به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

دانشکده مهندسی مکانیک

رباتیک و مکاترونیک

مینی پروژه شماره ۴

نام و نام خانوادگی	نورا زارعی – عرفان عسگری
شماره دانشجویی	۸۱۰۱۹۹۴۶۰ – ۸۱۰۱۹۹۴۳۳
تاریخ ارسال گزارش	۱۴۰۳/۰۴/۲۱

Table of Contents

Problem 1 – YOLO different versions

YOLO (You Only Look Once) is a popular object detection algorithm that has undergone several iterations, each improving upon its predecessor. Here’s a brief overview of key differences between each version.

YOLOv1 to YOLOv2:

- Batch Normalization: YOLOv2 introduced batch normalization on all convolutional layers, which helped stabilize training and significantly improved convergence.
- High-Resolution Classifier: YOLOv2 increased the input resolution to 416x416, compared to the 448x448 in YOLOv1, which improved the detection of smaller objects.

YOLOv2 to YOLOv3:

- Residual Blocks: YOLOv3 incorporated residual blocks from the ResNet architecture, which helped in training deeper networks by addressing the vanishing gradient problem.
- Feature Pyramid Network (FPN): YOLOv3 introduced a multi-scale detection feature, allowing it to predict bounding boxes at three different scales, improving accuracy for small and large objects.

YOLOv3 to YOLOv4:

- Cross Stage Partial Network (CSPNet): YOLOv4 used CSPNet to reduce computation by partitioning the feature map into two parts and merging them through a cross-stage hierarchy, enhancing the gradient flow.
- Mish Activation Function: YOLOv4 replaced some of the leaky ReLU activations with the Mish activation function, which provided smoother gradients and better overall performance.

YOLOv4 to YOLOv5:

- Ultralytics Implementation: YOLOv5, developed by Ultralytics, focused on ease of use and integration with PyTorch, leading to faster training and inference times.
- Auto-Learning Bounding Box Anchors: YOLOv5 included automated learning of bounding box anchors, optimizing them during the training process to improve detection accuracy.

YOLOv5 to YOLOv6:

- Anchor-Free Detection: YOLOv6 introduced anchor-free detection, which simplified the model architecture and reduced computational complexity.
- Decoupled Head: YOLOv6 employed a decoupled head, separating the classification and localization tasks, improving the detection performance.

YOLOv6 to YOLOv7:

- Reparametrized Convolutions: YOLOv7 introduced reparametrized convolutional layers that enhanced model efficiency by decoupling training and inference stages, resulting in faster inference without sacrificing accuracy.
- Dynamic Head: YOLOv7 implemented a dynamic head architecture, which dynamically adjusts the network structure during training, leading to improved performance on various object detection tasks.
- Additional Features: YOLOv7 included various architectural refinements, such as better activation functions and normalization techniques, to further boost detection accuracy and speed.

YOLOv7 to YOLOv8:

- Enhanced Backbone Network: YOLOv8 featured an improved backbone network with a focus on better feature extraction and reduced computational cost, leveraging advanced techniques like EfficientNet-based architectures.
- Attention Mechanisms: YOLOv8 integrated attention mechanisms, such as SE (Squeeze-and-Excitation) blocks or CBAM (Convolutional Block Attention Module), to better focus on relevant parts of the image, improving object detection accuracy.
- Improved Loss Function: YOLOv8 utilized a more sophisticated loss function that better balanced localization and classification errors, leading to enhanced overall detection performance.

YOLOv8 to YOLOv9:

- Transformer Integration: YOLOv9 incorporated transformer-based modules, inspired by advancements in vision transformers (ViTs), to better capture global context and relationships between objects in the image.
- Hybrid Model Architecture: YOLOv9 combined the strengths of convolutional neural networks (CNNs) and transformers, leading to a more powerful and flexible model capable of handling diverse object detection challenges.
- Advanced Training Techniques: YOLOv9 employed advanced training techniques, such as self-supervised learning and data augmentation strategies, to further enhance model robustness and generalization capabilities.

Problem 2 – mAP score

The mean Average Precision (mAP) is a popular evaluation metric used to assess the performance of object detection models, including YOLO.

Steps to Calculate mAP:

1. Precision and Recall:

- Precision is the ratio of true positive detections (correctly detected objects) to the total number of positive detections (both true positives and false positives). It indicates how many of the detected objects are actually correct.
- Recall is the ratio of true positive detections to the total number of ground truth objects (true positives and false negatives). It indicates how many of the actual objects are detected by the model.

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)}$$

$$Recall = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Negatives\ (FN)}$$

2. Precision-Recall Curve:

- For each class, compute the precision and recall at various thresholds. These thresholds determine whether a detection is considered a positive or a negative.
- Plot the precision-recall curve, with precision on the y-axis and recall on the x-axis.

3. Average Precision (AP):

- To calculate the AP for each class, integrate the area under the precision-recall curve. This can be done using different interpolation methods, but the most common is the 11-point interpolation.
- The 11-point interpolation method involves calculating precision at 11 equally spaced recall levels (0, 0.1, 0.2, ..., 1.0). The precision at each recall level is the maximum precision obtained for that recall level or any higher recall level.

$$AP = \frac{1}{11} \sum_{r \in (0,0.1,0.2,\dots,1.0)} \max_{\tilde{r} \geq r} maxPrecision(\tilde{r})$$

4. Mean Average Precision (mAP):

- Calculate the AP for each class in the dataset.
- The mAP is the mean of the AP values for all classes.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

Where N is the number of classes, and AP_i is the average precision for class i.

Problem 3 – Project dataset

1. To analyze this dataset regarding its even distribution throughout different classes, we can summarize the data provided in the screenshots.

Dataset Summary:

Total Images: 1408

- Training Set: 2517 images (82%)
- Validation Set: 212 images (7%)
- Test Set: 357 images (12%)

Average Image Size: 9.14 MP (ranging from 0.05 MP to 12.19 MP)

- Median Image Ratio: 3024x3024 (square)

Class Distribution:

Here is the number of images and instances for each class:

1. Logo: 551
2. Cup (standing): 384
3. Tea: 343
4. Cup (laying): 337
5. Fork: 314
6. Biscuit: 304
7. Knife: 289
8. Spoon: 282
9. Juice (laying): 240
10. Nescafe: 240
11. Tangerine: 222
12. Straw: 221
13. Banana: 201
14. Cake: 191
15. Rani (laying): 146
16. Nescafe (square): 144
17. Juice (standing): 138
18. Pack: 114
19. Rani (standing): 71

Observation:

- The class distribution is not even. Some classes like "Logo" (551 instances) and "Cup (standing)" (384 instances) are well-represented, whereas classes like "Rani (standing)" (71 instances) and "Pack" (114 instances) are underrepresented.
- Several classes are marked as underrepresented, indicating a need for balancing the dataset to avoid bias in model training.

2.

Increase Dataset Size: Augmentation artificially increases the size of the dataset by generating new images through transformations of the original images. This is particularly useful when the available data is limited.

Enhance Model Generalization: Augmented data introduces variability in the training data, helping the model to generalize better to unseen data. This means the model can perform well not just on the training data but also on new, unseen images.

Reduce Overfitting: By providing more diverse examples, augmentation reduces the chances of the model memorizing the training data. This helps in reducing overfitting, where the model performs well on training data but poorly on validation and test data.

Simulate Real-World Variations: Real-world data can have various transformations like rotations, translations, noise, and lighting changes. Augmentation helps simulate these variations, making the model robust to such changes when deployed in real-world scenarios.

Improve Model Performance: More diverse and extensive training data generally leads to better-performing models. Augmentation can help achieve higher accuracy and better performance metrics by providing a richer training set.

3. In the provided dataset:

- Rotation: Between -15° and +15°
- Bounding Box Noise: Up to 5% of pixels

These augmentations ensure that the model sees slightly varied versions of the same image, helping it to learn more robust features.

Problem 4 – Object detection

We start by importing the dataset from Roboflow. We use YOLOv8 for this project. YOLOv8 is preferred due to its improved accuracy and speed over previous versions. We train the model on the dataset for 50 epochs. The training process involves data augmentation and logging of results.

The training report provides a summary of the model architecture and training progress. The YOLOv8 model consists of 225 layers with 3,014,553 parameters. Box Loss measures the error in predicted bounding boxes. Class Loss measures the error in class predictions. DFL Loss measures the error in object localization. During training, various performance metrics (Precision, Recall, mAP) were logged. For instance: Epoch 1: Precision: 0.416, Recall: 0.491, mAP50: 0.449, mAP50-95: 0.379. Epoch 50: Precision: 0.898, Recall: 0.878, mAP50: 0.936, mAP50-95: 0.828. The average training time per epoch was approximately 58 seconds. This was influenced by the available GPU resources (Tesla T4).

After training, we evaluate the model on the provided test images.

Here is a summary of the results for the YOLOv8 prediction on the test set. Images: 212, Instances: 620, Precision: 0.896, Recall: 0.878, mAP@50: 0.936, mAP@50-95: 0.828, Speed of Preprocess: 0.6ms, Speed of Inference: 6.2ms, Speed of Postprocess: 5.5ms.

Many classes have precision close to or at 1, indicating few false positives. While some classes like banana, juice-laying, and rani-laying have perfect recall, others like nescafe and tea have lower recall values, indicating some false negatives.

The YOLOv8 model shows good performance on the training and validation sets, with increasing precision and recall over the epochs. However, some classes like "tea" had lower performance due to potential under-representation in the dataset or high variability in object appearance.

Common Issues: Certain classes were underrepresented, affecting model accuracy for those classes. While augmentation helps, excessive transformations can introduce noise. Accurate annotations are crucial; any errors can lead to incorrect predictions.

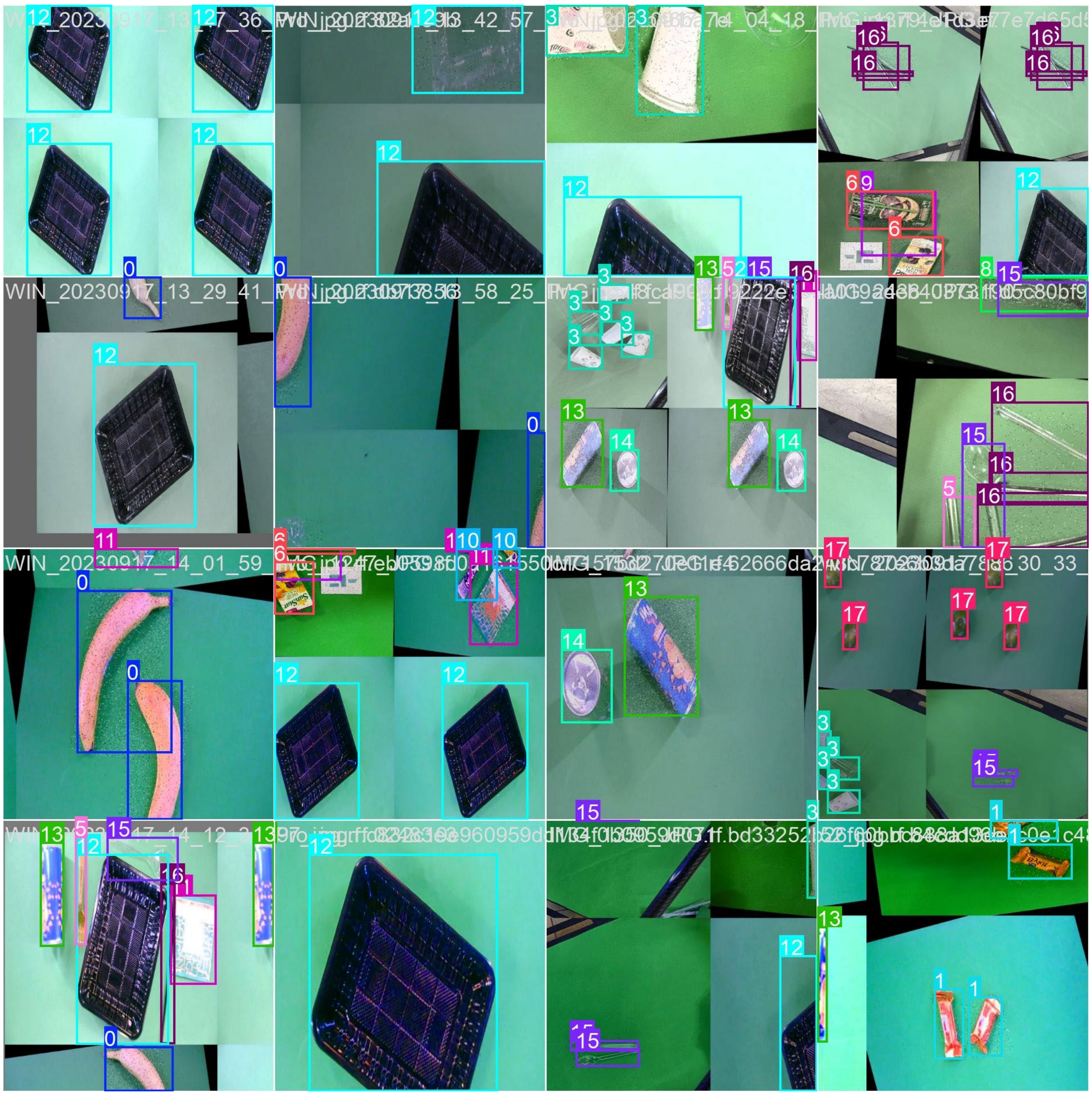
Figure 1 Applying model on train set

Now, we have some validation data, in left pictures the actual labels, and in right pictures the predicted labels are shown.

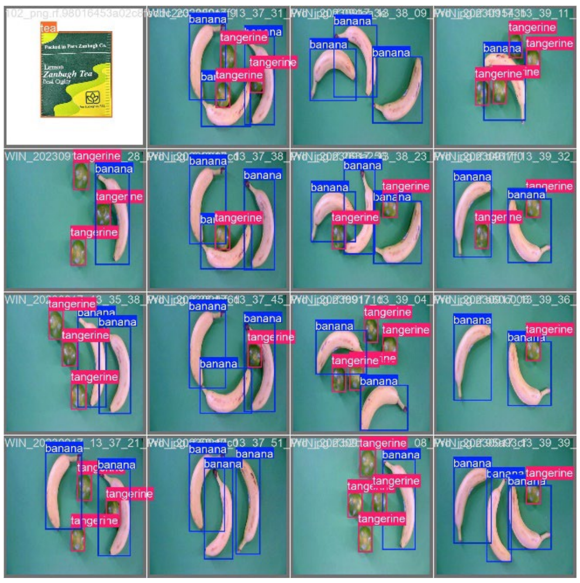

Figure 2 actual labels

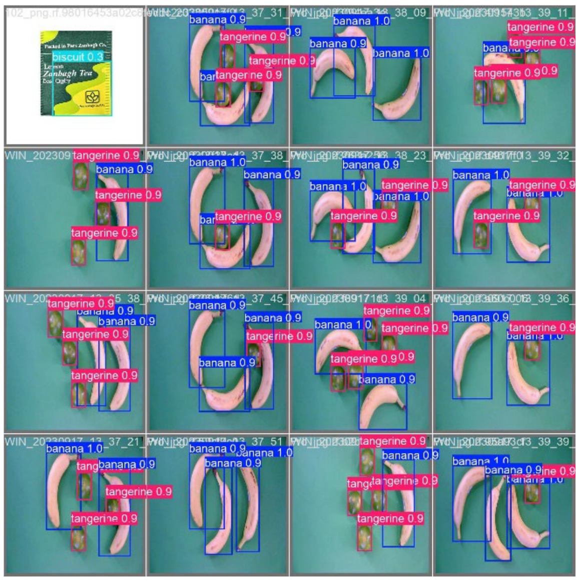

Figure 3 predicted labels

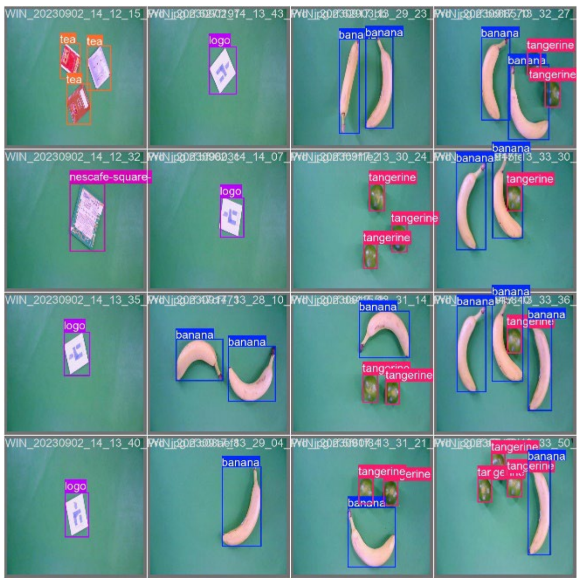

Figure 4 actual labels

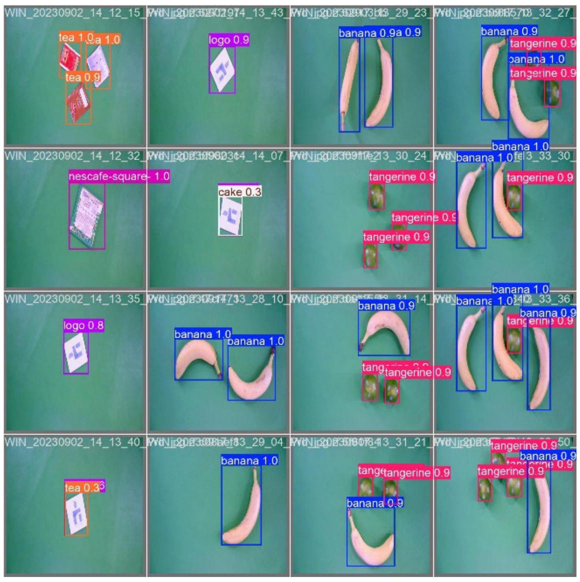

Figure 5 predicted labels

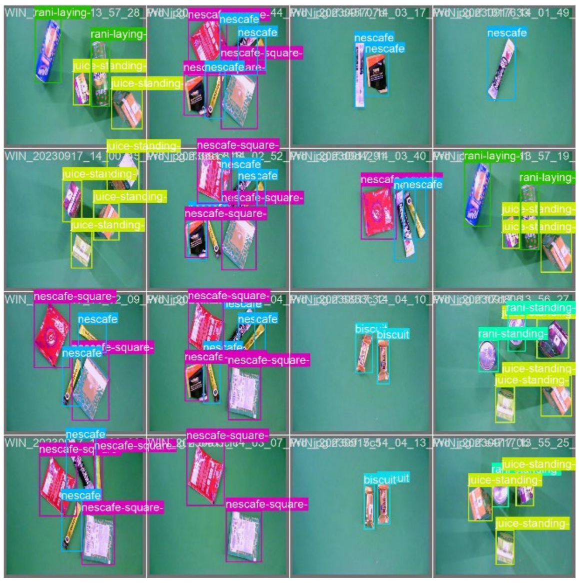

Figure 6 actual labels

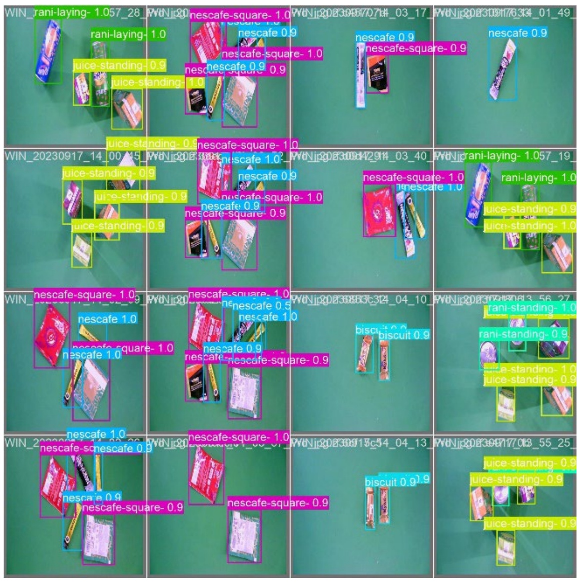

Figure 7 predicted labels

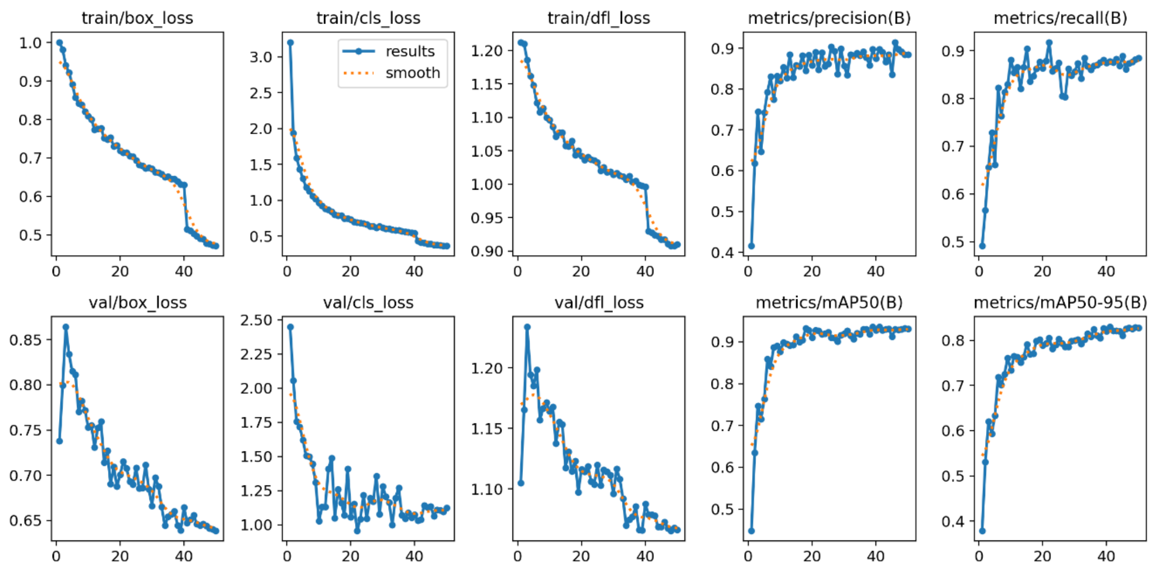

Figure 8 Results of training model

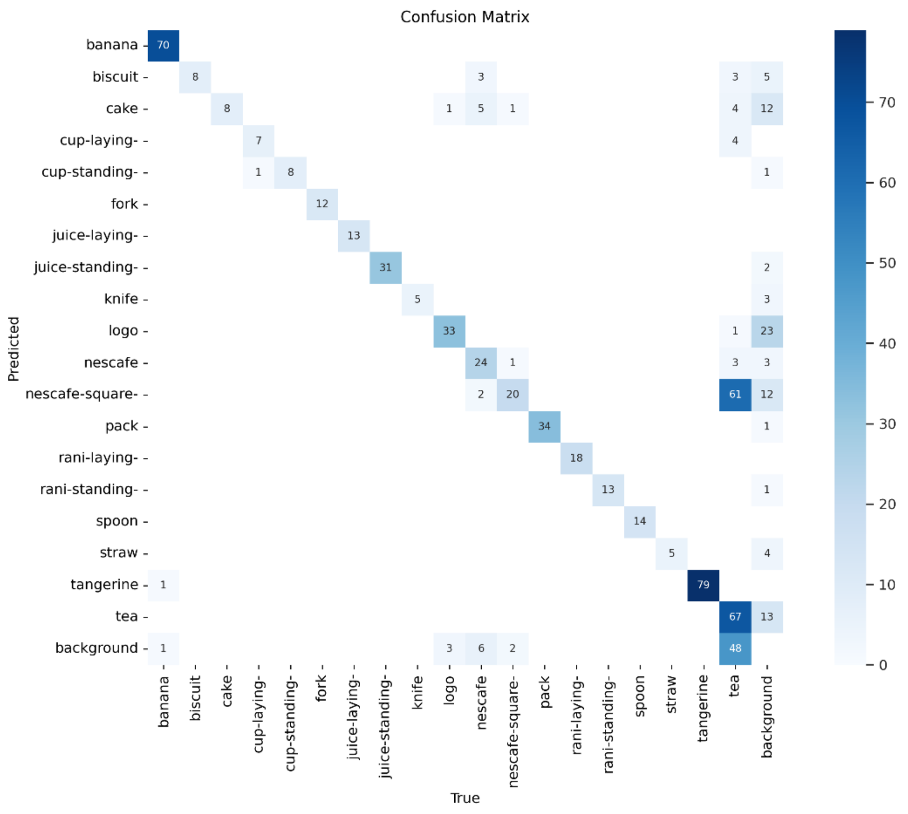

Figure 9 Confusion Matrix

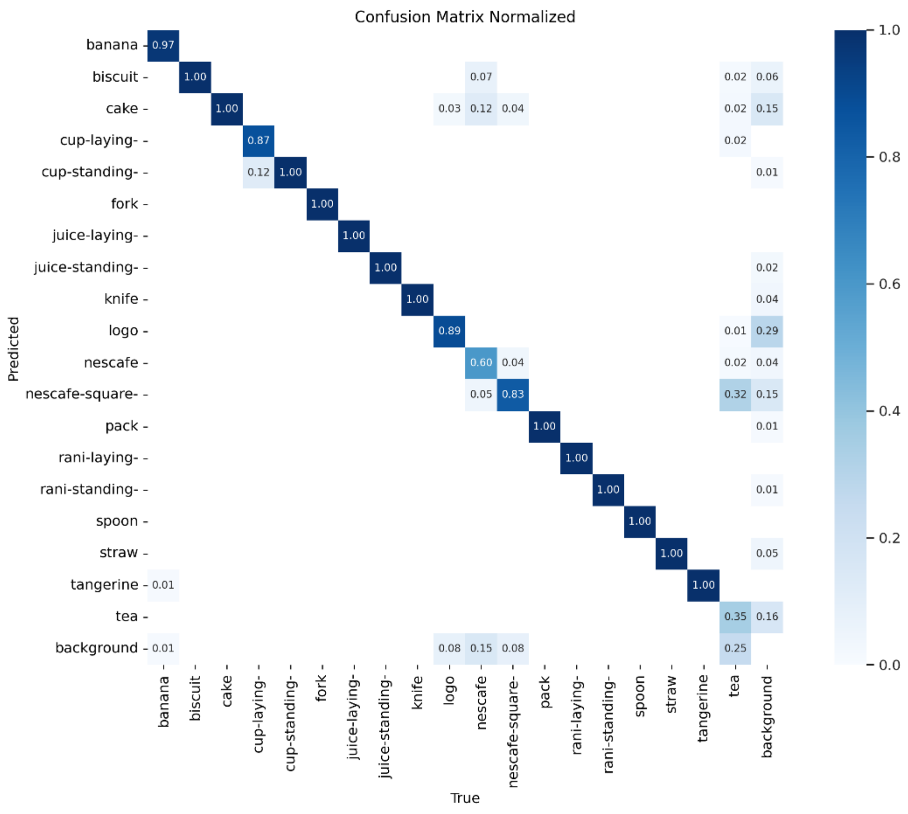

Figure 10 Confusion Matrix Normalized

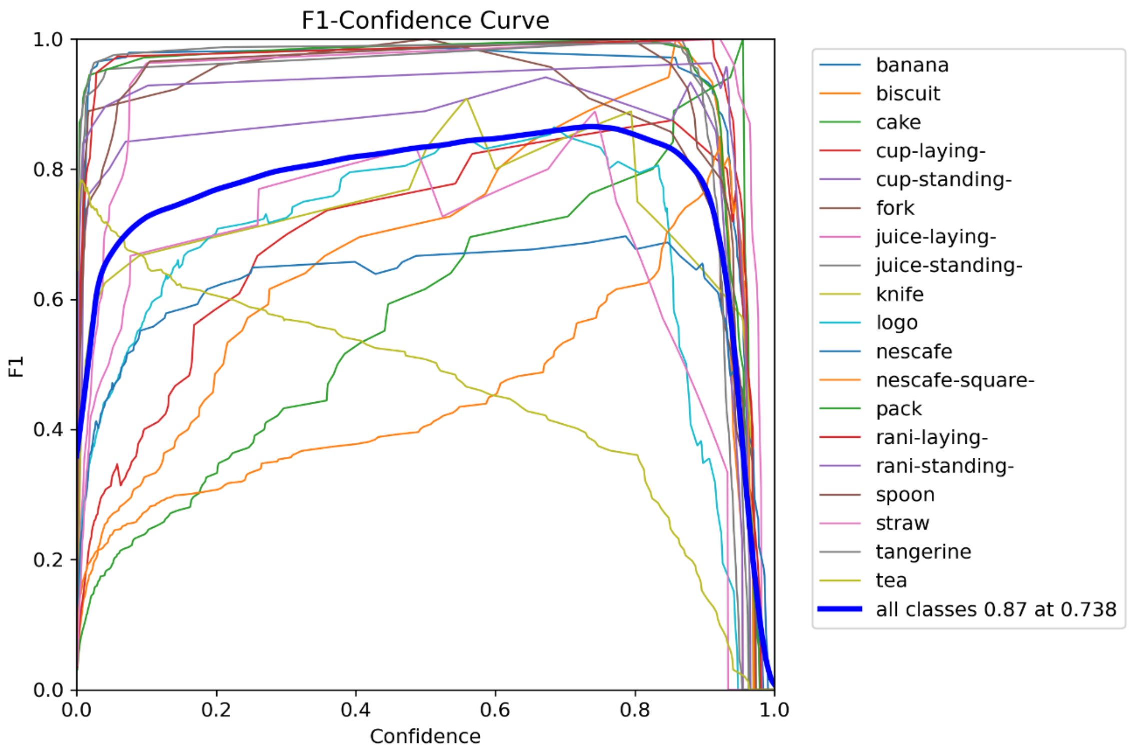

Figure 11 F1-Confidence curve for model

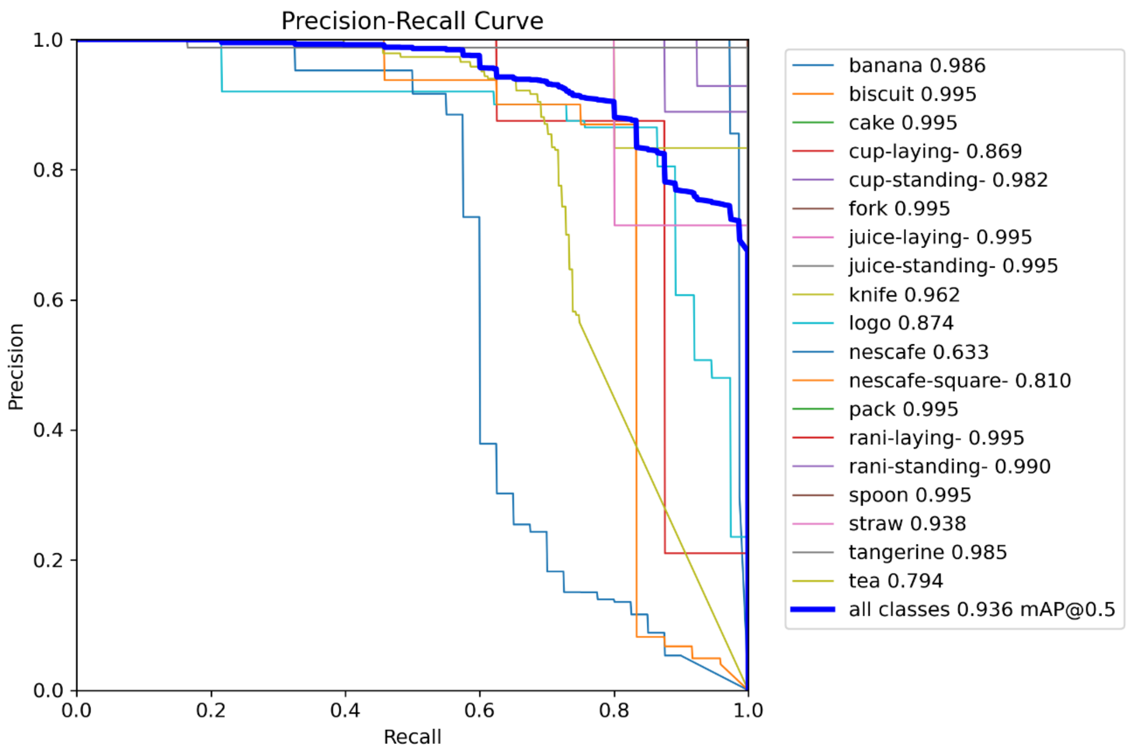

Figure 9 Precision-Recall curve for model

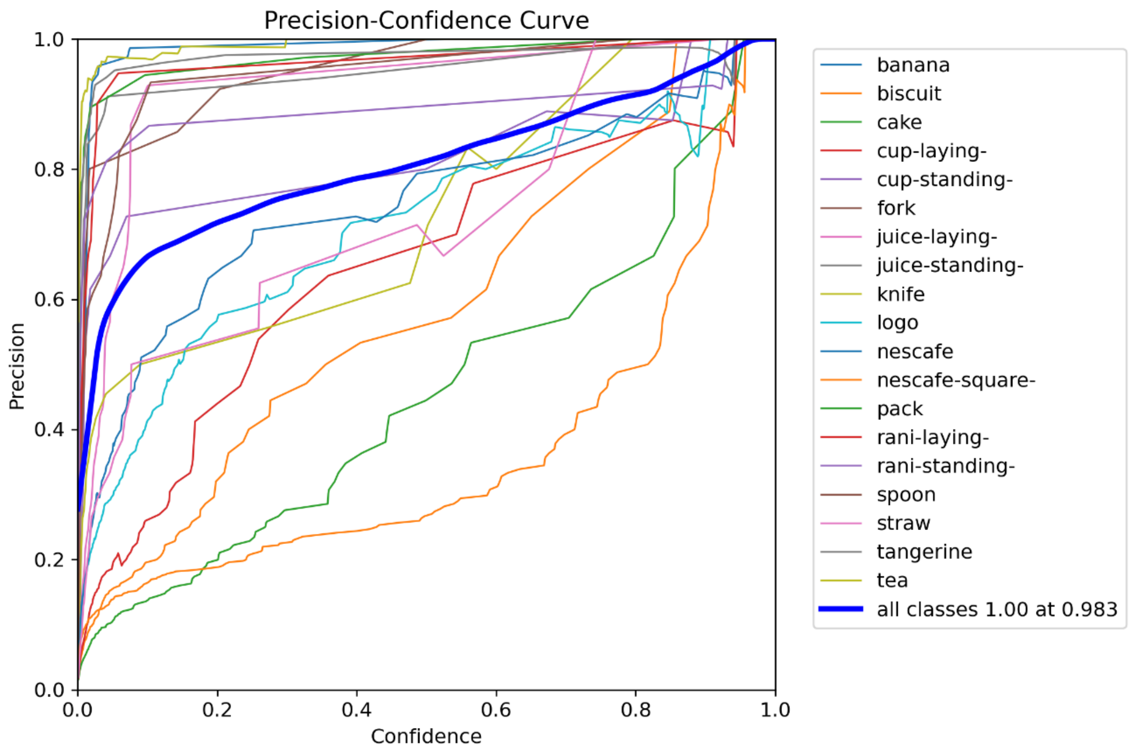

Figure 10 Precision-Confidence curve for model

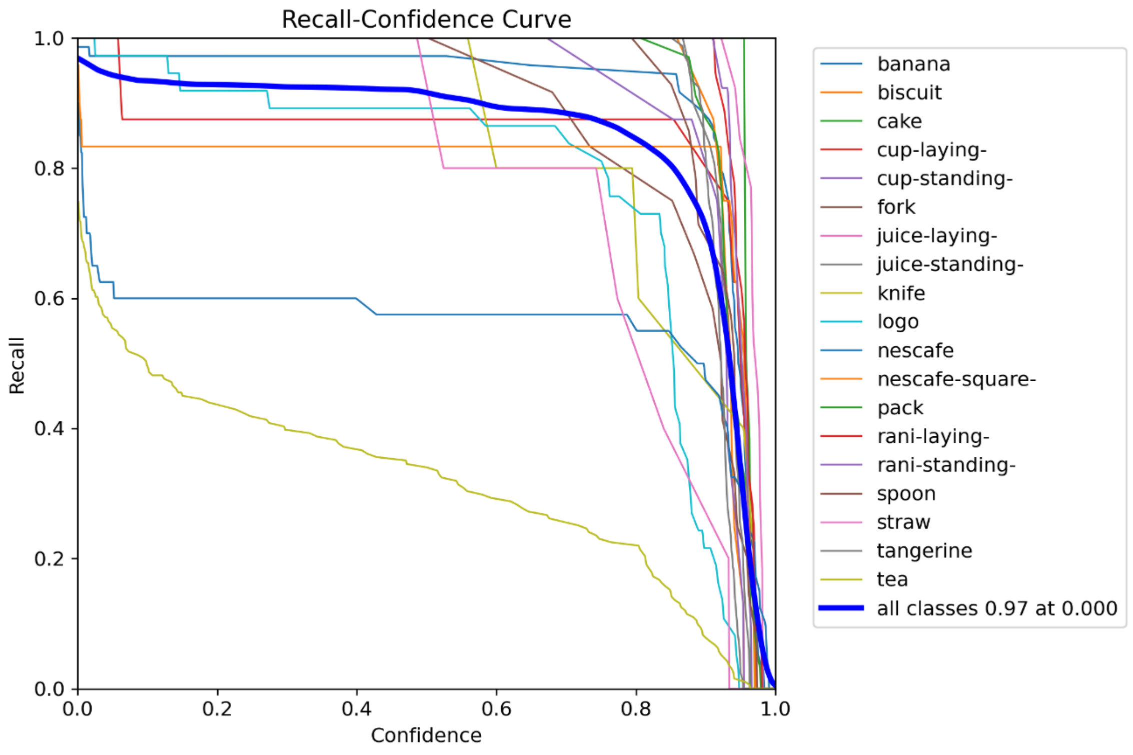

Figure 11 Recall-Confidence curve for model

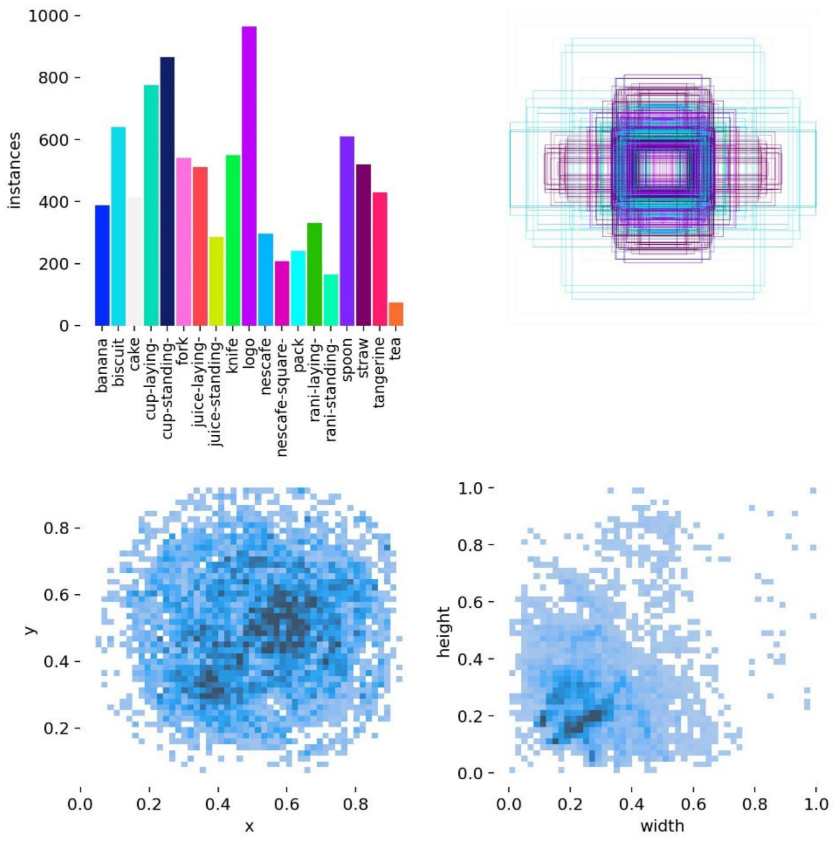

Figure 13 labels prediction

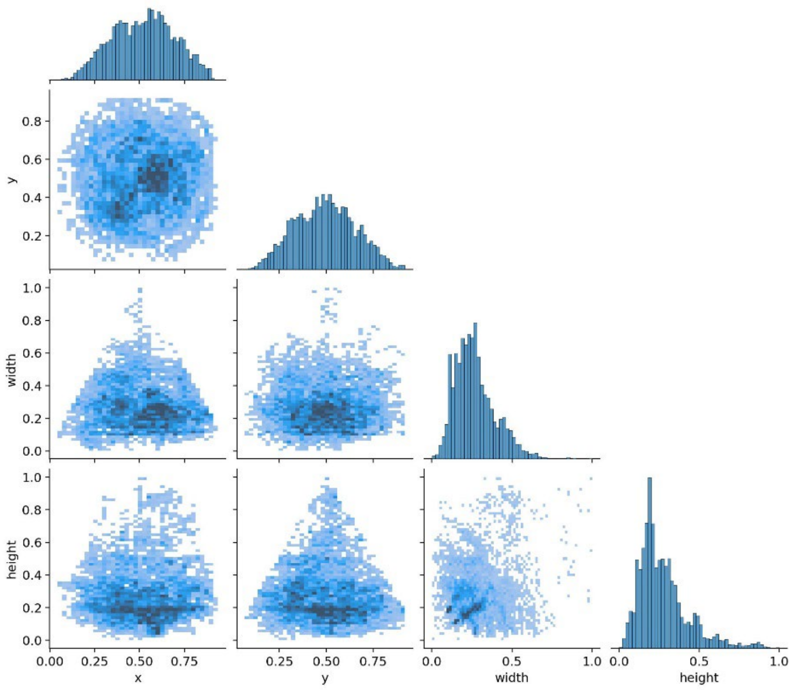

Figure 14 labels correlogram

Problem 5 – Object segmentation

In the field of computer vision, object detection and image segmentation are two pivotal tasks. While object detection provides bounding boxes around objects in an image, it does not offer information about the shape of the objects. Image segmentation, on the other hand, assigns a pixel-wise mask to each object, providing granular details about its shape.

Fast Segment Anything (FastSAM) is an advanced image segmentation model developed by Meta AI. This model can precisely identify either specific objects or every object in an image using various user prompts such as bounding boxes, text, points, or segment-everything inputs. This report outlines the implementation and testing of FastSAM for segmenting images based on bounding box outputs from a fine-tuned YOLO model.

The FastSAM model was initialized with the appropriate device (CUDA for GPU support). Several helper functions were defined to convert masks, annotate images, and encode images for visualization. Bounding boxes generated by the fine-tuned YOLO model were loaded. Each image was processed using the bounding boxes as prompts for FastSAM. The masks generated by FastSAM were used to annotate the images.

The output displayed the segmented images with pixel-wise masks generated by FastSAM based on the bounding box prompts. We use This method provided detailed information about the shapes of the objects in the images, which is not possible with bounding boxes alone.

We sent the xyxys of the bounding boxes identified in the previous question as a prompt box to fastSAM, and the following samples were produced in the output:

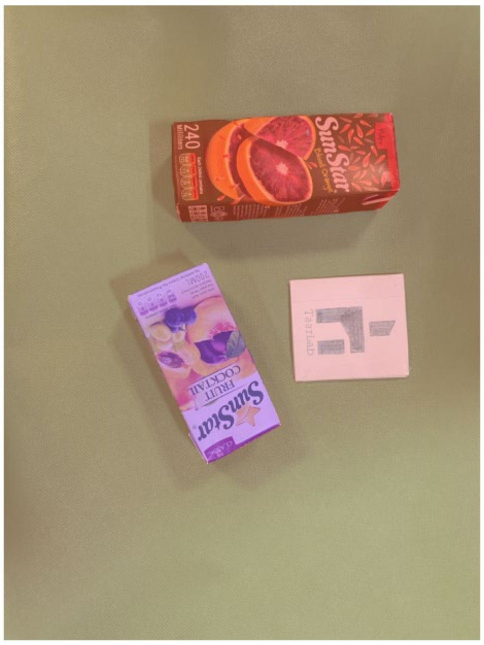

Figure 15 correct output of model

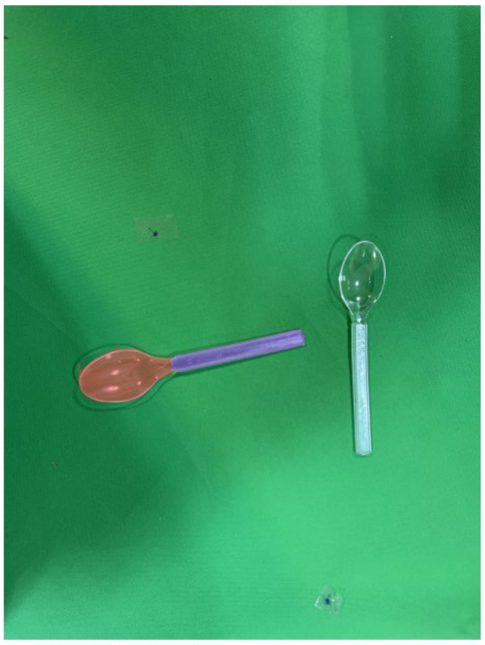

Figure 16 wrong output of model

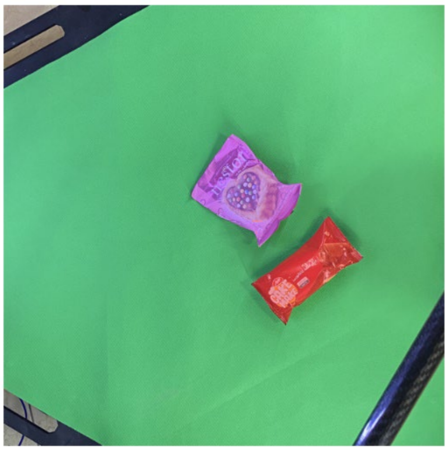

Figure 17 correct output of model

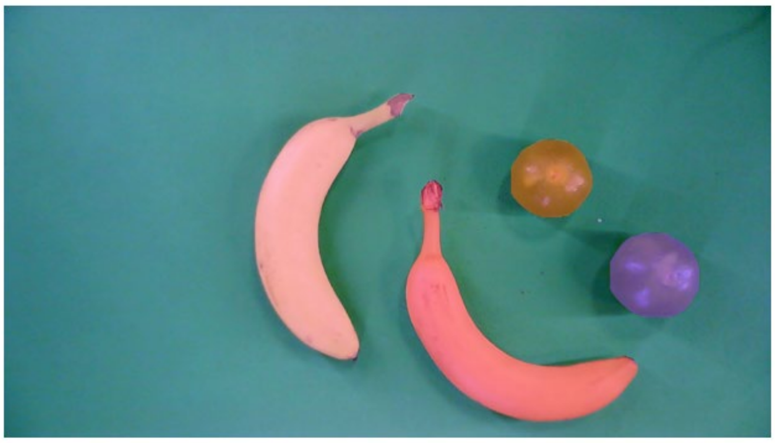

Figure 18 correct output of model