



Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Survey paper

State-of-the-art review and benchmarking of barcode localization methods

 Enrico Vezzali ^a, Federico Bolelli ^a,* , Stefano Santi ^b, Costantino Grana ^a
^a Department of Engineering "Enzo Ferrari", University of Modena and Reggio-Emilia, Via P. Vivarelli 10, Modena 41125, MO, Italy

^b Datalogic S.r.l., Via San Vitalino, 13, Lippo 40012, BO, Italy


ARTICLE INFO

Dataset link: ditto.ing.unimore.it/barber

Keywords:

 Barcodes
 Benchmarking
 Quick response codes
 Object detection

ABSTRACT

Barcodes, despite their long history, remain an essential technology in supply chain management. In addition, barcodes have found extensive use in industrial engineering, particularly in warehouse automation, component tracking, and robot guidance. 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 hinders 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. Among the supported localization methods, there are multiple deep-learning detection models, that will be used to assess the recent contributions of Artificial Intelligence to this field. 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 provide a thorough summary of the history and literature on barcode localization and share the results obtained from running the benchmark on our dataset, offering valuable insights into the performance of different algorithms when applied to real-world problems.

1. Introduction

Barcodes are a visual data representation with the property of being easily readable by a machine. They are an automatic identification technology that greatly improves the accuracy and speed of data collection and identification (Weng and Yang, 2012). For this reason, plus their cost-effectiveness, barcodes have found extensive use in various real-world engineering applications. First of all, they are a cornerstone of supply chain management (McCathie, 2004), playing a crucial role in managing the flow of goods from manufacturers to consumers. They help in tracking inventory, managing logistics, and improving efficiency (Weng and Yang, 2012). 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 (Kubáňová et al., 2022). Barcodes are also a valuable tool in automation, with applications ranging from unmanned retail (Niu et al., 2023) to robot guidance (Kalinov et al., 2020; Soliman et al., 2023). Other notable applications are component tracking in manufacturing (Weng and Yang, 2012) and product recognition in retail (Melek et al., 2024). Despite their inception over seven decades ago, barcodes continue to hold their ground in today's

digital age. The AIDC 100 projects a consistent utilization of barcodes in the forthcoming years (Kapsambelis, 2005), a prediction that is corroborated by scholarly literature (Kubáňová et al., 2022). 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 (Vaishnavi Shyamsundar Mate, 2023).

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 (Taveerad and Vongpradhip, 2015). The process of reading a barcode can usually be divided into two macro steps: localization and decoding. Some papers focus on both steps (Gallo and Manduchi, 2010; Tekin and Coughlan, 2012; Klimek and Vamossy, 2013). However, most of the publications just focus on the localization part. Especially in recent times, it has become

* Corresponding author.

E-mail addresses: enrico.vezzali@unimore.it (E. Vezzali), federico.bolelli@unimore.it (F. Bolelli), stefano.santi@datalogic.com (S. Santi), costantino.grana@unimore.it (C. Grana).

<https://doi.org/10.1016/j.engappai.2025.110259>

Received 6 May 2024; Received in revised form 23 December 2024; Accepted 6 February 2025

Available online 21 February 2025

0952-1976/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

the norm to use public third-party libraries to handle the decoding step (Wudhikarn et al., 2022). The two most used libraries are ZXing¹ and Zbar.² Each software tool can handle both 1D and 2D barcodes. Therefore, our primary focus from now on will be on the localization.

Barcode recognition is often used in industrial applications, where accuracy and speed are paramount. Until recently, real-time speed for a localization algorithm was achievable solely through the computation of hand-crafted features from the image. In this regard, linear barcodes have two main features: they are made of high-contrast lines and these lines are parallel. To exploit these two features, most barcode localization methods typically involve an initial edge-detection phase and an aggregation phase where edges with similar directions are grouped (Viard-Gaudin et al., 1993; Tekin and Coughlan, 2012; Yun and Kim, 2017). Two-dimensional barcodes are instead made up of two sets of parallel lines rotated 90° from one another. A common strategy is to use the Hough Transform to find a set of perpendicular lines (Hu et al., 2009; Szentandrás et al., 2012; Klimek and Vamossy, 2013). With the significant breakthrough of AlexNet in 2012 (Krizhevsky et al., 2012), deep-learning methods have come to dominate the field of Computer Vision (Bhatt et al., 2021). The recent increase in the significance of deep learning has been fueled by the appearance of large, high-quality, publicly available labeled datasets, coupled with the huge advancements in GPU computing (Voulodimos et al., 2018). Initially, neural networks were used to process extracted features of the image, since end-to-end models require more processing time. This was the case for the method proposed by Zamberletti et al. (2013), which used a neural network to process the Hough Transform of the image. However, in the following years, the use of end-to-end models became more prominent. According to Wudhikarn et al. (2022), between the years 2015 and 2021, 25 publications introduced a method for barcode localization (either 1D, 2D, or both) that utilized deep learning techniques. Out of the 25 papers reviewed, 9 utilized a custom CNN model. The remaining papers employed publicly available architectures, with YOLO (Redmon et al., 2016) being the most popular among them, followed by Faster R-CNN (Ren et al., 2015) and SSD (Liu et al., 2016). Numerous detection methods for barcodes and QR codes based on deep learning have been proposed, with many papers featuring comparisons between two or more methods (Sörös and Flörkemeier, 2013; Yun and Kim, 2017; Kamnardsiri et al., 2022). However, several issues prevent definitive conclusions about the methods' effectiveness.

Dataset availability. Most of the literature relies mostly on just two public datasets for 1D barcodes (Wudhikarn et al., 2022): *Arte-Lab* (Zamberletti et al., 2010) and *WWU Muenster* (Wachenfeld et al., 2008), comprising 430 and 1 055 images respectively. For 2D barcodes, the most used public dataset is *Dubská QR* (Dubská et al., 2016) of 400 images. These datasets are relatively small, and as subsequent publications continue to improve their scores on these datasets, it becomes challenging to determine whether these improvements translate to real-world applications. Furthermore, these datasets are certainly too small to train Neural Networks for object detection. A few other public datasets have been used, but they are also quite small (Wudhikarn et al., 2022). Datasets of barcodes of more than 5 000 images are just of two types: private (Ventsov and Podkolzina, 2018; Yuan et al., 2019; Do and Pham, 2021; Zhang et al., 2021) or synthetic (Bodnár et al., 2018; Quenum et al., 2021; Monfared et al., 2021). However, the reliance on synthetic datasets for evaluation can lead to misleading results. For instance, the method proposed by Katona and Nyúl (2013) achieved an accuracy of 96.8% on a synthetic dataset, but when tested on the Muenster Dataset, the average accuracy dropped to 19.8% (Sörös and Flörkemeier, 2013). Another issue is the lack of standardization across datasets. Different datasets employ different formats, which complicates the use of multiple datasets without additional preprocessing.

Reproducibility. Most of the publications in this research area are not followed by a publication of the code used for the test. This makes it much more time-consuming to reproduce the experiments since it requires writing the code from scratch. Even in the rare instances where code is available, it is uncommon to find subsequent studies that employ the same algorithm, datasets, and metrics. The use of different coding languages and frameworks further complicates the comparison of different methods.

Metrics consistency. The third issue is that different studies use varying metrics, leading to contradictory results even with identical algorithms and datasets. For instance, Sörös and Flörkemeier (2013) compared his proposed method against Tekin and Coughlan (2012) and Gallo and Manduchi (2010) methods on the Muenster dataset. The Jaccard's Index (J) (Etude, 1901) was used as a metric, with a good detection defined as $J > 0.5$. Tekin's method ranked highest, followed by Gallo and then Sörös. Nevertheless, in a following paper (Yun and Kim, 2017) compared the same algorithms on the same dataset using the Dice Similarity Coefficient (DSC) (Dice, 1945) with a threshold of 0.8. This time, Sörös' method ranked first, followed by Gallo's and Tekin's. Changing the evaluation metric can completely alter the ranking of these methods. Therefore, it is crucial to use a consistent set of metrics when comparing different experiments and to employ multiple metrics for a comprehensive comparison.

The goal of this paper is to address these challenges in barcode localization research. In particular, this work offers several key contributions:

1. First, we will present an exhaustive review of existing methods for barcode localization, synthesizing the various approaches present in the literature. This will provide a strong foundation for understanding the current landscape of the field;
2. The public release of an annotated dataset of 8 748 images. This dataset merges multiple public datasets of 1D and 2D barcodes, standardizing the annotation formats. While all the annotations are provided in VGG (Dutta and Zisserman, 2019) format, they can be easily converted to COCO (Lin et al., 2014) or YOLO (Redmon et al., 2016) formats using a conversion script that we make available. Barcode regions are described with polygons, allowing both detection and segmentation;
3. The creation of BarBeR (Barcode Benchmark Repository), a public benchmark for barcode detection. This benchmark includes a set of default algorithms for comparison but can be readily extended to encompass any localization algorithm. In addition, our benchmark incorporates a range of metrics that can be utilized to evaluate the performance of barcode detection algorithms. The benchmark is open-source, as well as the scripts to train deep-learning models on the proposed dataset, promoting the reproducibility of our findings and facilitating further research in the field by allowing researchers to build upon our work.

This article is structured as follows. Section 2 describes the publicly available datasets we found, their characteristics, and our process for developing a unified annotation standard. Section 3 provides a history of proposed methods for both 1D and 2D barcode detection. In Sections 4 and 5 we detail the publicly available detection algorithms and the deep-learning architectures selected to be included in our benchmark, respectively. After that, Section 6 explores evaluation metrics used in object detection and our implementation choices. Then in Section 7, we can find a description of our benchmark's repository, its structure, and the available tests and methods. Sections 8, 9, and 10 present the collected benchmark results, covering single-class detection (1D or 2D), multi-class detection, and timing measurements. Finally, the ending conclusions will be presented in Section 11.

¹ <https://github.com/zxing/zxing>

² <https://github.com/ZBar/ZBar>



Fig. 1. Sample of images of the dataset. Some images contain a single linear barcode or a single bidimensional barcode. Other images, instead, contain multiple codes, sometimes of multiple classes.

2. Datasets

For this project, we needed a large enough dataset to reliably compare different algorithms and train object detection neural networks. For this reason, we reviewed the available literature on barcode detection and decoding to identify publicly available datasets.

WWU Muenster. Released in 2008, it is the oldest dataset we have found. It contains 1055 images of 1D barcodes taken with a Nokia N95 (Wachenfeld et al., 2008). All the images have a resolution of 2592×1944.

Artelab Medium Barcode 1D (Zamberletti et al., 2010). Published in 2010, it consists of 430 images of linear barcodes. All the images depict a single barcode near the center of the image, with a rotation of $\pm 30^\circ$.

Szentandrási QR. Published by Szentandrási et al. (2012), the dataset consists of 115 images of QR codes. Most of these images have a resolution of 15 MegaPixels and contain multiple QR codes.

Dubaska QR. The same group of the Szentandrási QR dataset published another dataset of 810 images containing QR codes, half of which are captured with a camera and the other half with a smartphone (Dubská et al., 2016). There are 25 images common to both the Szentandrási QR and Dubaska QR datasets; these were removed from the former to avoid duplication.

Arte-Lab Extended 1D. In 2013, the group that published the Arte-Lab dataset, released two additional datasets: Arte-Lab Rotated 1D and Arte-Lab Extended 1D, of 365 and 155 images each (Zamberletti et al., 2013). Both contain only images of 1D barcodes. However, as Arte-Lab Rotated uses the same objects captured in the original Arte-Lab dataset, we decided not to include it in our study due to its high similarity.

Bodnar-Huawei. Released in 2018, it consists of 98 images of QR codes taken with a Huawei smartphone, all of which contain a single code and have a resolution of 1600×1200 (Bodnár et al., 2018).

Skku Inyong DB. Released in 2017, the dataset contains 325 images at a resolution of 1440×2560 with multiple instances of linear barcodes (Yun and Kim, 2017).

ZVZ-Real. Another interesting dataset was published by Zharkov and Zagaynov (2019) under the name ZVZ-Real. The dataset contains 921

images of a large variety of 1D and 2D barcodes. Some images depict multiple barcodes.

DEAL KAIST. It is the largest barcode dataset that we collected (Do and Kim, 2021) and is usually referred to as DEAL KAIST Barcode or QuickBrowser dataset. The dataset contains 3308 images of barcodes (mostly linear) at various resolutions (from 141×200 to 3480×4640).

InventBar and ParcelBar. Finally, Kannardsiri et al. (2022) worked on developing other two datasets of linear barcodes, used for neural network training and testing. The two datasets have 527 and 844 images each. ParcelBar in particular, is one of the most difficult datasets for barcode detection that we have found since the barcodes are very small compared to the size of the images.

OpenFood Facts. In addition to these public datasets used in literature, we collected other 185 images of linear barcodes from Open Food Facts github.³

The collected datasets account for a total of 8748 images with 9818 annotated barcodes, 8062 linear, and 1756 two-dimensional. An example of images contained in the dataset is presented in Fig. 1, while Table 1 provides a breakdown of every dataset used, plus some information about them. The symbologies of 1D barcodes included are 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. For 2D barcodes, the included symbologies are Aztec, Datamatrix, PDF-417, and QR Code.

One immediate challenge we encountered was that not all datasets had annotations, and the ones available followed widely different formats. Some annotations were designed for object detection, others for segmentation. Furthermore, there were unannotated 2D barcodes in the datasets intended for 1D detection and vice versa. Consequently, we created new annotations for all the images in the dataset. Datalogic's proprietary software was used to automate the generation of the annotations. This tool generates a 4-point polygon for every barcode read and provides additional information about the code, such as its type,

³ <https://github.com/openfoodfacts/openfoodfacts-ai/issues/15>.

Table 1

List of the public datasets collected for the benchmark. # Images is the number of images in the dataset. Minimum and maximum resolution refers to 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 the dataset respectively.

Dataset name	# Images	Minimum resolution	Maximum resolution	# 1D	# 2D
Arte-Lab Medium 1D (Zamberletti et al., 2010)	430	1 152×864	2 976×2 232	430	7
Arte-Lab Extended 1D (Zamberletti et al., 2013)	155	648×488	648×488	165	3
Bodnár-Huawei QR (Bodnár et al., 2018)	98	1 600×1 200	1 600×1 200	0	98
DEAL KAIST Lab (Do and Kim, 2021)	3 308	141×200	3 480×4 640	3 378	76
Dubská QR (Dubská et al., 2016)	810	402×604	2 560×1 440	0	806
InventBar (Kamnardsiri et al., 2022)	527	480×640	480×640	530	33
Muenster 1D (Wachenfeld et al., 2008)	1 055	1 600×1 200	2 592×1 944	1 068	1
OpenFood Facts	185	390×520	5 984×3 376	187	5
ParcelBar (Kamnardsiri et al., 2022)	844	1 108×1 478	1 478×1 108	1 196	17
Skku Inyong DB (Yun and Kim, 2017)	325	1 440×2 560	1 440×2 560	368	10
Szentandrási QR (Szentandrási et al., 2012)	90	1 024×768	4 752×3 168	0	225
ZVZ-Real (Zharkov and Zagaynov, 2019)	921	407×576	3 288×4 930	740	475
Total	8748	200 × 141	5 984 × 3 376	8 062	1756



Fig. 2. (a) shows an element (or module) of a linear barcode, while (b) shows an element (or module) of a 2D barcode.

and the encoded string. In addition, we have information about the pixel density of the barcode, usually measured in pixels per element (PPE), i.e., the mean width of the smallest element in a barcode (shown in Fig. 2). This measure can be referred to as pixels per module (PPM).

While most codes were annotated this way (8096), a few (1722) were un-decodable due to blur, noise, or incorrect scale (either too high or too low resolution). These additional codes have been annotated by hand using VGG annotator (Dutta and Zisserman, 2019) and they lack some information like the PPE. The missing fields have been filled with a symbolic value of -1 . As said, the annotations use polygons instead of boxes and thus are suitable for both detection and segmentation.

3. Algorithms history

3.1. Early barcode localization efforts

Joseph Woodland and Bernard Silver invented the linear barcode in 1949 and patented it in 1952 (Norman J. Woodland, 1949). Alongside the barcode proposal, they described a method for reading it. The idea was to focus light directly on the code and a photocell was tasked to convert the reflective light into an analog signal. Finally, an analog circuit was used to decode this signal. Laser scanners became the primary decoding method in the '70s, leading to numerous optical innovations (Hildebrand, 1977; Reich, 1977; Neyroud et al., 1980). 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 a wider field of view, but to do so, the barcode must first be located. Viard-Gaudin et al. (1993) proposed an algorithm using a set of Sobel filters and blob detection. Liao et al. (1995) used an edge detector to find the barcode edges and their direction to separate the barcode from the background. The following year, Jain and Karu (1996) explored the use of Multi-Layer Perceptrons (MLP) (Bishop,

1995) for texture classification. In one of the experiments, the authors showed how this method could be used for barcode localization and segmentation. The Hough Transform (Duda and Hart, 1972) gained traction for linear barcode localization with Muniz et al. (1999)'s work. The same year, Ottaviani et al. (1999) presented the first QR code localization method which was based on gradient histograms.

3.2. Evolution and recent approaches

Over the years, numerous publications have addressed the limitations of barcode localization and expanded their effectiveness to more general applications. Chai and Hock (2005) proposed a method based on skeletonization (Gonzalez and Woods, 2002) to locate linear barcodes. A method based on texture direction analysis to localized 2D barcodes was tested by Hu et al. (2009). Gallo and Manduchi (2010) described a method for locating linear barcodes that is fast enough to be used on a mobile phone of the time. However, it lacked the characteristic of being rotation invariant. Tekin and Coughlan (2012) released an Android application called BLADE (Barcode Localization and Decoding Engine), that was orientation invariant and could work in real-time on smartphones. The same year, Szentandrási et al. (2012) showed the Hough Transform's applicability to QR code localization. Sörös and Flörkemeier (2013) proposed a matrix structure-based method for 1D and 2D barcodes, while Zamberletti et al. (2013) described a detection algorithm based on the Hough Transform and machine learning.

3.3. The deep learning era

Chou et al. (2015) marked a shift by utilizing a small convolutional neural network (CNN) for QR code detection. Deep-learning-based methods have since dominated the field, with the notable exception of Yun and Kim (2017), which relied on the orientation histogram. In the same year, Hansen et al. (2017) trained and tested a YOLO-v2 network on the ArteLab and WWU Muenster datasets, achieving impressive results. The following year, Li et al. (2018) reported even higher accuracy scores on the same datasets using a Faster R-CNN. Zharkov and Zagaynov (2019) proposed the use of a Dilated-Convolution network for the segmentation of various linear and bidimensional barcodes. Do and Kim (2021) introduced a unified model for real-time detection and decoding of barcodes and simple objects by integrating multi-digit recognition into a one-stage detection model. More recently, Quenum et al. (2021) tackled barcode detection in ultra-high-resolution images with a pipeline combining a modified Region Proposal Network (RPN) and a Y-Net segmentation network, surpassing YOLO-v4 and Mask R-CNN in both speed and performance. Numerous other papers on deep-learning-based barcode detection have been proposed in recent years (Wudhikarn et al., 2022).

Table 2
Characteristics of the public algorithms tested.

Method	1D	2D	Multi ROI	Rotation invariance	Rotated box
Gallo and Manduchi (2010)	✓	✗	✗	✗	✗
Sörös and Flörkemeier (2013)	✓	✓	✗	✓	✗
Zamberletti et al. (2013)	✓	✗	✓	✓	✓
Yun and Kim (2017)	✓	✗	✓	✓	✗

4. Available algorithms

Numerous algorithms for linear and two-dimensional barcodes have been proposed, yet many lack publicly available implementations. In total, we selected five functioning algorithms implemented in C++, which are the ones with the highest number of comparisons and have a public implementation available. These algorithms are: Gallo and Manduchi (2010), Tekin and Coughlan (2012), Sörös and Flörkemeier (2013), Zamberletti et al. (2013) and Yun and Kim (2017). Since Tekin's method generates scan lines rather than detection boxes, it was excluded from our comparison, but it is included in the repository for testing. Because our benchmark is written in Python, these methods have been compiled to be loaded in Python using Ctypes.

The rest of this section will detail how the available detection algorithms work. In Table 2, a summary of the main characteristics of these selected methods is also reported.

4.1. Gallo and Manduchi

This localization method was proposed by Gallo and Manduchi (2010). For brevity, we will refer to this algorithm with Gallo et al. or Gallo's method. This algorithm was engineered for speed, enabling it to operate on contemporary mobile devices. The method was designed to accurately identify linear barcodes, even in images compromised by blur or noise. However, it is tailored specifically for 1D barcodes and yields a single Region of Interest, making it unsuitable for detecting multiple barcodes within a single image. A key assumption underpinning this algorithm is the orientation of the barcode. It assumes that the barcode is positioned horizontally, with its parallel lines aligned vertically. Only rotations of less than $\pm 30^\circ$ are allowed for reliable detection.

The first step is to calculate a heatmap $I_e(n)$ which is the difference between the horizontal and vertical derivatives:

$$I_e(n) = |I_x(n)| - |I_y(n)| \quad (1)$$

$I_e(n)$ should reasonably have higher values in the barcode region than elsewhere. A box filter is applied to $I_e(n)$ to obtain the smoothed heatmap $I_s(n)$. After that, $I_s(n)$ is binarized with a single threshold using Otsu's method (Otsu, 1979). The binarized heatmap may contain multiple blobs, but the method assumes that only a single barcode is present in the image. Therefore, only one blob is selected, and it is the one that contains the pixel n_0 which is the one that maximizes $I_s(n)$. Vertical and horizontal lines, parallel to the image borders, are traced from n_0 , forming a rectangle with sides parallel to the axes of the image and containing the intersections of these lines with the edge of the blob. The horizontal line $l(n)$ that passes through the center of this rectangle is chosen as the scanline. Usually, the blob includes the quiet zone of the barcode too, because of the large size of the box filter. To remove it, the scanline is reduced from both sides until an intensity of less than 85% of the mean is found.

4.2. Soros and Florkemeier

Sörös and Flörkemeier (2013) proposed a barcode detection method designed for both 1D and 2D barcodes that is orientation invariant and is quite resistant to blur. For brevity, we will refer to this algorithm

with Soros et al. or Soros's method. However, this method can only output a single ROI for each barcode type. The idea is to compute two heatmaps, one for linear codes and one for two-dimensional codes. The first step is to compute the structure matrix M for every pixel p of the image:

$$M = \begin{bmatrix} C_{xx} & C_{xy} \\ C_{xy} & C_{yy} \end{bmatrix} \quad (2)$$

where the c_{ij} entries of M are computed from the horizontal and vertical derivatives of the image I_x and I_y over an image patch D around the pixel p using a window w :

$$C_{ij} = \sum_{(x,y) \in D} w(x,y) I_i(x,y) I_j(x,y) \quad (3)$$

The values c_{ij} are used to compute the Unidirectional Variance Detector and the Omnidirectional Variance Detector defined by Ando (2000). These two measures are indicated respectively with m_1 and m_2 and are defined as:

$$m_1 = \frac{(C_{xx} - C_{yy})^2 + 4C_{xy}^2}{(C_{xx} + C_{yy})^2 + \epsilon} \quad (4)$$

$$m_2 = \frac{4(C_{xx}C_{yy} - C_{xy}^2)}{(C_{xx} + C_{yy})^2 + \epsilon} \quad (5)$$

The values m_1 and m_2 are computed for every pixel, generating two heatmaps. The value m_1 (Unidirectional Variance Detector) is strong where edge structures are present, and m_2 (Omnidirectional Variance Detector) is high at corners. The value ϵ is a small constant that avoids 0/0 conditions in flat areas. After calculating m_1 and m_2 , a box filter is applied to each heatmap. The two box-filtered maps are linearly combined to get the two barcode saliency maps s_1 and s_2 , one for linear codes and the other for 2D codes. Finally, the resulting images are thresholded, and the barcode box is found by tracing the binary image from the pixel with maximal strength, following Gallo's method.

4.3. Zamberletti et al.

This method was proposed by Zamberletti et al. (2013). The first step of the algorithm is to apply Canny Edge Detector (Canny, 1986) to the image I obtaining the edge map I_e . Once the edge map has been determined, the Hough Transform of I_e is computed in the two-dimensional Hough Transform space H . A line in I is represented as a point in H and the (x, y) coordinates of this point represent ρ and θ , where ρ is the distance of this line from the origin and θ is its angle. The two-dimensional Hough transform of the image indicated with A_H , is divided into cells of size $m \times n$ that are processed by a Multi-Layer Perceptron (MLP) one at a time. The output of the MLP has the same dimension as the input, so that processing each cell we obtain a matrix with the same dimensions of A_H . Each value of A_H indicates the probability of that cell to contain the lines corresponding to a barcode. The algorithm assumes that all barcodes are oriented with the same angle and this angle is predicted by taking the angle θ_b with the most barcode lines, i.e. the column of A_H with the highest sum of its elements. After the angle prediction, the algorithm proceeds by finding the set S of all the segments in I by applying the same technique of Galamhos et al. (1999) to A_H . We call $S_b \subset S$ the set of all the segments with angles differing less than $\pm 5^\circ$ from θ_b . A binary image I_{s_b} is created, in which the intensity value assigned to the pixels of the segments of S_b is 1, and the others are assigned a 0. I_{s_b} is then rotated by $90^\circ - \theta_b$ degrees, so that most of the segments are vertical. Then, two histograms are defined to describe the intensity profile of the rows and columns of the binary image. Each bin of these histograms is computed as the sum of the elements of a row/column in the binary image. A smoothing filter is applied to each histogram. Finally, the bounding box of the barcode is determined as the intersection area between the rows and columns associated with the remaining non-zero bins in the histograms. This algorithm can generate multiple rotated boxes, but all

Table 3
Characteristics of the deep-learning models used in our tests.

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

detections have the same angle. This can be useful in the case of a single label with multiple barcodes, where every code has the same angle of rotation.

The open source code available online⁴ is compatible with OpenCV versions ≤ 2 . We modified it to make it compatible with OpenCV 4.

4.4. Yun and Kim

This detection method was described by Yun and Kim (2017). For brevity, we will refer to this algorithm with Yun et al. or Yun's method. The algorithm is designed for the detection of linear barcodes and supports multiple detections per image. The first step is to compute the image derivatives ∇I_x and ∇I_y using the Sobel operator to the gray-scale version of the image and use them to compute the module and angle of the gradient:

$$mag(p) = \nabla I_x + \nabla I_y \quad (6)$$

$$ang(p) = \arctan\left(\frac{\nabla I_y}{\nabla I_x}\right) \quad (7)$$

An orientation histogram h_G is computed by counting how many pixels with a magnitude bigger than the threshold T_{mag} there are for every orientation. The histogram has 18 bins in total, each of which covers 10° . The mapping $V_{h_G}^{map}(b)$ separates the principal orientation components from the weak orientation components analyzed in the h_G :

$$V_{h_G}^{map}(b) = \begin{cases} O_s, & \text{if } h_G(b) > T_{hist} \\ O_w, & \text{otherwise} \end{cases} \quad (8)$$

$$T_{hist} = \max_b(h_G(b)) \times \alpha \quad (9)$$

where O_s denotes a principal orientation component, O_w is a weak orientation component and T_{hist} is the threshold used to separate the principal orientation components. The constant α is a ratio constant used to calculate T_{hist} and $0 < \alpha < 1$. For detecting the salient regions, the entropy scheme is used (Chang and Yang, 1983). The image is divided into non-overlapping cells. To each cell, we assign a direction, and this direction is the orientation with the highest value in the local orientation histogram h_L . We indicate with i_{max} the index of the maximum component of h_L . The entropy of every patch f is calculated as follows:

$$E(f) = \begin{cases} J, & \text{if } V_{h_G}^{map}(i_{max}) = O_s \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

$$J = \sum_i h_L(i) - h_L(i_{max}) \quad (11)$$

$E(f)$ is small if the principal component of the patch is much stronger than the others, indicating a high probability of a barcode region. The entropy map $E(f)$ is thresholded to obtain the saliency map $S(f)$. After setting the important region, a box filter is used to blur $S(f)$ to eliminate the noise regions and to connect the separated barcode regions. Then the saliency map is binarized using Otsu's binarization. Finally, to determine each blob's central point and bounding box, connected components labeling is used.

5. Deep-learning models

As discussed earlier, most of the barcode detection algorithms proposed in the latest years relied on deep-learning detection models. Some authors released their trained architectures like Zharkov and Zagaynov (2019). We decided to include it in our benchmark, but due to the overlap between our test set and the training set of this network, we re-trained it from scratch. In addition, we selected a few mainstream architectures pre-trained on the MS-COCO (Lin et al., 2014) dataset. These networks were then fine-tuned on our dataset using transfer learning. In total, six different architectures have been tested: (Zharkov and Zagaynov, 2019), Faster R-CNN (Ren et al., 2015), RetinaNet (Lin et al., 2017b), YOLO Medium and Nano (Redmon et al., 2016) and RT-DETR (Lv et al., 2023). Table 3 contains a summary of the main characteristics of these architectures.

5.1. Zharkov and Zagaynov

In 2019, Zharkov and Zagaynov (2019), proposed a custom convolutional neural network architecture for the detection of 1D and 2D barcodes. For brevity, we will refer to this architecture with Zharkov et al. Their architecture is composed of three key modules:

- **Downscale module.** This module consists of three convolutional layers and two downscaling layers. It reduces the input image resolution by a factor of four, expanding the network's receptive field for more efficient processing;
- **Context module.** Inspired by the work of Yu and Koltun (2015), this module utilizes dilated convolutions to further increase the networks' receptive field. It comprises 9 convolutional layers with different dilation factors;
- **Classification layer.** It is a 1×1 convolutional layer, with $1 + n_{classes}$ outputs, where $n_{classes}$ it is the number of different barcode types (two in our tests).

The network produces a multi-channel output map at a quarter of the input resolution. The first channel indicates potential barcode locations through higher values. After applying a threshold, individual blobs are extracted, and bounding boxes are generated. The other channels of the output are used to classify every blob, with the highest-scoring channel determining the barcode's type. Finally, a confidence score based on the mean value of predicted pixels in the first channel is assigned to each detected box.

To train the network, we used the loss function proposed by its original paper. This loss function is designed to prioritize high recall over high precision.

5.2. Faster R-CNN

Faster R-CNN (Ren et al., 2015) has been the first near-realtime deep learning detector to be developed (Zou et al., 2023). Faster R-CNN is a two-stage object detection network, meaning that it generates regions of interest before defining candidate bounding boxes. Its architecture is comprised of two main components:

⁴ <https://github.com/SimoneAlbertini/BarcodeDetectionHough>

- **Region Proposal Network (RPN).** A fully convolutional network that proposes regions. A convolutional backbone is used to extract features from the image. Then, for each sliding window of the feature map, it proposes a set of regions. The proposals are parameterized relative to a set of reference anchor boxes;
- **Fast R-CNN detector.** A neural network that given an image and a set of possible ROIs detects up to k objects per ROI. For every object, a bounding box and a classification label is generated (Girshick, 2015).

The main contribution of Faster R-CNN is that the RPN runs nearly cost-free because it runs directly on the extracted features. In this way, the feature extraction backbone needs to run only one time, and its output is used by both the RPN and Fast R-CNN. A following improvement to Faster R-CNN was the Feature Pyramid Network (Lin et al., 2017a), which links high-level and bottom-level feature data, improving small-sized object detection.

This neural network was selected for our tests because it is the second most used for barcode detection (Kamnardsiri et al., 2022). In addition, Faster R-CNN is one of the most cited papers for deep-learning-based object detection. In our experiments, we used ResNet-50 (He et al., 2016) with FPN as a backbone for Faster R-CNN.

5.3. RetinaNet

The RetinaNet model was first described by Lin et al. (2017b). RetinaNet is a one-stage network composed of a backbone network and two task-specific subnetworks:

- **Backbone network.** It is composed of a bottom-up pathway and a top-down pathway with lateral connections. The bottom-up pathway is used for feature extraction, calculating the feature maps at different scales. The top-down pathway upsamples the spatially coarser feature maps in subsequent steps. Same-scale features from both pathways are then merged together.
- **Subnetwork for object detection.** It does class-agnostic bounding box regression. Detection is done relative to translation-invariant anchor boxes at different scales.
- **Subnetwork for object classification.** Predicts the probability of object presence at each spatial position for each of the anchors and object classes. It does not share weights with the object detection subnetwork.

RetinaNet introduces a novel loss function called Focal Loss, which is designed to address the issue of class imbalance during training. Despite being a very famous object detector, there are no barcode localization papers that have used RetinaNet. In our experiments, we used ResNet-50 FPN as a backbone, the same that we selected for Faster R-CNN.

5.4. YOLO

YOLO, introduced by Redmon et al. (2016), revolutionized object detection with its speed, paving the way for real-time applications. It is a one-stage network that predicts bounding boxes and probabilities for each region of the image. Despite initial localization accuracy issues, especially for small objects. The following iterations of the networks have paid more attention to this problem, significantly increasing its performance (Zou et al., 2023):

- **Yolo v2** (Redmon and Farhadi, 2017) incorporated batch normalization, anchor boxes, and dimension clusters;
- **Yolo v3** (Redmon and Farhadi, 2018) enhanced the backbone and enabled multi-scale detection by making predictions at three different scales of granularity (Hussain, 2023);
- **Yolo v4** (Bochkovski et al., 2020) introduced feature aggregation (Lin et al., 2017a) and an SPP block (He et al., 2015) to increase the receptive field and feature separation.

Other incremental changes in architecture, like anchor-free detection and improvements in the loss functions, have been applied from YOLO-v5 (Jocher, 2020) to YOLO-v7 (Wang et al., 2022).

In January 2023, Ultralytics, the same team that released YOLO-v5 (Jocher, 2020), confirmed the newest member of the YOLO family with the launch of YOLO-v8 (Jocher et al., 2023). While a detailed paper and additional features are still in the pipeline for the YOLO-v8 repository, preliminary comparisons indicate that it surpasses its predecessors, establishing a new benchmark in the YOLO series (Hussain, 2023).

We decided to include YOLO in our benchmark since it is one of the most famous object detection architectures and the one most used in barcode detection literature (Wudhikarn et al., 2022). In addition, the focus on performance and efficiency makes YOLO an ideal candidate for industrial applications (Hussain, 2023).

For our tests, we used the latest version of this network i.e. YOLO-v8. In particular, we tested two architectures: YOLO-v8 Medium and YOLO-v8 Nano. The former is the standard architecture for YOLO-v8. The latter, while maintaining a similar structure, is a more compact network with fewer layers and channels, resulting in an eightfold reduction in weight.

5.5. RT-DETR

Transformers, introduced by Vaswani et al. (2017), have revolutionized the field of Natural Language Processing (NLP). They are based on the attention mechanism, which allows the model to focus on different parts of the input sequence when generating the output. This ability to handle dependencies regardless of their position in the input makes Transformers particularly effective for NLP tasks. The incredible achievements of Transformers in NLP motivated researchers to explore their applications in computer vision tasks. Today, the highest mAP score on the MS-COCO dataset (Lin et al., 2014) for object detection has been achieved by Co-DETR (Zong et al., 2023), a transformer-based detector (DETR) (Shah and Tembhurne, 2023). However, the high computational cost of DETRs makes them hardly suitable for real-time applications. To solve this problem, Lv et al. (2023) proposed in 2023 a faster DETR, called RT-DETR, that could work in real time. RT-DETR architecture is made up of three main components:

- **Backbone network.** It is a convolutional neural network that extracts features from the image at different scales;
- **Hybrid encoder.** It transforms the multi-scale features from the backbone into a sequence of image features;
- **Transformer decoder.** First, an IoU-aware query selection is employed to select a fixed number of image features from the encoder. These selected features serve as initial object queries for the decoder. Finally, the decoder, equipped with auxiliary prediction heads, iteratively refines these object queries to generate bounding boxes and confidence scores.

It is important to note that this architecture, along with the other transformer-based networks, eliminates the need for non-maxima suppression, thereby accelerating the post-processing stage. We opted to incorporate RT-DETR into our evaluations to ensure the inclusion of a transformer-based network in our tests. As pointed out before, these networks are currently excelling in object detection, yet no existing papers have applied transformers to barcode detection.

6. Evaluation metrics

One of the primary objectives of our research is to introduce a suite of metrics for barcode detection and localization that are needed to evaluate the test results. In particular, we have described two main types of algorithms so far:

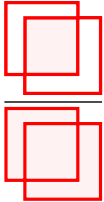
- **Non-deep-learning-based.** These methods utilize traditional computer vision techniques for object detection. The image features are typically hand-crafted. While they may incorporate smaller machine-learning models for feature processing, they are not trained end-to-end;
- **Deep-learning-based.** Rely on deep-learning models for both feature extraction and processing.

The key difference is that non-deep-learning algorithms output boxes and classes, but not confidence scores. Deep-learning detection models, however, generate confidence scores for each predicted bounding box. Consequently, we will divide the metrics into two categories: metrics that do not require confidence scores and metrics that do.

6.1. Metrics that do not require confidence

6.1.1. Intersection over union

To evaluate the quality of detection we want to measure how close the detected bounding boxes are to the ground-truth bounding boxes. By far the most common metric used to do so is the Intersection over Union (IoU), also called Jaccard Index (Etude, 1901). This measurement is done independently for each object class (Padilla et al., 2021). The IOU is equal to the area of the overlap (intersection) between the predicted bounding box B_p and the ground-truth bounding box B_{gt} divided by the area of their union:

$$IoU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area of } B_p \cap B_{gt}}{\text{area of } B_p \cup B_{gt}} \quad (12)$$


An ideal match has an IoU of 1, while no intersection results in an IoU of 0. The closer to 1, the better the detection. IoU values are usually expressed in percentages, with 50% and 75% being the most used thresholds (Padilla et al., 2021).

6.1.2. Precision and recall

Precision measures a model's ability to identify only relevant objects, while recall assesses its success in finding all existing objects. To calculate the precision and recall values, each detected bounding box must first be classified as:

- **True Positive (TP).** A correct detection matching a ground-truth object;
- **False Positive (FP).** An incorrect detection in an empty area or a misplaced detection of an existing object;
- **False Negative (FN).** An undetected ground-truth object.

Given a dataset of G ground truth and a model that outputs a total of N detections, we define as S the number of correct predictions ($S \leq G$). Precision and recall can be computed as follows:

$$Precision = \frac{TP}{TP + FP} = \frac{S}{N} \quad (13)$$

$$Recall = \frac{TP}{TP + FN} = \frac{S}{G} \quad (14)$$

Matching ground truth and prediction boxes can be complex, as there could be multiple predictions with an IoU over the threshold for a single ground truth box or a single prediction can overlap multiple ground truth boxes. For the purpose of this study, we have adopted the same methodology used in the COCO API, which was developed to evaluate detections on the MS-COCO dataset (Lin et al., 2014). Essentially, it employs a greedy algorithm that examines the detection boxes individually. For each detection box, the algorithm finds the unmatched ground-truth box with the highest Intersection over Union (IoU) score. If the IoU exceeds a threshold, a match is established,

and the corresponding ground-truth box is removed from the pool of unmatched ones. If the IoU falls below the threshold, it is considered a false positive. Any remaining unmatched ground-truth boxes at the end are deemed false negatives.

6.1.3. F_1 score

The F_1 score is the harmonic mean of precision and recall, and synthesize the prediction performance in a single scalar value:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (15)$$

6.1.4. Curves

As of now, we have considered a fixed threshold for the IoU (T_{IoU}). This approach results in a static level of false positives and negatives, limiting our understanding of the model's performance across different levels of strictness. In particular, precision and recall are monotonically decreasing functions of the IoU threshold T_{IoU} . To gain a deeper insight into the model's behavior, visualizing the relationship between precision, recall, or F_1 -score and the varying IoU threshold can be highly beneficial.

6.2. Metrics that require confidence

6.2.1. Average precision

By setting a confidence threshold τ , detections with confidence greater than τ are considered positive, and the rest are negatives. This allows us to express precision, recall, and F_1 -score as functions of τ :

$$Precision(\tau) = \frac{TP(\tau)}{TP(\tau) + FP(\tau)} \quad (16)$$

$$Recall(\tau) = \frac{TP(\tau)}{TP(\tau) + FN(\tau)} \quad (17)$$

$$F_1(\tau) = 2 \times \frac{Precision(\tau) \times Recall(\tau)}{Precision(\tau) + Recall(\tau)} \quad (18)$$

Both $TP(\tau)$ and $FP(\tau)$ decrease with τ , while $FN(\tau)$ increases. For this reason, the recall is a decreasing function of τ , while nothing can be said a priori about precision. Indeed, the graph of $Precision(\tau)$ versus $Recall(\tau)$ often exhibits a zig-zag pattern in real-world scenarios (Padilla et al., 2021). The Average Precision (AP) is defined as the Area Under the Curve (AUC) of the Precision-Recall curve. To handle the curve's zig-zag pattern, we used the COCO API's N-point interpolation method with $N=101$ to compute AP. In the N-point interpolation, the first step is to take N points equally spaced in the interval $[0, 1]$, that is:

$$Re(n) = \frac{N - n}{N - 1}, n = 1, 2, \dots, N \quad (19)$$

where $Re(n)$ is the n th recall value. Now, we would like to have a value of precision Pr for every value of recall, to compute the Riemann Integral of the function $Pr(Re(\tau))$. The problem is that we do not always have a single value of recall given a value of precision. To solve the issue, we define the function $Pr_{interp}(R)$, a continuous function of Re , as follows:

$$Pr_{interp}(R) = \max_{k | Re(\tau(k)) \geq R} \{Pr(\tau(k))\} \quad (20)$$

Finally, we can compute the AP with the following equation:

$$AP = \frac{1}{N} \sum_{n=1}^N Pr_{interp}(R_r(n)) \quad (21)$$

6.2.2. Mean average precision

For datasets with many classes, the mean Average Precision (mAP) is defined as the average AP over all classes. It is used to have a single AP score for all classes:

$$mAP = \frac{1}{C} \sum_{i=1}^C AP_i \quad (22)$$

where AP_i is the AP value for the i th class and C is the total number of classes being evaluated.

Table 4
Number of 1D barcodes with module size within a specified range (in pixels).

PPE Range	0-0.5	0.5-1.0	1.0-1.5	1.5-2.0	2.0-2.5	2.5-3.0	3.0-3.5	3.5-4.0	> 4.0	Undefined
Num. Examples	14	1035	1340	1405	1176	545	163	61	28	1044

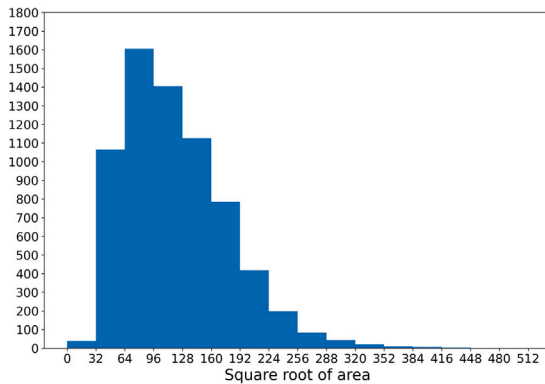


Fig. 3. This histogram depicts the distribution of object sizes (calculated as the square root of their area in pixels) within the dataset after images were resized to a 640-pixel longest side. The x -axis displays the square root of the area, the y -axis indicates the object count. Each bin has a range of 32 pixels.

6.2.3. $AP@T_{IoU}$

We indicate with $AP@T_{IoU}$ the average precision given an IoU threshold T_{IoU} . Usually, the threshold T is 0.5.

6.2.4. $AP@[0.5:0.05:0.95]$

The $AP@[0.5:0.05:0.95]$ is the mean of all the values $AP@T_{IoU}$ with T_{IoU} in the range 0.5 to 0.95 with a step size of 0.05. Can also be shortened to $AP@[0.5:0.95]$ or $AP@[.5:.95]$.

6.2.5. AP across scales

The AP across scales is a set of three metrics, indicated with AP_S , AP_M and AP_L . These metrics are equal to the $AP@[.5:.95]$, but taking into consideration the area of the ground-truth object:

- AP_S only evaluates small ground-truth objects (area < 32² pixels);
- AP_M only evaluates medium ground-truth objects (32² < area < 96² pixels);
- AP_L only evaluates large ground-truth objects (area > 96² pixels).

When evaluating objects of a given size, objects of other sizes (both ground-truth and predicted) are not considered in the evaluation.

7. Benchmark description

7.1. Repository content

As part of this project, we have developed BarBeR, a benchmark for barcode localization algorithms. Its code is publicly accessible and it can be downloaded from GitHub.⁵ Our dataset, used for running our tests, can be downloaded from the same GitHub repository. The project contains the files necessary to build the following publicly available detection methods: Gallo and Manduchi (2010), Zamberletti et al. (2013), Sörös and Flörkemeier (2013), Tekin and Coughlan (2012) and Yun and Kim (2017). Additionally, the project includes scripts for training (Zharkov and Zagaynov, 2019)'s neural network and other deep-learning models with Ultralytics or Detectron2 (Wu et al., 2019) and supports early stopping. The repository contains multiple test

Table 5

Precision, Recall and F1-score with an IoU threshold of 0.5. All images contain a single 1D barcode and were resized to have their longest side of 640 pixels. Upward arrows mean that higher values of the metric indicate better performance.

Detection Method	Precision ↑	Recall ↑	F1-score ↑
Gallo et al.	0.533	0.533	0.533
Soros et al.	0.658	0.658	0.658
Zamberletti et al.	0.234	0.340	0.278
Yun et al.	0.806	0.714	0.757
Zharkov et al.	0.725	0.952	0.823
Faster R-CNN	0.981	0.996	0.989
RetinaNet	0.988	0.991	0.990
YOLO Nano	0.978	0.997	0.987
YOLO Medium	0.984	0.998	0.991
RT-DETR Large	0.987	0.999	0.993

scripts and each supports multiple 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. The script can be configured to only allow linear barcodes or two-dimensional barcodes. It is also possible to include only images with a single Region Of Interest (ROI) or allow multiple ROIs per image. In addition, it is possible to set 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 algorithms 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.

Test scripts are in Python format. The repository also contains bash scripts used to run a pipeline of tests. This is useful, for example, for k-fold cross-validation.

7.2. Methodology

This article will present the results from detection accuracy tests in both single and multi-class modes. We used k-fold cross-validation (k=5) for a comprehensive accuracy assessment, dividing the dataset into five equal sections. Each section was used as a test set, with the remainder for training. Deep-learning models were trained using 75% of the training set, with the rest as a validation set for early stopping (patience 10 epochs). The networks were trained using a batch size of 16 and the Adam optimizer (Kingma, 2014), configured with a learning rate of 0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. To augment training data we applied horizontal and vertical flips, as well as random adjustments to brightness, contrast, and saturation. The methods of Gallo, Soros, and Yun were tuned using the entire training set, selecting the optimal size for the box filter's window (15 pixels for Soros and Gallo, 30 pixels for Yun). Zamberletti's method employs a pre-trained MLP model that works on the Hough Transform. The original MLP network was trained on the ArteLab Rotated dataset, which is not part of our dataset, thus preventing any information leakage. Finally, we will measure the timing for each detection method. For each image, the detection is run three times and the lowest timing value is taken. This is done to remove the effect of external factors such as background processes, CPU loads, one-time initialization overheads, caching, and garbage collection cycles. All tests were conducted on a PC with an AMD Ryzen

⁵ <https://github.com/Henzezz95/BarBeR>

Threadripper Pro 5965WX CPU (24 cores), 128 GB DDR4 RAM, and an RTX 4090 GPU. To provide an example of performance on embedded systems, we also ran the time performance test on a Raspberry Pi 3B+.

8. Single-class benchmarks

First, the available detection algorithms are tested by considering just images of a single class, linear barcodes, or 2D barcodes.

8.1. Single 1D barcode

This evaluation focuses on images with a single linear barcode, allowing us to test all available algorithms, including those that output a single Region of Interest (ROI). The total number of images included in this test was 6811. All images are resized using the method “resize by longest side”. For this test, we decided to set the value of the longest side to 640 pixels. This is the same size used to test the methods of Gallo et al. and Zamberletti et al. in their original paper. Soros and Yun’s methods used instead a higher resolution, 960×723 and 1024×768 respectively. This is also the default resolution for YOLO-v8 (Jocher et al., 2023) and other object detection networks. At this resolution, our dataset comprises 42 small objects (area $< 32^2$), 2665 medium objects ($32^2 < \text{area} < 96^2$), and 4104 large objects (area $> 96^2$). Fig. 3 provides additional information about object size distribution within the dataset.

Most objects fall into the medium and large categories at the selected resolution, suggesting that they are likely easily detectable by a neural network. However, all traditional methods we have found in public repositories implement some form of texture detection for localization. The texture of a barcode, whether linear or bidimensional, primarily depends on the pixels per element. Barcodes with large modules have recognizable edges and corners, but if the modules are small, the texture becomes much more uniform. Ideally, a barcode detector should always find a barcode that could be decoded. However, defining the minimum number of pixels per module for a reliable decode is challenging, as many other factors, such as contrast and blur, are involved. We can see from the annotations of the dataset that the automatic labeler was able to decode linear barcodes in the range of 0.88 to 24.33 pixels per element and 2D barcodes in the range of 1.21 to 71.1 pixels per element (PPE). However, decoding at less than 1 PPE for linear barcodes and 2 PPE for 2D barcodes is usually not guaranteed. In cases of blur or low contrast, the number of required pixels per element increases significantly. When resizing, the pixel density scales proportionally to the applied scaling factor. A more detailed breakdown of the various PPE values at the selected resize resolution can be seen in Table 4. As can be seen, most of the barcodes have a pixel density between 1 and 3 pixels per module, which is typically sufficient for a linear barcode decoder. There are also a few barcodes with more than 3 pixels per module, peaking at 5.13 pixels per module. However, over 1000 barcodes have a PPE < 1 , indicating that decoding them at this resolution would be very challenging or even impossible. Additionally, there are 1044 barcodes without PPE information, suggesting that the automatic labeler was unable to decode them, and this would likely remain true even after resizing the image. We can conclude that this dataset would pose a significant challenge for a barcode reader. However, having a few barcodes with a resolution lower than required is useful for understanding the limits of a detector and determining if it could be applied to a downsampled image, for real-time purposes. Not all the methods tested generate a confidence score, so, for a fair comparison, we decided to use precision, recall, and F1-score as metrics. In Table 5 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 can be displayed with the precision, recall, and F1-score curves at different values of T_{IoU} . Fig. 4

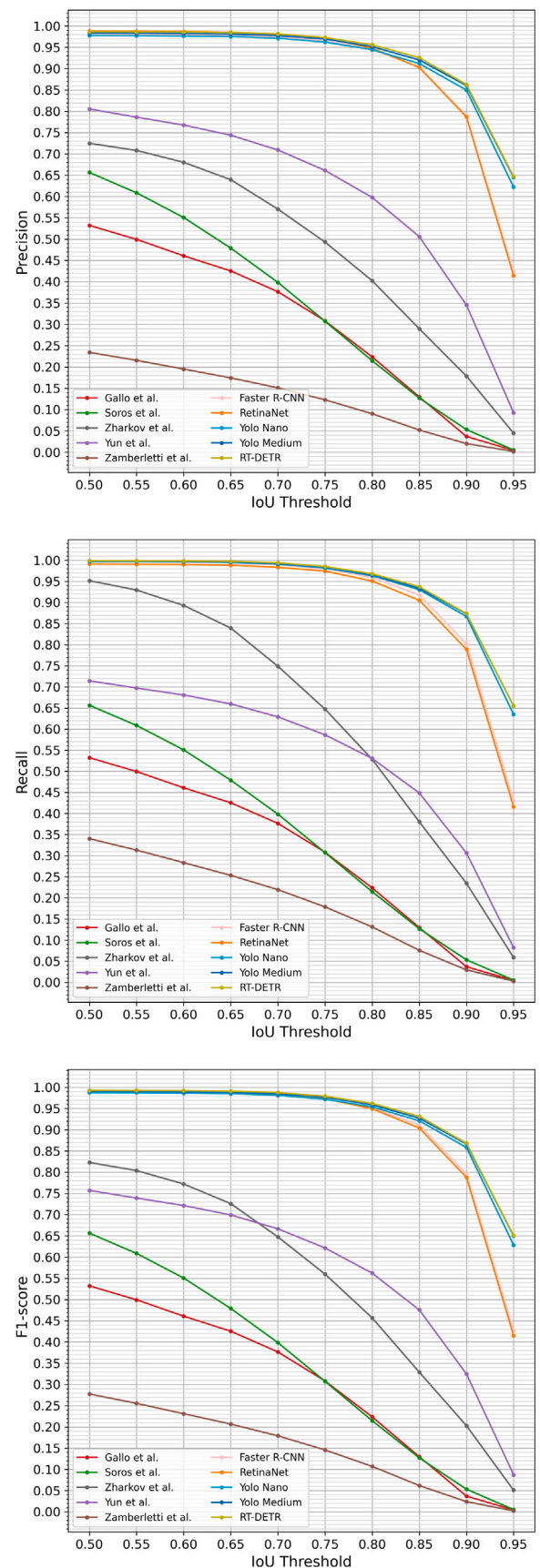


Fig. 4. Precision, recall and F1-score curves of detection algorithms at different thresholds. Images contain one 1D barcode and were resized to have the longest side equal to 640 pixels.

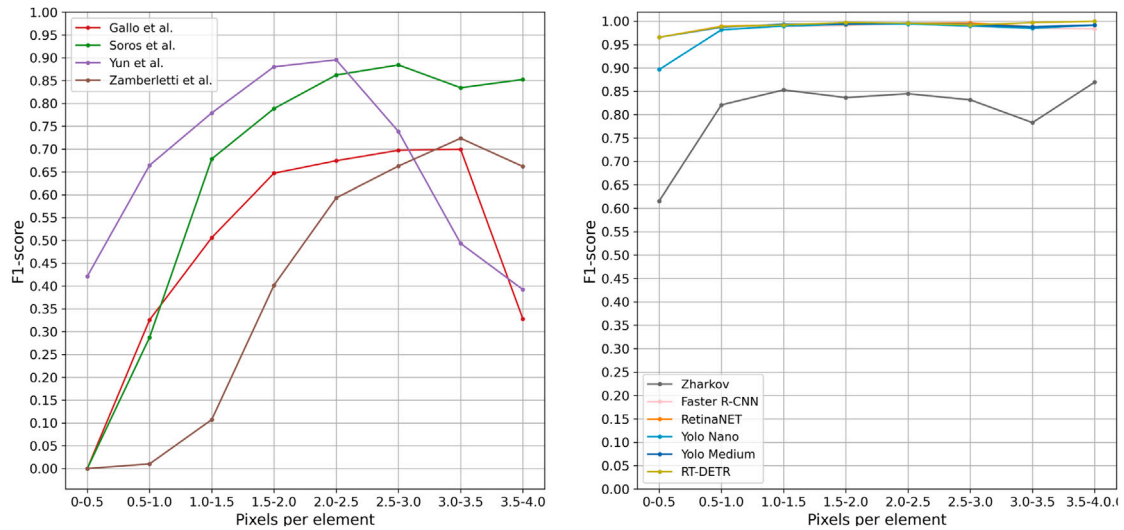


Fig. 5. F1-score of the tested detection methods on linear barcodes at different ranges of pixels per element. The picture on the left shows all non-deep-learning-based methods while the one on the right shows only the deep-learning-based methods.

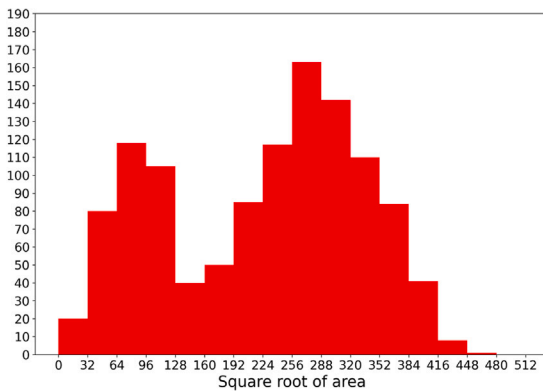


Fig. 6. This histogram depicts the distribution of object sizes (calculated as the square root of their area in pixels) within the dataset after images were resized to a 640-pixel longest side. The x -axis displays the square root of the area, the y -axis indicates the object count. Each bin has a range of 32 pixels.

represents the precision, recall and F1-score curves of the different methods. Apart from Zharkov's architecture, all the other end-to-end neural networks always outperform the other methods in all three graphs. This was expected since these methods are much more computationally intensive and well-versed for complex detection problems. Among the tested classic algorithms, Yun and Kim (2017) is by far the one that performs better at every IoU threshold, meaning this is probably the suggested method every time a neural network would be too cumbersome for the task. 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 overall the weakest performer. Zharkov's architecture 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 a bit less precise. Overall, RT-DETR is at the top of the leaderboard, but by an extremely small margin. Interestingly, YOLO Nano has a very similar performance to YOLO Medium and RT-DETR, despite having nearly 10 times fewer parameters. This indicates that the simplicity of

Table 6

Number of 2D barcodes with module sizes within a specified range (in pixels).

PPE Range	0-1	1-2	2-3	3-4	4-5	5-6	6-7	> 7	Undefined
Num. Examples	6	138	133	102	218	225	155	97	90

Table 7

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. Upward arrows mean that higher values of the metric indicate better performance.

Detection Method	Precision \uparrow	Recall \uparrow	F1-score \uparrow
Soros et al.	0.140	0.140	0.140
Zharkov et al.	0.727	0.900	0.804
Faster R-CNN	0.981	0.992	0.987
RetinaNet	0.981	0.995	0.988
YOLO Nano	0.962	0.989	0.975
YOLO Medium	0.980	0.990	0.985
RT-DETR Large	0.972	0.997	0.984

this detection task allows small networks to perform very well, without compromising on accuracy.

Finally, we can study the performance of different algorithms depending on the pixel density of the barcodes. We consider the range from 0 to 4 pixels per module and divide it into bins of 0.5 pixels per module. As a single performance metric, we consider the F1-score at $T_{IoU} = 0.5$. For every bin, we have a different number of examples, as we can see from Table 4. Fig. 5 shows that methods not based on deep learning have an optimal pixel-per-module range for accurate detection. This is because they rely on edge features for localization. We can see that the range of 1.5 to 3.5 pixels per module is the optimal range for both Gallo's and Soros' detection algorithms. Zamberletti's method requires a bit more pixels per element, while the one proposed by Yun performs better at 1.5 to 2.5 pixels per module. This method is also the only one that is able to detect some barcodes with < 0.5 PPE and is moderately reliable at $0.5 < ppe < 1.0$ without the use of deep learning. On the other hand, we can see that the deep-learning-based methods have near-constant performance in the pixels-per-element range that goes from 0.5 to 4.0. The performance seems to drop a bit under 0.5 pixels per module, but it is hard to draw definitive conclusions with only 14 samples.

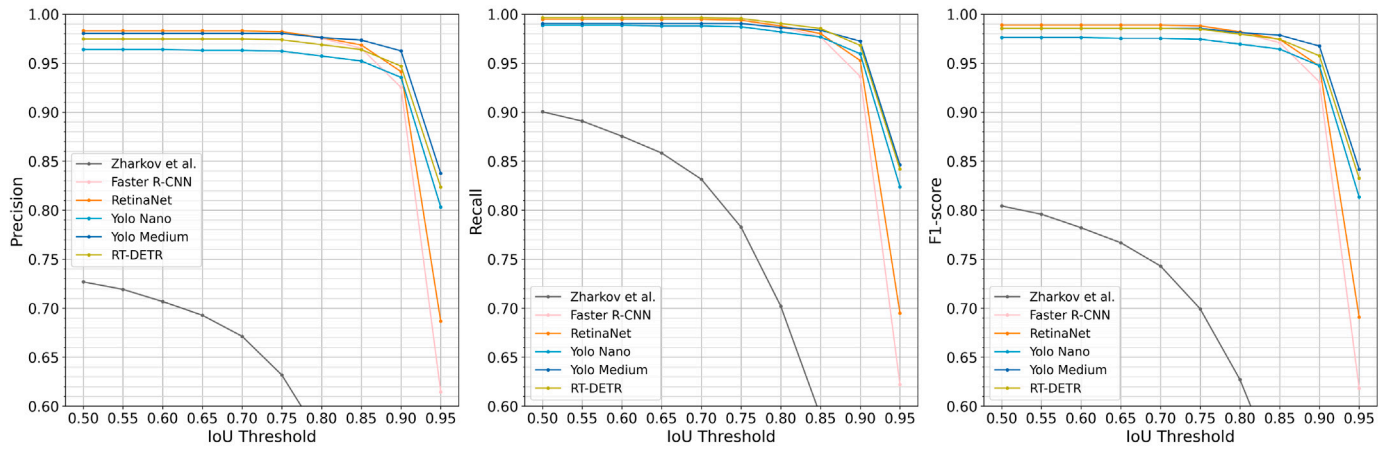


Fig. 7. Precision, recall, and F1-score curves of 2D barcode detection algorithms at different values of IoU threshold. All images contain one 2D barcode and were resized to have their longest side of 640 pixels.

8.2. Single 2D barcodes

In this test, we only include examples with a single two-dimensional barcode. Sörös and Flörkemeier (2013)'s method is the only non-deep-learning-based method available that also detects 2D barcodes and has been tested alongside the six neural networks presented so far. The dataset contained 1164 images, resized to a maximum edge length of 640 pixels. At this resolution, our dataset included 19 small objects ($\text{area} < 32^2$), 202 medium objects ($32^2 < \text{area} < 96^2$), and 943 large objects ($\text{area} > 96^2$).

A more detailed breakdown of the area distribution is presented in Fig. 6. The dataset exhibits a bimodal area distribution, with clusters of larger barcodes, with one group of larger barcodes with an area between 240^2px^2 and 380^2px^2 and the other group of tinier barcodes with an area between 20^2px^2 and 140^2px^2 . The largest barcodes come mostly from the Dubska (Dubská et al., 2016) dataset, while the smaller codes are contained mostly in the ZVZ-Real dataset (Zharkov and Zagaynov, 2019). Most codes are classified as “large”. Alongside the object’s area, module density remains crucial for determining the dataset’s difficulty. High pixel density in 2D barcodes aids in detecting corners and edges, even with some image blur. Conversely, low pixel-per-element (PPE) values obscure these features, making detection harder. As said before, a density of at least 2 pixels per module is needed for reliable decoding, but in case of blur and noise, a higher pixel density is required. However, decodings between 1 and 2 pixels per module are sometimes possible. Table 6 details the dataset’s PPE distribution, which ranges from 0.68 to 9.58. Most codes have a density $> 2.0 \text{px/el}$, suggesting probable readability, but there are still 144 codes with less than 2 PPE that would present a decoding challenge. Additionally, 90 barcodes lack PPE information, implying that even with resizing, reliable decoding is unlikely. We can conclude that a good portion of the dataset could be decoded by a two-dimensional barcode reader, but there are still a good amount of hard cases to make the test more challenging. Not all the methods tested generate a confidence score, so, for a fair comparison, we decided to use precision, recall and F1-score as metrics. In Table 7 we can see the results of the different methods considering an IoU threshold of 0.5. It is clear that the (Sörös and Flörkemeier, 2013) 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 Precision, Recall, and F1 curves in Fig. 7. Zharkov et al. achieves good results, especially in recall, but falls short of the other deep learning architectures. At a lower T_{IoU} threshold RetinaNet is the best method, in terms of F1-score. On the other hand, for $T_{IoU} > 0.75$ YOLO Medium and RT-DETR are the best performers, meaning that they generate a more precise bounding box. YOLO Nano has a similar performance to YOLO Medium, but now the gap is a

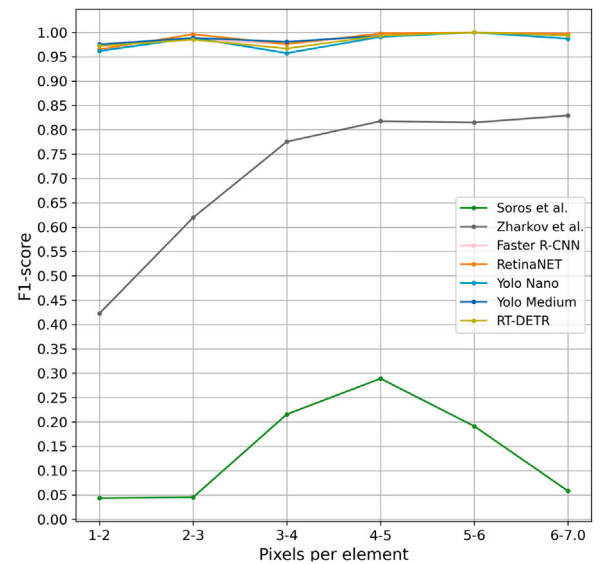


Fig. 8. F1-score of the tested detection methods on 2D barcodes at different ranges of pixels per element.

bit larger with respect to the 1D case. Finally, we can analyze the performance of different algorithms depending on the pixel density of the 2D barcodes. We consider the range from 1 to 7 pixels per module and divide it into bins of 1 pixel per module. As a single performance metric, we consider the F1-score at $T_{IoU} = 0.5$. For every bin, we have a different number of examples, as we can see from Table 6. We can see from Fig. 8 that Soros’s method reaches its peak performance in the range of 3–5 pixels per module. While the performance degrades fast outside this range, especially for lower pixel densities. Zharkov et al. performs better if the PPE is higher than 3, while the other deep-learning architectures seem to not be affected much by pixel density.

9. Multi-class benchmarks

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

Table 8

Average precision scores for the tested models across all images of the dataset, resized at different scales. Upward arrows mean that higher values of the metric indicate better performance.

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 Large	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 Large	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 Large	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	0.890	0.793	0.888	0.766

Table 9

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

applications could be challenging due to the high computational costs. As we will better investigate in Section 10, running these models on embedded devices requires a lot of time. 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 images will be resized to have their longest side equal to 640 pixels, the same scaling policy adopted for the previous tests. Then, we will resize the longest side of the images to 480 pixels and 320 pixels, to measure their performance on downscaled images. For each scale, we re-trained the models using a training set with the same scale. In Table 9 we see the number of instances divided by class and size. As expected, at lower resolution there will be more small objects and less large objects. 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 will calculate 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 will consider the corresponding mean Average Precision values (mAP@0.5 and mAP@[.5:.95]) for each model.

The results in terms of AP@0.5 and AP@[.5:.95] are presented in Table 8. An overall comparison of the results obtained is depicted in the bar chart in Fig. 9. Zharkov's method remains the weaker model among the tested ones but still achieves a respectable mAP@0.5 score of 0.823 at the 640 pixels scale. However, we can see quite a huge drop in performance at the other two scales. The other models perform well at all tested resolutions. The drop in performance from 640 pixels to 480 pixels is small for most models while downscaling to 320 pixels has a more noticeable impact on performance. At the 640 pixels scale, Faster R-CNN and RetinaNet achieve lower scores than other models

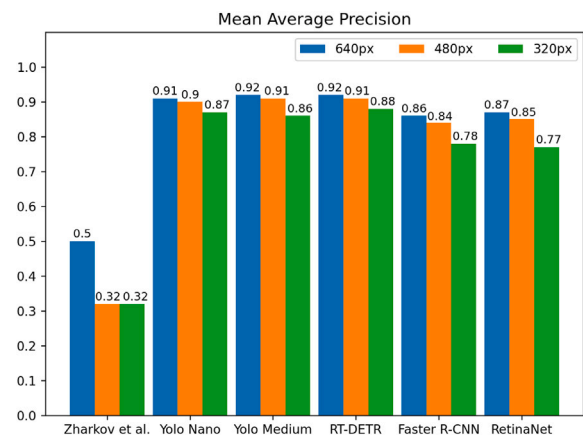


Fig. 9. Values of mAP@[.5:.95] of different models at three different scales: longest side resized to 640 pixels, longest side resized to 480 pixels, and longest side resized to 320 pixels. The tests were conducted considering all images in the dataset.

while YOLO Medium and RT-DETR deliver the highest mAP@0.5 and mAP@[.5:.95] respectively. At the two other scales, Faster R-CNN and RetinaNet remain at the bottom of the leaderboard, but Faster R-CNN seems to perform a bit better. 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 concerning YOLO Medium at 320 pixels resize, while this is not true at 480 pixels resize. Finally, we measure the Average Precision across scales for the lowest resolution (longest side resized to 320 pixels). Results are shown in Table 10. As expected, the small object category is the one with the lowest scores overall. Large 2D codes seem to be quite easier to detect than large 1D barcodes. The ranking amongst different models remains the same across scales. The only exception is that RetinaNet struggles more than Faster R-CNN at small scales, but has better scores for the medium and large categories.

10. Time measurements

In this section, we present inference times for the barcode detection algorithms under evaluation. This analysis is crucial for barcode detection applications, many of which operate on embedded devices with limited computational resources. To assess performance across diverse use cases, we conducted benchmarks on two contrasting hardware

Table 10

Average precision across scales. AP_S , AP_M , AP_L are the AP[.5:.95] for objects with small ($< 32^2$), medium ($> 32^2 \wedge < 96^2$) and large ($> 96^2$) ground truth areas respectively. Test images have been resized to have a maximum side length of 320 pixels. Upward arrows mean that higher values of the metric indicate better performance.

Model	1D barcodes			2D barcodes			Average		
	$AP_S \uparrow$	$AP_M \uparrow$	$AP_L \uparrow$	$AP_S \uparrow$	$AP_M \uparrow$	$AP_L \uparrow$	$mAP_S \uparrow$	$mAP_M \uparrow$	$mAP_L \uparrow$
Zharkov et al.	0.050	0.360	0.439	0.005	0.139	0.678	0.028	0.25	0.558
YOLO Nano	0.641	0.886	0.922	0.613	0.845	0.966	0.627	0.866	0.944
YOLO Medium	0.633	0.878	0.900	0.633	0.841	0.947	0.633	0.859	0.923
RT-DETR Large	0.681	0.897	0.929	0.669	0.878	0.975	0.675	0.887	0.952
Faster R-CNN	0.464	0.810	0.841	0.526	0.748	0.853	0.495	0.779	0.847
RetinaNet	0.362	0.813	0.838	0.311	0.799	0.941	0.337	0.806	0.890

Table 11

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. Downward arrows mean that lower values of the metric indicate better performance.

Detection method	Times on PC (ms)			Times on Raspberry PI (ms)		
	Single-thread CPU \downarrow	Multi-thread CPU \downarrow	GPU \downarrow	Single-thread CPU \downarrow	Multi-thread CPU \downarrow	GPU \downarrow
Gallo et al.	1.632	–	–	53.45	–	–
Soros et al.	11.25	–	–	397.5	–	–
Zamberletti et al.	48.20	–	–	1 360	–	–
Yun et al.	7.598	–	–	146.3	–	–
Zharkov et al.	25.85	5.974	1.447	2 120	1 949	–
YOLO Nano	64.99	17.40	18.66	3 034	1 803	–
YOLO Medium	478.9	51.36	23.91	20 083	15 813	–
RT-DETR	985.4	141.0	37.55	39 882	33 224	–
Faster R-CNN	1 271	237.9	30.27	∞	∞	–
RetinaNet	1 124	105.2	36.00	∞	∞	–

setups: a deep-learning optimized PC (AMD Ryzen Threadripper Pro 5965WX with 24 cores, 128 GB DDR4 RAM, RTX 4090 GPU) and a Raspberry Pi 3B+ (1.2 GHz quad-core ARMv8 CPU, 1 GB DDR2 RAM). The algorithms we tested, implemented in C++, were not specifically optimized for multi-threading. However, they do leverage certain OpenCV functions capable of running on multiple threads. To provide a clear understanding of their performance, we ran these methods on a single CPU thread. This approach ensures that the timings are not skewed by the limited parallelization of only a few sections of the code. For a balanced comparison, we also recorded the inference times of deep-learning methods running on a single CPU thread. In addition, we report, for informational purposes, the times of deep-learning methods when running on GPU or on CPU with multi-threading enabled. Finally, all C++ implementations were compiled with -O3 optimization, which includes auto-vectorization, to ensure maximum performance. For this benchmark, we run all the detection methods on all the images of the dataset. Every detection is repeated three times and the lowest timing is considered. This is done to minimize the influence of background processes on the measurement. The final time is the average of the times recorded for every image. Since the images have different aspect ratios we will also report the mean resolution in megapixels of the images used after scaling.

10.1. Time on PC

These are the reported times when running on a PC with an AMD Ryzen Threadripper Pro 5965WX CPU and an RTX 4090 GPU. All the tests were conducted after scaling the images following the rule that the longest side must be 640 pixels. In total, we have 8 748 images, with a mean resolution after resizing of 0.284 Megapixels, equivalent to the resolution of an image of 640×444 pixels. The inference is always conducted on a single image at a time. Table 11 reports the times required to run the detection methods on a single thread on the CPU. For the deep-learning methods, we also report the multi-threaded performance on 24-cores and the GPU. We will first focus on single-threaded performance on the CPU since this is the only way to confront all the methods. As expected, there is a huge difference between the methods involved, with the fastest method being 780 times faster than the slowest one. Gallo et al. is by far the fastest one and

can run more than 600 times a second on a single thread. This was expected since this is the oldest method and its main focus was to run on limited hardware. This incredible speed is possible because the method is not rotation invariant. Yun et al. is the second fastest method (7.598 ms), despite having a better detection accuracy than Soros et al. and Zamberletti et al. Soros et al. is a bit slower with a performance of 11.25 ms. Zamberletti's method is quite slower at 48.20 ms. Since it uses MLP in the Hough Transform space, its performance is in between classic CV methods and deep-learning methods. 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 models with a mean time of 64.99 ms. YOLO Medium is more than 7x slower than the Nano version 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 are even slower with a time of 1 124 ms and 1 271 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 method is still Zharkov et al. at 1.447 ms while the slowest one is RetinaNet at 36.00 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 for barcode detection where using a high-end GPU makes economical and logistical sense.

Previously, we have seen that deep-learning-based detectors work quite well even at lower resolution. For this reason, we also recorded the single-thread performance when resizing the longest side to 480 pixels and 320 pixels. All these tests have been conducted using a single thread on the CPU and are shown in Table 12.

We can see that the times more or less scale linearly with the number of pixels of the images. Indeed, the resolution at 480 pixels is around 1.8 less, and at 320 pixels is 4 times less than at 640 pixels. On the lowest resolution, we could easily run a small network like Zharkov et al. or YOLO Nano in real-time, while the other deep-learning-based models are still too slow.

10.2. Time on embedded device

As we have seen, a PC CPU is usually more than enough to run detection algorithms for barcodes in real-time. However, many barcode reading applications rely on an embedded CPU. An example is retail barcode readers, which should be small enough to be handheld devices. Another example is part identification marking in industry, where every component is marked with a barcode and multiple readers are used along the pipeline. This method is used in industries where precision and safety are paramount, such as aerospace, automotive, medical devices, and electronics. Part identification is used to cross-reference part specifications, ensuring that the correct components are used in assembly. This helps avoid mistakes, reduces the risk of faulty products, and minimizes costly recalls. The use of embedded devices instead of PCs to handle the processing ensures a reduction in costs and space requirements. In addition, offloading computation to external machines would increase latency. To measure the performance on embedded devices we run our benchmark on a Raspberry PI 3B+ system that uses a quad-core ARM Cortex A53 CPU at 1.2 GHz and 1 GB of 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. We conducted single-core CPU tests for all detection algorithms on the selected images, resizing them to a maximum edge length of 640 pixels. Deep-learning methods have also been tested using all the 4-cores of the CPU. Results are presented in Table 11. Compared to the PC results, execution times on the Raspberry PI increased by 40-50x. Insufficient RAM (1 GB) prevented Faster R-CNN and RetinaNet from running. Consequently, 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), and then we have Yun's (146.3 ms) and Soros' (397.5 ms) algorithms followed by the one proposed by Zamberletti (1360 ms). All the deep-learning methods are slower than that, with multi-second time requirements. Zharkov et al. is still the fastest network at 2120 ms, followed by YOLO Nano at 3034 ms. YOLO Medium and RT-DETR are incredibly slow with processing times of 20083 ms and 39882 ms respectively. Multi-core execution yielded a modest speed-up of roughly 1.5x, 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 12. 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 1050 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 are not currently optimized for multi-core processing. However, this could be readily achieved with libraries such as OpenMP (OpenMP Architecture Review Board, 2008). Furthermore, C++ code can be made much faster by employing SIMD intrinsics while, software toolkits like OpenVINO (OpenVINO, 2024) and TFLite (Abadi et al., 2015) can speed up the execution of deep-learning models, particularly on embedded CPUs. Lastly, techniques such as 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.

11. Concluding remarks and future research directions

In this paper we presented a comprehensive review of the field of barcode localization and released a public benchmark for barcode localization, addressing existing challenges in reproducibility and dataset

standardization within the field. Our core contributions are summarized as follows.

Dataset consolidation and standardization. We have collected a dataset of 8748 images from public sources and supplied it with standardized annotations. We decided to make the dataset public so that it could be used for future contributions to this field.

Reproducible benchmarking. BarBeR, our publicly accessible benchmark, features a suite of algorithms from the literature (thoroughly described in Section 4), scripts for deep learning model training, and diverse performance metrics (presented in Section 6). 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, verifying its reliability and usability. In particular, we can draw some interesting conclusions from the tests we carried out so far. 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. On the other hand, our findings suggest that small neural networks, such as YOLO Nano, perform nearly as well as much bigger architectures like RT-DETR and RetinaNet. Lastly, among the publicly available methods tested, Yun et al. proposal offers the optimal blend of accuracy and speed, surpassing Soros' and Zamberletti's methods in both metrics. The fastest method, instead, was the one described by Gallo et al., showing that decent accuracy could be achieved even on very constrained devices. As stated in the introduction, barcode decoding technology plays a vital role in industries such as logistics, supply chain management, retail, and robotics, with a market value in the billions. This paper aims to provide an academic perspective on the field, focusing on barcode localization — the crucial first step in any decoding process. By advancing open-source research in this area, we hope to foster growth in industries leveraging barcode technology and support academic endeavors that rely on it. For the future, we envision a set of possible improvements to our benchmark that could be used to further push the field of barcode reading.

Instance segmentation. The current software could easily be extended to support image segmentation benchmarking. The dataset metadata already defines ROIs with polygons.

Barcode decoding. Assessing the decoding capabilities of integrated localization-decoding systems offers a broader evaluation for practical use cases. Indeed, the final metric for a barcode reading system is the number of decoded barcodes and the time required to achieve that result.

Image enhancement. Studying the impact of image enhancement techniques could potentially improve barcode reading success rates under challenging conditions. An image processing step could be added to the pipeline before or after the localization phase.

Video support. In many applications, barcode reader inputs are sequences of images rather than single frames. Adding a video dataset to the benchmark could open new evaluation possibilities, enabling researchers to explore the trade-offs between speed and accuracy. For example, faster and less precise decoders might process more frames, while slower but more accurate algorithms might rely on fewer. Benchmarking could reveal which approach is more effective in different scenarios. Additionally, leveraging temporal information across frames could improve the accuracy or efficiency of localization algorithms, offering new directions for video-based barcode detection research.

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 this project.

Table 12

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. Downward arrows mean that lower values of the metric indicate better performance.

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.	1.632	0.919	0.409	53.45	32.04	14.31
Soros et al.	11.25	6.260	2.782	397.5	205.5	92.02
Zamberletti et al.	48.20	29.66	17.42	1360	1357	855.7
Yun et al.	7.598	4.498	2.171	146.3	103.8	52.80
Zharkov et al.	25.85	14.56	6.725	2120	882.5	340.9
YOLO Nano	64.99	40.20	20.82	3034	2108	1050
YOLO Medium	478.9	284.6	135.2	20083	12091	5570
RT-DETR	985.4	604.0	329.2	39882	25371	13427
Faster R-CNN	1271	892.3	599.1	∞	∞	∞
RetinaNet	1124	665.0	319.1	∞	∞	∞

CRedit authorship contribution statement

Enrico Vezzali: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Federico Bolelli:** Writing – review & editing, Validation, Supervision, Investigation, Conceptualization. **Stefano Santi:** Writing – review & editing, Supervision, Data curation. **Costantino Grana:** Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This project has received funding from the University of Modena and Reggio Emilia and Fondazione di Modena, through the FAR 2024 and FARD-2024 funds (Fondo di Ateneo per la Ricerca).

Data availability

Data are available online at ditto.ing.unimore.it/barber.

References

- Abadi, M., et al., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL <https://www.tensorflow.org/>, Software available from [tensorflow.org](https://www.tensorflow.org/).
- Ando, S., 2000. Image field categorization and edge/corner detection from gradient covariance. *IEEE Trans. Pattern Anal. Mach. Intell.* 22 (2), 179–190.
- Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S., Modi, K., Ghayvat, H., 2021. CNN variants for computer vision: History, architecture, application, challenges and future scope. *Electronics* 10 (20), 2470.
- Bishop, C., 1995. *Neural networks for pattern recognition*. Clarendon Press 2, 223–228.
- Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y.M., 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.
- Bodnár, P., Grósz, T., Tóth, L., Nyúl, L.G., 2018. Efficient visual code localization with neural networks. *Pattern Anal. Appl.* 21, 249–260.
- Canny, J., 1986. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* (6), 679–698.
- Chai, D., Hock, F., 2005. Locating and decoding EAN-13 barcodes from images captured by digital cameras. In: 2005 5th International Conference on Information Communications & Signal Processing. *IEEE*, pp. 1595–1599.
- Chang, S.-K., Yang, C.-C., 1983. Picture information measures for similarity retrieval. *Comput. Vis. Graph. Image Process.* 23 (3), 366–375.
- Chou, T.-H., Ho, C.-S., Kuo, Y.-F., 2015. QR code detection using convolutional neural networks. In: *International Conference on Advanced Robotics and Intelligent Systems*. ARIS, *IEEE*, pp. 1–5.

- Dice, L.R., 1945. Measures of the amount of ecologic association between species. *Ecology* 26 (3), 297–302.
- Do, T., Kim, D., 2021. Quick browser: A unified model to detect and read simple object in real-time. In: 2021 International Joint Conference on Neural Networks. *IJCNN, IEEE*, pp. 1–8.
- Do, H.-T., Pham, V.-C., 2021. Deep learning based goods management in supermarkets. *J. Adv. Inf. Technol.* 12 (2).
- Dubská, M., Herout, A., Havel, J., 2016. Real-time precise detection of regular grids and matrix codes. *J. Real-Time Image Process.* 11, 193–200.
- Duda, R.O., Hart, P.E., 1972. Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM* 15 (1), 11–15.
- Dutta, A., Zisserman, A., 2019. The VIA annotation software for images, audio and video. In: *Proceedings of the 27th ACM International Conference on Multimedia*. pp. 2276–2279.
- Etude, P.J., 1901. Comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull. Soc. Vaud. Sci. Nat* 37, 547.
- Galamhos, C., Matas, J., Kittler, J., 1999. Progressive probabilistic Hough transform for line detection. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 1, *IEEE*, pp. 554–560.
- Gallo, O., Manduchi, R., 2010. Reading 1D barcodes with mobile phones using deformable templates. *IEEE Trans. Pattern Anal. Mach. Intell.* 33 (9), 1834–1843.
- Girshick, R., 2015. Fast R-CNN. In: *Proceedings of the IEEE International Conference on Computer Vision*. pp. 1440–1448.
- Gonzalez, R.C., Woods, R.E., 2002. *Digital image processing*. Digit. Image Process..
- Hansen, D.K., Nasrollahi, K., Rasmussen, C.B., Moeslund, T.B., 2017. Real-time barcode detection and classification using deep learning. In: *International Joint Conference on Computational Intelligence*. SCITEPRESS Digital Library, pp. 321–327.
- He, K., Zhang, X., Ren, S., Sun, J., 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 37 (9), 1904–1916.
- He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 770–778.
- Hildebrand, A.P., 1977. Laser scanning of the new standardized bar code. In: *Practical Applications of Low Power Lasers*. Vol. 92, *SPIE*, pp. 61–65.
- Hu, H., Xu, W., Huang, Q., 2009. A 2D barcode extraction method based on texture direction analysis. In: 2009 Fifth International Conference on Image and Graphics. *IEEE*, pp. 759–762.
- Hussain, M., 2023. YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection. *Machines* 11 (7), 677.
- Jain, A.K., Karu, K., 1996. Learning texture discrimination masks. *IEEE Trans. Pattern Anal. Mach. Intell.* 18 (2), 195–205.
- Jocher, G., 2020. Ultralytics YOLOv5. <http://dx.doi.org/10.5281/zenodo.3908559>, URL <https://github.com/ultralytics/yolov5>.
- Jocher, G., Chaurasia, A., Qiu, J., 2023. Ultralytics YOLOv8. URL <https://github.com/ultralytics/ultralytics>.
- Kalinov, I., Petrovsky, A., Ilin, V., Pristanskiy, E., Kurenkov, M., Ramzhaev, V., Idrisov, I., Tsetserukou, D., 2020. Warevision: Cnn barcode detection-based UAV trajectory optimization for autonomous warehouse stocktaking. *IEEE Robot. Autom. Lett.* 5 (4), 6647–6653.
- Kamnardsiri, T., Charoenkwan, P., Malang, C., Wudhikarn, R., 2022. 1D barcode detection: Novel benchmark datasets and comprehensive comparison of deep convolutional neural network approaches. *Sensors* 22 (22), 8788.
- Kapsambelis, C., 2005. Bar codes aren't going away!

- Katona, M., Nyúl, L.G., 2013. Efficient 1D and 2D barcode detection using mathematical morphology. In: *Mathematical Morphology and Its Applications To Signal and Image Processing: 11th International Symposium, ISMM 2013, Uppsala, Sweden, May 27-29, 2013. Proceedings 11*. Springer, pp. 464–475.
- Kingma, D.P., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
- Klimek, G., Vamossy, Z., 2013. QR code detection using parallel lines. In: *2013 IEEE 14th International Symposium on Computational Intelligence and Informatics. CINTI, IEEE*, pp. 477–481.
- Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* 25.
- Kubáňová, J., Kubasáková, I., Čulík, K., Štítik, L., 2022. Implementation of barcode technology to logistics processes of a company. *Sustainability* 14 (2), 790.
- Li, J., Zhao, Q., Tan, X., Luo, Z., Tang, Z., 2018. 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)*. Springer, pp. 261–267.
- Liao, H.-Y., Liu, S.-J., Chen, L.-H., Tyan, H.-R., 1995. A bar-code recognition system using backpropagation neural networks. *Eng. Appl. Artif. Intell.* 8 (1), 81–90.
- Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017a. Feature pyramid networks for object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2117–2125.
- Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollár, P., 2017b. Focal loss for dense object detection. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2980–2988.
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft coco: Common objects in context. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, pp. 740–755.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C., 2016. SSD: Single shot multibox detector. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, the Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, pp. 21–37.
- Lv, W., Xu, S., Zhao, Y., Wang, G., Wei, J., Cui, C., Du, Y., Dang, Q., Liu, Y., 2023. DETRS beat YOLOs on real-time object detection. arXiv preprint arXiv:2304.08069.
- McCathie, L., 2004. The advantages and disadvantages of barcodes and radio frequency identification in supply chain management.
- Melek, C.G., Battini Sönmez, E., Varlı, S., 2024. Datasets and methods of product recognition on grocery shelf images using computer vision and machine learning approaches: An exhaustive literature review. *Eng. Appl. Artif. Intell.* 133, 108452. <http://dx.doi.org/10.1016/j.engappai.2024.108452>, URL <https://www.sciencedirect.com/science/article/pii/S0952197624006109>.
- Monfared, M., Koochari, A., Monshianmotlagh, R., 2021. QR-DN1.0: A new distorted and noisy QRs dataset. *Data Brief* 39, 107605.
- Muniz, R., Junco, L., Otero, A., 1999. A robust software barcode reader using the Hough transform. In: *Proceedings 1999 International Conference on Information Intelligence and Systems (Cat. No. PR00446)*. IEEE, pp. 313–319.
- Neyroud, J., Metayer, P., Danel, F., 1980. Laser beam scanning for remote control. In: *2nd European Congress on Optics Applied To Metrology*. Vol. 210, SPIE, pp. 107–115.
- Niu, Y., Wang, L., Yu, Z., Huang, J., Huang, B., Su, Y., 2023. Vision-based automatic order check method for online medicine dispensing cabinet under incomplete data. *Eng. Appl. Artif. Intell.* 123, 106204.
- Norman J. Woodland, S.B., 1949. Classifying apparatus and method. (2612994A), URL <https://patents.google.com/patent/US2612994A/en>.
- OpenMP Architecture Review Board, 2008. OpenMP application program interface version 3.0. URL <http://www.openmp.org/mp-documents/spec30.pdf>.
- OpenVINO, 2024. OpenVINO™ Toolkit, <https://github.com/openvinotoolkit/openvino>, Accessed: [Date Accessed].
- Otsu, N., 1979. A threshold selection method from gray-level histograms. *IEEE Trans. Syst. Man Cybern.* 9 (1), 62–66.
- Ottaviani, Pavan, Bottazzi, Brunelli, Caselli, Guerrero, 1999. 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, IET, pp. 652–655.
- Padilla, R., Passos, W.L., Dias, T.L.B., Netto, S.L., da Silva, E.A.B., 2021. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics* 10 (3), <http://dx.doi.org/10.3390/electronics10030279>, URL <https://www.mdpi.com/2079-9292/10/3/279>.
- Quenum, J., Wang, K., Zakhori, A., 2021. Fast, accurate barcode detection in ultra high-resolution images. In: *2021 IEEE International Conference on Image Processing. ICIP, IEEE*, pp. 1019–1023.
- Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788.
- Redmon, J., Farhadi, A., 2017. YOLO9000: better, faster, stronger. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7263–7271.
- Redmon, J., Farhadi, A., 2018. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.
- Reich, S., 1977. The use of electro-mechanical mirror scanning devices. In: *Laser Scanning Components and Techniques: Design Considerations/Trends*. Vol. 84, SPIE, pp. 47–56.
- Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* 28.
- Shah, S., Tembhurne, J., 2023. Object detection using convolutional neural networks and transformer-based models: a review. *J. Electr. Syst. Inf. Technol.* 10 (1), 54.
- Soliman, A., Al-Ali, A., Mohamed, A., Gedawy, H., Izham, D., Bahri, M., Erbad, A., Guizani, M., 2023. AI-based UAV navigation framework with digital twin technology for mobile target visitation. *Eng. Appl. Artif. Intell.* 123, 106318.
- Sörös, G., Flörkemeier, C., 2013. 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.
- Szentandrás, I., Herout, A., Dubská, M., 2012. Fast detection and recognition of QR codes in high-resolution images. In: *Proceedings of the 28th Spring Conference on Computer Graphics*, pp. 129–136.
- Taveerad, N., Vongpradhip, S., 2015. Development of color QR code for increasing capacity. In: *2015 11th International Conference on Signal-Image Technology & Internet-Based Systems. SITIS, IEEE*, pp. 645–648.
- Tekin, E., Coughlan, J., 2012. BLADE: Barcode localization and decoding engine. *Tech. Rep. 2012-RERC. 01*.
- Vaishnavi Shyamsundar Mate, S.M., 2023. Barcode reader market size, share, competitive landscape and trend analysis report by type, by application: Global opportunity analysis and industry forecast, 2023–2032.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. *Adv. Neural Inf. Process. Syst.* 30.
- Ventsov, N., Podkolzina, L., 2018. Localization of barcodes using artificial neural network. In: *2018 IEEE East-West Design & Test Symposium. EWDTS, IEEE*, pp. 1–6.
- Viard-Gaudin, C., Normand, N., Barba, D., 1993. A bar code location algorithm using a two-dimensional approach. In: *Proceedings of 2nd International Conference on Document Analysis and Recognition. ICDAR'93, IEEE*, pp. 45–48.
- Voulodimos, A., Doulamis, N., Bebis, G., Stathaki, T., 2018. Recent developments in deep learning for engineering applications. *Comput. Intell. Neurosci.* 2018.
- Wachenfeld, S., Terlunen, S., Jiang, X., 2008. Robust recognition of 1-d barcodes using camera phones. In: *2008 19th International Conference on Pattern Recognition. IEEE*, pp. 1–4.
- Wang, C.-Y., Bochkovskiy, A., Liao, H.-Y.M., 2022. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv preprint arXiv:2207.02696.
- Weng, D., Yang, L., 2012. Design and implementation of barcode management information system. In: *Zhu, R., Ma, Y. (Eds.), Information Engineering and Applications. Springer London, London*, pp. 1200–1207.
- Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., Girshick, R., 2019. Detectron2. <https://github.com/facebookresearch/detectron2>.
- Wudhikarn, R., Charoenkwan, P., Malang, K., 2022. Deep learning in barcode recognition: A systematic literature review. *IEEE Access* 10, 8049–8072.
- Yu, F., Koltun, V., 2015. Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122.
- Yuan, B., Li, Y., Jiang, F., Xu, X., Guo, Y., Zhao, J., Zhang, D., Guo, J., Shen, X., 2019. MU R-CNN: A two-dimensional code instance segmentation network based on deep learning. *Futur. Internet* 11 (9), 197.
- Yun, I., Kim, J., 2017. Vision-based 1D barcode localization method for scale and rotation invariant. In: *TENCON 2017-2017 IEEE Region 10 Conference. IEEE*, pp. 2204–2208.
- Zamberletti, A., Gallo, I., Albertini, S., 2013. Robust angle invariant 1D barcode detection. In: *2013 2nd IAPR Asian Conference on Pattern Recognition. IEEE*, pp. 160–164.
- Zamberletti, A., Gallo, I., Carullo, M., Binaghi, E., 2010. Neural image restoration for decoding 1-D barcodes using common camera phones. In: *VISAPP (1)*, pp. 5–11.
- Zhang, L., Sui, Y., Zhu, F., Zhu, M., He, B., Deng, Z., 2021. Fast barcode detection method based on ThinYOLOv4. In: *Cognitive Systems and Signal Processing: 5th International Conference, ICCSIP 2020, Zhuhai, China, December 25–27, 2020, Revised Selected Papers 5*. Springer, pp. 41–55.
- Zharkov, A., Zagaynov, I., 2019. Universal barcode detector via semantic segmentation. In: *2019 International Conference on Document Analysis and Recognition. ICDAR, IEEE*, pp. 837–843.
- Zong, Z., Song, G., Liu, Y., 2023. Detsr with collaborative hybrid assignments training. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6748–6758.
- Zou, Z., Chen, K., Shi, Z., Guo, Y., Ye, J., 2023. Object detection in 20 years: A survey. *Proc. IEEE*.



Enrico Vezzali received both B.Sc. and M.Sc. degrees in Electrical Engineering from Università degli Studi Bologna, Italy. He is currently pursuing an Industrial Ph.D. at Università degli Studi di Modena e Reggio Emilia in Italy granted by Datalogic while working as a Machine Learning Engineer in the same company. His main research interests are Industrial applications of Artificial intelligence and Computer Vision, with particular emphasis on Image Enhancement.



Federico Bolelli received the B.Sc. and M.Sc. degrees in Computer Engineering from Università degli Studi di Modena e Reggio Emilia, Italy. He pursued a Ph.D. degree from the same university where he is currently working as a Tenure Track Assistant Professor within the AlmageLab group at Dipartimento di Ingegneria “Enzo Ferrari”. His research interests include image processing, algorithms and optimization, and medical imaging. He is currently involved in a H2020 European Project.



Stefano Santi, Ph.D., is a senior technology advisor to the CTO at Datalogic. He previously led the company’s North American advanced research in new technology and deep learning. His interests range from machine learning and deep learning technology applied to the state-of-the-art barcode decoding methods, to code optimization for strict real-time performance on embedded systems. Stefano Santi holds nine patents and several other publications related to acquisition systems and computer vision applied to barcode technology.



Costantino Grana graduated from Università degli Studi di Modena e Reggio Emilia, Italy in 2000 and achieved a Ph.D. in Computer Science and Engineering in 2004. He is currently a Full Professor at Dipartimento di Ingegneria “Enzo Ferrari” of the same university. His research interests are mainly in medical imaging, optimization of image processing algorithms, and computer vision applications. He published 6 book chapters, 47 papers in international peer-reviewed journals, and more than 130 papers at international conferences.