataset. Or artificial Data Generation is another technique which can help generate some images using simulations along Sim2Real models.

    a. Capture bowls in various lighting conditions
    b. Include different bowl types, sizes, and materials
    c. Vary camera angles and distances
    d. Include partially occluded bowls
2. Annotation strategy is probably the next most important thing to focus on, by using well through-out and consistent annotation guidelines, it can allow us to maintain a good dataset over a long time while there are changes in the robot, hardware, etc.
3. Large datasets also need to be stored appropriately, we can use DVC or git-LFS to store versioned datasets.

Model Training

For model training, I set up the training framework using Pytorch Lightning along with Hydra for config management and Wandb for experiment tracking. I've added the code references below.

References:

- https://github.com/ashleve/lightning-hydra-template
- https://github.com/Iywie/pl_YOLO

Architecture Selection and Justification

Main Models:

- Faster R-CNN with ResNet-50 backbone
- RetinaNet: Single-stage detector with focal loss for handling class imbalance

Two-stage detectors had been the bread and butter of object detection models before YOLO based models came in and they have been proven to work well with simple implementations, which is why I started experimenting with them. Their main advantages: Good accuracy, mature implementation, good generalization. But, they can be slower in inference compared to single-stage detectors which is why I wanted to try and integrate newer YOLO models which didn't work out as I expected. In hindsight, experimenting with the original Darknet YOLO models without BiFPN, and multi-scale backbones would have been the better route.

Alternative Models Implemented (not working):

- YOLOX and YOLOv7: Attempted integration but encountered configuration issues with third part pl_YOLO. Alternative is to obviously integrate original YOLOX and YOLOv7 codebases but that takes time.

Evaluation

Key Metrics

- Primary metric: Mean Average Precision (mAP)
- Secondary metric: Complete Intersection over Union (CIOU)

Test Metric	Test Data	Validation Data
test/ciou	-0.3018110990524292	-0.3248564302921295
test/mAP	0.3550005853176117	0.5670316815376282

The model doesn't perform very well given the limited size of the dataset, but it can be improved with more data and possibly better loss functions in this case.

Extension

1. **Quad Detections/Bowl Orientation Prediction :** Extend current object detection to figure out the orientation of the detected bowls. There are different approaches that can be pursued here:
    a. Keypoint detection : Define rim points for orientation estimation, and those points can also be used to detect quads on the bowl instead of axis-aligned bounding boxes.
    b. Multi-task learning with an additional classification head or within the same classification head define custom output vectors with additional parameters like orientation, emptiness, etc.
    c. Implement 6DoF pose estimation using PnP algorithms
2. **Extend current model to detect both bowls and ingredients :** Classify the food that has already been placed inside the bowls. This can allow the robot to interact with a variety of ingredients and place them in the correct bowl depending upon the type of food already in them.
    a. This can be done through a two stage detector where the first stage detects the bowl, and the second stage focusses (maybe through some heuristics depending upon the type of the bowl) on cropping the image to the ingredients and identifying them using the second stage.
    b. Multi-task learning model (Something like Hydranet)