به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

دانشکده مهندسی مکانیک

رباتیک و مکاترونیک

مینی پروژه شماره ۴

نام و نام خانوادگی	محیا شهشهانی-شیرین جمشیدی
شماره دانشجویی	۸۱۰۱۹۹۵۹۸-۸۱۰۱۹۹۵۷۰
تاریخ ارسال گزارش	۲۱ تیر ۱۴۰۳

Table of Contents

Problem 1: YOLO different versions

YOLO (You Only Look Once) is a popular real-time object detection system that has evolved significantly over time. Here are key differences in each version compared to its predecessor:

- **YOLO v1**: Introduced a single convolutional network for simultaneous object detection and classification.
- **YOLO v2**: Added batch normalization and anchor boxes, improving accuracy and stability.
- **YOLO v3**: Implemented multi-scale predictions with a Feature Pyramid Network (FPN) for better detection of small objects.
- **YOLO v4**: Integrated "Bag of Freebies" and "Bag of Specials" techniques for improved speed and accuracy without extra computational cost.
- **YOLO v5**: Focused on ease of use, deployment efficiency, and integration with frameworks like PyTorch.
- **YOLO v6**: Optimized for hardware accelerators, enhancing deployment on edge devices.
- **YOLO v7**: Introduced E-ELAN for better gradient flow and network performance, balancing speed and accuracy.
- **YOLO v8**: Added Transformer layers and a modular design for advanced feature representation and customization.

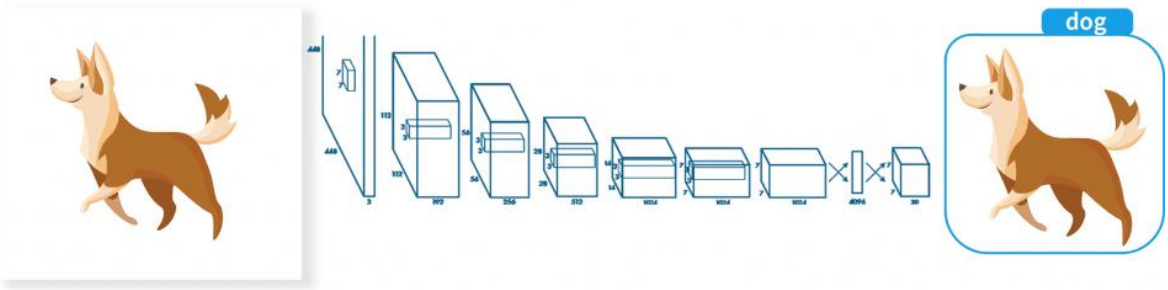


Figure 1-1: YOLO; a real-time object detection network

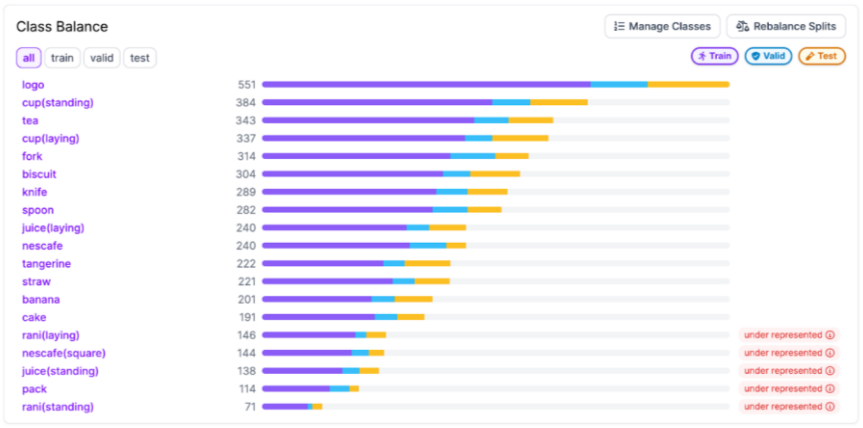
Problem 2: mAP score

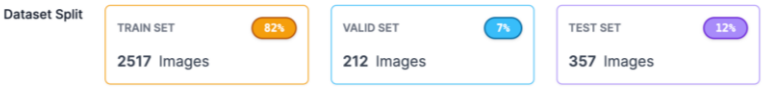
Here's a step-by-step description of how mAP is calculated:

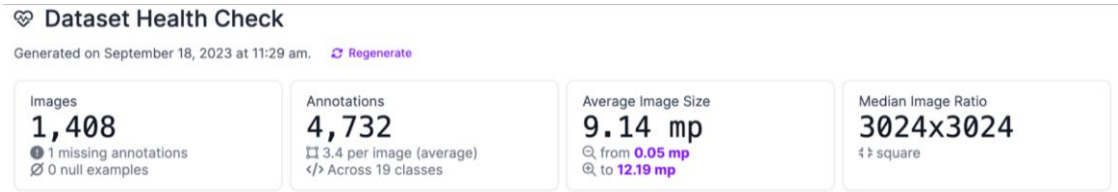
1. Prediction and Ground Truth:
    - For a given dataset, the model makes predictions for the locations and classes of objects in each image.
    - The ground truth data contains the actual bounding boxes and classes for these objects.
2. Compute IoU (Intersection over Union):
    - For each predicted bounding box, compute the Intersection over Union (IoU) with the ground truth bounding boxes.
    - IoU is calculated as the area of overlap between the predicted and ground truth bounding boxes divided by the area of their union.
3. Match Predictions with Ground Truth:
    - Assign predictions to ground truth objects based on a threshold IoU value (commonly 0.5). If IoU > threshold, the prediction is considered a true positive (TP).
    - Predictions with no matching ground truth boxes are false positives (FP).
    - Ground truth boxes with no corresponding predictions are false negatives (FN).
4. Calculate Precision and Recall:
    - Precision: The ratio of true positives to the total number of positive predictions (TP / (TP + FP)).
    - Recall: The ratio of true positives to the total number of actual positives (TP / (TP + FN)).
5. Precision-Recall Curve:
    - For each class, plot the precision-recall curve by varying the confidence threshold for the predictions from 0 to 1.
    - Calculate precision and recall at multiple points along this curve.
6. Average Precision (AP):
    - Calculate the Average Precision (AP) for each class. This is typically done by computing the area under the precision-recall curve.
    - One common approach is to use the 11-point interpolation method where precision is sampled at recall levels 0, 0.1, 0.2, ..., 1.0 and then averaged.
7. Mean Average Precision (mAP):
    - Compute the AP for each class in the dataset.
    - The mean Average Precision (mAP) is the average of the AP values for all classes.

Problem 3: Project dataset

1. There are 1408 images, with only 1 image missing annotations, ensuring high data quality. An average of 3.4 annotations per image across 19 classes, providing a rich set of labeled data for training. 3024x3024, indicating that most images are square-shaped and high-resolution. 82% of data is used for training, 7% for validation and 12% for test set.







2.

- Increase Dataset Size: Augmentation helps create more training data from existing data, making the model more robust.

- Enhance Model Generalization: By introducing variations, the model learns to generalize better to unseen data.

- Simulate Real-World Variability: Augmentation can mimic different lighting, angles, and distortions that the model might encounter in real-world scenarios.

3. Based on the health check data on Roboflow, the applied preprocessing and augmentation and their help is like below:

Preprocessing:

Auto Orient: Automatically adjusts the orientation of the images. Ensures that all images have a consistent orientation, preventing errors due to incorrect image orientation and making the training process smoother.

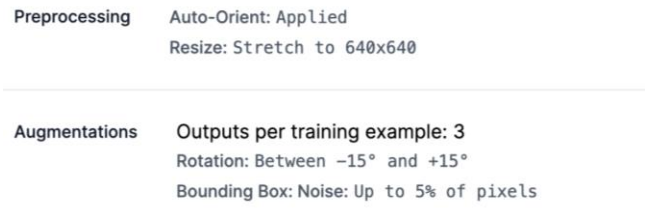
Resize: Stretching images to a standard size of 640x640 pixels. ensuring uniformity in input dimensions for the model.

Augmentations:

Outputs per Training Example: 3 augmented versions are generated for each training example, increasing the dataset size and variability.

Rotation: Images are randomly rotated between -15 and +15 degrees. This helps the model learn to recognize objects from slightly different angles, which is crucial for robustness.

Bounding Box Noise: Adds noise up to 5% of the bounding box dimensions, simulating minor inaccuracies in bounding box placement. This helps the model become more tolerant of slight errors in object localization.



Problem 4: Object detection

With the code below we were able to train our model.

```
!yolo mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=50 imgsz=640 plots=True
```

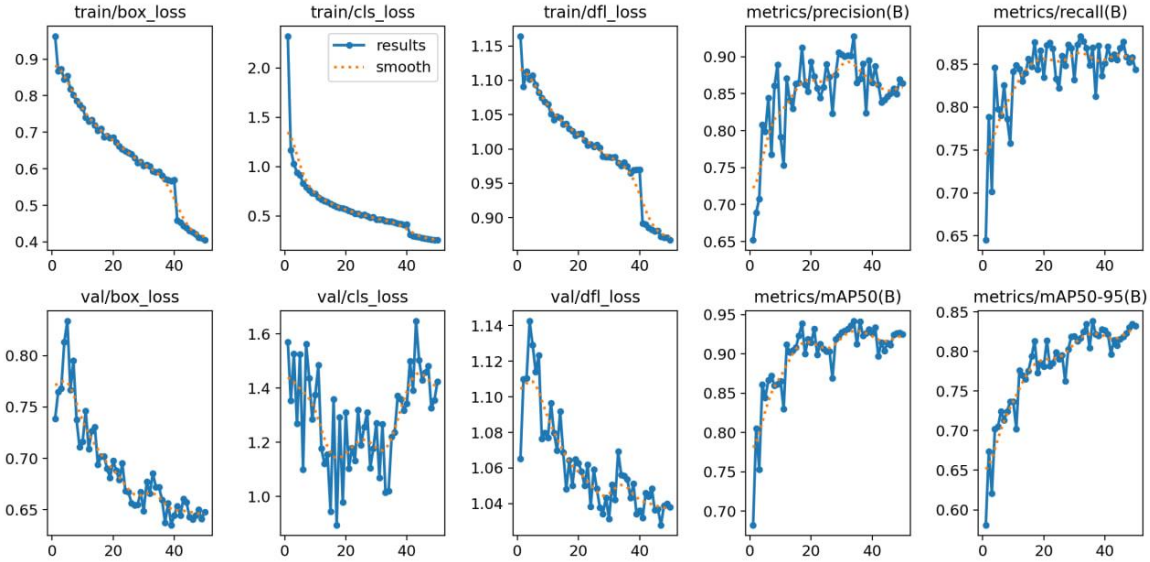


Figure 4-1: Training evaluation plot

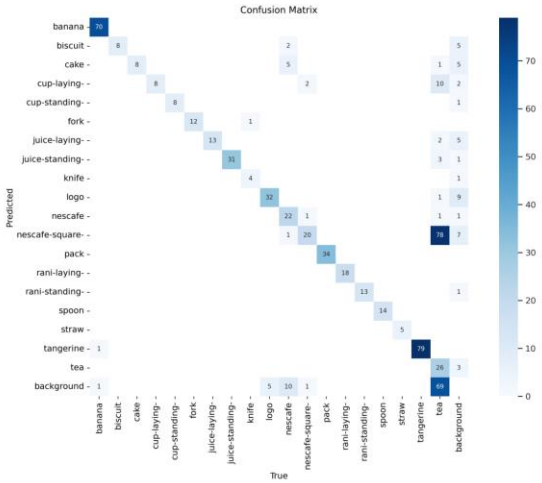


Figure 4-2: Configuration Matrix

Training configuration:
Model: YOLOv8
Training Dataset: Custom dataset (grasp-6-7)
Epochs: 50
Batch Size: 16
Image Size: 640
Optimizer: AdamW (auto configured)
GPU: Tesla T4

Model performance:

The model shows a steady improvement in mAP (mean Average Precision) across the epochs, indicating better performance in detecting objects over time. Notably, the mAP@50-95 metric, which considers various IoU thresholds, improves from 0.581 in the first epoch to 0.786 by the 20th epoch, reflecting enhanced localization and classification accuracy.

Loss Metrics:

Box Loss: Decreases steadily from 0.9628 to around 0.6839, showing better bounding box regression.

Class Loss: Drops significantly, indicating improved classification accuracy.

DFL Loss: Shows slight variations but generally trends downward, demonstrating better distributional alignment for object localization.

Precision and Recall: Both metrics exhibit an upward trend, with precision improving more consistently than recall. By epoch 20, precision reaches 0.853 and recall 0.866, suggesting the model's predictions are both accurate and consistent.

4. Average time per epoch is approximately 62 seconds.

5. Here are the results for testing the model:

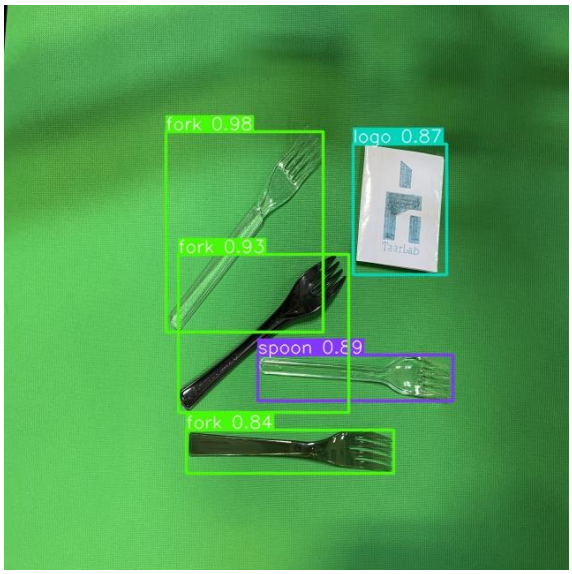


Figure 4-3: A good bounding box but one false classification

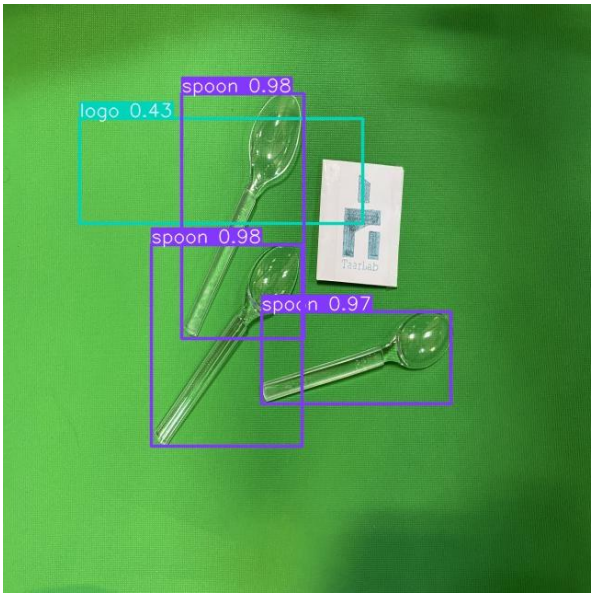


Figure 4-4: Location of the located logo is false

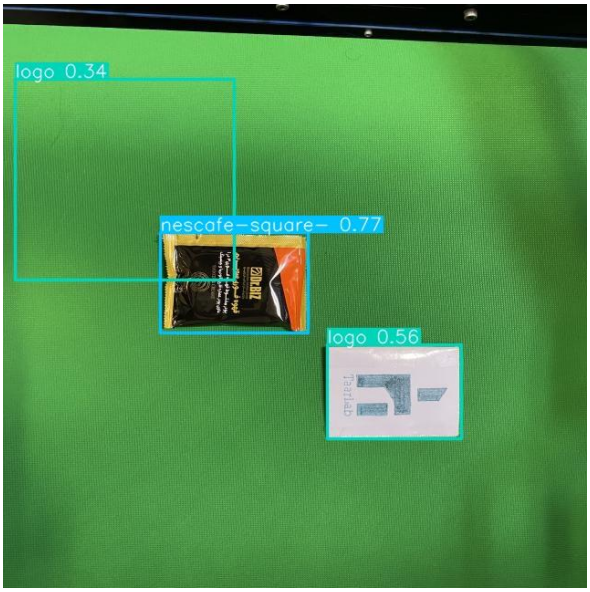


Figure 4-5: two correct detections. One false.

But there were some tests that our model predicted correctly.

Figure 4-6: Correct Classification

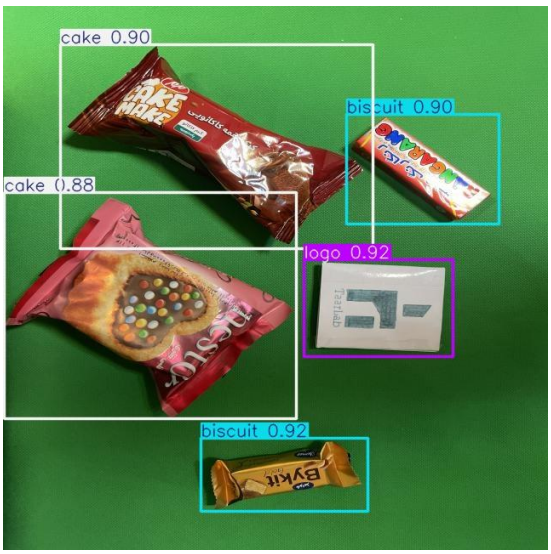


Figure 4-7: Correct Classification

Figure 4-8: Correct Classification

But why do false positives happen?

1. Ambiguous or Similar Classes:

Classes that look very similar to each other (e.g., forks and spoons) can be challenging for the model to differentiate, especially if they have not been clearly distinguished in the training data.

2. Insufficient or Imbalanced Training Data:

Insufficient Data: The model might not have seen enough examples of certain objects during training.

Imbalanced Data: If some classes are overrepresented compared to others, the model might perform better on those classes and worse on underrepresented ones.

3. Quality of Annotations:

Poorly annotated training data (e.g., incorrect labels, imprecise bounding boxes) can lead to poor model performance.

4. Environmental Factors:

Shadows, reflections, and other environmental factors can confuse the model, leading to false positives.

Figure 4-9: Correct Classification

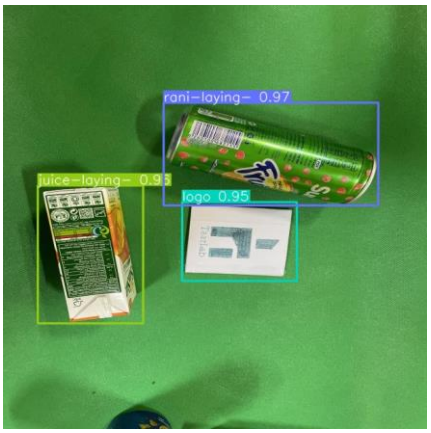


Figure 4-10: Correct Classification

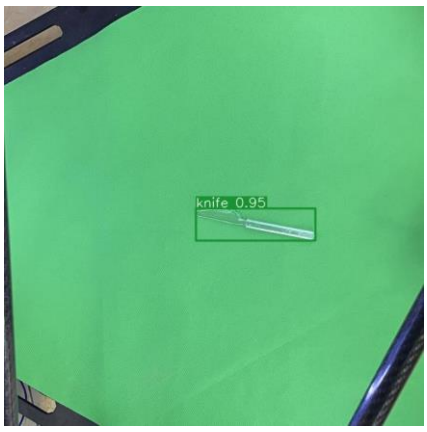


Figure 4-11: Correct Classification

Problem 5: Object segmentation

To do the segmentation using FastSam, we picked some correctly detected pictures and saved their bounding boxes xy. By applying FastSam on them, we were able to do the object segmentation.

More details can be found in our notebook file.

Here are some object segmentation outputs, visualizing the output of SAM on picture:

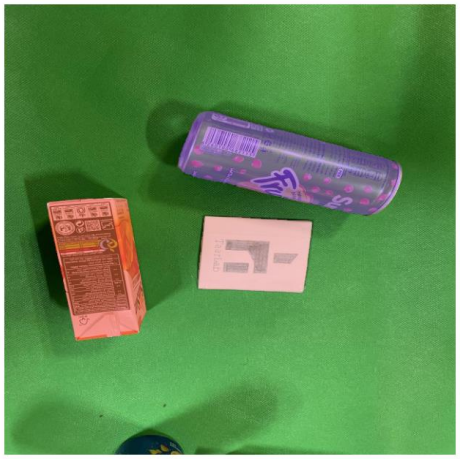


Figure 5-1: Image segmentation output no.1

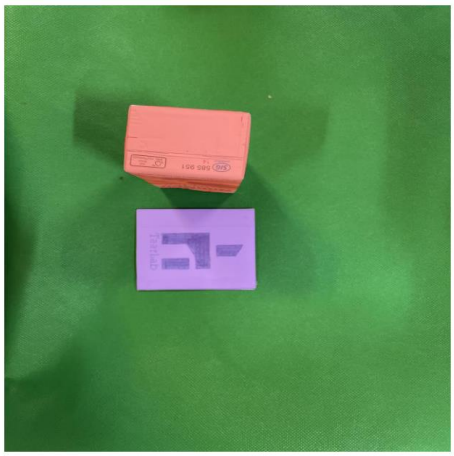


Figure 5-2: Image segmentation output no.2

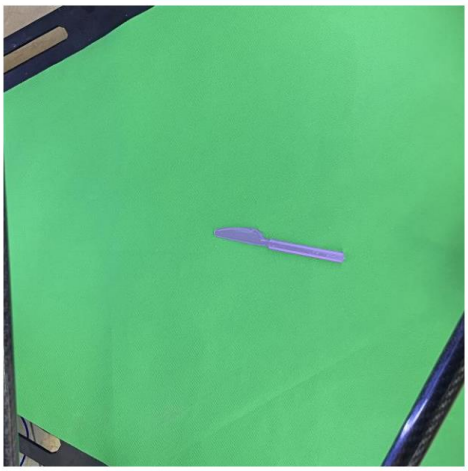


Figure 5-3: Image segmentation output no.3

Figure 5-4: Image segmentation output no.4

Problem 6: Grasp point generation

We did this process in the following steps:

- Boundary Extraction:
  Using `cv2.findContours`, we extract object boundaries from segmentation masks and fit rotated rectangles to these contours for orientation.

- Center Determination:
  The center of each object is found using `cv2.minAreaRect`, providing the reference point for grasp calculations.

- Grasp Point Calculation:
  Lines are drawn through the object's center in two directions based on its orientation angle. These lines intersect with the object's boundary, identified using Shapely's geometric operations.

- Selection of Grasp Points:
  The intersection points farthest from the center along each line are selected as grasp points. These points are chosen by computing their distances from the center and selecting the maximum values.

- Visualization:
  Grasp points are visualized on the image using green circles, aiding in understanding their placement for robotic manipulation tasks.

Grass points are generated for objects in images. (if there are several just for one):

Figure 6-1: Grasp point for a juice

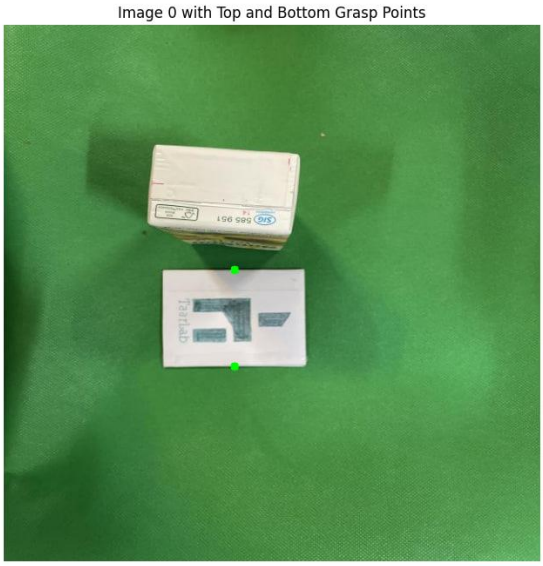


Figure 6-2: Grasp point for a logo (card)

Figure 6-3: Grasp point for a knife

Figure 6-4: Grasp point for a cake (package)

You can find our codes in the notebook attached for more details of the whole project.