Performance comparison of YOLOv7 and YOLOv8 using the YCB datasets YCB-M and YCB-Video

Samuel Hafner[0000−1111−2222−3333], Prof Dr. Markus Schneider[1111−2222−3333−4444], and Benjamin Stähle[2222−−3333−4444−5555]

RWU Hochschule Ravensburg-Weingarten University of Applied Sciences, Doggenriedstraße 70, 88250 Weingarten, Germany

Abstract. In this paper, the two YOLO frameworks, YOLOv7 and YOLOv8, are compared using the two labeled YCB datasets, YCB-M and YCB-Video. Additionally, a custom test dataset is created in the robotics context to observe how the two YOLO frameworks perform on dissimilar data (compared to the training data). The performance is measured by considering the mean average precision (mAP), average detection time, and resource consumption. Furthermore, the impact of different amounts of training data on performance is observed. For comparability, a training and validation pipeline is established that every trained model undergoes.

We was able to show that both frameworks perform very well and similarly on similar datas (test data from the two datasets YCB-M and YCB-Video) and on dissimilar data (the custom-created test dataset), YOLOv7 significantly outperforms YOLOv8 by 22% mAP.

The code of the datasets divison, the training and validation pipeline and the trained models can be found here: https://github.com/iki-wgt/yolov7_yolov8_benchmark_on_ycb_dataset

Keywords: Object Detection · YOLO · Benchmark · YCB Dataset · Service Robotics · MS COCO · MAP

1 Introduction

Manipulation of objects is one of the most important and complex tasks in service robotics and represents the most substantial interaction a robot can have with its environment. Many developers are working to find good algorithms and solutions intended to facilitate the manipulation of objects [7][4]. To better benchmark these algorithms, the YCB Object and Model set was released in 2015, featuring a total of 77 household Items [5].

Before a robot can manipulate objects, it first needs to recognize them, and currently there are hardly any available models that can reliably recognize the objects of the YCB Object and Model set. Each year, newer and improved object detection frameworks are released, including the additions to the YOLO family in 2022 with YOLOv7[19] and YOLOv8[13]. The advantage of the YOLO frameworks is their real-time capability and the low computational power required during the detection process.

In this paper, the two YOLO frameworks are compared in terms of their performance using the YCB dataset, as it is a common benchmark in robotics. To do this, two already labeled datasets (YCB-Video[20] and YCB-M[11]) are utilized, covering 21 of the 77 objects. Figure 1 illustrates the distribution of the two datasets. The comparison is based on various points. First, the influence of different amounts of data on the two frameworks is examined by initially training and comparing the frameworks with the individual datasets and then with both datasets combined. Secondly, the performance is evaluated using test datas for the YCB-M and YCB-Video datasets and additionally a specially own labeled test dataset from the robotics context, which is completely independent of the two datasets. This has the advantage of examining the performance of the models in an unknown field with a robotics context. Each model uses the same training and test pipeline, and no hyperparameters of the respective frameworks are changed, except for the batch size and epochs, which are determined once for all models at the beginning. This examines how the two frameworks function "out of the box." All models are tested and compared based on performance, average detection time, and resource consumption.

2 Related Work

The benchmarks for the two Frameworks are by default, based on the MS COCO dataset [19] [13].

In August 2023, a comprehensive comparison of "YOLO-based object detection models" titled YOLOBench was published [14]. In this comparison, the developers evaluated various YOLO models (from YOLOv3 to YOLOv8) across four different datasets, on four different hardware platforms, and with different backbones. The four datasets are the VOC dataset[8], the SKU110k dataset[10], the MS COCO dataset[15], and the WIDER FACE dataset[21][14].

In June 2023, another comparison of the YOLO frameworks from YOLOv5 to YOLOv8 in an underwater environment was released [9]. The paper is not publicly available.

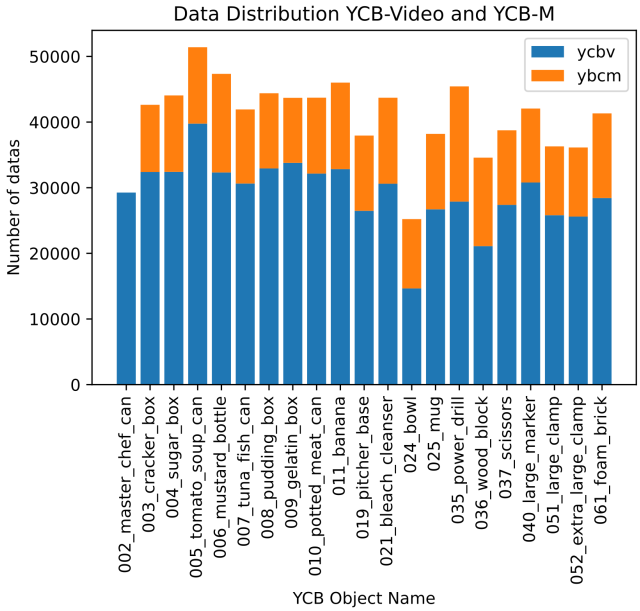

Fig. 1. Data Distribution of YCB-Video and YCB-M. The X-axis indicates the name of each object, and the Y-axis shows the frequency of each object in the respective dataset.

Furthermore, there are additional comparisons between YOLOv7 and YOLOv8, such as helmet detection [3] and smoke and wildfire detection [6].

All these comparisons do not rely on the YCB dataset, the models are not tested "out of the box," and they do not examine the impact of varying amounts of data on the models. Likewise, none of the benchmarks compare the performance of the two models in a robotics context (through the specially created dataset) as already mentioned in Chapter 1.

So far, there are no YOLOv7 or YOLOv8 models that recognize YCB objects or have been trained on the two datasets. The most recent model released on the YCB-Video dataset is a YOLOX model from the "BOP: Benchmark for 6D Object Pose Estimation" [12] Challenge [16]. Therefore, this paper releases the first YOLOv7 and YOLOv8 models for 21 of the 77 YCB objects. These models are available in my Git repository as google drive link.

3 Methods

In this chapter, a brief overview of the aspects of YOLOv7 and YOLOv8 relevant to this paper is provided. This is followed by an explanation of the used metrics, as well as the training and test dataset.

3.1 Theory

The Origin The development of the two YOLO versions (7 and 8) occurred in parallel, as they were created by different individuals. One of the co-founders of YOLOv7 (Alexey Bochkovskiy) also worked on YOLOv4, while at the same time, the founder of YOLOv8 (Glenn Jocher) was working on YOLOv5. Both implementations were inspired by YOLOv5 and were released around the same time, YOLOv8 coming out about two months later. This means that a newer YOLO version does not necessarily indicate better performance.

YOLOv7 The developers of YOLOv7 aimed to increase detection accuracy without requiring additional computing power (inference costs) and while reducing the model's parameters. They successfully achieved this goal and were able to reduce the parameters by 40% compared to other state-of-the-art models [19].

YOLOv8 Since the developers did not publish a paper, their motivation, goals, and the theory behind their work are not clearly evident. There are attempts to explain the architecture and theory [18]. They released a plot showing that YOLOv8 performs better than all previous YOLO models, and they have faster and smaller models. However, it is not clear how they achieved this better performance [13].

3.2 Model Types

Both models feature various model types, which differ in size and, consequently, in performance (the more parameters, the better the performance and the higher the detection time). For example, the YOLOv8n model has only 3.2 million parameters, while their largest model, YOLOv8x, has 68.2 million parameters, but the YOLOv8n model is faster in detection. Overall, YOLOv8 has five different model types which use as input 640 pixel image size, which is changed by the framework itself in the preprocess (so you can put every size of image in it). YOLOv7 originally had the same number, but in the meantime, they have narrowed it down to just the standard YOLOv7 and YOLOv7x as model types, which use also as input image size 640 pixel. But YOLOv7 has additional model types that run on an image size of 1280 pixels. [19] [13]. In this paper we use for both frameworks the X model type. The reasoning will be in Chapter 4.3.

3.3 Metrics

As already mentioned in Chapter 1, the models are compared based on three factors: model performance, average detection time, and resource consumption.

Model Performance A very common metric for evaluating the performance of a model is the mean average precision (mAP). There are various standardized ways of calculating this, such as the VOC Evaluation Metric [8] or the MS COCO Evaluation Metric [1].

In this paper, the MS COCO Evaluation Metric is used, as it is also utilized in the benchmarks of YOLOv7 and YOLOv8 [19][13].

The MS COCO Metric calculates the mAP using various IOU thresholds, sums up the results, and divides by the number of thresholds. The thresholds start at 0.5 and go up to 0.95 with a step size of 0.05, leading to Formula 1. It is often also written as AP@[.5:.05:.95] [1].

$$mAP_{coco} = \frac{(mAP_{0.50} + mAP_{0.55} + ... + mAP_{0.95})}{10} \tag{1}$$

In addition, the PascalVOC Metric mAP_{50} and the strict metric mAP_{75} are also calculated as supplementary metrics. These are also part of the MS COCO Evaluation Metric [1].

Average Detection Time To measure the average detection time, the time taken for all detections on the test dataset is measured and divided by the number of detections. This provides the average detection time for the respective model.

Resource Consumption To measure resource consumption, the GPU usage of the models is monitored during detection.

3.4 The YCB-Video and YCB-M Dataset

YCB-Video Dataset The YCB-Video dataset, with 92 videos and a total of 133,827 frames, is the largest available labeled dataset of the YCB objects. The dataset contains 21 of the 77 YCB objects. The creators of the YCB-Video dataset provide a 3D model for each object, as well as the 6D poses, 2D and 3D semantics, bounding box labels, and respective depth images per scene. The scenes were recorded with the RGB-D camera Asus Xtion Pro Live and have a resolution of 640x480. Each scene features between 3-5 objects [20].

YCB-M Dataset The YCB-M dataset, in contrast, consists of 32 labeled scenes with a total of approximately 47 thousand frames and 20 instead of 21 YCB objects. The objects are the same as those in the YCB-Video dataset, except for the "Master Chef Can," which was not available at the time of recording. Like the YCB-Video dataset, this dataset also includes 3D models, 6D poses, 2D and 3D semantics, bounding box labels, and depth images for each scene. A unique feature of this dataset is that the scenes were recorded with 6 different cameras simultaneously, providing camera diversity. Each scene features between 3 - 8 objects, with an average of 5 objects [11].

In Figure 1, the data distribution of the YCB-M and YCB-Video datasets can be seen.

3.5 Test Dataset

As mentioned in Chapter 1, in addition to the testing datasets of the YCB-M and YCB-Video datas, a separate test dataset is created and labeled. The test dataset comprises a total of 3 scenes (couch table, table, and shelf) with 4 recordings per scene, and in each recording, there are 5 objects, thus covering all 20 objects. The selection of which objects appear in each recording is randomized. All scenes are without the Sugar Box, as it was not available at the time of the recordings. In total, the test dataset contains 251 labeled images and is recorded with the RGB-D camera Asus Xtion Pro Live, which is often used in the robotics field.

In Figure 2, a few example frames from the test dataset can be seen.

4 Experiments

In this chapter, we first address the division of the datasets, followed by the description of the hardware used for training and testing. Next is the explanation of the training pipeline and validation pipeline. The implementation of these tasks can be found in my Git repository, which is linked in the abstract.

4.1 Division of the datasets

When dividing the data, attention is paid to the distribution of the data. For example, the YCB-Video dataset contains 92 scenes, each with a different number of images. If, for instance, 10% are taken as validation data, it is determined per scene what number of images from that scene constitutes 10%. Additionally, the data division is systematic, meaning that in a scene with 100 images, every 10th image is included in the validation dataset, rather than, for example, the last 10 images.

YCB-Video The YCB-Video dataset already provides a division between training data and test data. From the training data, 10% are taken as validation data, as described above [20].

YCB-M For the YCB-M dataset, there is no predefined division into training and test data. Therefore, the division is 90% training data and 10% test data. From the training data 10% are taken as validation data.

Combination When combining both datasets, the divisions from each dataset are retained and merged together.

In Table 1, the division of the individual datasets into training, validation, and test data can be seen.

Fig. 2. Some frames of the test dataset (above: couch table, middle: shelf, below: table)

4.2 Hardware

For training, the in-house training server of the Ravensburg-Weingarten University of Applied Sciences from the Institute for Artificial Intelligence (IKI) is used [2]. It is equipped with a server-based A40 graphics card with 48 GB of graphics memory, 512 GB of RAM, and a CPU with 128 cores.

Table 1. Distribution into Training, Validation and Test dataset per dataset

Dataset	Train	Val	Test
YCB-M	38287	4259	4732
YCB-Video	74967	18788	40181
Combination	113254	23047	44913

The test is carried out on a laptop with fewer resources, since for detection, simpler computers are mostly used. Compared to the training server described above, the laptop is equipped with an NVIDIA GeForce RTX 2070 with 8 GB of graphics memory. It also has 16 GB of RAM and an AMD Ryzen 7 3700X CPU with 8 cores.

4.3 Training pipeline

For better comparability, the training of each model always follows the same procedure.

Pretrained Model For each model, the corresponding existing MS COCO model is used as a pretrained model. For instance, when a YOLOv7x model is trained, the YOLOv7x MS COCO model is taken as the pretrained model. However, when the YOLOv8m model is trained, the YOLOv8m MS COCO model is used as the pretrained model, and so on.

The performance is about 20% mAP better when a MS COCO model is used as a pretrained model, as shown in Appendix A The test was conducted with a standard YOLOv7 model using the combined dataset for 10 epochs and a batch size of 40.

Model Type For the comparison, the respective X model type of YOLOv7 and YOLOv8 are used, as they demonstrably deliver better performance. Similar to the pretrained models, the difference in performance with various YOLOv7 model types was initially tested. Between the standard YOLOv7 model and the YOLOv7x model, is a difference of 4% mAP. This comparison can be seen in Appendix B

Batchsize and Epochs The training is conducted over 100 epochs, as is standard for both frameworks. Training stops early if the model does not improve after 10 epochs (Early Stopping). The best model on the validation dataset is used as the benchmark model. A batch size of 40 nearly maximizes the GPU usage. Training both models (YOLOv7x and YOLOv8x) requires about 40GB out of 48GB of GPU memory on the training server.

In summary, for the comparison of the two frameworks, the X model types of YOLOv7 and YOLOv8 are trained, using the respective MS COCO model

as a pretrained model, with a batch size of 40 and 100 epochs. The training is conducted first on the YCB-M dataset, then on the YCB-Video dataset, and lastly on the combined dataset.

4.4 Validation Pipeline

The validation pipeline is structured as follows:

Load Model Here, for further use, either the YOLOv7 or YOLOv8 model is loaded.

Prediction on the Test Dataset Subsequently, predictions are made on the respective test data for which the model was trained (see Chapter 4.1), and on the specially created test dataset (refer to Chapter 3.5). Also, the average detection time is calculated as explained in Chapter 3.3.

Calculate the metrics Finally, based on the predictions and the ground truth (GT) labels, the mean average precision is calculated. To calculate the mean average precision according to the MS COCO Evaluation Metric, we took the code from the paper "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit" [17] and modified it for my application so that it fits into the entire validation pipeline. The results are saved and compared with the other models. The output is a table with the results of mAP, mAP_{50}, mAP_{75}.

5 Results

As already highlighted in Chapter 3.3, the models are evaluated based on performance, average detection time, and resource consumption. These are presented in detail in this chapter.

5.1 Model Performance

The model performance is presented in two different tables. Table 2 displays the results of the model performance on the corresponding test dataset (refer to Chapter 4.1). Table 3 shows the results on my own created test dataset (see Chapter 3.5). In both tables, the results of the models that performed better on the respective dataset are highlighted in bold.

None of the two frameworks stopped early (Early Stopping) and always trained through the 100 epochs.

As shown in Table 2, YOLOv8 performs slightly better (otherwise the same and 1 time worse) than YOLOv7 on all three test datasets, both in terms of mAP, mAP_{50}, and mAP_{75}. Overall, however, it can be said that the performance on all test datasets by all models is very good (at least 90.58% mAP and

Table 2. Benchmark of YOLOv7 and YOLOv8 models (100 epochs, batchsize 40) on their corresponding Test Dataset

Train/Test Dataset	Model	mAP	mAP_{50}	mAP_{75}
YCB-M	YOLOv7x	90.58%	**98.43%**	96.72%
YCB-M	YOLOv8x	**91.60%**	98.42%	**96.74%**
YCB-V	YOLOv7x	97.45%	99.25%	99.15%
YCB-V	YOLOv8x	**97.77%**	99.25%	99.15%
Combination	YOLOv7x	96.63%	99.01%	98.90%
Combination	YOLOv8x	**97.00%**	**99.06%**	**98.91%**

up to 97.77% mAP) and on each dataset YOLOv7 and YOLOv8 perform most likely the same. The biggest difference between YOLOv7 and YOLOv8 is on the combination dataset with 0.37% mAP. Interestingly, YOLOv7 and YOLOv8 deliver better results when trained and tested only on the YCB-Video dataset than on the combination dataset, although the latter contains more and varied data. This is probably due to the dataset difference between the YCB-M dataset and the YCB-Video dataset. Since the YCB-Video dataset has significantly more data, the respective combination models are better trained on the YCB-Video dataset and thus perform worse on the combination test dataset, as it also contains YCB-M test data. However, the difference in YOLOv8 performance from the YCB-Video to the combination dataset is 0.77% mAP, which could also just be noise.

Table 3, on the other hand, indicates that YOLOv7 performs significantly better on the own created dataset with all three models. For the combination model, with a total of 66.19% mAP, the model is about 22% mAP better than the YOLOv8 model, and for the mAP_{50}, with a total of 84.39%, the difference is about 27%. This suggests that YOLOv7 better generalizes and performs very well on completely new and different data, while YOLOv8 generalizes worse and performs significantly worse across all three models. The best for YOLOv8 is 44.10% mAP on the combination model, and the worst is 13.05% mAP on the YCB-M model. Table 3 also clearly shows the impact of the amount of data on the performance of the models. For YOLOv7, there is a performance difference of 19.41% mAP between the YCB-M dataset with 47 thousand images and the YCB-V dataset with about 133 thousand images, and a performance difference of 23.5% mAP between the YCB-Video and the combination dataset. A similar pattern is observed with YOLOv8.

5.2 Average Detection Time

The results of the average detection time are presented in Table 4. It shows the average detection time in milliseconds for each model and dataset, and at the end, the average of the 3 values is calculated, with the result shown in the last

Table 3. Benchmark of YOLOv7 and YOLOv8 models (100 epochs, batchsize 40) on own Test Dataset

Train Dataset	Model	mAP	mAP_{50}	mAP_{75} d
YCB-M	YOLOv7x	**23.55%**	**33.37%**	**29.08%**
YCB-M	YOLOv8x	13.05%	18.76%	15.00%
YCB-V	YOLOv7x	**42.69%**	**60.17%**	**52.30%**
YCB-V	YOLOv8x	35.37%	48.77%	43.18%
Combination	YOLOv7x	**66.19%**	**84.39%**	**75.29%**
Combination	YOLOv8x	44.10%	57.14%	50.97%

column of the table. The average detection time of YOLOv7 is approximately 6.33ms better on average than that of YOLOv8.

Table 4. Average Detection Time of YOLOv7 and YOLOv8 models

Model	YCB-M	YCB-V	Combination	Average
YOLOv7x	30ms	21ms	21ms	**24ms**
YOLOv8x	30ms	32ms	29ms	30ms

5.3 Resource Consumption

The results of the resource consumption are shown in Table 5. The resource consumption is independent of the dataset with which it was trained; it only depends on the model type (YOLOv7 or YOLOv7x, etc.).

The resource consumption of both are very low, with the YOLOv8x models consuming 100MB less than the YOLOv7x models. This is likely because the YOLOv8x models have fewer layers and about 4 million fewer parameters than the YOLOv7x models [19][13].

Table 5. Resource Consumption of YOLOv7x and YOLOv8x models

Model Type	Resource Consumption
YOLOv7x	1.84GB
YOLOv8x	**1.74GB**

6 Conclusion and Future Work

In this paper, the performance of YOLOv7 and YOLOv8 on the YCB Object and Model set was compared. The comparison was made with the already labeled datasets YCB-M and YCB-Video. Various aspects were addressed, such as the mAP, average detection time, resource consumption, and the influence of different amounts of data. Additionally, a unique test dataset in the robotic context, was created to measure the performance of both frameworks on completely unseen and dissimilar data compared to the training data.

To ensure comparability, a training and validation pipeline was developed that takes each model through the exact same procedure. It was initially found that models with the MS COCO model as a pretrained model perform about 20% better, and that Model Type X performs the best of all. See Appendix A and B.

We was able to demonstrate that both YOLOv7 and YOLOv8 perform very well on similar data, with YOLOv8 almost always performing slightly better or equal to YOLOv7, as shown in Table 2. We also showed that YOLOv7 performs significantly better on dissimilar data, with a peak of about a 23.5% mAP performance difference, which speaks to YOLOv7's better generalization. See Table 3 for this.

Regarding average detection time and resource consumption, YOLOv7 is with 24 ms average detection time with a GPU consumption of 1.84GB, 6.33 ms faster than YOLOv8 but consumes 100MB more GPU, as already presented in Table 4 and 5.

As further work, it would be interesting to delve deeper into the reason why YOLOv8 significantly underperforms on dissimilar data compared to YOLOv7 (Table 3). Are 100 epochs too much for the YOLOv8 framework? Is it overfitting? Is there something amiss in the architecture? Or why does YOLOv7 comparatively perform so well on dissimilar data? Additionally, during my research, we made an interesting observation that both YOLO frameworks perform very well on my test dataset with the model created after the 1st epoch. YOLOv7 still performs significantly better than YOLOv8, but both are overall better. Here, YOLOv8 performs with a mAP of 67% and YOLOv7 with a mAP of 79%. Why this is the case would be another interesting question to explore.

7 Appendix

7.1 A

Table 6. Benchmark between pretrained and no pretrained YOLOv7 model (10 epochs, batchsize 40)

mAP	mAP_{50}	mAP_{75}	Dataset	Pretrained Model
0.478	0.681	0.544	Combination	None
0.671	**0.877**	**0.778**	Combination	MS COCO

7.2 B

Table 7. Benchmark between different YOLOv7 model types (10 epochs, batchsize 40)

mAP	mAP_{50}	mAP_{75}	Dataset	Model Type
0.671	0.877	0.778	Combination	yolov7
0.711	**0.909**	**0.809**	Combination	yolov7x

References

1. COCO Detection Evaluation. https://cocodataset.org/#detection-eval, accessed: 2023-09-17
2. Institut für Künstliche Intelligenz. https://forschung.rwu.de/institute/iki, accessed: 2023-09-18
3. Aboah, A., Wang, B., Bagci, U., Adu-Gyamfi, Y.: Real-time multi-class helmet violation detection using few-shot data sampling technique and yolov8. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5349–5357 (2023)
4. Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., Florence, P., Fu, C., Arenas, M.G., Gopalakrishnan, K., Han, K., Hausman, K., Herzog, A., Hsu, J., Ichter, B., Irpan, A., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Leal, I., Lee, L., Lee, T.W.E., Levine, S., Lu, Y., Michalewski, H., Mordatch, I., Pertsch, K., Rao, K., Reymann, K., Ryoo, M., Salazar, G., Sanketi, P., Sermanet, P., Singh, J., Singh, A., Soricut, R., Tran, H., Vanhoucke, V., Vuong, Q., Wahid, A., Welker, S., Wohlhart, P., Wu, J., Xia, F., Xiao, T., Xu, P., Xu, S., Yu, T., Zitkovich, B.: Rt-2: Vision-language-action models transfer web knowledge to robotic control (2023)
5. Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., Dollar, A.M.: The ycb object and model set: Towards common benchmarks for manipulation research. In: 2015 International Conference on Advanced Robotics (ICAR). pp. 510–517 (2015). https://doi.org/10.1109/ICAR.2015.7251504
6. Casas, E., Ramos, L., Bendek, E., Rivas-Echeverría, F.: Assessing the effectiveness of yolo architectures for smoke and wildfire detection. IEEE Access **11**, 96554–96583 (2023). https://doi.org/10.1109/ACCESS.2023.3312217
7. Coleman, D., Șucan, I.A., Chitta, S., Correll, N.: Reducing the barrier to entry of complex robotic software: a moveit! case study. Journal of Software Engineering for Robotics **5**(1), 3–16 (may 2014)
8. Everingham, M., Gool, L.V., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes challenge 2012 (voc2012) results (2012)
9. Gašparović, B., Mauša, G., Rukavina, J., Lerga, J.: Evaluating yolov5, yolov6, yolov7, and yolov8 in underwater environment: Is there real improvement? In: 2023 8th International Conference on Smart and Sustainable Technologies (SpliTech). pp. 1–4 (2023). https://doi.org/10.23919/SpliTech58164.2023.10193505
10. Goldman, E., Herzig, R., Eisenschtat, A., Ratzon, O., Levi, I., Goldberger, J., Hassner, T.: Precise detection in densely packed scenes (2019)
11. Grenzdorffer, T., Gunther, M., Hertzberg, J.: YCB-m: A multi-camera RGB-d dataset for object recognition and 6dof pose estimation. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE (may 2020). https://doi.org/10.1109/icra40945.2020.9197426, https://doi.org/10.1109%2Ficra40945.2020.9197426
12. Hodan, T., Michel, F., Brachmann, E., Kehl, W., Buch, A.G., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., Sahin, C., Manhardt, F., Tombari, F., Kim, T.K., Matas, J., Rother, C.: Bop: Benchmark for 6d object pose estimation (2018)
13. Jocher, G., Chaurasia, A., Qiu, J.: YOLO by Ultralytics. https://github.com/ultralytics/ultralytics, accessed: 2023-09-17
14. Lazarevich, I., Grimaldi, M., Kumar, R., Mitra, S., Khan, S., Sah, S.: Yolobench: Benchmarking efficient object detectors on embedded systems (2023)

15. Lin, T.Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P.: Microsoft coco: Common objects in context (2015)
16. Liu, X.: gdrnppbop2022. https://github.com/shanice-l/gdrnpp-bop2022, accessed: 2023-09-17
17. Padilla, R., Passos, W.L., Dias, T.L.B., Netto, S.L., da Silva, E.A.B.: A comparative analysis of object detection metrics with a companion open-source toolkit. Electronics **10**(3) (2021). https://doi.org/10.3390/electronics10030279, https://www.mdpi.com/2079-9292/10/3/279
18. RangeKing: Brief summary of YOLOv8 model structure. https://github.com/ultralytics/ultralytics/issues/189, accessed: 2024-03-07
19. Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M.: Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors (2022)
20. Xiang, Y., Schmidt, T., Narayanan, V., Fox, D.: Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes (2018)
21. Yang, S., Luo, P., Loy, C.C., Tang, X.: Wider face: A face detection benchmark. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)