A Deep-Learning-Based Method for Real-Time Barcode Segmentation on Edge CPUs

Enrico Vezzali[1,2], Lorenzo Vorabbi[2], Costantino Grana[1], and Federico Bolelli[1]✉

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

Abstract. Barcodes are a critical technology in industrial automation, logistics, and retail, enabling fast and reliable data capture. While deep learning has significantly improved barcode localization accuracy, most modern architectures remain too computationally demanding for real-time deployment on embedded systems without dedicated hardware acceleration. In this work, we present **BaFaLo** (**Ba**rcode **Fa**st **Lo**calizer), an ultra-lightweight segmentation-based neural network for barcode localization. Our model is specifically optimized for real-time performance on low-power CPUs while maintaining high localization accuracy for both 1D and 2D barcodes. It features a two-branch architecture—comprising a local feature extractor and a global context module—and is tailored for low-resolution inputs to improve inference speed further. We benchmark BaFaLo against several lightweight architectures for object detection or segmentation, including YOLO Nano, Fast-SCNN, BiSeNet V2, and ContextNet, using the BarBeR dataset. BaFaLo achieves the fastest inference time among all deep-learning models tested, operating at 57.62 ms per frame on a single CPU core of a Raspberry Pi 3B+. Despite its compact design, it achieves a decoding rate nearly equivalent to YOLO Nano for 1D barcodes and only 3.5 percentage points lower for 2D barcodes while being approximately nine times faster.

Keywords: Barcodes · Embedded Systems · Object Detection.

1 Introduction

Barcodes are a pivotal technology for automated data capture and identification, relying on simple visual codes to convey complex information cost-effectively. Their versatility has led to widespread adoption across numerous domains: from inventory tracking and logistics [8,19] to automated warehouse operations [7], manufacturing [19], retail product recognition [9], and even robot navigation [13]. Recognizing the vital role of barcodes, both academia and industry have increasingly focused on enhancing barcode localization techniques [20].

✉ Corresponding author: federico.bolelli@unimore.it

The first attempts at barcode localization in 2D images, dating back to the 1990s, relied on traditional computer vision techniques such as edge detection, Hough transforms, and texture direction analysis [5,10,18]. While computationally efficient, these approaches were sensitive to noise, lighting variations, and perspective distortions. The methods proposed by Sörös *et al.* [14] and Zamberletti *et al.* [23] improved the robustness of traditional techniques by incorporating structural analysis and learning-based refinement, but remained limited by resolution sensitivity and focused primarily on 1D barcodes.

The shift toward deep-learning-based methods began in the mid-2010s with the application of convolutional neural networks (CNN) [1,3]. Object detectors such as YOLO and Faster R-CNN were adapted for barcode localization, achieving significant improvements in accuracy and robustness. More recently, dedicated architectures for barcode segmentation have been proposed, such as the method introduced by Zharkov *et al.* [24], to achieve faster inference while remaining quite reliable. Recent work by Vezzali *et al.* [17] introduced the BarBeR benchmarking suite alongside a dataset of 8 748 images of barcodes, revealing that deep-learning models significantly outperform hand-crafted methods. However, these models often incur high computational costs, limiting deployment on low-power hardware such as the Raspberry Pi CPU, where even compact networks such as YOLO V8 Nano [6] can exceed one second of inference time [17], making real-time processing impractical.

In this paper, we introduce **BaFaLo** (**Ba**rcode **Fa**st **Lo**calizer), an ultra-lightweight architecture for barcode localization and segmentation. By selectively reducing model complexity while preserving critical performance, BaFaLo achieves real-time inference on embedded CPUs for both 1D and 2D barcodes, with less than a 3.5% drop in decoding rate compared to more computationally demanding models like YOLO Nano. Furthermore, BaFaLo significantly outperforms other lightweight segmentation models, such as the one proposed by Zharkov *et al.* [24], as well as traditional hand-crafted approaches from Yun *et al.* and Sörös *et al.*, while maintaining superior speed. This advance opens the door to cost-efficient solutions for industrial, retail, and robotics applications requiring rapid barcode scanning in the field. To support reproducibility and further research, we release all source code for BaFaLo, along with the training and validation scripts and the trained models: `https://github.com/Henvezz95/BarBeR`.

2 Proposed Method

In this work, we propose a deep learning framework to localize 1D and 2D barcodes on edge devices in real time (processing time < 60 ms). In particular, we want to show that deep-learning-based localization of barcodes is possible even without accelerators and that an ARM CPU is enough for the task (in our case, a Raspberry PI 3B+). Rather than relying on traditional detection networks (e.g., YOLO), our method employs a segmentation-based architecture that offers several distinct advantages. First, segmentation networks can leverage local texture information and, therefore, function effectively with a smaller

receptive field, whereas detection networks require having the entire object in view. Second, because segmentation operates on pixel-level classifications using convolutional blocks, the network more naturally adapts to different input resolutions and handles scale variations without relying on fixed-size anchor boxes. Finally, the segmentation output contains richer, pixel-wise information—such as orientation or region boundaries—potentially benefiting the downstream decoding phase, though we do not explore this in the current paper.

Our design takes inspiration from Fast-SCNN [12], an architecture known for its real-time performance on embedded devices. However, the original Fast-SCNN design remained too slow for the use case targeted in this paper. Our proposed approach is organized into four modules: (*i*) a *Learning to Downsample* module that quickly reduces spatial resolution while preserving low-level features; (*ii*) a *Coarse Feature Extraction* module that uses bottleneck residual blocks to capture a broader context. This component is a streamlined, lightweight adaptation of the Global Feature Extractor from Fast-SCNN; (*iii*) a *Feature Fusion* module to merge the high-resolution details from the downsample path with the global context extracted at lower resolution; and (*iv*) a lightweight *Classifier* that upsamples the fused features to produce the final segmentation map. This pipeline provides a strong balance of local detail and global context.

2.1 Balancing Low-Resolution and High-Resolution Features

To achieve the necessary speed, we reduced the number of layers and channels in the architecture. Vezzali *et al.* [17] demonstrated that even very low-resolution images can provide sufficient information to localize barcodes. For instance, YOLO Nano achieved a high mAP@50 (0.961) when localizing barcodes in the BarBeR dataset [15,16] using a 320×320 resolution. Inspired by these findings, we adopted a similar target resolution, which is much lower than the high-resolution Cityscapes dataset [2] originally used to train Fast-SCNN. Consequently, our first major modification was to significantly streamline the Global Feature Extraction module—reducing its nine linear bottleneck layers to three, lowering the output channels from (64, 64, 64, 96, 96, 96, 128, 128, 128) to (32, 32, 32), and decreasing the expansion ratio of each bottleneck from six to two. We also removed the Pyramid Pooling module at the end, as it offered no tangible benefit in our application. Since this branch now provides a more limited receptive field and captures coarser context, we refer to it as the *Coarse Feature Extractor* rather than a *Global Feature Extractor*. In contrast, the Learning to Downsample module plays a pivotal role in feeding subsequent layers with sufficiently low-level features; therefore, reducing it too aggressively would compromise overall network capacity. We preserved its three convolutional layers but lowered their channel counts from (32, 48, 64) to (12, 24, 32). Furthermore, we opted for regular 3×3 convolutions in the second and third layers rather than depthwise separable convolutions. Our tests showed that the latter would diminish the model capacity too much for a very modest speed improvement. As a result, this new module contains 9 828 parameters—compared to 6 192 in

the original Fast-SCNN—yet still operates more efficiently under our real-time constraints.

2.2 Pixel Shuffle

In the original Fast-SCNN design, the classifier produces a segmentation map at $\frac{1}{8}$ of the input resolution and then uses linear upscaling to match the input's dimensions. However, working with low-resolution images makes it difficult to preserve the fine details of small objects at such a low output resolution. To address this, we replace the linear upscaling step with a pointwise convolution followed by a pixel shuffle operation. First, the pointwise convolution expands the feature maps from 32 to 128 channels. Then, the pixel shuffle layer rearranges these channels spatially by a factor of eight, restoring the original input resolution. This process also reduces the final channel count to two—one channel for 1D barcodes and another for 2D barcodes—thereby enabling clear segmentation of both barcode types in a low-resolution setting.

2.3 Training

Fig. 1 illustrates our final proposed architecture. We trained this network on the BarBeR dataset for 300 epochs with a batch size of 16. Each image was resized during preprocessing so that its longest side measured 320 pixels while preserving the original aspect ratio. We then zero-padded the shorter dimension until the input reached 320×320. The Adam optimizer was used with an initial learning rate of 1×10^{-3}, which we decayed exponentially to 1×10^{-5} by the final epoch. To increase robustness, we applied random scaling (from $0.6\times$ to $1.4\times$), random flips (horizontal and vertical), and random adjustments to saturation and contrast. We used a pixel-wise binary cross-entropy as a loss function without adding any additional auxiliary loss.

2.4 Inference

During inference, the proposed neural network processes the input image and generates a two-channel heatmap of the same dimensions as the input. The first channel corresponds to 1D barcode regions, while the second corresponds to 2D barcode regions. Although the network produces pixel-level outputs and can be used for segmentation, in this work, we employ it exclusively for detection. To convert the heatmap into detection boxes, each channel is thresholded independently (we use a fixed threshold of 0.4 in all experiments). We then apply blob detection to the binarized map, and a bounding box is computed for each identified blob.

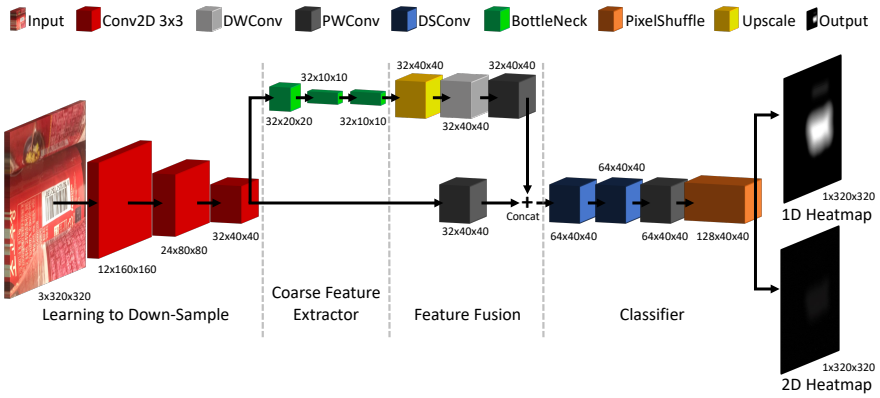

Fig. 1. Architecture of our proposed model. DWConv stands for depthwise convolution, PWConv for pointwise convolution, and DSConv for depthwise separable convolution. We indicate the size of the output tensor for each block, considering an input RGB image of size 320×320.

3 Results

3.1 Experimental Setup

All experiments were conducted on the BarBeR dataset,[3] and the full evaluation pipeline is available on our GitHub repository.[4] We used 5-fold cross-validation to ensure robust evaluation. The dataset was split into five equal subsets; each subset was used once as the test set, with the remaining four used for training. All images were resized so that the longest side measured 320 pixels before being processed in any test.

Localization Accuracy. We evaluated all deep-learning-based localization models, including ours, on the complete dataset of 8 748 images. For each method, we computed the average precision scores, AP@0.5 and AP@[0.5:0.95], separately for 1D and 2D barcodes. For segmentation models, region-level confidence scores were computed as the mean of the predicted pixel confidences within each detected region. We also included three traditional methods tailored to 1D barcode localization (Sörös *et al.* [14], Yun *et al.* [22], and Zamberletti *et al.* [23]), which are part of the BarBeR benchmark. Since these methods only support 1D barcodes, we excluded 2D barcodes from this evaluation. Moreover, as the method by Sörös *et al.* detects only one barcode per image, we restricted this evaluation to the 6 811 images containing a single 1D barcode.

Time Benchmark. For each localization method, we measured the inference time per image. Each image was processed three times, and the minimum inference time was recorded to minimize the impact of background processes. The final timing for each method was computed as the average over all images. We

[3] `https://ditto.ing.unimore.it/barber`
[4] `https://github.com/Henvezz95/BarBeR`

Table 1. Performance Comparison of Deep-Learning-Based Barcode Localization Methods. The table reports average precision (AP@0.5 and AP@[0.5:0.95]), decoding rate (% Dec), and single-threaded inference time on two platforms: a high-end PC and a Raspberry Pi 3B. Results are provided separately for 1D and 2D barcodes. All input images were resized such that the longest side measures 320 pixels.

Model	1D barcodes			2D barcodes			Times	
	AP@0.5 ↑	AP@[.5:.95] ↑	% Dec ↑	AP@0.5 ↑	AP@[.5:.95] ↑	% Dec ↑	PC (ms) ↓	Rasp Pi (ms) ↓
YOLO Nano	**0.976**	**0.860**	**59.3%**	**0.947**	**0.872**	**63,5%**	21.55	509.1
Zharkov *et al.*	0.530	0.254	50.9%	0.571	0.382	55.9%	4.521	180.4
ContextNet 0.25x	0.808	0.592	55.5%	0.413	0.295	33.0%	5.359	209.9
BisenetV2 0.25x	0.909	0.725	58.8%	0.834	0.657	60.9%	5.877	193.4
BisenetV2 0.125x	0.880	0.673	58.3%	0.681	0.479	49.2%	2.943	84.73
Fast SCNN 0.5x	0.863	0.629	57.9%	0.702	0.486	53.7%	3.313	120.0
Fast SCNN 0.25x	0.783	0.523	55.7%	0.534	0.354	42.1%	2.235	64.56
Ours	0.898	0.694	58.5%	0.822	0.628	60.0 %	**1.635**	**57.62**

conducted benchmarks on both a Raspberry Pi 3B+ (representing low-power embedded devices) and a high-end workstation equipped with an AMD Ryzen Threadripper Pro 5965WX CPU (24 cores) and 128 GB of DDR4 RAM. All deep-learning architectures have been converted to ONNX to accurately measure their maximum speed in possible real-world applications.

Decoding Test. To assess practical usability, we measured the decoding rate using a virtual barcode reader composed of two stages: localization and decoding. In the first stage, the tested localization method outputs bounding boxes. Each box defines a region, which is then passed to the `pyzbar` library for decoding. Before cropping, each box is extended by 20 pixels on each side since barcode readers usually require a quiet zone around the code. Since `pyzbar` supports only a subset of barcode formats (QR Code, EAN-13, EAN-8, UPC-E, Interleaved 2 of 5, Code 39, and Code 128), we restricted this test to those types. The decoding rate is defined as the proportion of images in which at least one barcode is successfully decoded. If decoding fails at the base resolution (320 pixels on the longest side), we attempt decoding again on a 4× larger version of the image (longest side = 1280 pixels) to accommodate barcodes that cover a small area of the overall image.

3.2 Experimental Results

We first evaluated our model against several deep-learning-based localization methods. As baselines, we included YOLO Nano and the architecture proposed by Zharkov *et al.*, both of which are integrated into the BarBeR benchmark. Additionally, we selected three real-time segmentation networks commonly used in lightweight vision tasks: ContextNet [11], BiSeNet V2 [21], and Fast-SCNN [12].

To ensure a fair comparison on embedded hardware, we tested reduced versions of these models using uniform channel-width scaling. This technique, commonly known as a width multiplier [4], uniformly reduces the number of filters in each convolutional layer by a fixed ratio. For example, a 0.25× configuration

Table 2. Comparison with traditional methods for 1D barcode localization. Each model is evaluated using the Precision (P), Recall (R), and F1 score at an IoU threshold of 0.5, decoding rate (% Dec), and single-threaded inference time on a PC and a Raspberry Pi 3B. All input images were resized such that the longest side measures 320 pixels.

Model	1D barcodes				Times	
	P@0.5 ↑	R@0.5 ↑	F1@0.5 ↑	% Dec ↑	PC (ms) ↓	Rasp Pi (ms) ↓
Sörös *et al.*	0.429	0.429	0.429	51.4%	2.782	92.07
Yun *et al.*	0.918	0.436	0.591	39.9%	2.171	**52.83**
Zamberletti *et al.*	0.103	0.125	0.113	11.0%	17.42	855.7
Ours	**0.976**	**0.947**	**0.962**	**66.0%**	**1.635**	57.62

reduces all channels to 25% of their original count, significantly lowering computational complexity. Based on this, we evaluated different variants: ContextNet (0.25×), Fast-SCNN (0.5× and 0.25×), and BiSeNet V2 (0.25× and 0.125×).

The performance of all deep-learning models is reported in Tab. 1, in terms of AP@0.5, AP@[0.5:0.95], processing times on both PC and Raspberry Pi (single-threaded), and decoding rate using `pyzbar`. Our method achieves the fastest inference time across all platforms, requiring only 57.62 ms per image on a single CPU core, making it capable of near real-time operation even on embedded devices. Despite its simplicity, our method ranks third among all models in both AP@0.5 and AP@[0.5:0.95] and in decoding rate.

The second-best decoding rate is achieved by BiSeNet V2 (0.25×), which outperforms our model by just 0.3 percentage points for 1D barcodes (58.8% vs. 58.5%) and 0.9 for 2D barcodes (60.9% vs. 60.0%), while being approximately 3.5 times slower. YOLO Nano achieves the highest accuracy overall, as expected, but its inference time of 509.1 ms per image makes it impractical for many real-time embedded applications. Notably, despite being over 9 times slower than our model, it achieves only a marginal improvement of 0.8 percentage points in 1D barcode decoding and 3.5 points in 2D barcode decoding.

Our method also outperforms several slower models, including Zharkov *et al.*, ContextNet (0.25×), Fast-SCNN (0.5× and 0.25×), and BiSeNet V2 (0.125×), making it the only approach to achieve such high accuracy at this speed. This result underscores that our architectural choices offer a substantial advantage over Fast-SCNN, from which we drew inspiration. Tab. 2 compares our method to three classical, hand-crafted approaches: Sörös *et al.*, Yun *et al.*, and Zamberletti *et al.*. Our method is the fastest on PC and the second-fastest on Raspberry Pi 3B+, with only a slight difference from the fastest traditional model (Yun *et al.*). However, it achieves significantly higher Precision, Recall, F1 scores, and decoding rates. Traditional methods are highly sensitive to resolution and require higher pixel density for accurate decoding [17]. Operating at a higher resolution could improve the results a bit but would make these methods around 4× times slower. Additionally, these methods are limited to 1D barcode detection, whereas our model supports both 1D and 2D barcode localization.

Table 3. Pipeline timing breakdown for barcode detection, comparing localization plus decoding (*Loc + Decoding*) against running `pyzbar` alone (*No Localization*). The table lists per-image processing times on a PC and a Raspberry Pi 3B, with all input images resized so their longest side is 320 pixels. Total time is the sum of localization and decoding.

Model	Times PC (ms)			Times Raspberry Pi (ms)		
	Loc ↓	Decoding ↓	Total ↓	Loc ↓	Decoding ↓	Total ↓
No Localization	-	35.944	35.944	-	299.22	299.22
YOLO Nano	21.558	5.369	27.197	509.16	31.17	540.33
Zharkov	4.521	13.056	17.577	180.41	66.29	246.70
ContextNet 0.25x	5.359	12.223	17.582	209.93	28,02	237.95
BisenetV2 0.25x	5.877	10.685	16.562	193.41	30.54	223.95
BisenetV2 0.125x	2.943	9.118	12.061	84.73	**29.80**	114.53
Fast SCNN 0.5x	3.313	**4.935**	8.248	120.00	30,09	150.09
Fast SCNN 0.25x	2.235	5.190	7.425	64.56	40.39	104.95
Ours	**1.635**	5.431	**7.066**	**57.62**	37.54	**95.16**

3.3 Decoding Time

In addition to measuring localization speed in isolation, we also evaluated the time required to run the entire detection pipeline, including both localization and decoding. This serves two main purposes. First, it reveals how effectively each model limits false positives and unnecessarily large bounding boxes. A model with overly generous detections may achieve high recall but will suffer increased decoding overhead, as the `pyzbar` library would process more (or larger) crops. Second, it clarifies how much overall latency is reduced by speeding up the localization step.

As shown in Tab. 3, running `pyzbar` directly on the entire image (*No Localization*) at a maximum scale of 1280 pixels for the longest edge, takes 35.94 ms on PC and 299.22 ms on Raspberry Pi. With our localization approach, the total pipeline time drops to 7.07 ms on PC and 95.16 ms on the Pi—the fastest among the methods tested on both platforms. For reference, YOLO Nano requires 27.20 ms on PC and 540.33 ms on the Raspberry Pi, highlighting that a higher-accuracy model can become bottlenecked by slow inference on edge devices. Notably, our method does not inflate decoding times with excess bounding boxes, indicating few false positives.

Finally, although 95.16 ms per image on the Pi is not yet real-time on a single CPU core, we anticipate that basic parallelization or multithreading could push the pipeline into real-time territory. Anyway, for most applications, a reading rate of 10 FPS is enough. On PC, the decoding phase appears to be more time-consuming relative to localization, suggesting that further optimizations should focus more on the decoding component than the localizer.

Table 4. Ablation study on key architectural components of our model. We report AP@0.5, AP@[0.5:0.95], decoding rate (% Dec), and single-threaded inference time on PC and Raspberry Pi 3B. All images were resized to have a longest side of 320 pixels. Variants include the removal of pixel shuffle and the re-introduction of the pyramid pooling module.

Model	1D barcodes			2D barcodes			Times	
	AP@0.5 ↑	AP@[.5:.95] ↑	% Dec ↑	AP@0.5 ↑	AP@[.5:.95] ↑	% Dec ↑	PC (ms) ↓	Rasp PI (ms) ↓
With PPM	0.897	0.691	**58.8%**	0.783	0.568	56.2%	2.518	74.87
No PixelShuffle	0.893	0.681	58.5%	0.817	0.622	59.6%	1.829	57.93
Ours	**0.898**	**0.694**	58.5%	**0.822**	**0.628**	**60.0** %	**1.635**	**57.62**

3.4 Ablation Studies

We evaluated the impact of two architectural modifications that differentiate our model from the original Fast-SCNN design, beyond the already reduced depth and channel width. Specifically, we assess (*i*) the use of a pixel shuffle module for upsampling and (*ii*) the removal of the pyramid pooling module (PPM) from the global feature extractor. Results are shown in Tab. 4. All tests were conducted at a resolution where the longest image side was set to 320 pixels.

Our full configuration achieves the best overall performance, with the fastest inference time (57.62 ms on Raspberry Pi 3B+) and the highest accuracy across most metrics. The version without pixel shuffle shows slightly lower performance on both AP and decoding rate, particularly on 2D barcodes (60.0% vs. 59.6%), suggesting that pixel shuffle contributes positively to precise localization.

Reintroducing the PPM results in a significant slowdown (74.8 ms vs. 57.6 ms) and mixed accuracy changes: a small improvement in 1D decoding rate (58.8% vs. 58.5%), but noticeably worse performance on 2D barcodes, with lower AP scores (0.783 vs. 0.822) and decoding rate (56.2% vs. 60.0%). These results confirm that removing the PPM and adopting pixel shuffle leads to a better overall balance between speed and performance in our final model.

4 Conclusion and Future Research

In this paper, we introduced **BaFaLo**, an ultra-lightweight neural network architecture for barcode localization and segmentation. Designed for real-time performance on embedded CPUs, BaFaLo achieves real-time speed while resulting in a similar decoding rate to much slower architectures. Unlike many deep-learning models that require powerful GPUs or accelerators, BaFaLo can localize both 1D and 2D barcodes in real time on a Raspberry Pi 3B+, requiring only 57.62 ms per image on a single CPU core.

Looking ahead, now that a high-speed and accurate localization framework is available, future research should explore the optimization of the decoding step. While `pyzbar` provides broad format compatibility, it is not optimized for speed and can become a bottleneck in real-time systems. Developing a fast,

lightweight decoder—potentially using deep-learning techniques—could significantly improve end-to-end performance. Moreover, additional tests could evaluate various multithreading strategies that leverage multiple cores on embedded CPUs, and further speed-ups may be achieved through model quantization (e.g., 8-bit or lower).

In parallel, more extensive testing on video streams would offer insights into real-world deployment scenarios. In such contexts, fast but slightly less accurate methods might still yield better results, as higher frame rates provide more decoding opportunities over time. Understanding the trade-off between speed and temporal redundancy will be crucial for robust barcode reading in industrial and retail applications.

Finally, because our network excels at pixel-level texture segmentation, BaFaLo can potentially address other challenges that hinge on subtle textural cues, ranging from surface defect detection in manufacturing to AR marker recognition or fast pattern analysis for robots and drones. This opens the door to a broad spectrum of applications beyond barcode localization.

Acknowledgments. This work was supported by the University of Modena and Reggio Emilia and Fondazione di Modena through the "Fondo di Ateneo per la Ricerca - FAR 2024" (CUP E93C24002080007).

Disclosure of Interests. The authors have no conflicts of interest to declare.

References

1. Chou, T.H., Ho, C.S., Kuo, Y.F.: QR code detection using convolutional neural networks. In: International Conference on Advanced Robotics and Intelligent Systems (ARIS) (2015)
2. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The Cityscapes Dataset for Semantic Urban Scene Understanding. In: Computer Vision and Pattern Recognition (2016)
3. Hansen, D.K., Nasrollahi, K., Rasmussen, C.B., Moeslund, T.B.: Real-Time Barcode Detection and Classification using Deep Learning. In: International Joint Conference on Computational Intelligence (2017)
4. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861 (2017)
5. Hu, H., Xu, W., Huang, Q.: A 2D Barcode Extraction Method Based on Texture Direction Analysis. In: International Conference on Image and Graphics (2009)
6. Jocher, G., Chaurasia, A., Qiu, J.: Ultralytics YOLOv8 (2023)
7. Kubáňová, J., Kubasáková, I., Čulík, K., Štítik, L.: Implementation of Barcode Technology to Logistics Processes of a Company. Sustainability **14**(2) (2022)
8. McCathie, L.: The advantages and disadvantages of barcodes and radio frequency identification in supply chain management. PhD Thesis, School of Information Technology and Computer Science (2004)
9. Melek, C.G., Battini Sönmez, E., Varlı, S.: Datasets and methods of product recognition on grocery shelf images using computer vision and machine learning approaches: An exhaustive literature review. Engineering Applications of Artificial Intelligence **133** (2024)

10. Muniz, R., Junco, L., Otero, A.: A Robust Software Barcode Reader using the Hough Transform. In: International Conference on Information Intelligence and Systems (1999)
11. Poudel, R.P., Bonde, U., Liwicki, S., Zach, C.: Contextnet: Exploring context and detail for semantic segmentation in real-time. arXiv preprint arXiv:1805.04554 (2018)
12. Poudel, R.P., Liwicki, S., Cipolla, R.: Fast-SCNN: Fast Semantic Segmentation Network. arXiv preprint arXiv:1902.04502 (2019)
13. Soliman, A., Al-Ali, A., Mohamed, A., Gedawy, H., Izham, D., Bahri, M., Erbad, A., Guizani, M.: AI-based UAV navigation framework with digital twin technology for mobile target visitation. Engineering Applications of Artificial Intelligence **123** (2023)
14. Sörös, G., Flörkemeier, C.: Blur-resistant joint 1D and 2D barcode localization for smartphones. In: International Conference on Mobile and Ubiquitous Multimedia (2013)
15. Vezzali, E., Bolelli, F., Santi, S., Grana, C.: Barber: A Barcode Benchmarking Repository. In: International Conference on Pattern Recognition. Springer (2025)
16. Vezzali, E., Bolelli, F., Santi, S., Grana, C.: BarBeR-Barcode Benchmark Repository: Implementation and Reproducibility Notes. In: International Workshop on Reproducible Research in Pattern Recognition (2025)
17. Vezzali, E., Bolelli, F., Santi, S., Grana, C.: State-of-the-art Review and Benchmarking of Barcode Localization Methods. Engineering Applications of Artificial Intelligence (2025)
18. Viard-Gaudin, C., Normand, N., Barba, D.: A bar code location algorithm using a two-dimensional approach. In: International Conference on Document Analysis and Recognition (1993)
19. Weng, D., Yang, L.: Design and Implementation of Barcode Management Information System. In: International Conference on Information Engineering and Applications (2012)
20. Wudhikarn, R., Charoenkwan, P., Malang, K.: Deep Learning in Barcode Recognition: A Systematic Literature Review. IEEE Access **10** (2022)
21. Yu, C., Gao, C., Wang, J., Yu, G., Shen, C., Sang, N.: BiSeNet V2: Bilateral Network with Guided Aggregation for Real-Time Semantic Segmentation. International Journal of Computer Vision **129** (2021)
22. Yun, I., Kim, J.: Vision-based 1D Barcode Localization Method for Scale and Rotation Invariant. In: TENCON - IEEE Region 10 Conference (2017)
23. Zamberletti, A., Gallo, I., Albertini, S.: Robust Angle Invariant 1D Barcode Detection. In: 2013 2nd IAPR Asian Conference on Pattern Recognition (2013)
24. Zharkov, A., Zagaynov, I.: Universal Barcode Detector via Semantic Segmentation. In: International Conference on Document Analysis and Recognition (2019)