University of Verona

Bunch Detection using YOLO and FasterRCNN Deep Learning Architectures

Davide Camponogara [VR472108]

June 26, 2023

Abstract

The applications of deep learning in agriculture are many, more and more companies use artificial intelligence to improve production processes. The technological transition of these companies takes place in conjunction with increasingly better innovations in the field of machine learning. In this report we talk about object detection models, and in particular, we compare FasterRCNN[6] and YOLOv5[4], we test their performance in a real object detection problem, furthermore we analyze their training processes and test results in two different datasets.

The problem is characterized by the detection grape bunches from an image. In this report we use two datasets: the WGISD dataset[7], and the wGrapeUNIPD-DL one, created by University of Padua specifically for object detection tasks.[9] When we show the results, the power of single-stage approaches will be clear, both in terms of speed and accuracy. Source Code on GitHub: https://github.com/dadecampo/CVDeepLearning.

Contents

1 Introduction

The applications of machine learning are virtually endless, with the loads of data that are collected every day, these processes are more and more prone to perfection, the theme is increasingly in vogue, and for this reason research on the subject is constantly growing. When we talk about deep learning, we often talk about computer vision, in particular in this paper we focus on an object detection task in the agricultural field, in an area that today is in strong technological growth, also regarding artificial intelligence. When we talk about object detection, we mean the process of automatic detection of one, or more objects, inside an image, in particular, what is investigated in this report is the identification of white grape bunches. In the past there have been other researches on the same field, these have compared single-stage models like YOLO, an algorithm much more applicable in a real-time context than a two-stage model like RCNN. However, in this report we investigate what RCNN offers from the point of view of the detection of grape bunches, at the expense of the well-known computational cost due to the complexity of the model. In addition to the precision analysis, then let's compare the FPS obtained from two models: YOLOv5 and FasterRCNN.

2 Related Works

We can subdivide the deep learning based object detectors set in two different families: one-stage detectors and two-stage detectors. In this section we speak about one architecture for each of these families, YOLOv5 for the one-stage detectors family, and Faster RCNN, which belongs to the two-stage one.

Before speak about these architectures is better to introduce the differences among two-stage and one-stage architectures in a more general way. Two-stage detectors involve the presence of two main procedures: the proposal generation, where the image detector generates region proposals, and the classification phase, where, based on the proposed regions, these are classified by checking if they contain or not an object. On the other hand, one-stage detectors are simpler architectures than two-stage ones, intermediate tasks are not included in their execution, and they based their execution on regression, so instead of selecting interesting parts of an image, they predict classes and bounding boxes for the whole image in one run of the algorithm.

The arrival of Convolutional Neural Networks[1] was a game changer in deep learning research, nowadays the CNN are the foundations on which all the pattern recognition analysis is built, from object detection to action recognition.

Applying Convolutional Neural Networks, it had been possible to develop the first of the Region-based Convolutional Neural Network family of algorithms[3], from the extraction of 2000 region proposals, through the feature computation by use of Convolutional Neural Networks, ending with the classification of each region using class-specify SVMs, it was possible to improve drastically OverFeat[8], a sliding-window detector based on a similar CNN architecture.

Subsequently this algorithm had some improvements, and the Fast RCNN was born.[2] The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, the input image is fed to CNN to generate a convolutional feature map, this had improved the speed of the algorithm. Finally we arrived to Faster RCNN algorithm. Since the mechanism

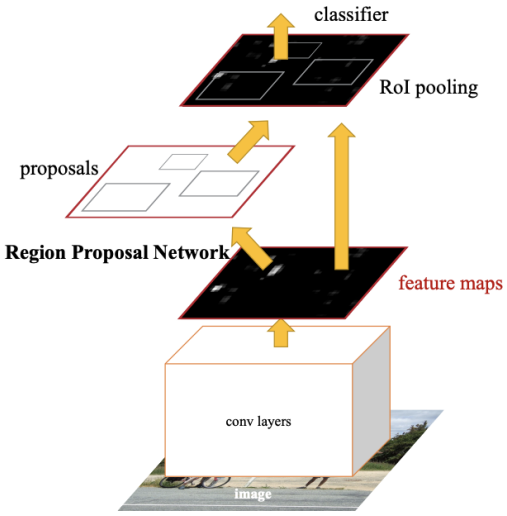

Figure 1: Faster RCNN Architecture.

of region proposal is the bottleneck of the Fast RCNN[6], then the creators introduce a Region Proposal Network (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals, nowadays we can think Fast RCNN as one of the best object detector algorithms. The Faster RCNN has two trainable layers: Region Proposal Network Layer and FasterRCNN Layer, both aims to minimize the classification loss and the bounding box regression loss, the first one loss function is:

$$\mathcal{L}\left(\{p_i\},\{t_i\}\right) = \frac{1}{N_{\text{cls}}}\sum_i \mathcal{L}_{\text{cls}}\left(p_i, p_i^*\right) + \frac{\lambda}{N_{\text{box}}}\sum_i p_i^* \cdot L_1^{\text{smooth}}\left(t_i - t_i^*\right) \tag{1}$$

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}} \tag{2}$$

where $\mathcal{L}_{\text{cls}}$ is the logarithmic loss function across the two classes. In the region proposal layer, the regions are classified into only two classes: the background, and the object.

$$L_{cls}\left(p_i, p_i^*\right) = -p_i^* \log p_i - \left(1 - p_i^*\right)\log\left(1 - p_i\right) \tag{3}$$

The variable p_i^* is the confidence target. It equals 1 when it is an object. Otherwise, it equals zero. In the COCO metrics, it is represented by loss_objectness.

For the Fast RCNN Layer there are only a few differences, the samples come from the output of the region proposal layer instead of from the anchors, the accuracy of a proposed box is based on the value of the IOU between proposed bounding boxes and the ground truth (GT). Furthermore, the classification loss of the Fast RCNN layer is defined as a multi-class loss function:

$$L_{cls}\left(p_i, p_i^*\right) = -\log\left(p_i^c\right) \tag{4}$$

The variable p_i^c is the predicted probability for the proposed region belonging to class c, that is, the GT class.[6] The total loss for the Faster R-CNN is a weighted linear combination of loss for the Fast R-CNN layer and the Region Proposal Layer.

$$\begin{aligned} \text{Loss}_{\text{FasterRCNN}} &= (\lambda_{\text{FastRCNN}} * \text{Loss}_{\text{FastRCNN}}) + \\ &(\lambda_{\text{RegionProposal}} * \text{Loss}_{\text{RegionProposal}}) \end{aligned} \tag{5}$$

where $\lambda_{\text{FastRCNN}}$ and $\lambda_{\text{RegionProposal}}$ are the weights for these two layers, and are usually set to 1.

Now it's time to talk about one-stage detectors, the most famous architecture that belongs to this family is You Look Only Once (YOLO).[5] The creators of YOLO devised as a regression problem to spatially separated bounding boxes and associated class probabilities. The algorithm is extremely fast and trainable end-to-end because of its single network structure. YOLO is particular indicated

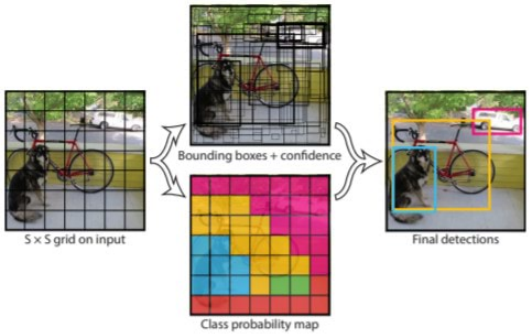

Figure 2: YOLO model.

to all the real-time applications, due to his speed. In the years following the publication of the YOLO paper several versions were released in which the

Redmon et al algorithm was improved. Now we analyze the YOLO loss function:

$$
\begin{aligned}
\text{Loss}_{\text{YOLO}} &= \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
\tag{6}
$$

where, the first term penalizes bad localization of the boxes, the second term penalizes wrong dimensions, the third and the forth losses modify confidence scores, and the last term is a classification loss. The λ terms are constants designed to unbalance the loss function more towards specific purposes.

Sozzi et al's work[9] illustrates a series of comparisons of YOLO architectures for the real time detection of white grape bunches, while Santos et al's work[7] introduces the problem of segmentation of the grape bunches through the usage of YOLOv1 and YOLOv2 algorithms.

3 Objectives

In this report, we compare Faster RCNN and YOLO architectures in a grape's bunches detection task. The objectives are to make a correct comparison between the YOLO and FasterRCNN architectures. We test their performance on two different datasets, for the Faster RCNN implementation we try different backbones, for the purpose of making comparisons among them.

The comparison between Faster RCNN and YOLO is done by analyzing the COCO metrics of the evaluation phases, we also report the qualitative results obtained directly from the detection phase on images and measurements of the model inference velocity.

We want to show how the compromises are in using one architecture instead of the other, we expect that Faster RCNN will be the most accurate algorithm between the two, but also the slowest one.

4 Methodology

Before we start talking about architectures, algorithms, or loss functions, it is better to accurately introduce the datasets we work on. The wGrapeUNIPD-DL is a dataset composed by a group of 373 images, these were acquired from lateral view in vertical shoot position vineyards in six different Italian locations

at different phenological stages. From these 373 images, 269 were then labeled in YOLO labeling format. The dataset was made available both in terms of images and labels, however, it is only one fraction of that which is used for the training phase in the correlated paper.[9] The WGISD is a dataset composed of 300 images, every one with its relative boxes in YOLO format, this consists in a txt file where every row is characterized by <object-class> <x-center> <y-center> <width> <height>, where:

- <object-class> is an integer representing the class of the object. The class index should start from 0 and increase by 1 for each unique class in the dataset, in our dataset, because of the presence of only one type of object, this value is always 0.
- <x-center> and <y-center> are the coordinates of the center of the bounding box, normalized by the width and height of the image, respectively. The values should be in the range of [0, 1].
- <width> and <height> are the width and height of the bounding box, normalized by the width and height of the image, respectively the values should be in the range of [0, 1].

We divide our dataset 80% into training images and labels and 20% test.

WGrapeUNIPD	Images	WGISD	Images
Train	216	Train	240
Test	53	Test	60
Total	269	Total	300

Table 1: Datasets subdivision.

Clearly, the creators of the Sozzi et al paper used a different dataset from our, so our aim is not to compare the results. We define 40 training epochs for YOLOv5, with a batch size of 16 and input images resized to 1216x1216, the optimizer used in this training is SGD, starting from a learning rate of 0.01.

We experiment a bit with different types of data augmentations, in the final execution of training we use the augmentations shown below: HueSaturationValue (hue_shift_limit=0.015, sat_shift_limit=0.7, val_shift_limit=0.4), Translate, Scale (p=0.9), HorizontalFlip (p=0.5), Mosaic (p=1.0). In the section on experiments we dedicate a part to the importance of data augmentations.

For the part that concerns Faster RCNN instead we tried several backbones and many different parameters, in the end the settings that gave us the most satisfaction were: ResNet50_FPN as backbone, training for 5 epochs and a very aggressive initial learning rate of 0.005, which descends through StepLR every 3 epochs with a gamma of 0.1, the optimizer used is SGD with a momentum of 0.9 and weight_decay equals to 0.0005.

Figure 3: Example of one image taken from the wGrapeUNIPD dataset, with relative boxes.

Also in this case we implement some data augmentation tactics, the same as YOLO implementation with the exception of Mosaic augmentation technique.

5 Experiments & Results

In order to implement the two architectures in the best possible way, we played a bit with parameter tuning and data augmentations. About augmentations, we compare the results obtained implementing it or not. You can see the summaries of this comparison (data collected from training on the wGrapeUNIPD dataset) in Figure 6.

Increasing the data at our disposal helps to prevent overfitting, in fact, it is evident that in training without data augmentation we arrived first at overfitting, this is marked by the rise of the obj_loss loss during evaluation. You can also see the benefits in the mAP metrics, where the evaluation which derives from the training with data augmentation has achieved better results than the one without augmentation.

We can compare the mAP values obtained on COCO evaluation using different backbones, like MobileNet, ResNet50, and VGG16, these results are represented in Figure 9. We train all the backbones in the same way, 5 epochs with a learning rate of 0.005, and a scheduler that reduces this rate to 0.0005 after 3 epochs. Clearly, backbones more complex like ResNet_101 may need more training to get better results, however, by doing various tests increasing the epochs and with different learning rates the results do not differ much. These results

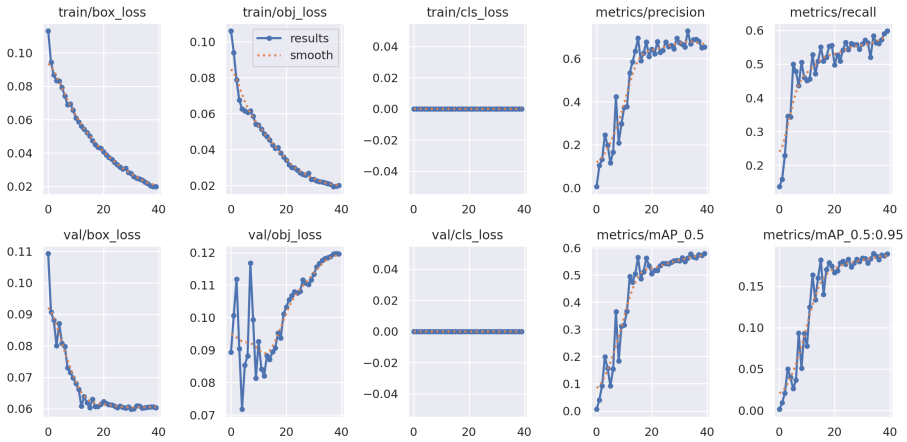

Figure 4: Without data augmentations.

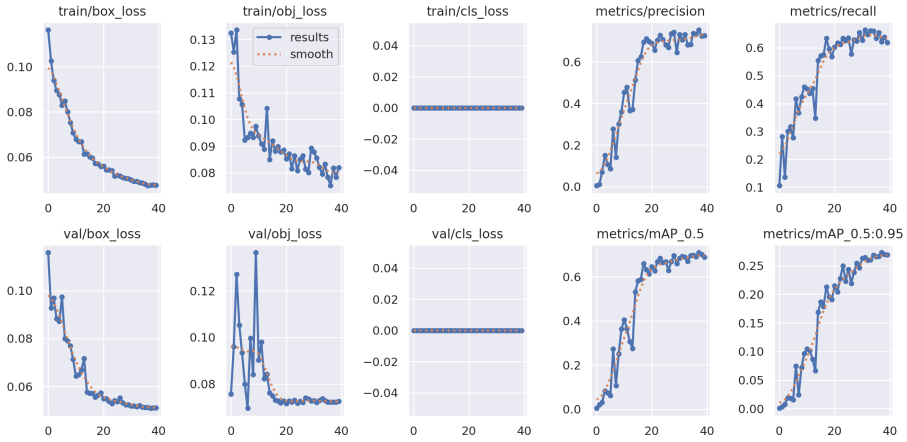

Figure 5: With data augmentations.

Figure 6: Summaries on the importance of data augmentations techniques.

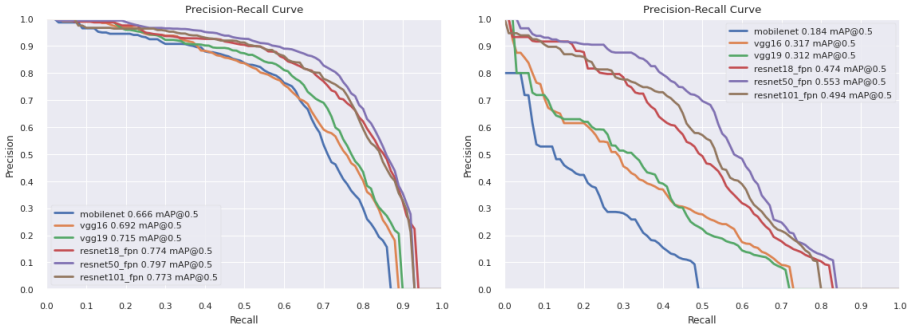

Figure 7: COCO evaluation results with different backbones, on WGISD dataset and wGrapeUNIPD dataset, from left to right: WGISD and wGrapeUNIPD.

are not particularly surprising as more complex models with higher parameters number have actually brought better results.

It is more interesting to look at the relationship between FPS and mAP obtained by FasterRCNN using the different backbones, in fact, in Figure 8 we can see the speed of inference of some backbones compared to others. For example, ResNet18 shows a very good map/FPS ratio compared to ResNet101 or ResNet50. Every test in which we have to compare the algorithms' speed is carried out on a Colab environment, with a Tesla T4 graphic card.

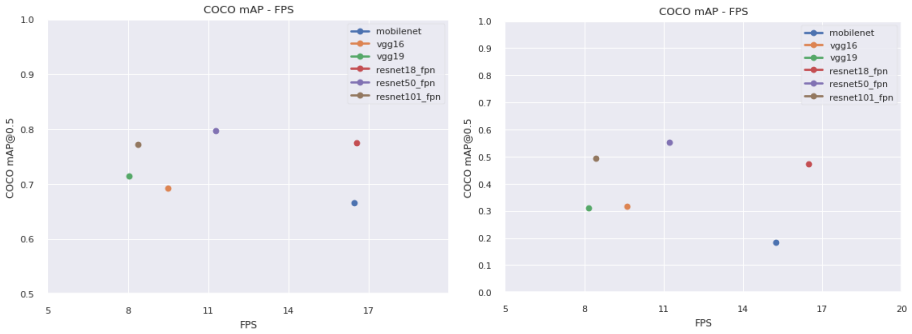

Figure 8: mAP COCO in relation with FPS for each backbone, from left to right: WGISD and wGrapeUNIPD.

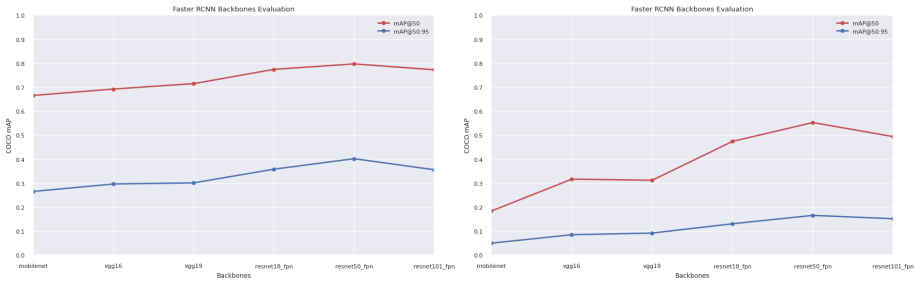

Figure 9: Comparison of mAP COCO evaluation metrics amongst different backbones, from left to right: WGISD and wGrapeUNIPD.

Speaking about YOLO, the comparison with the curve PR obtained from the training of YOLOv5 is merciless, YOLO shows a much better accuracy than the FasterRCNN one (7), in a field where we did not expect it could withstand the comparison, we show these results in Figure 10.

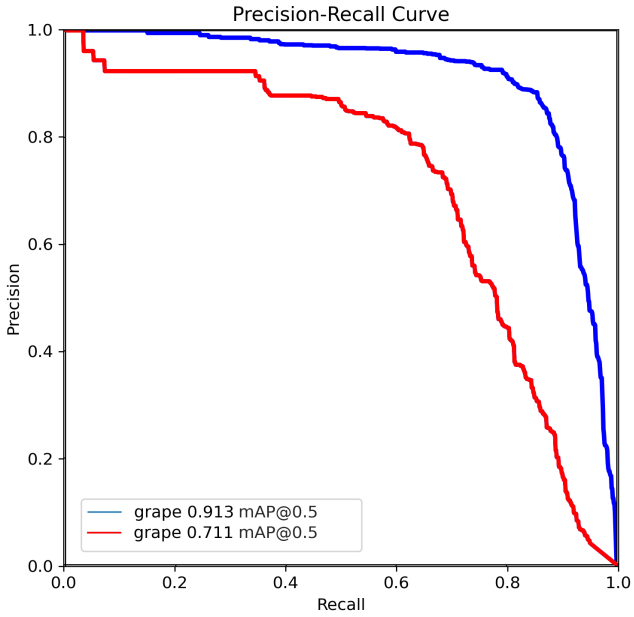

Figure 10: YOLOv5 precision-recall curves. The red curve refers to the training on wGrapeUNIPD, the blue refers to the WGISD one.

In the end, in Figure 11 and Figure 12 we show some inference results, obtained by using YOLOv5 and FasterRCNN (ResNet50_FPN backbone).

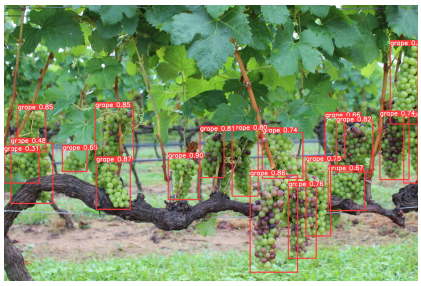

Figure 11: YOLO inference.

Figure 12: FasterRCNN inference.

In summary, the data collected on the accuracy of the algorithms and their speed of inference leave no doubt, YOLO is an approach that, at least in this case, brings great results at the price of a very low computational cost. Below, we leave a summary table 2 with all the metrics, collected for each architecture and backbone.

		WGISD Dataset			**wGrapeUNIPD Dataset**		
Model Architecture	**Backbone**	**mAP@50**	**mAP@50:95**	**FPS**	**mAP@50**	**mAP@50:95**	**FPS**
Faster RCNN	MobileNet	0.666	0.266	16.44	0.184	0.050	15.24
	VGG16	0.692	0.297	9.49	0.317	0.085	9.60
	VGG19	0.715	0.301	8.04	0.312	0.092	8.17
	ResNet18	0.774	0.358	16.53	0.474	0.131	16.47
	ResNet50	0.797	0.402	11.28	0.553	0.166	11.23
	ResNet101	0.773	0.356	8.36	0.494	0.152	8.43
YOLOv5		**0.913**	**0.604**	**19.96**	**0.711**	**0.264**	**20.20**

Table 2: Summary of collected data.

6 Conclusions

In this report we have seen how powerful single-stage approaches like YOLO can be, both in terms of accuracy and in terms of speed of inference, the results of the comparison were surprising and in such a task it was not expected that YOLO would outperform FasterRCNN in both datasets. The training of the models achieved much better results in the WGISD dataset than the wGrapeUNIPD dataset, this in fact was a predictable aspect as, even by a human eye it is not easy to recognize the grapes in the second dataset. Several backbones have been tested in order to make a comparison also on them, from this analysis the tradeoffs have been highlighted; everyone approaching deep learning must know before using one model regarding another.

7 Future Works

It would certainly be very interesting to understand in depth the reasons why these architectures perform so differently on these two datasets, in particular understand what they are, whether the light conditions, or the reduced size of the bunches in Sozzi et al, etc... the factors that induce these performance differences.

It would also be interesting to use the knowledge acquired from the WGISD training to increase the performance of the model on the wGrapeUNIPD dataset, using the Transfer of Learning. This could be done by freezing a few model layers trained on the WGISD dataset, so as to maintain some knowledge acquired to make the second training phase on wGrapeUNIPD easier.

Acknowledgements

[1] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36 (1980), pp. 193–202.

[2] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[3] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[4] Glenn Jocher et al. *ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation*. Version v7.0. Nov. 2022. DOI: 10.5281/zenodo.7347926. URL: https://doi.org/10.5281/zenodo.7347926.

[5] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[6] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[7] Thiago Santos et al. *Embrapa Wine Grape Instance Segmentation Dataset – Embrapa WGISD*. Version 1.0.0. The building of the WGISD dataset was supported by the Embrapa SEG Project 01.14.09.001.05.04, Image-based metrology for Precision Agriculture and Phenotyping, and the CNPq PIBIC Program (grants 161165/2017-6 and 125044/2018-6). Zenodo, July 2019. DOI: 10.5281/zenodo.3361736. URL: https://doi.org/10.5281/zenodo.3361736.

[8] Pierre Sermanet et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint arXiv:1312.6229* (2013).

[9] Marco Sozzi et al. *wGrapeUNIPD-DL: an open dataset for white grape bunch detection*. Zenodo, May 2022. DOI: 10.5281/zenodo.6757555. URL: https://doi.org/10.5281/zenodo.6757555.