BarBeR: A Barcode Benchmarking Repository

Enrico Vezzali[1], Federico Bolelli[1], Stefano Santi[2], and Costantino Grana[1]

[1] University of Modena and Reggio Emilia, Modena, Italy
{name.surname}@unimore.it
[2] Datalogic, S.p.A, Bologna, Italy
stefano.santi@datalogic.com

Abstract. Since their invention in 1949, barcodes have remained the preferred method for automatic data capture, playing a crucial role in supply chain management. To detect a barcode in an image, multiple algorithms have been proposed in the literature, with a significant increase of interest in the topic since the rise of deep learning. However, research in the field suffers from many limitations, including the scarcity of public datasets and code implementations, which hampers the reproducibility and reliability of published results. For this reason, we developed "BarBeR" (Barcode Benchmark Repository), a benchmark designed for testing and comparing barcode detection algorithms. This benchmark includes the code implementation of various detection algorithms for barcodes, along with a suite of useful metrics. It offers a range of test setups and can be expanded to include any localization algorithm. In addition, we provide a large, annotated dataset of 8 748 barcode images, combining multiple public barcode datasets with standardized annotation formats for both detection and segmentation tasks. Finally, we share the results obtained from running the benchmark on our dataset, offering valuable insights into the performance of different algorithms.

Keywords: BarBeR · Barcodes· Benchmark· QR Codes · Public Dataset

1 Introduction

Barcodes, a prevalent form of machine-readable data representation, have revolutionized the accuracy and speed of data collection and identification [36]. Their cost-effectiveness and efficiency have led to their widespread use in various engineering applications. First of all, barcodes serve as a cornerstone of supply chain management [24], facilitating the flow of goods from manufacturers to consumers by enabling efficient inventory tracking and logistics management. Secondly, barcodes are extensively used in warehouses to automate the process of goods receipt, storage, and dispatch, helping in reducing manual errors and improving the speed of operations [19]. Other notable applications are component tracking in manufacturing, product recognition in retail [25], and robot guidance [29]. Despite their inception over seven decades ago, barcodes continue to hold their ground in today's digital age, and their use is forecasted to increase in the future [17]. This is reflected in the projected growth of the barcode reader market, which was valued at $7.4 billion in 2022 and is expected to reach $13.3

billion by 2032, growing at a CAGR of 6.3% from 2023 to 2032 [33]. Barcodes come in two categories: one-dimensional (1D or linear) and two-dimensional (2D). Linear barcodes encode data with lines of varying widths and spacing, but have limited data storage capacity. To overcome this issue, 2D barcodes were introduced. Their structure allows data to be stored on both vertical and horizontal axes, offering greater capacity compared to 1D barcodes [32]. The process of reading a barcode can usually be divided into two macro steps: localization and decoding. While some papers focus on both steps [10,18] most of the publications just focus on the localization part [30,38,41]. Especially in recent times, it has become the norm to use public third-party libraries to handle the decoding step [37]. The two most used libraries are ZXing[3] and Zbar.[4] Each software tool can handle both 1D and 2D barcodes. Therefore, our primary focus from now on will be on localization. Until recently, real-time speed for a localization algorithm was achievable solely through the computation of hand-crafted features from the image. However, the recent advancements in edge deep learning fueled the interest in developing barcode localization solutions based on deep learning. Between the years 2015 and 2021, 25 publications introduced a method for barcode localization (either 1D, 2D, or both) that utilized deep learning techniques [37]. Despite the huge interest in the field, several issues prevent definitive conclusions about methods' effectiveness and applicability. The first is that existing research relies on small datasets that do not reflect real-world scenarios accurately and make training deep learning models difficult. Then there is the problem of reproducibility. The lack of public code implementations makes replicating results challenging. Finally, different studies use different metrics, leading to contradictory comparisons even with identical algorithms and datasets.

To address these challenges, we have developed "BarBeR" (Barcode Benchmark Repository) —an open-source benchmark for barcode localization with standardized test protocols and evaluation metrics. BarBeR contains the implementation of multiple localization algorithms tailored for barcodes that we selected after a thorough review of the literature. In addition, we are publicly releasing a large annotated dataset of 8 748 images of barcodes to be used with our benchmark. Our aim is to enhance reproducibility and facilitate more reliable algorithm comparisons within the research community.

2 Related Works

Early Barcode Localization Efforts. Joseph Woodland and Bernard Silver invented the linear barcode in 1949 and patented it in 1952. Early decoding methods relied on analog circuits, with laser scanners being the primary decoding method in the '70s. However, these systems required the reader to be directly aimed at the barcode. The 1990s saw the advent of 2D image barcode reading. A significant advantage of this approach is the ability to read a barcode from

[3] https://github.com/zxing/zxing
[4] https://github.com/ZBar/ZBar

Table 1. List of the public datasets collected for the benchmark. The table reports the number of images per dataset and the resolution of the image with the minimum and the maximum number of pixels in the dataset respectively. # 1D and # 2D represent the number of linear and two-dimensional barcode instances in each dataset.

Dataset Name	# Images	Minimum Resolution	Maximum Resolution	# 1D	# 2D
Arte-Lab Medium 1D [39]	430	1 152×864	2 976×2 232	430	7
Arte-Lab Extended 1D [40]	155	648×488	648×488	165	3
Bodnár-Huawei QR [3]	98	1 600×1 200	1 600×1 200	0	98
DEAL KAIST Lab [7]	3 308	141×200	3 480×4 640	3 378	76
Dubska QR [8]	810	402×604	2 560×1 440	0	806
InventBar [16]	527	480×640	480×640	530	33
Muenster 1D [35]	1 055	1 600×1 200	2 592×1 944	1 068	1
OpenFood Facts [1]	185	390×520	5 984×3 376	187	5
ParcelBar [16]	844	1 108×1 478	1 478×1 108	1 196	17
Skku Inyong DB [38]	325	1 440×2 560	1 440×2 560	368	10
Szentandrasi QR [31]	90	1 024×768	4 752×3 168	0	225
ZVZ-Real [41]	921	407×576	3 288×4 930	740	475
Total	**8 748**	**200×141**	**5 984×3 376**	**8 062**	**1 756**

a wider field of view, but to do so, the barcode must first be located. Detection methods for linear barcodes included Sobel filters for texture analysis [34], Gabor filters [13], and even early machine learning techniques for texture classification [14]. The Hough Transform also gained popularity for linear barcodes [26], and gradient analysis was used for QR codes [27].

Recent Approaches. Research continued to address limitations and expand barcode localization applications. Methods explored skeletonization [4] and texture direction analysis [12], while the use of the Hough Transform was extended to the 2D barcodes [31]. Huge efforts were directed into increasing the algorithm's speed, to allow the use on mobile phones [10].

The Deep Learning Era. Chou's 2015 work marked a turning point with the introduction of CNNs for QR code detection [6]. Deep learning has since become dominant, with notable successes using YOLO [11] and Faster R-CNN [20]. Many also proposed custom CNN architectures adapted to the task [41].

3 Dataset

For this project, we required a large dataset to accurately compare algorithms and train object detection neural networks. For this reason, we conducted a thorough literature review to identify publicly available datasets of barcodes. Table 1 lists these datasets and their sources. The collected datasets account for a total of 8 748 images with 9 818 annotated barcodes, 8 062 linear, and 1 756 two-dimensional. A significant challenge was the lack of annotations in some datasets and the wide variation in annotation formats. To address this, we generated new annotations for all images using Datalogic's proprietary software, which generates a 4-point polygon for each barcode read and provides additional information, such as its type, and the encoded string. In addition, we have information about the pixel density of the barcode, usually measured in pixels per

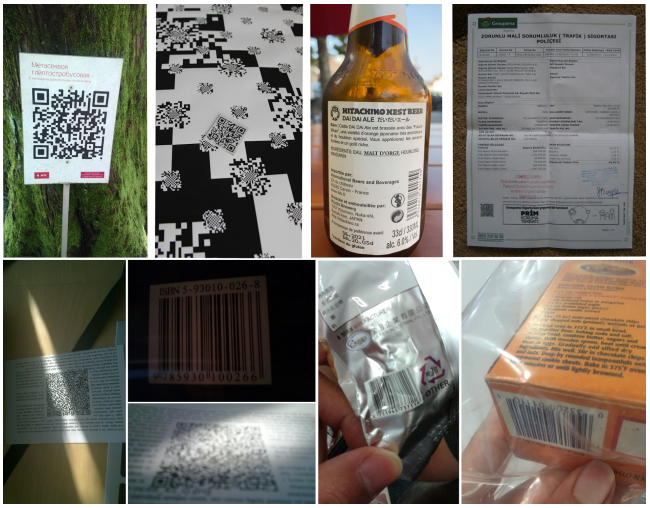

Fig. 1. Example of images taken from the proposed dataset. Multiple types of items are portrayed in different settings. In addition, we have some examples of hard cases, such as confusing patterns, small bounding boxes, variable lighting, underexposure, and blur. Some barcodes are also non-planar or partially obstructed.

element (PPE), i.e., the mean width of the smallest element in a barcode. This measure can also be referred to as pixels per module (PPM). While most codes were annotated in this way (8 096), a few (1 722) were un-decodable due to blur, noise, or incorrect scale. These codes were manually annotated, and thus they lack some information like the PPE. Since the annotations use polygons instead of boxes, they are suitable for both detection and segmentation.

The final dataset presents an extensive diversity of subjects and environments. It contains barcodes of 18 categories, 14 of which are considered linear symbologies (Code 128, Code 39, EAN-2, EAN-8, EAN-13, GS1-128, IATA 2 of 5, Intelligent Mail Barcode, Interleaved 2 of 5, Japan Postal Barcode, KIX-code, PostNet, RoyalMail Code, and UPC) and 4 are considered 2D symbologies (Aztec, Datamatrix, PDF417, and QR Code). The images have been captured with different devices such as a Nokia N95 [35], a Huawei Smartphone from 2014 [3], and a 15MP professional camera [8]. The dataset also features different settings and subjects. Skku Inyong DB [38] was captured inside a supermarket and represents items found there. DEAL KAIST Lab [7] also represents market items, but the settings change widely, some indoors, others outdoors. Dubska [8] dataset represents mostly QR codes printed on paper, captured indoors in a controlled setting. ZVZ-Real [41] is one of the most diverse datasets in the collection, with images taken indoors and outdoors, with subjects ranging from market items, product labels, receipts, and letters as well as book photos and scans. In addition, our dataset contains both planar barcodes and skewed or warped barcodes. The dataset contains barcodes in different lighting conditions, some are underexposed or overexposed, and others have variable lighting

throughout the code. Other codes have specular reflections. Finally, some codes are affected by blur and noise or are partially covered or obstructed, as shown in Fig. 1.

4 Benchmark Description

As part of this project, we have developed BarBeR, a benchmark for barcode localization algorithms available on GitHub.[5] It includes various detection methods and scripts to train neural networks for barcode detection. Our dataset, used for running our tests, can be downloaded from the same GitHub repository or from our website.[6]

4.1 Tests and Metrics

The repository is equipped with a variety of test scripts, each supporting diverse configurations. Here is a breakdown of the test scripts and their main configuration parameters:

- **Single Class Detection**: runs all the selected algorithms considering only images with the selected type of barcodes. It can be tailored to permit only linear or two-dimensional barcodes. It is also possible to include only images with a single Region Of Interest (ROI) or multiple ROIs per image. In addition, we can decide the target resolution used to rescale the images in the test set. Finally, we can specify which algorithms to use in the test and with which arguments;
- **Multi-Class Detection**: runs all the selected algorithms on all the images of the test set. As for Single Class Detection, we can choose the resizing resolution and which algorithms are included in the test;
- **Timing Performance**: measures the time required to run the algorithms. The times can be taken from the average times on all datasets or a subsection of it. It is possible to measure the algorithms' performance on a single core or multiple cores as well as on GPU.

All test scripts are written in Python and take as input argument a YAML configuration file and output a YAML file containing multiple metrics for every tested algorithm. The available metrics are precision, recall, and F1-score at different IoU scores. For algorithms that also output a confidence score, the Benchmark computes the Average Precision (AP@.5, AP@[.5:.95]) for each class, the mean Average Precision (mAP@.5, mAP@[.5:.95]) and the Average Recall (AR100, AR10, AR1). Finally, the benchmark allows to filter these metrics depending on the size of the ground truth and its pixel density. The repository also contains bash scripts used to run a pipeline of tests. This is useful, for example, for k-fold cross-validation.

[5] https://github.com/Henvezz95/BarBeR
[6] https://ditto.ing.unimore.it/barber

Table 2. Characteristics of the deep-learning models used in our tests. Two-stage detectors first propose regions, then classify and refine bounding boxes. One-stage detectors perform detection and classification in a single step.

Network	Type	Backbone	# Parameters [M]	GFlops @(640x640)
Zharkov *et al.* [41]	One-Stage	dilated-net	0.0424	1.528
Faster R-CNN [28]	Two-Stage	resnet50_fpn_3x	41.755	134.38
RetinaNet [21]	One-Stage	resnet50_fpn_3x	34.014	151.54
YOLO-v8 [15]	One-Stage	yolov8 medium	25.903	39.66
YOLO-v8 Nano [15]	One-Stage	yolov8 nano	3.157	4.429
RT-DETR [23]	One-Stage	HGNetv2-L	31.005	54.17

4.2 Available Localization Methods

Gallo *et al.* The localization method proposed by Gallo and Manduchi in 2011 is a rapid algorithm that localizes a single 1D barcode per image. It assumes the barcode is horizontally positioned with vertically aligned parallel lines and is not rotation invariant. The process begins by calculating a heatmap $I_e(n)$, representing the difference between the magnitudes of the horizontal and vertical derivatives. After smoothing and binarizing the heatmap, the blob containing the pixel that maximizes $I_e(n)$ is used to compute the barcode's bounding box.

Soros *et al.* This algorithm was proposed in 2013 by Sörös and Flörkemeier. It is a method designed for both 1D and 2D barcodes that is orientation invariant and is quite resistant to blur [30]. However, this method can only output a single ROI for each barcode type. It is based on the UNIVAR detector and OMNIVAR detector proposed by Ando [2]. The first detector finds areas with strong unidirectional edges and can be used to find linear barcodes, while the latter can find corners and is useful for 2D barcode localization.

Zamberletti *et al.* The method introduced by Zamberletti *et al.* in 2013 is capable of detecting multiple linear barcodes. It generates several rotated boxes, all sharing the same angle of rotation. This proves beneficial in scenarios where a single label contains multiple barcodes, each exhibiting the same rotational angle. It uses a multi-layer perceptron to process the Hough Transform of the image and predict the angle of the barcodes in the image. Once the angle is found, the technique of Galamhos *et al.* [9] is used to find all lines with that angle of orientation. Finally, the areas with the highest concentration of these lines are located using a method based on histograms and labeled as barcodes.

Yun *et al.* This detection method was described in 2017 by Yun and Kim. The algorithm is designed for the detection of linear barcodes and supports multiple detections per image. For detecting the salient regions, the entropy scheme is used [5]. The idea is to divide the image into non-overlapping cells, and for each cell the local orientation histogram is computed. The histogram is used to compute the entropy of the cell. Cells with high entropy have high directionality and a high probability of being part of a barcode.

Zharkov *et al.* In 2019, Zharkov et al. proposed a custom Convolutional Neural Network for 1D and 2D barcodes segmentation employing dilated convolution. The network is trained using a loss function that prioritizes high recall over high precision.

Table 3. Precision, Recall and F1-score with an IoU threshold of 0.5. Employed images contain a single 1D barcode and were resized to have their longest side of 640 pixels.

Detection Method	Precision ↑	Recall ↑	F1-score ↑
Gallo *et al.* [10]	0.533	0.533	0.533
Soros *et al.* [30]	0.658	0.658	0.658
Zamberletti *et al.* [40]	0.234	0.340	0.278
Yun *et al.* [38]	0.806	0.714	0.757
Zharkov *et al.* [41]	0.725	0.952	0.823
Faster R-CNN [28]	0.981	0.996	0.989
RetinaNet [21]	**0.988**	0.991	0.990
YOLO Nano [15]	0.978	0.997	0.987
YOLO Medium [15]	0.984	0.998	0.991
RT-DETR [23]	0.987	**0.999**	**0.993**

Open Source Object Detection Models. In addition, we included five open-source object detection models in our benchmark. Each model was pre-trained on the MS COCO dataset [22] and fine-tuned on our training set. The selected architectures are Faster R-CNN, RetinaNet, YOLO-v8 Medium, and Nano and RT-DETR. The details of the selected architectures are presented in Table 2.

5 Benchmark Results

5.1 Methodology

We assess the detection accuracy using 5-fold cross-validation for both single-class and multi-class modes. End-to-end deep learning models are trained with 75% of the training set, using the remaining 25% as a validation set for early stopping. Zamberletti's method leverages a pre-trained MLP trained on the Arte-Lab Rotated dataset, that is not included in our dataset, thus preventing any unfair comparison. Timing measurements are taken as the best of three runs to minimize external factor interference.

5.2 Single 1D Barcode Localization

First, we tested the available detection algorithms by considering just images of a single class, linear barcodes, or 2D barcodes. This evaluation focuses on images containing a single linear barcode, allowing us to test all the available algorithms. The total number of images included in this test was 6 811. For this test, we resized all images to have their longest side of 640 pixels. This is the same size used to test the methods of Gallo [10] and Zamberletti [40] in their original paper. This is also the default resolution for YOLO-v8 [15] and other object detection networks. At this resolution, our dataset comprises 42 small objects (area $< 32^2$), 2 665 medium objects ($32^2 <$ area $< 96^2$), and 4 104 large objects (area $>96^2$). Traditional methods often rely on some form of texture detection for localization, where barcode texture depends on the number of pixels per element (PPE). After resizing, the PPE ranges from 0.35 to 5.13, with most barcodes in the dataset having a pixel density between 1 and 3 pixels

per element. Additionally, there are 1 044 barcodes without PPE information, suggesting that the automatic labeler was unable to decode them.

Since not all methods generate a confidence score, we used precision, recall, and F1-score as metrics for a fair comparison. In Table 3 we can see the results of the different methods considering an IoU threshold of 0.5. Gallo and Soros' algorithms produce a single prediction every time, so their precision, recall, and F1 scores are always the same. However, considering a single IoU threshold could not be enough for a fair comparison. A more complete evaluation is displayed in Fig. 2, with the F1-score curves at different values of T_{IoU}. Apart from Zharkov *et al.*, all the other end-to-end neural networks always outperform the other methods. This was expected since these methods are more computationally intensive and adept at complex detection problems. Among the tested classic algorithms, Yun *et al.* is by far the one that performs better at every IoU threshold, making it a valid choice when a neural network is too resource-heavy. The methods of Gallo and Soros have similar performance, with a moderate edge in favor of the second one at low T_{IoU}. Zamberletti's method is the weakest performer overall. Zharkov *et al.* reaches a very high recall, much higher than what is achieved by the classic algorithms, but scores lower in precision. All the other deep-learning-based methods reach a near-perfect precision and recall for $T_{IoU} < 0.75$. Despite being the two biggest models, Faster R-CNN and RetinaNet underperform a bit compared to other networks for $T_{IoU} > 0.75$, meaning that the generated boxes are less precise. Overall, T-DETR leads the leaderboard, albeit by a small margin. Interestingly, YOLO Nano, despite having nearly 10 times fewer parameters,

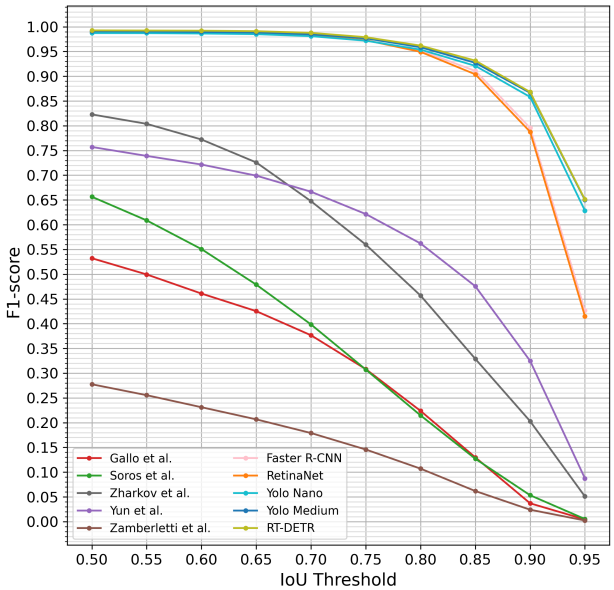

Fig. 2. F1-score of detection algorithms at different thresholds. Employed images contain a single 1D barcode and were resized to have the longest side equal to 640 pixels.

Table 4. Precision, Recall and F1-score with IoU threshold of 0.5. All images contain a single 2D barcode and were resized to have their longest side equal to 640 pixels.

Detection Method	Precision ↑	Recall ↑	F1-score ↑
Soros *et al.* [30]	0.140	0.140	0.140
Zharkov *et al.* [41]	0.727	0.900	0.804
Faster R-CNN [28]	**0.981**	0.992	0.987
RetinaNet [21]	**0.981**	0.995	**0.988**
YOLO Nano [15]	0.962	0.989	0.975
YOLO Medium [15]	0.980	0.990	0.985
RT-DETR [23]	0.972	**0.997**	0.984

performs similarly to YOLO Medium and RT-DETR, suggesting that smaller networks can excel in this detection task without sacrificing accuracy.

5.3 Single 2D Barcode Localization

In this test, we only include examples with a single two-dimensional barcode. Soros's method [30] is the only non-deep-learning-based method available that also detects 2D barcodes. The employed dataset contains 1 164 images, resized to a maximum edge length of 640 pixels. At this resolution, our dataset included 19 small objects (area $< 32^2$), 202 medium objects ($32^2 <$ area $< 96^2$), and 943 large objects (area $> 96^2$). Alongside the object's area, module density remains crucial for determining the dataset's difficulty. After resizing, the PPE ranges from 0.48 to 9.98, with most codes being uniformly distributed in the range 1.5 to 7.0. Additionally, 90 barcodes lack PPE information. As for the linear barcode case, we present the values of precision, recall, and F1-score of the tested methods considering an IoU threshold of 0.5. The results are presented in Table 4.

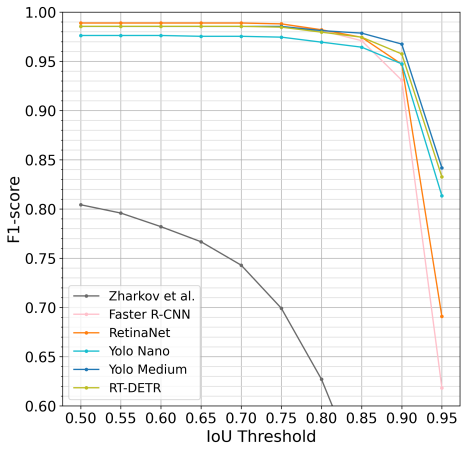

Fig. 3. F1-score curves of 2D barcode detection algorithms at different values of IoU threshold. Employed images contain a single 2D barcode and were resized to have their longest side of 640 pixels.

Table 5. Number of objects per class and size category across the entire dataset, with images resized at different resolutions.

Longest Side Resolution	Type	Small Objects	Medium Objects	Large Objects	Total
640 px	1D	172	3 613	4 277	8 062
	2D	85	611	1 060	1 756
	Total	257	4 224	5 337	9 818
480 px	1D	478	4 789	2 795	8 062
	2D	157	712	887	1 756
	Total	635	5 501	3 682	9 818
320 px	1D	1 813	5 447	802	8 062
	2D	421	574	761	1 756
	Total	2 234	6 021	1 563	9 818

It is clear that the Soros *et al.* method, with an F1 score of 0.14, is not a reliable 2D barcode detector. To better understand how the other methods perform at different IoU thresholds, we present their F1 curves in Fig. 3.

Zharkov *et al.* achieves good results, especially in recall, but falls short of the other deep learning architectures. At $T_{IoU} < 0.75$, RetinaNet performs the best in terms of F1-score, while YOLO Medium and RT-DETR have the highest score for $T_{IoU} > 0.75$. YOLO Nano has a similar performance to YOLO Medium, but now the gap is a bit larger with respect to the 1D case.

5.4 Multi-class Detection

We expand our analysis to the entirety of the dataset, encompassing both 1D and 2D barcode classes. The task is now not only about detection, but also classification. The available methods for multi-class and multi-ROI detection are the deep-learning-based models. As previously observed, deep-learning models significantly outperform classical methods in this domain. However, implementing them in industrial applications could be challenging due to the high computational costs. A potential solution is to detect barcodes at a lower resolution and execute the decoding phase at full resolution. We thus decided to run our tests at three different resolutions, to test the viability of this strategy. First, all the images are resized to have their longest side equal to 640 pixels, then to 480 pixels and 320 pixels. For each scale, we re-trained the models using a training set with the same scale used for testing. In Table 5 we see the number of instances divided by class and size. In total, 8 748 images are included, with 8 062 instances of 1D barcodes and 1 756 instances of 2D barcodes. To evaluate model performance, we calculated the Average Precision at an IoU threshold of 0.5 (AP@0.5) and the Average Precision across IoU thresholds from 0.5 to 0.95 with a step size of 0.05 (AP@[.5:.95]) for each class. In addition, we considered the corresponding mean Average Precision values (mAP@0.5 and mAP@[.5:.95]) for each model. The results are presented in Table 6. Zharkov *et al.*'s model, while not as robust as the others, achieves a respectable mAP@0.5 score of 0.823 at the 640 pixels scale. However, its performance drops significantly at the other two scales. Other models perform well at all tested resolutions. The performance

Table 6. Average precision scores for the tested models across all images of the dataset resized at different scales.

Longest Side Resolution	Model	1D barcodes		2D barcodes		Average	
		AP@0.5 ↑	AP@[.5:.95] ↑	AP@0.5 ↑	AP@[.5:.95] ↑	mAP@0.5 ↑	mAP@[.5:.95] ↑
640 px	Zharkov *et al.*	0.905	0.536	0.741	0.468	0.823	0.502
	YOLO Nano	0.986	0.902	0.960	0.910	0.973	0.906
	YOLO Medium	0.988	0.909	**0.976**	**0.930**	**0.982**	0.920
	RT-DETR	**0.989**	**0.914**	0.973	**0.930**	0.981	**0.922**
	Faster R-CNN	0.982	0.857	0.967	0.866	0.974	0.862
	RetinaNet	0.973	0.848	0.968	0.894	0.970	0.871
480 px	Zharkov *et al.*	0.380	0.180	0.661	0.465	0.521	0.322
	YOLO Nano	0.982	0.889	0.961	0.901	0.972	0.895
	YOLO Medium	**0.988**	0.899	0.966	0.917	0.977	0.908
	RT-DETR	0.987	**0.900**	**0.968**	**0.919**	**0.977**	**0.910**
	Faster R-CNN	0.979	0.843	0.953	0.843	0.966	0.843
	RetinaNet	0.963	0.830	0.948	0.866	0.955	0.848
320 px	Zharkov *et al.*	0.530	0.254	0.571	0.382	0.551	0.318
	YOLO Nano	0.976	0.860	0.947	0.872	0.961	0.866
	YOLO Medium	0.975	0.853	0.946	0.862	0.960	0.857
	RT-DETR	**0.980**	**0.875**	**0.955**	**0.893**	**0.968**	**0.884**
	Faster R-CNN	0.929	0.764	0.928	0.787	0.928	0.775
	RetinaNet	0.887	0.740	0.89	0.793	0.888	0.766

drop from 640 pixels to 480 pixels is small for most models, while downscaling to 320 pixels has a more noticeable impact. At the 640 pixels scale, Faster R-CNN and RetinaNet achieve lower scores than other models, while YOLO Medium and RT-DETR deliver the highest mAP@0.5 and mAP@[.5:.95], respectively. At the other two scales, the scores of Faster R-CNN and RetinaNet decrease more than those of YOLO and RT-DETR. RT-DETR is the best model across all metrics considered, with an increase in lead at the lowest resolution. Surprisingly, YOLO Nano has better metrics across all categories compared to YOLO Medium at 320 pixels resize, while this is not true for the other scales.

5.5 Time Measurement

In this section, we evaluate barcode detection algorithm inference times. This analysis is essential for applications running on devices with limited resources. For a comprehensive assessment, we benchmark the algorithms on two contrasting platforms: a high-end PC and a Raspberry Pi 3B+. The algorithms we tested, implemented in C++, were not specifically optimized for multi-threading, but employ a few OpenCV functions capable of multi-threaded execution. To provide a clear understanding of their performance, we ran these methods on a single CPU thread. For a balanced comparison, we also recorded the inference times of deep-learning methods running on a single CPU thread. In addition, we also report the times of deep-learning methods when running on GPU or CPU with multi-threading enabled. All C++ implementations were compiled with -O3 optimization for maximum performance. For this benchmark, we run all the detection methods on all the images of the dataset. To reduce the impact of the background processes, we repeat detections three times per image and take the lowest time. The final time is the average for every image.

Table 7. Average time required for detection on PC and on Raspberry PI. All images have been resized to have the longest side to 640 pixels. The ∞ symbol indicates that there was not enough RAM to run the algorithm.

Detection Method	Times on PC (ms)			Times on Raspberry PI (ms)	
	Single-Thread CPU ↓	Multi-Thread CPU ↓	GPU ↓	Single-Thread CPU ↓	Multi-Thread CPU ↓
Gallo *et al.* [10]	**1.63**	-	-	**53.45**	-
Soros *et al.* [30]	11.25	-	-	397.53	-
Zamberletti *et al.* [40]	48.20	-	-	1 360.23	-
Yun *et al.* [38]	7.59	-	-	146.31	-
Zharkov *et al.* [41]	25.85	**5.97**	**1.45**	2 120.43	1 949.08
YOLO Nano [15]	64.99	17.40	18.66	3 034.27	1 803.09
YOLO Medium [15]	478.92	51.36	23.91	20 083.87	15 813.46
RT-DETR [23]	985.41	141.06	37.55	39 882.45	33 224.15
Faster R-CNN [28]	1 271.93	237.91	30.27	∞	∞
RetinaNet [21]	1 124.11	105.20	36.00	∞	∞

Time on PC. We measured times when running on a PC with a 24-core AMD Ryzen Threadripper Pro 5965WX CPU, 128 GB of DDR4 RAM, and an RTX 4090 GPU. All the tests were conducted after scaling the images to have their longest side of 640 pixels. In total, we have 8 748 images, with a mean resolution of 0.284 Megapixels after resizing. Inference is conducted on a single image at a time. Table 7 presents the times required to run detection methods on a single CPU thread. For deep-learning methods, we also report multi-threaded performance and GPU performance. Focusing on single-threaded performance on the CPU, there's a significant difference between the methods, with Gallo *et al.* being the fastest (1.63 ms). This was expected since this is the oldest method, and its main focus was to run on limited hardware. Yun *et al.* is the second fastest method (7.59 ms), despite having a better detection accuracy than Soros and Zamberletti's algorithms. Zharkov *et al.* is the only deep-learning model that could run in real-time on a single core with a recorded time of 25.85 ms. YOLO Nano is also quite faster than the other deep-learning models with a mean execution time of 64.99 ms. YOLO Medium is much slower at 478.9 ms in single-thread. As expected, RT-DETR is slower with a time of 985.4 ms, and both RetinaNet and Faster R-CNN require even more time (1 124 ms and 1 272 ms respectively). Using multiple threads, all neural networks become 5-10 times faster, except YOLO Nano which becomes only 4 times faster with a time of 17.4 ms. On GPU, the ranking remains the same, but bigger models receive a bigger boost than smaller models. The fastest model is still Zharkov *et al.* at 1.45 ms while the slowest one is RetinaNet at 36 ms. All barcode detection methods could be used for real-time applications on a high-end PC. However, it is hard to find a real-world application where this makes economic sense.

We also recorded the single-thread performance when resizing the longest side to 480 pixels and 320 pixels, as deep-learning-based detectors work well even at lower resolutions. The results are shown in Table 8. At lower resolution, the ranking remains the same, but shorter times are required. Indeed, time scales more or less linearly with the amount of pixels.

Table 8. Average times required for detection on PC and on Raspberry PI, using a single thread on the CPU, at different longest side resolutions. The ∞ symbol indicates that there was not enough RAM to run the algorithm.

Detection Method	Times on PC (ms)			Times on Raspberry PI (ms)		
	Time at 640px ↓	Time at 480px ↓	Time at 320px ↓	Time at 640px ↓	Time at 480px ↓	Time at 320px ↓
Gallo *et al.* [10]	**1.63**	**0.92**	**0.41**	**53.45**	**32.04**	**14.31**
Soros *et al.* [30]	11.25	6.26	2.78	397.53	205.51	92.02
Zamberletti *et al.* [40]	48.20	29.66	17.42	1 360.23	1 357.17	855.78
Yun *et al.* [38]	7.59	4.49	2.17	146.31	103.84	52.80
Zharkov *et al.* [41]	25.85	14.56	6.72	2 120.43	882.50	340.92
YOLO Nano [15]	64.99	40.20	20.82	3 034.27	2 108.00	1 050.38
YOLO Medium [15]	478.92	284.62	135.24	20 083.87	12 091.44	5 570.13
RT-DETR [23]	985.41	604.01	329.26	39 882.45	25 371.39	13 427.26
Faster R-CNN [28]	1 271.93	892.33	599.15	∞	∞	∞
RetinaNet [21]	1 124.11	665.03	319.17	∞	∞	∞

Time on Embedded Device. Many barcode reading applications rely on embedded CPUs, such as identification marking and retail automatic checkouts. The use of embedded devices instead of PCs ensures a reduction in costs, latency, and space requirements. To measure the performance on embedded devices we run our benchmark on a Raspberry PI 3B+ (1.2GHz quad-core ARMv8 CPU, 1GB DDR2 RAM). Since the tested system is now much slower, we had to test on a subset of 500 randomly selected images of the dataset, to make the test run in a reasonable time. The mean area remained 0.284 Megapixels. Single-core CPU tests were conducted for all detection algorithms, with deep-learning methods also tested using all four cores of the CPU. Results are presented in Table 7. Compared to the PC results, execution times increased by 30-50x. Insufficient RAM prevented Faster R-CNN and RetinaNet from running. No method currently achieves real-time performance, with Gallo's method being close. The comparison between the various methods in terms of timings remains unchanged. Gallo's method is the fastest (53.45 ms), followed by Yun's (146.3 ms), Soros' (397.5 ms), and Zamberletti's (1 360 ms) algorithms. All the deep-learning methods are slower. Zharkov *et al.* is still the fastest network at 2 120 ms, followed by YOLO Nano (3 034 ms). YOLO Medium and RT-DETR are incredibly slow, with processing times of 20 084 ms and 39 882 ms respectively. Multi-core execution yielded a modest speed-up of roughly 1.5×, potentially limited by unoptimized libraries or system bottlenecks such as RAM. We also recorded the single-thread performance when resizing the longest side to 480 pixels and 320 pixels. The results are shown in Table 8. The ranking remains the same, apart from Zharkov *et al.* surpassing Zamberletti *et al.* at 320 pixels scaling. At this resolution, the time required by the smaller neural networks, Zharkov *et al.* and YOLO Nano, becomes more reasonable (340.9 ms and 1 050 ms respectively), but still far from the real-time applications target.

It is crucial to acknowledge that the speed of these methods could be significantly enhanced through optimization. For instance, the C++ methods we have tested could be optimized with SIMD intrinsics and multi-threaded code,

while the use of software toolkits for Edge AI or techniques like quantization and pruning can be employed to boost the speed of neural networks with minimal impact on accuracy. However, this goes beyond the scope of our paper.

6 Conclusion

The paper contributions include a comprehensive review of the field of barcode localization, the release of a large dataset of 8 748 images of barcodes with standardized annotations, and the public release of our benchmark. This benchmark includes multiple localization algorithms, scripts for training deep learning models, and diverse performance metrics. This ensures transparency and enables researchers to easily replicate and expand upon our work. Finally, we performed multiple tests with our benchmark, using our dataset and trained models, from which we can draw some interesting conclusions. First, our tests confirmed the significant accuracy advantage of deep learning methods over hand-crafted approaches. However, the computational complexity of most deep learning models remains a challenge for real-time embedded applications, since even fairly small models require more than one second per detection. Downscaling the image before localization gives a huge speed-up, but does not solve the problem entirely. Our findings suggest that small neural networks, such as YOLO Nano, perform nearly as well as much bigger architectures like RT-DETR and RetinaNet. Our tests also highlight the big advantage of using pre-trained general models, like YOLO or RetinaNet, over custom-built models like Zharkov's. Lastly, among the methods tailored to barcodes, Yun *et al.* proposal offers an optimal blend of accuracy and speed, surpassing Soros' and Zamberletti's methods in both metrics. The fastest method was Gallo *et al.*, showing that decent accuracy could be achieved even on very constrained devices.

As a closing remark, we hope this benchmark will be a valuable asset for further research in this field. Its modular design facilitates the integration of new algorithms, metrics, and data. We welcome feedback and contributions to further enhance the proposed benchmark.

Acknowledgements. This work was supported by the University of Modena and Reggio Emilia and Fondazione di Modena, through the FAR 2023 and FARD-2023 funds (Fondo di Ateneo per la Ricerca).

References

1. Generate a large labelled dataset of barcodes from open food facts data. https://github.com/openfoodfacts/openfoodfacts-ai/issues/15 (2018)
2. Ando, S.: Image field categorization and edge/corner detection from gradient covariance. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(2), 179–190 (2000)
3. Bodnár, P., Grósz, T., Tóth, L., Nyúl, L.G.: Efficient visual code localization with neural networks. Pattern Analysis and Applications **21**, 249–260 (2018)
4. Chai, D., Hock, F.: Locating and Decoding EAN-13 Barcodes from Images Captured by Digital Cameras. In: 2005 5th International Conference on Information Communications & Signal Processing. pp. 1595–1599 (2005)

5. Chang, S.K., Yang, C.C.: Picture information measures for similarity retrieval. Computer Vision, Graphics, and Image Processing **23**(3), 366–375 (1983)
6. Chou, T.H., Ho, C.S., Kuo, Y.F.: QR code detection using convolutional neural networks. In: International Conference on Advanced Robotics and Intelligent Systems (ARIS). pp. 1–5 (2015)
7. Do, T., Kim, D.: Quick Browser: A Unified Model to Detect and Read Simple Object in Real-Time. In: 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2021)
8. Dubská, M., Herout, A., Havel, J.: Real-time precise detection of regular grids and matrix codes. Journal of Real-Time Image Processing **11**, 193–200 (2016)
9. Galamhos, C., Matas, J., Kittler, J.: Progressive probabilistic Hough transform for line detection. In: 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. vol. 1, pp. 554–560 (1999)
10. Gallo, O., Manduchi, R.: Reading 1D Barcodes with Mobile Phones Using Deformable Templates. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(9), 1834–1843 (2010)
11. Hansen, D.K., Nasrollahi, K., Rasmussen, C.B., Moeslund, T.B.: Real-Time Barcode Detection and Classification using Deep Learning. In: International Joint Conference on Computational Intelligence. pp. 321–327 (2017)
12. Hu, H., Xu, W., Huang, Q.: A 2D barcode extraction method based on texture direction analysis. In: 2009 Fifth International Conference on Image and Graphics. pp. 759–762 (2009)
13. Jain, A.K., Chen, Y.: Bar code localization using texture analysis. In: Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR'93). pp. 41–44 (1993)
14. Jain, A.K., Karu, K.: Learning texture discrimination masks. IEEE Transactions on Pattern Analysis and Machine Intelligence **18**(2), 195–205 (1996)
15. Jocher, G., Chaurasia, A., Qiu, J.: Ultralytics YOLOv8 (2023)
16. Kamnardsiri, T., Charoenkwan, P., Malang, C., Wudhikarn, R.: 1D Barcode Detection: Novel Benchmark Datasets and Comprehensive Comparison of Deep Convolutional Neural Network Approaches. Sensors **22**(22), 8788 (2022)
17. Kapsambelis, C.: Bar Codes Aren't Going Away! (2005)
18. Klimek, G., Vamossy, Z.: QR code detection using parallel lines. In: 2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI). pp. 477–481 (2013)
19. Kubáňová, J., Kubasáková, I., Čulík, K., Štítik, L.: Implementation of barcode technology to logistics processes of a company. Sustainability **14**(2), 790 (2022)
20. Li, J., Zhao, Q., Tan, X., Luo, Z., Tang, Z.: Using Deep ConvNet for Robust 1D Barcode Detection. In: Advances in Intelligent Systems and Interactive Applications: Proceedings of the 2nd International Conference on Intelligent and Interactive Systems and Applications (IISA2017). pp. 261–267 (2018)
21. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal Loss for Dense Object Detection. In: Proceedings of the IEEE international conference on computer vision. pp. 2980–2988 (2017)
22. Lin, T.Y., et al.: Microsoft COCO: Common Objects in Context. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13. pp. 740–755 (2014)
23. Lv, W., et al.: DETRs Beat YOLOs on Real-time Object Detection. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2024)

24. McCathie, L.: The advantages and disadvantages of barcodes and radio frequency identification in supply chain management. Phd thesis, School of Information Technology and Computer Science (2004)
25. Melek, C.G., et al.: Datasets and methods of product recognition on grocery shelf images using computer vision and machine learning approaches: An exhaustive literature review. Engineering Applications of Artificial Intelligence **133** (2024)
26. Muniz, R., Junco, L., Otero, A.: A robust software barcode reader using the Hough transform. In: Proceedings 1999 International Conference on Information Intelligence and Systems (Cat. No. PR00446). pp. 313–319 (1999)
27. Ottaviani, E., et al.: A common image processing framework for 2D barcode reading. In: Image Processing And Its Applications, 1999. Seventh International Conference on (Conf. Publ. No. 465). vol. 2, pp. 652–655 (1999)
28. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Advances in Neural Information Processing Systems **28** (2015)
29. Soliman, A., Al-Ali, A., Mohamed, A., Gedawy, H., Izham, D., Bahri, M., Erbad, A., Guizani, M.: AI-based UAV navigation framework with digital twin technology for mobile target visitation. Engineering Applications of Artificial Intelligence **123**, 106318 (2023)
30. Sörös, G., Flörkemeier, C.: Blur-resistant joint 1D and 2D barcode localization for smartphones. In: Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia. pp. 1–8 (2013)
31. Szentandrási, I., Herout, A., Dubská, M.: Fast Detection and Recognition of QR codes in High-Resolution Images. In: Proceedings of the 28th Spring Conference on Computer Graphics. pp. 129–136 (2012)
32. Taveerad, N., Vongpradhip, S.: Development of Color QR Code for Increasing Capacity. In: 2015 11th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS). pp. 645–648 (2015)
33. Vaishnavi Shyamsundar Mate, S.M.: Barcode Reader Market Size, Share, Competitive Landscape and Trend Analysis Report by Type, by Application : Global Opportunity Analysis and Industry Forecast, 2023-2032 (2023)
34. Viard-Gaudin, C., Normand, N., Barba, D.: A bar code location algorithm using a two-dimensional approach. In: Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR'93). pp. 45–48 (1993)
35. Wachenfeld, S., Terlunen, S., Jiang, X.: Robust recognition of 1-d barcodes using camera phones. In: 2008 19th International Conference on Pattern Recognition. pp. 1–4 (2008)
36. Weng, D., Yang, L.: Design and Implementation of Barcode Management Information System. In: Information Engineering and Applications: International Conference on Information Engineering and Applications. pp. 1200–1207 (2012)
37. Wudhikarn, R., Charoenkwan, P., Malang, K.: Deep Learning in Barcode Recognition: A Systematic Literature Review. IEEE Access **10**, 8049–8072 (2022)
38. Yun, I., Kim, J.: Vision-based 1D barcode localization method for scale and rotation invariant. In: TENCON - IEEE Region 10 Conference. pp. 2204–2208 (2017)
39. Zamberletti, et al.: Neural Image Restoration for Decoding 1-D Barcodes using Common Camera Phones. In: VISAPP (1). pp. 5–11 (2010)
40. Zamberletti, et al.: Robust Angle Invariant 1D Barcode Detection. In: 2013 2nd IAPR Asian Conference on Pattern Recognition. pp. 160–164 (2013)
41. Zharkov, A., Zagaynov, I.: Universal Barcode Detector via Semantic Segmentation. In: 2019 International Conference on Document Analysis and Recognition (ICDAR). pp. 837–843 (2019)