BarBeR: A Barcode Benchmarking Repository Implementation and Reproducibility Notes

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

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

The BarBer Project

- Barcodes are a cornerstone of automatic data capture, critical in retail, manufacturing, and logistics.
- However, research in this field is hindered by the **limited availability of public datasets** and **code implementations**.
- *To address these issues, we present :*

Benchmark

BarBeR

BARCODE BENCHMARK REPOSITORY

An open-source benchmark for barcode detection algorithms.

Dataset

A public annotated dataset of 8,748 images of barcodes, with 9,818 barcodes istances.

Dataset

- **Barcode types:** 19 classes, including linear (Code 128, UPC) and 2D (QR Code, DataMatrix).
- **Annotations:** VGG format with polygon shapes, barcode type, pixels-per-module (PPM), and encoded strings.
- A total of 8748 images, with 8062 instances of 1D barcodes and 1756 2D barcodes.

The BarBer Project

- Barcodes are a cornerstone of automatic data capture, critical in retail, manufacturing, and logistics.
- However, research in this field is hindered by the **limited availability of public datasets** and **code implementations**.
- *To address these issues, we present :*

Benchmark

BarBeR

BARCODE BENCHMARK REPOSITORY

An open-source benchmark for barcode detection algorithms.

Dataset

A public annotated dataset of 8,748 images of barcodes, with 9,818 barcodes istances.

Benchmark

BarBeR

BARCODE BENCHMARK REPOSITORY

- **Available Algorithms**: Supports four traditional CV methods (**Gallo *et al.*, Soros *et al.*, Yun *et al.*, Zamberletti *et al.***) and three deep learning frameworks (**Torchvision, Ultralytics** and **Detectron2**).
- **Available Tests**: Single-Class **Localization** (1D or 2D), Multi-Class **Detection**, **Time Measurement.**
- **Available Metrics:** Precision, Recall, F1 Score, mAP@IoU.

Available Algorithms

BarBeR
BARCODE BENCHMARK REPOSITORY

- Some methods work only for 1D, others for both 1D and 2D
- Some methods support multi-ROI detection
- As mainstream architectures we tested YOLO, Faster RCNN, RetinaNET and RT-DETR

Algorithm	File	1D Detection	2D Detection	Multi-Label
Gallo *et al.* [1]	gallo_detector.py	✓	✗	✗
Soros *et al.* [2]	soros_detector.py	✓	✓	✗
Yun *et al.* [3]	yun_detector.py	✓	✗	✗
Zamberletti *et al.* [4]	zamberletti_detector.py	✓	✗	✓
Zharkov *et al.* [5]	zharkov_detector.py	✓	✓	✓
Ultralytics Models	ultralytics_detector.py	✓	✓	✓
Detectron2 Models	detectron2_detector.py	✓	✓	✓
Pytorch Models	pytorch_detector.py	✓	✓	✓

References:

[1] Gallo, Orazio, and Roberto Manduchi. "Reading 1D barcodes with mobile phones using deformable templates." *IEEE transactions on pattern analysis and machine intelligence* 33.9 (2010): 1834-1843.

[2] Sörös, Gábor, and Christian Flörkemeier. "Blur-resistant joint 1D and 2D barcode localization for smartphones." Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia. 2013.

[3] Yun, Inyong, and Joongkyu Kim. "Vision-based 1D barcode localization method for scale and rotation invariant." TENCON 2017-2017 IEEE Region 10 Conference. IEEE, 2017.

[4] Zamberletti, Alessandro, et al. "Neural 1D barcode detection using the Hough transform." Information and Media Technologies 10.1 (2015): 157-165.

[5] Zharkov, Andrey, and Ivan Zagaynov. "Universal barcode detector via semantic segmentation." 2019 International Conference on Document Analysis and Recognition (ICDAR). IEEE, 2019.

The BarBer Project

- Both Benchmark and dataset are available on GitHub.

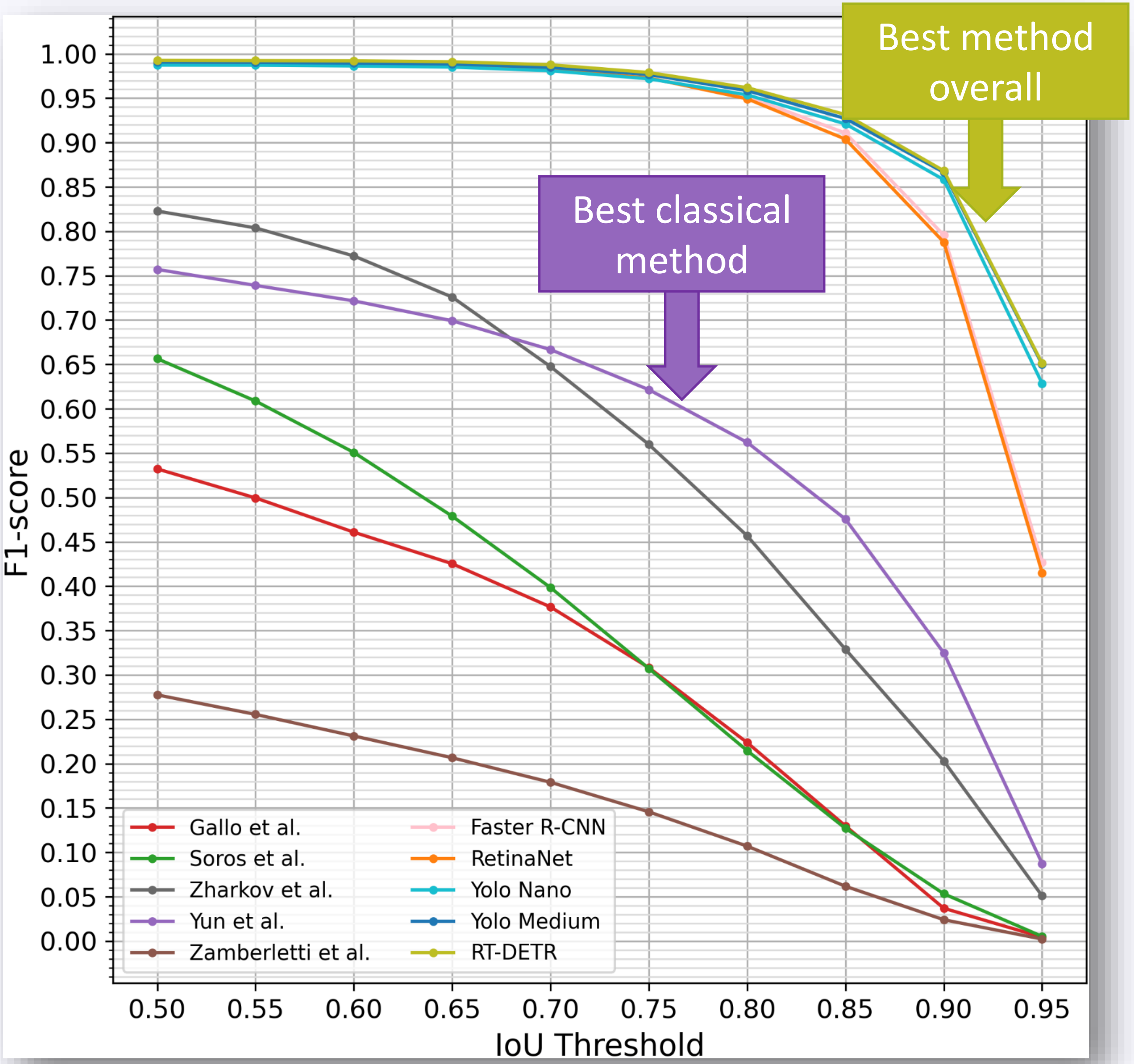
Benchmark

Link to repository

Dataset

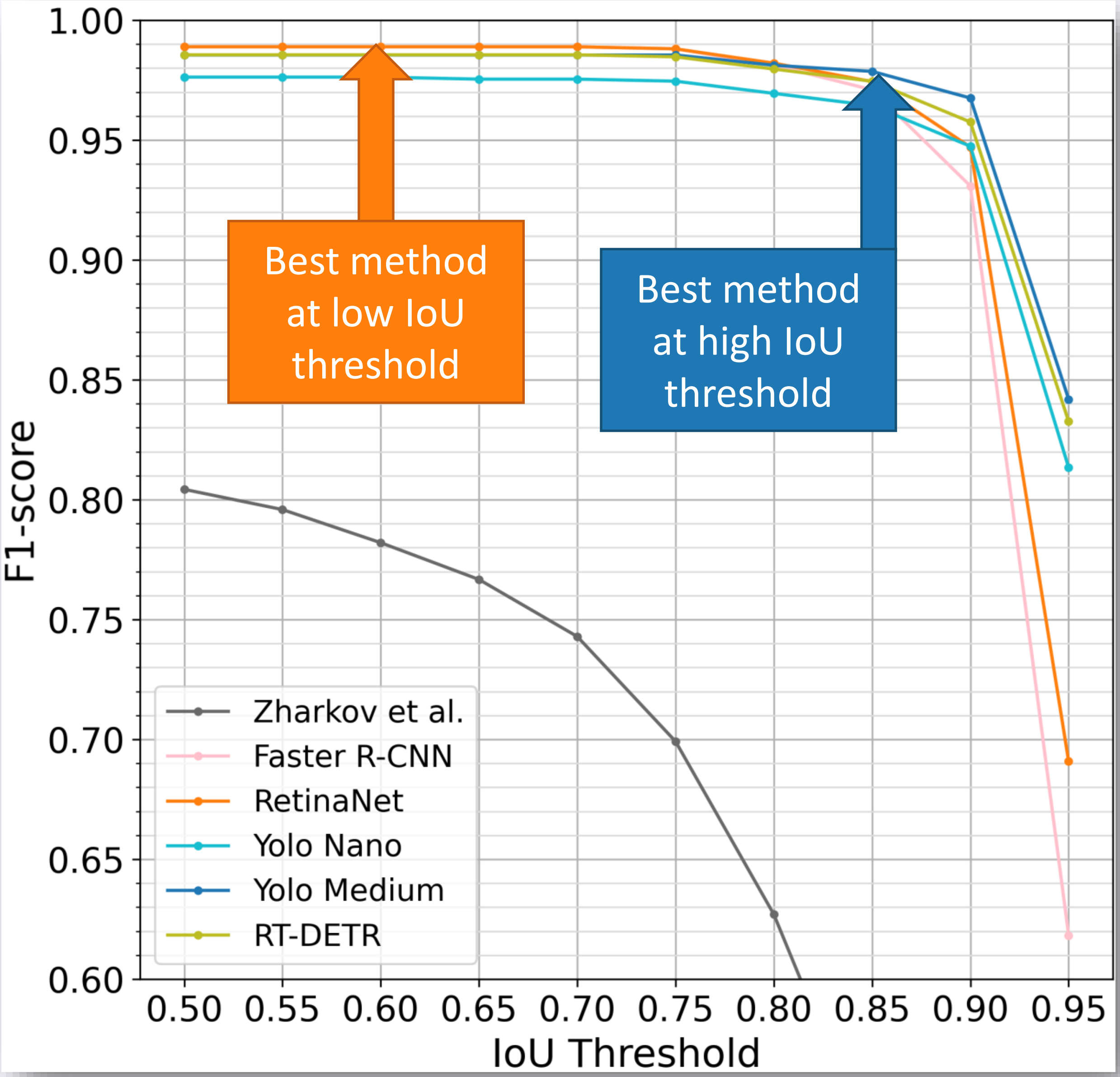
Link to dataset

1D Localization



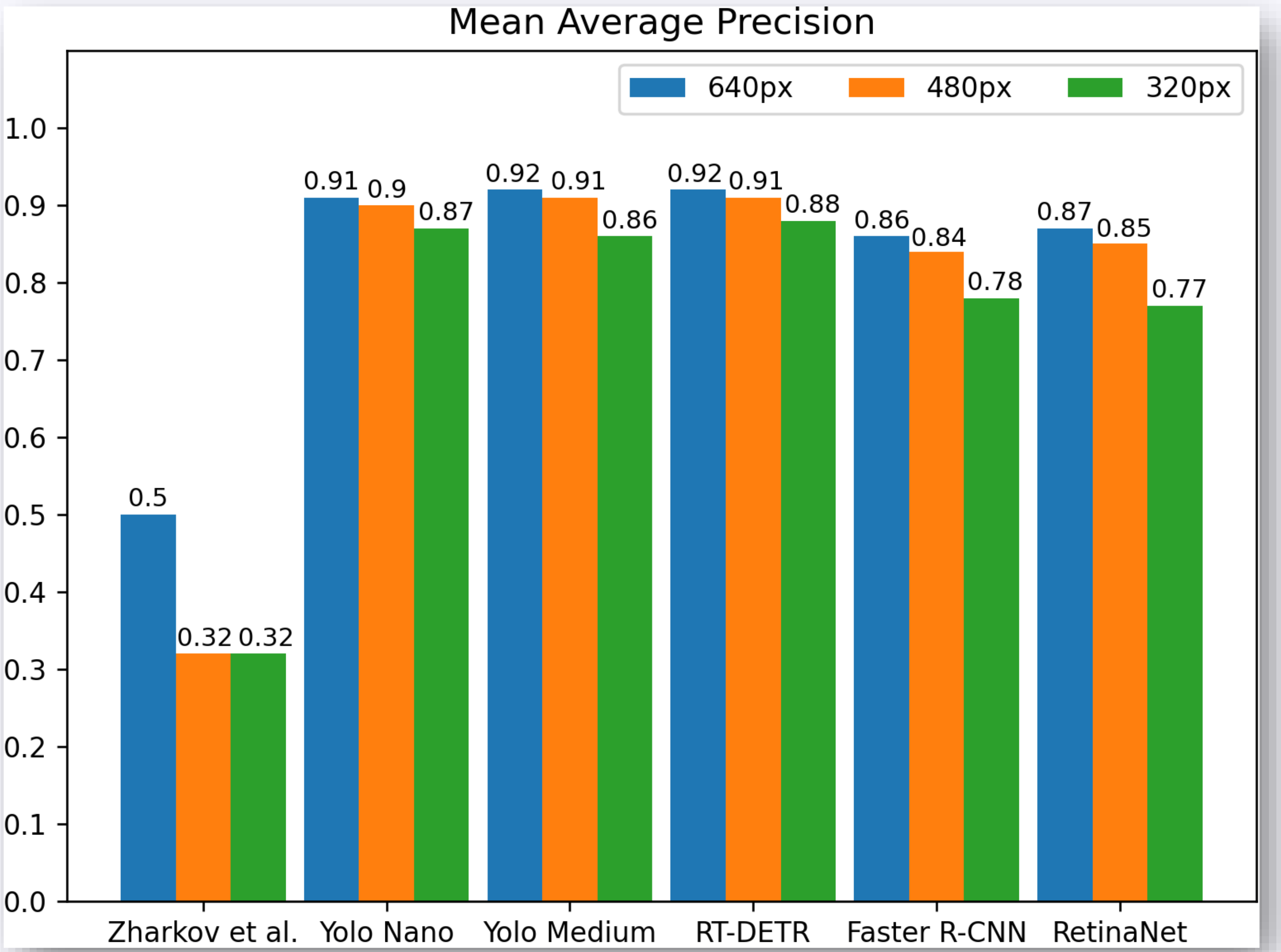
- 1D Barcodes can be located with all proposed methods.
- In this first test, only images with a single ROI are considered.
- The metric used is the F1-score considering an IoU threshold of 0.5.
- Images have been resized to have their longest edge of 640 pixels.

2D Localization



- Soros et al. is the only classical method that also supports 2D barcode detection.
- But gets an F1-score of just 0.141, and thus was not included in the graph.
- The leaderboard of deep-learning models remains more or less the same, but Faster-RCNN and RetinaNet perform better this time.

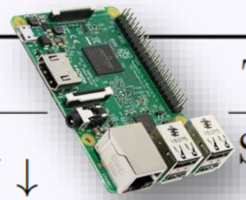
Multi-Class Detection



- Only deep-learning methods support multi-class detection.
- The test was conducted at 3 different resolutions for the longest edge of the images:
  - 640 Pixels
  - 480 Pixels
  - 320 Pixels
- The performance metric used is the mAP[0.5:0.95].

Time measurement

- Algorithms were tested on a high-end PC and a Raspberry PI 3B+. Images were scaled to have the longest edge of 640.
- This highlights significant differences in processing times based on the hardware and method used.



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

Repository Setup

- Git clone the repository.
- Install the needed C++ libraries: OpenCV, OpenCV-contrib, Boost.
- Download the dataset. Unzip the file and put the 2 folders Annotations and dataset in the repository.
- Build the repository using Cmake:

```
mkdir build
cd build
cmake ..
cmake --build .
```

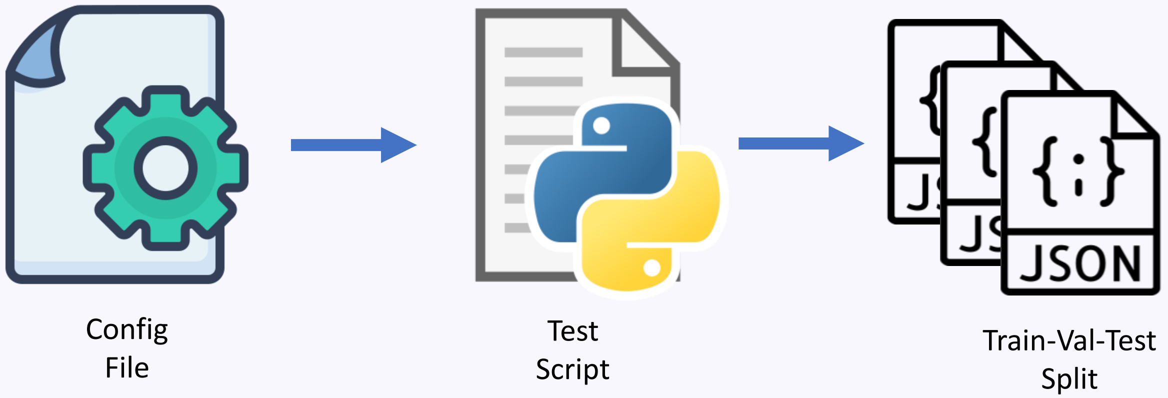
```
> algorithms
> annotations
> build
> config
> dataset
> Gallo2011-Soros2013-Yun2017
> python
> results
> scripts
> Tekin2012
> Zamberletti2013
> Zharkov2019
```

The repository should have this structure after the setup.

Converting the Annotations

- To run a test, COCO annotations must be divided into *train.json*, *val.json*, and *test.json*.
- A YAML configuration file is used to specify the annotation split settings, including the files and annotations to use, train-test split size, and whether K-fold cross-validation is applied.
- To run the annotation convertion:

```
python3 python/generate_coco_annotations.py -c
./config/generate_coco_annotations_config.yaml  -k 0
```



Single-Class and Multi-Class Localization

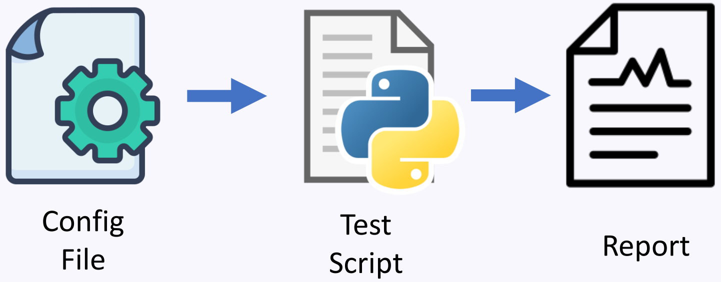
- The Python script that runs the Single-Class detection test is test_single_class.py: inputs are a configuration file and a path pointing to where the output report will be saved.

```
python3 python/test_single_class.py -c
./config/test_config.yaml -o ./results/test_results.yaml
```

- The script test_multiclass.py runs multi-class detection, taking a configuration file and an output path for the report.

```
python3 python/test_multiclass.py -c
./config/test_config.yaml -o ./results/test_results.yaml
```

- The output file will be a YAML file with all the metrics measured during the test.



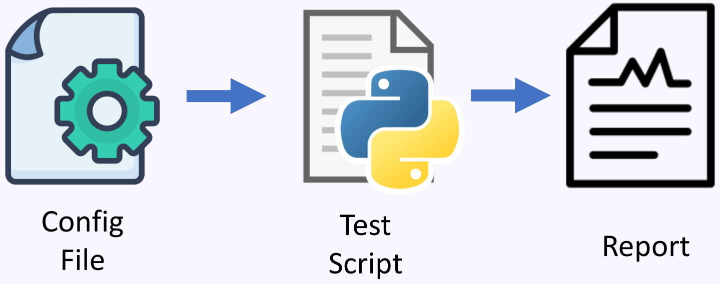

```yaml
coco_annotations_path: ./annotations/COCO/
longest_edge_resize: 640
class: 1D
single_ROI: true
bins: [-100, 0, 0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,100]
algorithms:
- args:
    imgsz: 640
    model_path: ./Saved Models/yolon_640_0.pt
  class: YOLO_detector
  library: ultralytics_detector
  name: Yolo Nano
- args:
    imgsz: 640
    model_path: ./Saved Models/yolom_640_0.pt
  class: YOLO_detector
  library: ultralytics_detector
  name: Yolo Medium
```

Example of configuration file for 1D localization.

Time Measurement

- The script *time_benchmark.py* measures the time required to run the localization algorithms.
- It is possible to measure the algorithms' performance on a single core or multiple cores as well as on GPU.
- As for the previous test scripts, the required inputs are a path to a configuration file and a path to the destination folder for the generated report.
- To run the test:

```
python3 python/time_benchmark.py -c
.config/timing_config.yaml -o ./results/timing_results.yaml
```



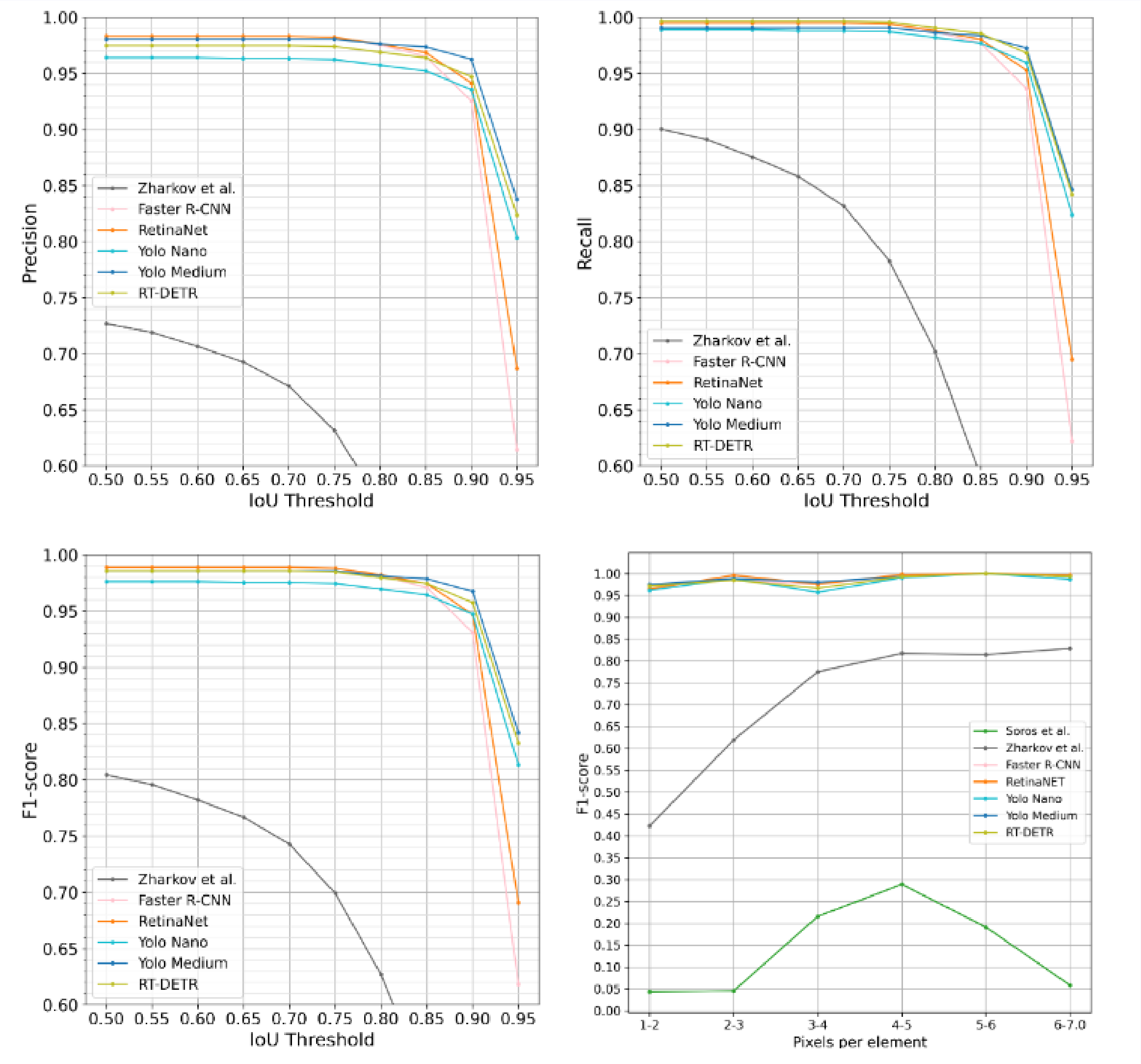

```yaml
coco_annotations_path: ./annotations/COCO/
define: &img_size 640
longest_edge_resize: *img_size
num_repeats: 3
num_threads: 1
step: 32
define: &device 'cpu'

algorithms:
- args:
    model_path: ./Saved_Models/zharkov_640_0.pt
    device: *device
  class: Zharkov_detector
  library: zharkov_detector
  name: Zharkov
```

Example of configuration file.

Graph Visualization

- The repository has two scripts for graph generation:
  - single_class_graphs.py: for single-class localization tests
  - multi_class_graphs.py: for multi-class detection tests

- Running the scripts will automatically generate .png files of the graphs.



Example of generated graphs

Test New Algorithms

- To test a new algorithm, you have to define a new file in the algorithms folder.
- Define a class with the implementation of the algorithm. To ensure compatibility, the new class should inherit from the abstract class "BaseDetector“.
- A detector must have at least these two methods: detect and get_timing.

```
# Defining the new class inside algorithms/new_algorithm.py
from detectors_abs import BaseDetector

class NewDetector(BaseDetector):
 def __init__():
   ...
 def detect(self, img):
   ...
 def get_timing(self):
   ...
```

Conclusions

BarBeR fosters innovation as an open and reliable tool for evaluating barcode detection. We invite contributions to drive advancements in automatic data capture technology.

Link to repository

Thank You!

Link to dataset